

CA-ADS[®]

Application Design Guide

15.0



Computer Associates

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

THIS DOCUMENTATION MAY NOT BE COPIED, TRANSFERRED, REPRODUCED, DISCLOSED, OR DUPLICATED, IN WHOLE OR IN PART, WITHOUT THE PRIOR WRITTEN CONSENT OF CA. THIS DOCUMENTATION IS PROPRIETARY INFORMATION OF CA AND PROTECTED BY THE COPYRIGHT LAWS OF THE UNITED STATES AND INTERNATIONAL TREATIES.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.

THE USE OF ANY PRODUCT REFERENCED IN THIS DOCUMENTATION AND THIS DOCUMENTATION IS GOVERNED BY THE END USER'S APPLICABLE LICENSE AGREEMENT.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227.7013(c)(1)(ii) or applicable successor provisions.

First Edition, December 2000

© 2000 Computer Associates International, Inc.
One Computer Associates Plaza, Islandia, NY 11749
All rights reserved.

All trademarks, trade names, service marks, or logos referenced herein belong to their respective companies.

Contents

About This Guide	vii
Chapter 1. Introduction	1-1
1.1 Overview	1-3
1.2 Application Guidelines	1-4
1.3 Tools for Designing and Developing Applications	1-5
1.3.1 CA-ADS Application Compiler (ADSA)	1-5
1.3.2 CA-ADS Dialog Compiler (ADSC)	1-6
1.3.3 CA-ADS Runtime System	1-6
1.3.4 IDD	1-7
1.3.5 CA-IDMS/DC Mapping Facility	1-8
1.3.6 Batch and Online Reporting Facilities	1-8
1.4 The Design and Development Team	1-11
Chapter 2. Design Methodology	2-1
2.1 Development of Effective Design	2-3
2.2 Step One: Analyzing the Problem	2-5
2.3 Step Two: Developing the Design	2-6
2.3.1 External/Functional Specifications	2-6
2.3.2 Internal/Technical Specifications	2-10
2.4 Step Three: Building a Prototype	2-12
2.4.1 Uses for the Prototype	2-12
2.4.2 Unique Features of the Prototype	2-13
2.4.3 Creating the Prototype	2-13
2.5 Step Four: Writing Process Code for the Dialogs	2-15
2.5.1 Writing the Dialog Specifications	2-15
2.5.2 Writing the Source Code	2-18
2.6 Step Five: Testing and Implementing the Application	2-24
2.6.1 Test Plan	2-24
2.6.2 Test Procedures	2-24
2.7 Underlying Issues	2-26
2.8 Data Definition and Database Design	2-28
Chapter 3. Building a Prototype	3-1
3.1 Three-stage Approach	3-3
3.2 Stage I: Building the Basic Prototype	3-4
3.2.1 Compiling the Application (ADSA)	3-4
3.2.2 Compiling the Maps	3-5
3.2.3 Compiling the Dialogs (ADSC)	3-5
3.2.4 User Review	3-6
3.3 Stage II: Adding Process Logic and Data Retrieval	3-7
3.3.1 ADSA Enhancements	3-7
3.3.2 Populating the Dictionary	3-7
3.3.3 CA-IDMS Mapping Facility Enhancements	3-8
3.3.4 ADSC Enhancements	3-8
3.4 Stage III: Refining the Maps and Processes	3-9

Chapter 4. Designing Maps	4-1
4.1 Attributes of Successful Maps	4-3
4.2 Design Standards for a Dialog Map	4-4
4.3 Online Mapping Procedures	4-5
4.4 Choosing Menu Maps	4-6
4.4.1 System-Defined Menu Maps	4-6
4.4.2 User-Defined Menu Maps	4-6
4.5 Designing Dialog Maps	4-9
Chapter 5. Designing Dialogs	5-1
5.1 Overview	5-3
5.1.1 Dialog Level	5-3
5.1.2 Dialog Status	5-4
5.1.3 Dialog Control	5-5
5.2 Design Considerations	5-7
5.2.1 Record Buffer Management	5-7
5.2.2 Working Storage Areas	5-9
5.2.3 Global Records	5-10
5.3 Dialogs That Issue Navigational DML	5-17
5.3.1 Database Currencies	5-17
5.3.2 Extended Run Units	5-18
5.3.3 Longterm Locks	5-19
5.3.4 Record Buffer Management for Logical Records	5-20
Chapter 6. Naming Conventions	6-1
6.1 Overview	6-3
6.2 Naming Application Entities	6-4
6.3 Naming Database Information Entities	6-7
Chapter 7. Performance Considerations	7-1
7.1 Overview	7-3
7.2 System Generation Parameters	7-4
7.2.1 ADSO Statement Parameters	7-4
7.2.2 PROGRAM Statement Parameters	7-4
7.2.3 TASK Statement Parameters	7-4
7.2.4 Allocating Primary and Secondary Storage Pools	7-5
7.2.5 Setting the Fast Mode Threshold	7-6
7.2.6 Specifying the Number of Internal and External Run Units	7-7
7.3 Resource Management	7-8
7.3.1 Monitoring Resource Consumption	7-9
7.3.2 Conserving Resources	7-14
Appendix A. Application Concepts	A-1
A.1 Overview	A-3
A.2 Application Components	A-4
A.2.1 Functions	A-4
A.2.2 Responses	A-6
A.3 Dialog Features	A-7
A.3.1 Dialog Components	A-7
A.3.2 Dialog Procedures	A-8
A.4 Control Commands	A-9

A.5 The Flow of Control	A-11
Glossary	X-1
A	X-1
B	X-1
C	X-2
D	X-2
E	X-3
F	X-3
G	X-3
I	X-4
K	X-4
L	X-4
M	X-4
O	X-5
P	X-5
Q	X-5
R	X-6
S	X-7
T	X-7
U	X-7
V	X-8
Index	X-9

About This Guide

What this manual is about

This document applies to CA-ADS 15.0 and should be used with existing CA-ADS documentation.

Who should use this manual

This manual is designed for those individuals responsible for designing and developing online applications in a CA-ADS environment. The information in this manual applies regardless of the database access language used by the application (that is, navigational DML or SQL DML).

How this manual is organized

This manual has seven chapters, an appendix, and a glossary, as follows:

- **Introduction (Chapter 1)** — A general presentation of the design process, including a discussion of design guidelines, the tools available for developing application designs, and the roles that need to be filled by members of the design and development team
- **Design Methodology (Chapter 2)** — Discussion of the five steps that can be used in the design and development of applications and an overview of the necessary procedures for defining data
- **Building a Prototype (Chapter 3)** — Instructions for building a prototype application
- **Designing Maps (Chapter 4)** — Discussion of screen design considerations, mapping procedures, and the choices available when designing menu maps
- **Designing Dialogs (Chapter 5)** — Discussion of dialog design considerations, including database currencies, record buffer management, and the use of work areas
- **Naming Conventions (Chapter 6)** — Suggestions for the establishment of standardized naming conventions
- **Performance Considerations (Chapter 7)** — Discussion of the system generation parameters that affect or can be affected by CA-ADS applications, and the monitoring and conservation of system resources in a CA-ADS environment
- **Application Concepts (Appendix A)** — Overview of application terms and concepts in a CA-ADS environment
- **Glossary** — A glossary of the terms and abbreviations associated with the design and development of applications in the CA-ADS environment

Related documentation

- *CA-ADS Reference*
- *CA-ADS User Guide*
- *CA-IDMS SQL Programming*
- *CA-IDMS Reports*
- *IDD DDDL Reference*
- *CA-IDMS Database Design*
- *CA-IDMS System Generation*
- *CA-IDMS System Operations*
- *CA-IDMS Mapping Facility*
- *CA-OLQ User Guide*

Chapter 1. Introduction

- 1.1 Overview 1-3
- 1.2 Application Guidelines 1-4
- 1.3 Tools for Designing and Developing Applications 1-5
 - 1.3.1 CA-ADS Application Compiler (ADSA) 1-5
 - 1.3.2 CA-ADS Dialog Compiler (ADSC) 1-6
 - 1.3.3 CA-ADS Runtime System 1-6
 - 1.3.4 IDD 1-7
 - 1.3.5 CA-IDMS/DC Mapping Facility 1-8
 - 1.3.6 Batch and Online Reporting Facilities 1-8
- 1.4 The Design and Development Team 1-11

1.1 Overview

This manual is designed for those individuals responsible for designing and developing online applications in a CA-ADS environment. A methodology is presented that covers the design process and the implementation of a design in an application prototype.

Separate chapters discuss design features to be considered when creating the maps and dialogs that are an integral part of the application. Also included is a discussion of factors to be considered when defining data for the application and when establishing the application database.

To benefit fully from the materials presented, the reader should be knowledgeable about CA-IDMS and have experience writing dialogs in a CA-ADS environment. CA-ADS concepts that are basic to creating applications are summarized in Appendix A, "Application Concepts." Additional concepts are reviewed throughout the manual and, where appropriate, the reader is referred to other CA documentation for further information. A glossary is included as a resource for any readers who might be unfamiliar with the CA-ADS terminology.

This introductory chapter covers the following topics:

- Application guidelines
- Tools for designing and developing applications
- The design and development team

Each of these topics is discussed below.

1.2 Application Guidelines

The following guidelines should be considered when developing an application:

- **User needs** — An application must satisfy the requirements of the user. To accomplish this goal, the developer must consult frequently with the user, remembering that all ramifications of an application are often not apparent in the initial stages of development. Additionally, specifications may be subject to change as the user reacts to the prototype application, or as new aspects of the application become evident. A successful application requires strong user involvement throughout the design process.
- **Human factors** — A user-friendly application increases productivity. An application should be designed so that the end user feels capable of responding, knows how to proceed after each step, and knows how to get assistance if there is any confusion. The screens should be straight-forward, uncomplicated, and uncluttered.
- **Flexibility** — An application must be easy to maintain and modify. The structured design methods used by the CA-ADS Application Compiler (ADSA) help the developer to accomplish this goal in the following ways: short, compact modules are used to perform the given functions; and the code that performs the processing logic is kept separate from the information about data (for example, format of records and elements, editing criteria). The implementation of naming, coding, and map formatting standards is strongly recommended, both for purposes of maintenance as well as for future enhancements of the application.
- **Performance** — The ultimate test of a design lies in its performance capabilities. The measures of what constitutes good performance are site-specific and vary with the needs and expectations of the user. Optimally, a good design should have acceptable throughput, should have reasonable response times, and should use the available resources as efficiently as possible.

1.3 Tools for Designing and Developing Applications

The following tools are available for designing, developing, and implementing applications in the CA database environment:

- CA-ADS Application Compiler (ADSA)
- CA-ADS Dialog Compiler (ADSC)
- CA-ADS runtime system
- IDD (Integrated Data Dictionary)
- CA-IDMS/DC Mapping Facility
- Batch and online reporting facilities

Each of these design and development tools is discussed below.

1.3.1 CA-ADS Application Compiler (ADSA)

An application can be defined and compiled by using the CA-ADS Application Compiler. ADSA also serves as a design tool and an automatic prototyping tool for the CA-ADS application developer.

Facilitates structured application planning: As a design tool, ADSA facilitates structured application planning at an early stage in the design process. When the basic application design has been resolved, the developer initiates an application compilation session and defines the application functions and responses (the application components) to the dictionary.

At any stage, the developer can query the dictionary as to the status of the design by using CA-IDMS dictionary reports, CA-OLQ, or IDD to access the definitions. Even if an application compilation session is suspended (that is, the application is not compiled), the dictionary still contains the component definitions and relationships defined up to this point.

Provides online preview: As a prototyping tool, ADSA enables the user to have an online preview of what the application looks like and what it can do. These walk-throughs can begin at an early stage in the design, before any process code needs to be written. To compile a prototype and create the appropriate load modules, ADSA only needs the dictionary definitions of any global records associated with an application; if no global records are specified, then no other definitions are necessary. To execute a prototype, only rudimentary dialogs and maps are required. Prototypes are readily modified and, therefore, can respond quickly to the needs of the user as the application design is being developed. Once the final design is approved, the existing prototype is enhanced with the requisite dialog code, and the completed application can be executed.

1.3.2 CA-ADS Dialog Compiler (ADSC)

Compiles dialogs: Dialogs are defined and compiled using the CA-ADS Dialog Compiler (ADSC). In a ADSC session, the application developer uses a series of screens to provide CA-ADS with information such as the dialog's name, subschema, maps, work records, and premap and response processes. Once the dialog has been compiled successfully, it is stored as a load module in the dictionary for use by the CA-ADS runtime system.

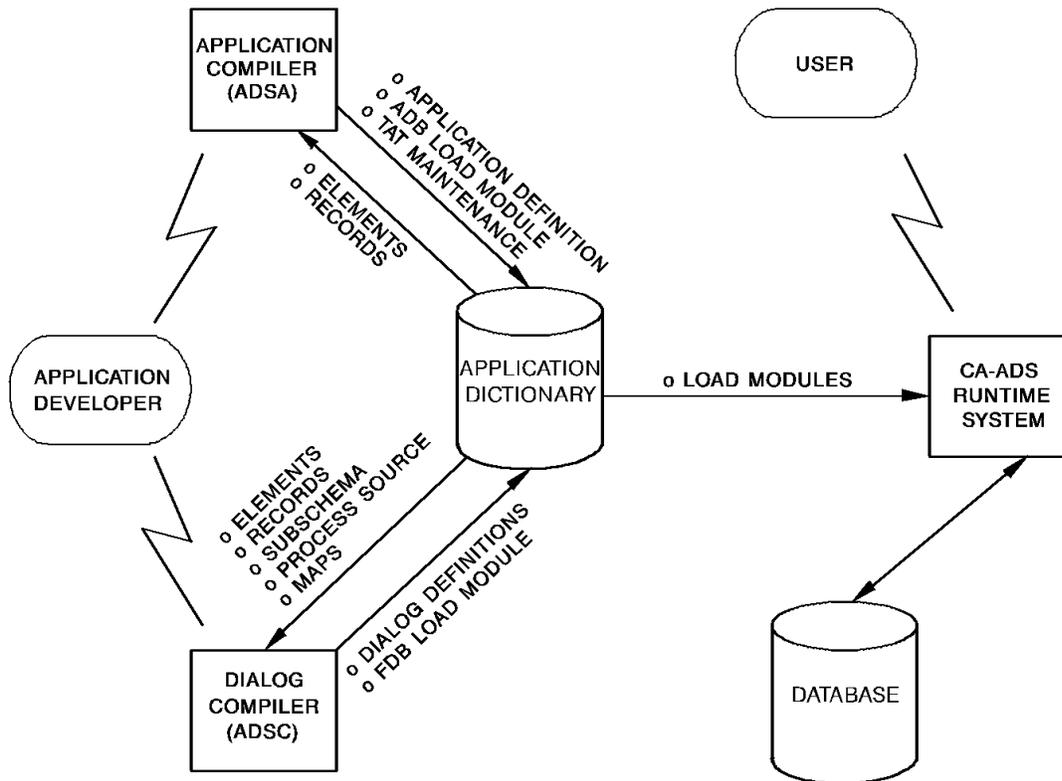
1.3.3 CA-ADS Runtime System

An application can be executed after the user signs on to the DC/UCF system and uses the necessary task code to initiate the CA-ADS runtime system. This task code will either display the CA-ADS menu screen or will begin executing a predefined dialog. The menu screen contains the list of available mainline dialogs that can be selected by the user.

Creates record buffers, control blocks: During dialog execution, the CA-ADS runtime system dynamically creates record buffers for the subschema and dictionary records used by the dialog, and automatically initializes each field in the newly created buffers. The runtime system also creates control blocks that provide information pertaining to the executing dialog's map and database access activities. The application can include process code to test certain fields in these control blocks and specify the action to be taken, based on the test outcome.

At runtime, the sequence of events is controlled by the user's selection of processing. The following figure shows the interrelationships of the CA-ADS Application Compiler (ADSA), the CA-ADS Dialog Compiler (ADSC), and the CA-ADS runtime system.

CA-ADS



Accesses record, element definitions: The CA-ADS Application Compiler accesses record and element definitions stored in the dictionary. ADSA supplies the dictionary with the application definition; the updated Task Activity Table (TAT), the DC/UCF load module that associates task codes and the invoked tasks; and the Application Definition Block (ADB), the application load module. The CA-ADS Dialog Compiler (ADSC) accesses record, element, subschema, map, and source process definitions stored in the dictionary. ADSC supplies the dictionary with the dialog definition and with the Fixed Dialog Block (FDB), the dialog load module. When the application is executed, the CA-ADS runtime system accesses the application, dialog, map, subschema, and edit and code table load modules stored in the dictionary.

1.3.4 IDD

Central repository: The Integrated Data Dictionary (IDD) acts as a central repository of information about data. The developer uses the dictionary to store definitions of the application's data elements, records, tables, and maps, as well as the processing modules associated with an application. IDD maintains information about the data stored in the application database and makes this information directly available to the applications. IDD comprises the dictionary itself (that is, the repository of information about data) and software components for accessing (that is, adding, modifying, deleting, and displaying) the dictionary-stored information.

IDD enables batch and online entry and examination of entity definitions stored in the dictionary.

For example, the application designer can request the display of an element definition, a record definition, or a user-defined entity (a site-specific data category defined by the DBA). The displayed information shows the definition of the entity itself as well as contextual information.

►► For information on how data can be defined for an application, see Chapter 2, “Design Methodology” on page 2-1.

1.3.5 CA-IDMS/DC Mapping Facility

Defines screen layout: The CA-IDMS/DC mapping facility is used to define the layout of the terminal screens (that is, maps) used for communication between the application and the user. A map definition, in addition to determining the appearance of the screen, associates fields on the screen (map fields) with record elements in the data dictionary, and defines display attributes (such as color and intensity) for map fields. All map definitions are stored in the dictionary.

Because maps are defined in the dictionary as separate entities, a CA-ADS dialog can use a map simply by naming it on the appropriate CA-ADS Dialog Compiler screen; the dialog itself does not perform any screen formatting.

At runtime, the mapping facility can perform automatic editing and error handling. When these facilities are enabled, input is validated automatically and output is formatted on the basis of dictionary-stored information on record elements (that is, internal picture, external picture, edit table, and code table). When a map is defined, the developer can specify different editing criteria for any field. The developer can also define stand-alone edit and code tables as modules in the dictionary. During map generation, these tables can be associated with map fields and external pictures can be defined for the fields.

►► For further information on the CA-IDMS Online Mapping Facility and the automatic editing and error-handling capabilities available to the application, refer to *CA-IDMS Mapping Facility*.

1.3.6 Batch and Online Reporting Facilities

This section describes reporting capabilities that are available to the designer for assistance throughout the development process.

CA-IDMS reports: CA-IDMS provides an extensive series of standard reports on information stored in the dictionary. These include summary, detail, and key reports of the elements and records in the dictionary. Reports are also available for dialogs and applications, and their associated components. Dictionary reports comprise a valuable tool for finding inconsistencies and redundant element types.

►► For more information, refer to *CA-IDMS Reports*.

Subschema compiler: The subschema compiler enables batch and online examination of subschema definitions.

►► For further information on the use of the subschema compiler, refer to *CA-IDMS Database Administration*.

IDMSRPTS utility: The IDMSRPTS utility provides a series of reports on database definitions (for example, schema definitions, logical record definitions).

►► For further information, including a complete list of the reports available with the IDMSRPTS utility, refer to *CA-IDMS Utilities*.

CA-OLQ: CA-OLQ is an online system for interrogating an CA-IDMS/DB, and for displaying and formatting the resulting information at a terminal. CA-OLQ accommodates ad hoc queries. With the use of q-files (CA-OLQ modules stored in the dictionary), users can obtain formatted reports at the terminal simply by supplying the name of the desired q-file.

►► For further information on using CA-OLQ to query the dictionary, and on the storing and accessing of q-files, refer to *CA-OLQ User Guide*.

CA-CULPRIT: CA-CULPRIT is a parameter-driven system for generating batch reports. CA-CULPRIT actively uses dictionary-stored element, record, and subschema definitions. Reports can be packaged and stored as CA-CULPRIT modules in the dictionary, enabling users to obtain a report simply by supplying the module name.

►► For further information on the use of CA-CULPRIT as an application reporting tool, refer to *CA-CULPRIT User Guide*.

The dialog reporter (ADSORPTS utility): The CA-ADS dialog reporter utility, ADSORPTS, is available to request batch reports that provide summary and/or detailed information about one or more dialogs. Reports can include: information on the records and processes of the named dialogs; a list of the contents of the Fixed Dialog Block (FDB); and a summary that includes map, schema, subschema, and version number information.

►► For further information on the ADSORPTS utility, refer to *CA-OLQ Reference*.

The DC/UCF map utility: In addition to generating and deleting map load modules, and producing map source code, the map utility provides mapping reports. These reports display the decompiled source code, a list of the attributes assigned to each map field, and a list of the records used by the named map.

►► For further information on using the map utility, refer to *CA-IDMS Mapping Facility*.

1.4 The Design and Development Team

Personnel: The personnel involved in the development of an application reflect the range of responsibilities involved in the creation of a successful design. The manner in which these responsibilities are assigned varies widely from installation to installation, with some individuals often assuming more than one role.

The remainder of this chapter discusses the roles that should be included in the team that develops an application.

Project leader: The role of the project leader is to orchestrate and coordinate the project. The project leader is ultimately responsible for producing the system to specifications and on time.

DBA/DCA: Both the database administrator (DBA) and the data communications administrator (DCA) should be involved in the design and development process, particularly from the aspect of maintaining consistent site-specific standards. The DBA is responsible for the data resources (that is, the application database and the dictionary), designing and implementing the database records, defining the logical records, and establishing naming conventions and data dictionary standards. The DCA is involved in the network and communication needs, helping to plan for space requirements, performance, and system tuning.

Data administrator: The data administrator interfaces with all members of the design and development team, running any reports that are needed as well as populating the dictionary. The data administrator is also responsible for enforcing the standards and conventions laid out by the DBA, entering the dictionary elements, records, maps, and edit and code tables as needed for the application.

Systems analyst: The systems analyst helps analyze and document the needs of the end users. The analyst often works with the data administrator and also with the DBA in designing the database. Additionally, the analyst defines the requirements for the applications that will access the database.

Programmers: Application programmers write the processing logic that accesses the database, interpreting the dialog requirements given to them by analysts and designers. Working from design specifications, the programmer determines map data fields, field edits, map and work record elements, and the messages needed for a given dialog. This information is then submitted to the data administrator for approval and, subsequently, for inclusion in the dictionary. The dialog source code is written and stored in modules in the dictionary.

End users: The end users of the application provide valuable input to the data administrator, DBA, systems analyst, and application programmers. They define what their present data needs are and try to predict future needs. There should be constant interaction between the end users and the other members of the development team, to ensure maximum usefulness of the applications developed.

Chapter 2. Design Methodology

- 2.1 Development of Effective Design 2-3
- 2.2 Step One: Analyzing the Problem 2-5
- 2.3 Step Two: Developing the Design 2-6
 - 2.3.1 External/Functional Specifications 2-6
 - 2.3.2 Internal/Technical Specifications 2-10
- 2.4 Step Three: Building a Prototype 2-12
 - 2.4.1 Uses for the Prototype 2-12
 - 2.4.2 Unique Features of the Prototype 2-13
 - 2.4.3 Creating the Prototype 2-13
- 2.5 Step Four: Writing Process Code for the Dialogs 2-15
 - 2.5.1 Writing the Dialog Specifications 2-15
 - 2.5.2 Writing the Source Code 2-18
- 2.6 Step Five: Testing and Implementing the Application 2-24
 - 2.6.1 Test Plan 2-24
 - 2.6.2 Test Procedures 2-24
- 2.7 Underlying Issues 2-26
- 2.8 Data Definition and Database Design 2-28

2.1 Development of Effective Design

There are a number of ways to approach the design of a CA-ADS application. This chapter provides information that may be useful to those involved in the development stage of an effective design. This chapter also presents information on how data is defined and stored in the CA-IDMS/DB environment.

Note that the procedures presented in this manual represent one possible approach to a design and should be used as a guideline. Application developers must determine their system's specific needs and the design procedures that will best meet those needs.

Three phases: The sample approach to application design methodology that is presented throughout this manual comprises the following three phases:

1. **Data definition** — The DBA and the systems analyst determine what element types the application needs. After defining the elements in the dictionary, the DBA then determines how the elements should be grouped into records and defines the records in the dictionary. As a result of this phase, the dictionary is populated with the element and record definitions required by the schema and subschema definitions, and with the application dialogs.
2. **Database design and definition** — The project leader, with the help of the DBA, designs and defines the application database, creating a schema that reflects the data access needs of the application system as a whole (that is, all the programs in the application system); subschemas are then developed that reflect the data access needs of a specific application. The database design and definition phase also deals with the physical structure of the database (that is, how the database exists on disk storage). As a result of this phase, the schema, DMCL, and subschema are defined in the dictionary.
3. **Application design and development** — The application development group designs and develops the applications. Dialogs are written using CA-ADS process code, and dialog maps are created with the DC/UCF mapping facility. The CA-ADS process code can link to routines written in source languages such as COBOL, PL/I, or Assembler. As a result of this third phase, applications exist in a form that end users can execute.

How tasks are performed: These phases can be implemented in chronological sequence, but they usually overlap, because certain tasks within each phase can be performed concurrently.

For example, an application prototype can be defined and executed while the database is being designed and data is being defined in the dictionary. However, each phase must be completed before the next phase can be fully implemented.

Five-method design: The design method proposed in this manual is organized into the following five steps:

1. Analyzing the problem
2. Developing the design

3. Building a prototype
4. Writing process code for the dialogs
5. Testing and implementing the application

These steps are discussed below, followed by a presentation of issues that underlie the entire design process.

2.2 Step One: Analyzing the Problem

Team approach: Problem analysis involves defining end-user needs and agreeing upon the functional requirements of the application. To generate an effective application, it is essential to have the users involved as members of the team throughout the entire design and development process.

You should seek information: During this stage, the team seeks answers to questions that help define the need for the application. Information regarding the following is required:

- Who is the end user?
- What departments use these transactions?
- Who performs a given activity?
- What data does the user need to see?
- What activities need to be automated?
- How is the activity usually performed?
- What information is referenced by these activities?
- Where is the output information used?
- What improvements are anticipated?
- What types of reports will be needed? When are reports usually run?
- How often will the application be used? By how many?

Develop two lists: In the process of analyzing the problem, the following lists are developed:

- Lists of activities that the user wants to be able to perform
- Lists of information available to or necessary for the identified activities

2.3 Step Two: Developing the Design

DBA incorporates related data: In the second step, a design is created to meet the needs that have been identified. During the actual design process, information begins to fall into groups of related data that can be incorporated by the DBA into dictionary elements, records, schemas, subschemas, and logical records. At the same time, activities combine into predictable functions (for example, update, modify, delete) that logically work together and begin to form a step-by-step design.

When developing a design, the application and development group must consider the external/functional specifications and the internal/technical specifications. The external/functional specifications reflect the user's view of the application, indicating what functions will be performed by the application; the internal/technical specifications reflect the developer's view of the application, indicating how the application will operate. Each of these considerations is discussed separately below.

For the purposes of this manual, the discussion of the specifications assumes that the database has already been designed and that subschema views and other site-specific information have been obtained from the DBA.

2.3.1 External/Functional Specifications

You should choose format: Decisions need to be made about the format of the intended application. The developer must decide what activities will take place and the response choices that will be available to the user at each stage of the application.

Once the application components have been developed, it is helpful to develop a structural chart that depicts the application graphically. Finally, the design details need to be documented. Each of these stages is discussed below.

Identifying the application components: The following list suggests a few of the questions that need to be answered to establish relationships within and between the functions and responses that make up an application:

- What online transactions need to be performed by the terminal user? For example, in the sample application, the user needs to be able to update the address, phone number, job code, or skill level of an employee.
- What information or processing is needed before a given function can be implemented? For example, the appropriate employee record needs to be obtained from the database and displayed online before the record can be modified by the terminal user.
- What are the possible results of a given function? For example, when the user chooses to update a record, will it be possible to delete the displayed record or can the record only be modified and stored?
- After completing a function, what should be the next step? For example, will the application return to the menu screen after the employee record has been updated

or will a new employee record be displayed? What response will the user have to make to effect either of these actions?

- What relationships can be established between functions? For example, can the same map be used for both the update and browse functions?
- How do these parts relate to the available or planned database entities? For example, is there a record in the database that provides information on the skill of an employee? If an employee has more than one skill or many employees have the same skill, will the application be able to access this information?

Developing a structural diagram: At this point in the design, it is useful to develop a graphic representation of the application, identifying the functions and responses, and incorporating them in a structural diagram that illustrates their interrelationships.

In addition to identifying the functions and responses of the application, the developer needs to be concerned with the following design items:

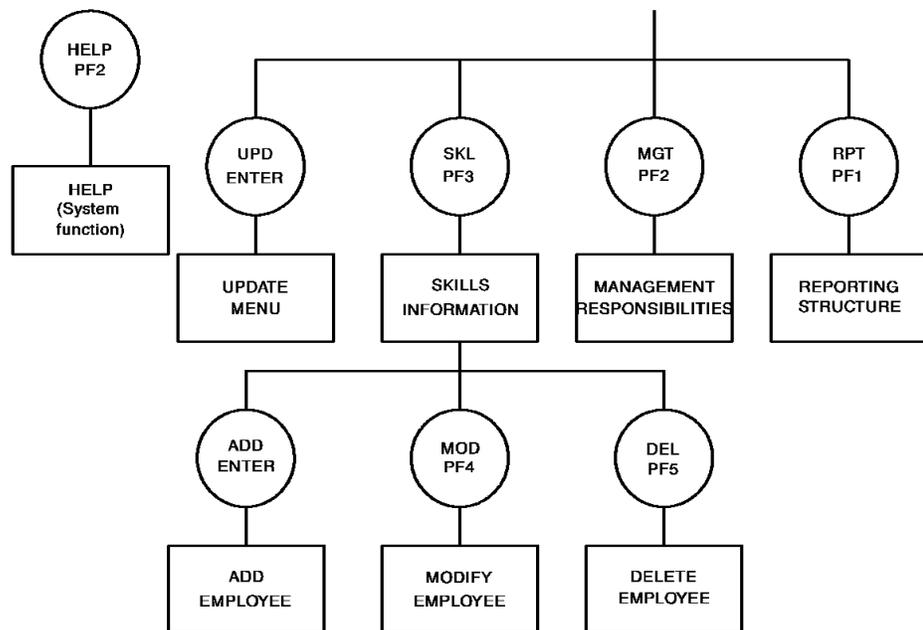
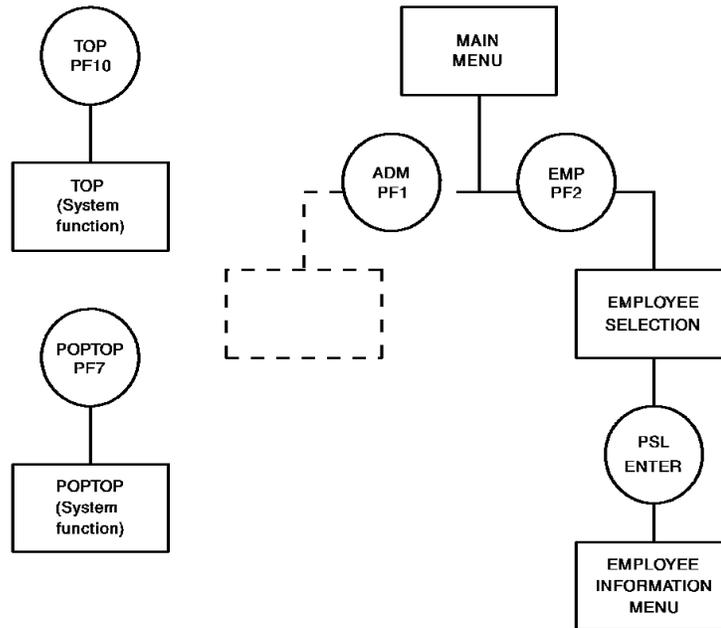
- The number of levels the application will contain.
- The commands that will be used to pass control between dialogs.
- The system-provided functions (for example, POP, POPTOP, QUIT) that will be incorporated into the design.
- The assignment of function keys and response codes.

The diagram presents one way in which the developer can begin to sketch out the application and graphically depict the flow between functions and responses. The management information system being developed in this sample diagram has administrative and personnel applications; only the personnel application is represented in the flowchart. The user begins by selecting an application from the main menu. After obtaining the record on a particular employee, the user can select the appropriate response from the employee information menu to add, modify, and display the skills of the employee; obtain information on employee rank within the company's organizational structure; and update the personnel data on the employee.

Returning to the main menu: At any point, the user can use the system functions defined for this application to return to the main menu (POPTOP); display a screen that supplies the valid responses for the current function (HELP); or return to the previous function (TOP).

Before proceeding to the next step in the design and development of the application, the flowchart should be reviewed with end users and modified as necessary.

The following diagram illustrates the partial structuring of a sample management information system. The circles in the flowchart represent the application responses and the rectangles represent the functions. Within each circle is the response code and control key that will be defined to initiate the given function (for example, SKL/PF3 will initiate the display of information on employee skills). The system functions to be used in this application (that is, HELP, TOP, and POPTOP) are indicated.



Documenting the design: When the user approves the basic design, the developer needs to document the details of that design. The application worksheets are examples of the types of forms that can be used to document the dialogs, maps, records, and

processes required by an application. The lists serve as helpful reference tools/checklists when the application is being defined online in the prototyping step (see "Step Three: Building a Prototype" in this chapter).

Application worksheet (responses)

APPLICATION COMPILER WORKSHEET				****RESPONSES****		
APPLICATION: <i>AO5201A1</i>		VERSION: <i>1</i>		PAGE OF		
DATE: <i>8/4/92</i>				PREPARED BY: <i>AHK</i>		
RESPONSE NAME	CONTROL KEY	FUNCTION INITIATED	CONTROL COMMAND	RESPONSE DESCRIPTION	SECURITY CLASS	GLOBAL/ LOCAL
<i>COM</i>	<i>PF7</i>	<i>AO5209F3</i>	<i>TRANS</i>	<i>COMPANY INFO MENU</i>	<i>0</i>	<i>L</i>
<i>DEP</i>	<i>PF3</i>	<i>AO5409F8</i>	<i>INVOKE</i>	<i>DEPARTMENT DISPLAY</i>	<i>0</i>	<i>L</i>
<i>EMP</i>	<i>PF4</i>	<i>AO5209F5</i>	<i>INVOKE</i>	<i>EMPLOYEE DISPLAY</i>	<i>0</i>	<i>L</i>
<i>HELP</i>	<i>PF2</i>	<i>HELP</i>	<i>-----</i>	<i>HELP SCREEN</i>	<i>0</i>	<i>G</i>

Application worksheet (functions)

APPLICATION COMPILER WORKSHEET

****FUNCTIONS****

APPLICATION: *A05209A1* VERSION: *1*

PAGE OF

DATE: *8/4/92*

PREPARED BY: *AHK*

FUNCTION NAME	FUNCTION DESCRIPTION	TYPE (M/D/P)	ASSOCIATED WITH (PROGRAM OR DIALOG)	EXIT DIALOG	VALID RESPONSES DEFAULT
<i>AO5209F1</i>	<i>MAIN MENU</i>	<i>M</i>	----		<i>COM GMP SIGNON</i>
<i>AO5209F2</i>	<i>EMPLOYEE INFO MENU</i>	<i>M</i>	----		<i>COM PER POP HELP</i>
<i>AO5209F3</i>	<i>COMPANY DATA MENU</i>	<i>M</i>	----		<i>DEP OFF SKI POP HELP POPTOP</i>
<i>AO5209F7</i>	<i>OFFICE DISPLAY</i>	<i>D</i>	<i>AO5209D5</i>		<i>POP POPTOP EMP</i>

Charts or checklists, such as those shown in the previous graphic, also serve as excellent documentation for an application, because all pieces of the application, as well as their relationships, are detailed.

Additionally, the use of naming conventions is helpful: consistent use of naming standards makes it easier to keep track of application and dialog components as they are created and maintained.

►► For suggestions on the use of standard naming techniques, see Chapter 6, “Naming Conventions” on page 6-1.

2.3.2 Internal/Technical Specifications

You should examine six areas: After the application format has been determined, decisions need to be made about how the application will work. The developer must consider the following:

- **Records** — What subschema, map, and work records are to be part of this application?
- **Menu Screens** — Will standard system-defined menus be used or will the menus be user-defined? If system-defined, which format of the system menu will be chosen? If user-defined, how will the menus be formatted and what will they do?

►► For information about the three types of system-defined menu maps, refer to *CA-IDMS Mapping Facility*.

Chapter 4, “Designing Maps” on page 4-1 discusses methods that can be used when designing user-defined menu maps.

- **Map formatting** — What maps will be needed? What will the maps look like? Are there site-specific standards that need to be considered?
- **Automatic editing** — What edit and/or code tables are necessary? Will the data be displayed as it is stored? How will the internal and external pictures be defined? How will the date display be formatted?

►► For further information on automatic editing and error handling, refer to *CA-IDMS Mapping Facility*.

- **Messages** — What informational and error messages, other than those supplied by the runtime system, should be conveyed to the terminal user?
- **Security** — What levels of security will be assigned? Will user, program, or subschema registration be implemented? Will a user id and password be required to sign on to an application?

►► For further information on the security that can be implemented, refer to *CA-ADS User Guide*.

2.4 Step Three: Building a Prototype

An application prototype in a CA-ADS environment is a representation of an online application system. As such, it is a tool that can be used throughout the design and development phases. Even after the implementation of an application, prototyping can be used as a vehicle for agreeing on revisions and enhancements.

Uses for a prototype, the unique features of a prototype, and creating a prototype are each discussed below.

2.4.1 Uses for the Prototype

Provides six benefits: The prototype provides the following benefits:

- **Aids in the design process** — The prototype helps to build relationships between the basic information entities (data items, records) of the business application, and between the information entities and the activities to be automated (for example, online screens/transactions, reports, batch jobs).
- **Maximizes end-user participation** — The prototype provides an end-user view of the application from an early point in the development process. Most importantly, the users are actually seeing the prospective system online.

Additionally, the user can participate in the step-by-step progress being made and can give valuable feedback while the application is still in its formative stage. As a review mechanism, the online screen walk-through provides a concrete means of checking to see if the application meets user needs.

- **Enhances project control** — The prototype provides an effective tool for monitoring the progress of the application development process.
- **Enables training** — The prototype can be used as a training tool for the data administrator and programmers on the development team. It enables them to become familiar with design techniques, dialog specifications, and documentation. The use of naming conventions, standardized coding procedures, and boilerplate process code facilitate the learning process. Additionally, the prototype can be employed by end users as a tool for training their own staff prior to implementation of the application in their production environment.
- **Establishes security procedures** — The prototype can incorporate the desired security standards without waiting for the source process code to be developed; thus, security procedures become established and understood by the end users at an early stage in the development of an application.
- **Provides an adaptable marketing tool** — A prototype can be developed as a demonstration model for use with prospective customers. As only a minimal amount of source code needs to be created, it is easy to adjust the prototype in response to specific user requests.

2.4.2 Unique Features of the Prototype

ADSA builds prototype: The prototype uses all the standard application components: dialogs that have been compiled with the CA-ADS Dialog Compiler; maps that have been created with the DC/UCF system's mapping facility; and data elements that have been defined in the dictionary with DDDL statements. Most importantly, the prototype is built with the CA-ADS Application Compiler.

ADSA provides the following capabilities that add considerable flexibility to the application, in general, and to dialogs, in particular:

- Security controls that can be put into effect for the application itself and for responses within the application
- Standard menus that are automatically created by the system at runtime and allow the use of fewer dialogs
- The EXECUTE NEXT FUNCTION command, which helps to control the flow of an application and allows process code to be more independent of its position within the application
- Global records that enable the developer to use fewer levels in the application thread
- Defined responses that reduce the number of response processes needed per dialog
- Function-related task codes that facilitate multiple entry points into the application
- Signon capabilities that make it possible for the end user to bypass the ENTER NEXT TASK CODE prompt from the DC/UCF system

2.4.3 Creating the Prototype

A prototype application can be built in three stages, as follows:

- Stage I: Building the basic prototype
- Stage II: Adding process logic and data retrieval
- Stage III: Refining the maps and processes

Each progressive stage contains enhancements that more closely approximate the final application. Note that it is possible to demonstrate the prototype online as soon as the first stage is completed successfully.

Information required: The developer must have the following information to format the prototype:

- The screens needed to support the functional requirements
- The processing activities taking place before and after communication with the user
- The number of dialogs included in the application
- The activities associated with each dialog

- The manner in which processing selections will be made by the user
- The control key and/or response code that will be associated with each selection

Worksheets can be developed to record all of the above information. Refer to the graphic, Sample Application Worksheets, earlier in this chapter for examples of sample worksheets. Chapter 3, “Building a Prototype” on page 3-1 provides the step-by-step procedure for creating an online prototype.

2.5 Step Four: Writing Process Code for the Dialogs

Step Four is the stage at which the technical design is translated into specific dialogs that can be coded and unit tested. Writing the dialog specifications and writing the source code are each discussed separately.

2.5.1 Writing the Dialog Specifications

Before any code is written, it is necessary to write dialog specifications for each dialog defined in the technical design. This process can be standardized (and simplified) if the programmer has access to a template that provides the accepted format for these specifications. The following text illustrates an example of a template that a design team might develop for its programmers.

Sample template for dialog specifications

```

*** HRIS SYSTEM ***

SPECIFICATION FOR DIALOG CEMDxxxx (...description of dialog...)

*****
*          **** UPDATE LOG ****          *
* WHO      WHEN      WHAT                *
* ===     =====     ===                *
* MCS      7/10/83   WROTE SPEC           *
* MMC      8/06/83   REVISED BASED ON NEW DATABASE DESIGN *
*                                                *
*                                                *
*****
DICTIONARY      : DOCUNET
SCHEMA          : EMPSCM
SUBSCHEMA       : EMPSS07
MAP             : CEMMXXX
MAP RECORD      : CEMMXXX-MAP-RECORD
DIALOG RECORD   : CEMDXXX WORK-RECORD, CEMDXXX-WORK2-RECORD
SYSTEM RECORD   : CEM-SYS-RECORD
MSG WORK RECORD : CEM-MESSAGE-WORK-RECORD
DB-ERROR RECORD : DATABASE-ERROR-RECORD

WORK INPUT/OUTPUT: CEM-MESSAGE-WORK-RECORD
  Record Layout: 05 CEM-MSG-MESSAGE-GET.
                   10 CEM-MSG-PROJECT-CODE PIC X(2).
                   10 CEM-MSG-MESSAGE-ID PIC S9(7) COMP-3.
                   05 CEM-MSG-SUB PIC S9(7) COMP-3.
                   05 CEM-MSG-MESSAGE-AREA.
                   10 CEM-MSG-MESSAGE
                       OCCURS 4 TIMES PIC X(40).

DATABASE INPUT : record names
DATABASE OUTPUT : record names

GENERAL DESCRIPTION:

*** PREMAP PROCESS: CEMDXXX-PREMAP

DESCRIPTION:

*** RESPONSE PROCESS: CEMDXXX-RESPONSE

DESCRIPTION:

```

Dialog specifications provide synopsis: Dialog specifications provide a synopsis of the dialog that includes descriptions of the premap and response processes; names of the dictionary, schema, and subschema; and the map and work records used by the dialog. Dialog specifications can be included at the beginning of the dialog's premap process.

Guidelines for writing the specifications and the importance of a review process are each discussed separately below.

Guidelines for dialog specifications: The following guidelines are suggested when writing the specifications:

- Ensure that the specification narrative has all the information needed to write the program.
- Use the structure diagram and worksheets to obtain the proper dialog, record, and map names.
- Adhere to naming conventions.
- Use the data structure diagram and reports of the elements and records for details about the individual dialogs.
- Store the completed specification in the data dictionary, as comments in the premap process. Within process source, use the exclamation point (!) to lead all comments.
- If the specification is particularly long, store it as a separate module in the dictionary and copy it into the premap process code with an INCLUDE statement. In this way, the specifications are included in reports, but do not have to be viewed when the programmer is working on the source code.
- Refer to maps by name and location. As the design of the dialog maps would have been completed when building the prototype, it is unnecessary to duplicate the layouts in the specifications. If further definitions on map fields are required in order to write the code, these definitions should be included in the specifications and given to the data administrator.
- Incorporate other comments in the process source, as needed, especially at the beginning of response processes and subroutines. Batch programs and reports should also have their specifications included as comments within the code, unless the specifications are very long.

Some sites find it worthwhile to create a partitioned data set (PDS) or library for storing the specifications for each dialog. Such a data set can also be useful for central storage of the map templates and boilerplate code developed as programming aids.

Reviewing the specifications: Coding should not begin until the project leader has reviewed and approved the dialog specifications. This is also the time to provide answers for questions that might have arisen during specification development. For example, in developing the specifications, it might become necessary to add some dialogs not already identified in the application structure. If so, this should be discussed and approved; changes can affect other screen layouts, as well as the manner in which the application has been defined to ADSA.

2.5.2 Writing the Source Code

Once the specifications have been approved, the programmer can write the source code. The use of test version numbers, procedures to aid the programmer, and dialog debugging aids are each discussed separately below.

Test version numbers: The DC/UCF system provides facilities for establishing a runtime environment in which test and production copies of the same application components can execute under one system. Programmers can be assigned a unique version number to be used when generating their own versions of maps, edit and code tables, and dialogs. When the application is fully tested and working, the version number can be changed for production purposes.

►► For a detailed discussion of the preparations necessary when establishing a test environment, refer to *CA-IDMS System Operations*.

Programming aids: To improve the efficiency of the development process and to help maintain standards, an installation might institute some of the following procedures for the programming staff:

- Create templates of dialog premap and response processes. The programmer can obtain a copy of the template, rename it, and add the specific dialog logic.
- Provide a list of the standard (site-specific) work records that are to be used by each dialog.
- Provide process code for standard routines. Identify commonly performed activities and decide how they are to be handled. Develop process code for these activities and store as modules in the dictionary so that any dialog can link to them when necessary. For example, routines can be developed to handle date conversions, forward and backward paging, database error routines, and message formatting.

The following text illustrates a type of boilerplate code that can be developed for a premap process; the template demonstrates a type of boilerplate code that can be developed for a response process. These templates contain the standard logic for interfacing with common subroutines.

Sample premap process template

```

      ADD
      MODULE NAME IS xxxDxxxx-PREMAP VERSION IS 1 LANGUAGE IS PROCESS
      MODULE SOURCE FOLLOWS
!*****
!*           THE PREMAP PROCESS FOR THE xxxx DIALOG
!*****
      INIT REC (xxx-message-work-record).
      KEEP LONGTERM ALL RELEASE.
!
! THE ACTUAL LOGIC FOR SCROLLING BACKWARDS AND FORWARDS WILL
! BE DIFFERENT FOR EVERY DIALOG. THEREFORE, THESE ROUTINES HAVE
! NOT BEEN CODED IN THIS TEMPLATE.
!
IF FIRST-TIME
  INIT REC (.....).
  MOVE SPACES TO xxx-function.

IF xxx-function EQ 'NEXT'
  THEN
    CALL forwr02.
ELSE
  IF xxx-function EQ 'PREV'
  THEN
    CALL backwd03.

! THE FOLLOWING CODE IS TO BE USED WHEN YOU WANT TO BE NOTIFIED
! THAT ANOTHER USER IS UPDATING THE SAME RECORDS THAT YOUR
! DIALOG IS UPDATING.
! SUBSTITUTE THE ACTUAL DIALOG NAME FOR 'dialog name' AND THE
! ACTUAL RECORDS OF CONCERN FOR record-name.
! IF MORE THAN ONE LOCK IS REQUESTED, INCLUDE A NUMERIC IDENTIFIER
! WITH THE DIALOG NAME (e.g., CEMD1LIS, CEMD2LIS).
! KEEP LONGTERM SHOULD BE CODED DIRECTLY AFTER AN OBTAIN.

KEEP LONGTERM 'dialog name' NOTIFY CURRENT record-name.

!           (Main premap logic goes here)

IF AGR-CURRENT-FUNCTION EQ 'DELETE FUNCTION'
  THEN DO.
    MOVE 98xxxx TO xxx-message-id.
    CALL messge98.
  END.
!
! THIS MESSAGE WILL READ
!       'TO COMPLETE DELETE ENTER PROPER RESPONSE'
!
IF xxx-msg-sub GT 0
  THEN
    DISPLAY MESSAGE TEXT xxx-msg-message-area.
  DISPLAY.

!

```

Sample premap process template

```
!*****
DEFINE SUBROUTINE dberr99.
!*****
!
!*****
!***** ABEND ROUTINE FOR BAD DB CALLS. *****
!*****
        KEEP LONGTERM ALL RELEASE.
        ACCEPT RECORD      INTO der-record-name.
        ACCEPT AREA        INTO der-area-name.
        ACCEPT ERROR SET   INTO der-error-set.
        ACCEPT ERROR RECORD INTO der-error-record.
        ACCEPT ERROR AREA  INTO der-error-area.
        MOVE ERROR-STATUS  TO der-error-status.
        ROLLBACK.
        DISPLAY MESSAGE CODE IS 799999
                PARS = (der-error-status
                        ,der-record-name
                        ,der-area-name
                        ,der-error-set
                        ,der-error-record
                        ,der-error-area).

        GOBACK.
        MSEND.
```

What templates provide: Templates provide a means of supplying site-specific information to programmers. For example, the installation using this template specifies the name of the dialog as the unique identifier for longterm locks.

Sample response process template: The following figure shows the response process template that corresponds to the premap process template.

```

ADD
MODULE NAME IS xxxDxxxx-RESPONSE VERSION IS 1 LANGUAGE IS PROCESS
MODULE SOURCE FOLLOWS
!*****
!? THE RESPONSE PROCESS FOR THE xxxx DIALOG
!*****
INIT REC (xxx-message-work-record).

IF AGR-NEXT-FUNCTION EQ 'NEXT'
THEN DO.
MOVE 'NEXT' TO xxx-function.
DISPLAY CONTINUE.
END.
IF AGR-NEXT-FUNCTION EQ 'PREV'
THEN DO.
MOVE 'PREV' TO xxx-function.
DISPLAY CONTINUE.
END.

IF AGR-NEXT-FUNCTION EQ AGR-CURRENT-FUNCTION AND
AGR-CURRENT-FUNCTION EQ 'delete'
THEN DO.
CALL .....
END.

IF NO FIELDS ARE CHANGED
THEN
EXECUTE NEXT FUNCTION.

! THE FOLLOWING CODE WILL RETURN A VALUE INTO A SPECIFIED FIELD
! IN THE SYSTEM RECORD. THE VALUE GIVES NOTIFICATION OF ANY
! ACTIVITY AGAINST ANY RECORDS WHICH WERE SPECIFIED IN THE PREMAP
! PROCESS OF THE DIALOG.

KEEP LONGTERM 'dialog name' TEST RETURN NOTIFICATION INTO
xxx-notify.

! IF APPROPRIATE, THE FOLLOWING VALUES OF xxx-notify SHOULD BE
! CHECKED:
!
! VALUE OF xxx-notify          MEANING
!
! 0          NO DATABASE ACTIVITY FOR RECORD
! 1          RECORD WAS OBTAINED
! 2          RECORD'S DATA EAS MODIFIED
! 4          THE RECORD'S PREFIX WAS MODIFIED
!           (I.E. A SET OPERATION OCCURRED
!           INVOLVING THIS RECORD)
! 8          THE RECORD WAS LOGICALLY DELETED
! 16         THE RECORD WAS PHYSICALLY DELETED
!
! MULTIPLE ACTIVITIES WILL CAUSE A COMBINATION OF THESE VALUES.
! THE MAXIMUM POSSIBLE VALUE IS 31 (MEANING ALL OF THE ABOVE
! OCCURRED).
!
! (I.E.)
!

```

2.5 Step Four: Writing Process Code for the Dialogs

```
! IF xxx-notify GT 7
!   THEN DO.
!     MOVE 98xxxx TO xxx-msg-message-id.
!     CALL messge98.
!     DISPLAY MESSAGE TEXT xxx-msg-message-area.
! END.

! IF xxx-notify GT 1
!   THEN DO.
!     MOVE 98xxxx TO xxx-msg-message-id.
!     CALL messge98.
!     DISPLAY CONTINUE MESSAGE TEXT xxx-msg-message-area.
! END.
!
! IN THE FIRST EXAMPLE, THE RECORD HAS BEEN DELETED.
!
! IN THE SECOND EXAMPLE, THE RECORD WAS MODIFIED BY
! ANOTHER USER. THE DISPLAY CONTINUE WILL NOT ONLY
! DISPLAY A MESSAGE, BUT WILL ALSO REEXECUTE THE
! PREAMP TO SHOW THE USER THE MODIFIED RECORD.

IF AGR-CURRENT-FUNCTION EQ 'function a'
  THEN
    CALL .....
IF AGR-CURRENT-FUNCTION EQ 'function b'
  THEN
    CALL .....

(Other processing code specific to the dialog goes here)

IF AGR-STEP-MODE AND xxx-msg-sub GT 0
  THEN
    DISPLAY MESSAGE TEXT xxx-msg-message-area.
  EXECUTE NEXT FUNCTION.

!*****
DEFINE SUBROUTINE messge98.
!*****
IF xxx-msg-sub LT 4
  THEN
    LINK PROGRAM 'xxxxxxx' USING (xxx-message-work-record).
  ELSE
    DISPLAY MESSAGE TEXT xxx-message-area.
GOBACK.
```

```

!
!*****
DEFINE SUBROUTINE dberr99.
!*****
!
!*****
!***** ABEND ROUTINE FOR BAD DB CALLS. *****
!*****
      KEEP LONGTERM ALL RELEASE.
      ACCEPT RECORD      INTO der-record-name.
      ACCEPT AREA        INTO der-area-name.
      ACCEPT ERROR SET   INTO der-error-set.
      ACCEPT ERROR RECORD INTO der-error-record.
      ACCEPT ERROR AREA  INTO der-error-area.
      MOVE  ERROR-STATUS TO der-error-status.
      ROLLBACK.
      DISPLAY MESSAGE CODE IS 799999
          PARS = (der-error-status
                ,der-record-name
                ,der-area-name
                ,der-error-set
                ,der-error-record
                ,der-error-area).

      GOBACK.
      MSEND.

```

Debugging aids: The following debugging aids are available to the CA-ADS programmer:

- The ABORT and SNAP commands
- The diagnostic screen (if enabled)
- The print log utility (RHDCPRLG)
- CA-OLQ
- The ADSORPTS utility (particularly the DIALOG report and the FDBLIST)
- The mapping report utility (RHDCMPUT)

Errors can be resolved by signing on to IDD, making changes to the process code, and signing on to ADSC to recompile the dialog.

2.6 Step Five: Testing and Implementing the Application

The final stage of application design and development deals with the testing and implementation of the application. This is an important step that requires careful planning. Testing should not begin until a comprehensive test plan has been formulated; the testing itself should be thoughtfully structured.

Guidelines for developing a test plan and the procedures involved in the final testing of an application are discussed below.

2.6.1 Test Plan

A definitive test plan should be drawn up after the technical design is finalized. This plan is particularly important when performing acceptance testing for the user because it must reflect the expectations of the user. The plan should include the following information:

- Division of the application for testing purposes
- Plans for testing interfaces
- The order of testing, taking into account the planned implementation
- Approval criteria for user and operations acceptance of the system
- Test data to be used and the method of creating this data
- Operational and technical support required
- A list of all testing and related tasks
- The people involved and their specific responsibilities
- A schedule for testing and acceptance of the system

2.6.2 Test Procedures

Procedures for testing applications typically fall into the following phases:

- Unit testing by the programmer
- Integrated system testing by the programming team
- User acceptance testing

Each of these phases is described below.

Unit testing: Each dialog should be tested in isolation for all possible error conditions. This should be done either by the person who developed the dialog, or, preferably, by another member of the project team. The following lists should be drawn up beforehand:

- The conditions to be tested
- The data used to test these conditions

- The expected results

The documented results of the testing should be approved by the project leader. When the unit testing is completed successfully, the dialog should be submitted for subsystem or integration testing. For dialogs that operate independently, no testing should be required beyond unit testing.

Integration testing: Integration testing determines whether the dialogs within each subsystem are functioning in accordance with the specifications. Interdependent dialogs should be grouped together and tested as a unit, using the same principle as for unit testing. To avoid duplication of effort, this phase should use the same data as that used for unit testing whenever possible. The application, in its entirety, should be tested to ensure that all paths through the application are traversed correctly.

Regression testing is a useful practice to implement. Test results are saved from each of the test procedures to be compared with subsequent test results if/when changes are made to an application. Comparison of the test results can provide an efficient way to monitor the effectiveness of the changes.

Acceptance testing: Users determine the acceptance test criteria and should approve all system outputs. Acceptance testing ensures that the system is functionally acceptable to the users and will operate successfully in the production environment. Testing should be performed using live or simulated-live data provided by the users.

2.7 Underlying Issues

Five key considerations: The following aspects of the application need to be kept in focus as the application is designed and developed:

- **Establishing design standards** — Each site should develop design standards with respect to items such as the formatting of maps, naming conventions for applications and their components, and the construction of process source code. Chapter 6, “Naming Conventions” on page 6-1 suggests some conventions that can be adapted to meet specific user needs. For information on templates that can be used as aids when writing dialogs, refer to the discussion of programming, under “Writing the Source Code” earlier in this chapter.
- **Using the dictionary** — The dictionary is an integral part of the application design process. Optimum utilization of this tool requires thoughtful preparation and the development of dictionary procedures that are enforced consistently. The dictionary can be used in the following ways:
 - Run DDR reports to obtain information on the elements, records, maps, tables, and process modules that are already in the dictionary.
 - Establish naming standards for dictionary entries; alternatively, obtain a copy of existing standards.
 - Build and populate the dictionary with the necessary elements and records.

►► For further information on the most effective ways to use the dictionary for storing application components, refer to Chapter 2, “Design Methodology” on page 2-1.

Chapter 6, “Naming Conventions” on page 6-1 suggests ways in which standardized dictionary naming conventions can be used to minimize the chance of redundancy.
- **Using the prototype** — The developer needs to remember that the prototype is meant to be a vehicle for and not an obstacle to system development. Therefore, any process source that is written for the prototype should be short and uncomplicated. The developer wants to be able to demonstrate how the application will work, but the prototype does not have to be refined to the point where it can run in a production environment.
- **Dialog design considerations** — To have a well-designed application in a CA-ADS environment, it is essential to provide well-designed dialogs that fully utilize the attributes of the CA-ADS process code and make efficient use of system resources. Chapter 5, “Designing Dialogs” on page 5-1 discusses ADSA features (for example, the use of global records, the extended run unit) to take into account when designing dialogs for ADSA-compiled applications. Additionally, Chapter 5, “Designing Dialogs” on page 5-1 reviews the basic procedures, components, and characteristics of dialogs that affect the manner in which a design is created.
- **Security considerations** — Security needs and procedures are, of course, site-specific. CA-IDMS provides a centralized security facility that can be used to

secure resources associated with a CA-ADS application, such as tasks and application activities.

►► For more information about CA-IDMS security, refer to *CA-IDMS Security Administration*.

When establishing security guidelines for a CA-ADS application, consider the following, which can be part of the application definition:

- Required signon to the application
- Security classes to be assigned to specific responses

►► For further information on specifying security in the application definition, refer to *CA-ADS User Guide*.

2.8 Data Definition and Database Design

Because CA-ADS operates in the CA-IDMS/DB environment, it is important to review how data is defined and stored in that environment. In a traditional application development environment, the application programs comprise both processing logic and information about the data accessed. Processing logic, which determines the action taken by a program to produce the desired output, correctly belongs in the realm of programming. Defining information about data (such as the format of records and elements, and editing criteria) can be handled more easily and efficiently as a separate function.

The CA-IDMS/DB environment uses the dictionary to accomplish this separation of information about data from process logic. The dictionary maintains information about data and makes this information directly available to the application maps and dialogs that need it.

Advantages of separating information: Separating information about data from process logic has the following advantages:

- **Allows control of data resources** — The site has better control over data resources because the control is centralized. Centralized control provides the following benefits:
 - Eliminates unwanted data redundancy
 - Controls the data that is available
 - Determines where data elements are used and by whom
 - Establishes standards for data element names, input and output formats, and editing criteria
- **Facilitates the design, development, and maintenance of CA-ADS applications** — The application can use data from the dictionary and can store application-specific data in the dictionary where it can be maintained. The data can be accessed by a variety of reporting facilities and software components, and can be populated and updated automatically.
- **Increases productivity** — Productivity is increased because activities are not duplicated. Information about an element type is defined once and does not have to be defined separately by every programmer using that element in a dialog or map.

The DBA staff can concentrate on defining information for the applications, and the programming staff can concentrate on the processing logic. For example, the dictionary can maintain editing and display information for each element. The DBA can simply define in the dictionary that the external format of social security numbers is 999-99-999, and the application programmers need not be concerned about editing and formatting the element when they use it. On all input operations for this element, the automatic editing facility will verify that user input conforms to this picture; on all output operations, it will format the data and insert hyphens.

Defining the elements in this way is easier, less error prone, and less time consuming than coding process logic.

Definition of information: The definition of information for an application can be divided into the following two phases:

- Data definition
- Database design and definition

Each of these phases in the design process is outlined below.

►► For further information on this topic, refer to *CA-IDMS Database Design*.

Chapter 3. Building a Prototype

- 3.1 Three-stage Approach 3-3
- 3.2 Stage I: Building the Basic Prototype 3-4
 - 3.2.1 Compiling the Application (ADSA) 3-4
 - 3.2.2 Compiling the Maps 3-5
 - 3.2.3 Compiling the Dialogs (ADSC) 3-5
 - 3.2.4 User Review 3-6
- 3.3 Stage II: Adding Process Logic and Data Retrieval 3-7
 - 3.3.1 ADSA Enhancements 3-7
 - 3.3.2 Populating the Dictionary 3-7
 - 3.3.3 CA-IDMS Mapping Facility Enhancements 3-8
 - 3.3.4 ADSC Enhancements 3-8
- 3.4 Stage III: Refining the Maps and Processes 3-9

3.1 Three-stage Approach

The development of a prototype can be approached in a variety of ways, depending upon the needs of the design team. The procedures suggested in this manual are based on a three-stage approach: the initial stage performs rudimentary navigation of the application; the second stage begins to perform data retrieval and update; and the final stage incorporates refinements that reflect the more complex requirements of an application running in a production environment.

Each stage of the prototype is discussed below.

3.2 Stage I: Building the Basic Prototype

Prototype can be developed quickly: The first stage of the prototype can be developed quickly and easily because only skeletal maps and dialogs are needed for execution by the CA-ADS runtime system. Typically, maps are created with just enough information to identify their use in the application process, and one dialog is created for each map. The dialogs do not need a premap process or a response process. With a minimum of time and effort, the designer has the opportunity to see how the application is going to work even before data processing takes place.

Activities to perform: To build an executable prototype, the developer needs to provide load modules for the runtime system by performing the following activities:

- **Compiling the application** — The application and its components (the functions and responses) are defined and compiled with ADSA.
- **Compiling the maps** — Each map is formatted, defined, and compiled with the online mapping facility.
- **Compiling the dialogs** — Each dialog is identified, associated with the appropriate map, and compiled with ADSC.

The prototype can be executed when the application, map, and dialog load modules are available for use by the CA-ADS runtime system. At this point, the developer has a meaningful version of the prototype that can be presented for user review and modification.

Each of the activities for building the basic prototype is discussed separately below, followed by user review considerations.

3.2.1 Compiling the Application (ADSA)

Five necessary steps: The amount of detail provided for a prototype can be as extensive as the developer wishes, but the basic prototype does not have to be elaborate. After initiating an ADSA session, the developer can define and compile an application as follows:

1. **Specify the application** — ADSA must be supplied the name of the application and related information such as version number.
2. **Name the task code** — The task code designates an entry point into the application. If there are multiple entry points, each task code must be defined individually.
3. **Define the responses** — The responses that initiate the functions of the application must be specified.
4. **Define the functions** — Menu and dialog functions that are initiated by the responses must be specified.

Note: Every function that you define as a dialog function in ADSA you must also define to ADSC as a dialog.

5. Compile the application.

When the above-named activities are completed successfully, ADSA defines an Application Definition Block (ADB) for the application and updates the Task Activity Table. Both the ADB and the TAT are stored as load modules in the dictionary and are used by the CA-ADS runtime system when the application is executed.

3.2.2 Compiling the Maps

How to produce prototype screens: Maps that are compiled for the first stage of the prototype usually contain all literal fields. The developer signs on to the online mapping facility (MAPC) and takes the following steps to produce the prototype screens:

- **Specify the map and map options** — The map name and related information such as version number must be supplied to MAPC. Certain options, such as display options, may also be appropriate to specify for the prototype.
- **Produce a screen layout** — A layout can be produced automatically if the developer specifies existing dictionary records to MAPC. Otherwise, the layout can be produced manually. Literal values (such as hyphens or underscores) can be assigned to represent variable data fields.
- **Compile the map**

A map load module is stored in the DDLDCLOD area of the dictionary when the map has been compiled successfully.

3.2.3 Compiling the Dialogs (ADSC)

One dialog for each map: You must compile one dialog for each map used by the prototype. To compile a prototype dialog, initiate an ADSC session and:

1. Add the dialog
2. Associate the map with the dialog
3. Compile the dialog.

Considerations: The following considerations should be noted when compiling dialogs for an application:

- If a dialog is defined as a function in ADSA, it must be defined in ADSC
- If a dialog is associated with a task code, it must be defined as a mainline dialog
- The associated map must be compiled before the dialog can be compiled

ADSC defines a Fixed Dialog Block (FDB) for every dialog that is compiled successfully. The FDB is stored as a load module in the dictionary and is used by the CA-ADS runtime system when the application is executed.

3.2.4 User Review

After the application, map, and dialog load modules have been compiled, the prototype is ready to be presented to the user for careful online review. Modifications based on review should be made to the existing prototype, the necessary load modules recompiled, and the prototype resubmitted for review until the users are satisfied.

3.3 Stage II: Adding Process Logic and Data Retrieval

The prototype becomes more functional in the second stage. The developer might add activities such as the following to the prototype:

- Global records (ADSA)
- Security restrictions such as signon menus (ADSA)
- Display capabilities (online mapping and IDD)
- Premap and response process logic (ADSC and IDD)

The ADSA, online mapping facility, ADSC, and IDD activities used for these enhancements are described separately below.

3.3.1 ADSA Enhancements

Four features can be added: The following ADSA features can be added to the prototype at this point:

- Global records (that is, records that are available for use by all dialogs in the application) can be defined
- User-program records (that is, records that are to be passed to a user-program) can be defined if needed
- Valid responses listed for a function can be resequenced or their display can be suppressed
- Signon can be specified as required or optional. If either is specified, these steps must be taken:
 - The signon function must be identified
 - The function type of the signon function must be specified as menu
 - The function must be defined as a menu
 - The SIGNON system function must be specified as the function initiated by the user's response from the signon screen
 - The response that initiates the SIGNON system function must be specified as a valid response for the named menu function

When these changes have been made, recompile the application.

3.3.2 Populating the Dictionary

Three necessary components: The dictionary must contain the following components if they are to be used by the prototype:

- **Dialog premap and response processes** — Premap and response processes must be stored as process modules in the dictionary. If premap or response processes are associated with a dialog, process modules must be defined in the dictionary

before the dialog can be compiled. Modules are added to the dictionary with the `IDD MODULE` statement specifying `LANGUAGE IS PROCESS`.

- **Map records and dialog work records** — All work records used by a dialog and all records associated with maps must be defined in the dictionary before the dialogs and maps can be compiled. Similarly, an application cannot be compiled unless all global records associated with the application are defined in the dictionary. Records are added with the `IDD RECORD` statement.
- **Edit and code tables** — All stand-alone edit and code tables associated with map records must be defined in the dictionary before the map is compiled. Edit and code tables are added with the `IDD TABLE` statement.

3.3.3 CA-IDMS Mapping Facility Enhancements

Variable map fields that were defined as literals for the first stage of the prototype should be redefined as data fields and edited accordingly. When the appropriate enhancements have been made, the map should be recompiled.

3.3.4 ADSC Enhancements

The developer now uses ADSC, recompiling the dialog to include the premap and response processes, as well as the changes made to the map associated with this dialog. After initiating an ADSC session and naming the appropriate dialog, the developer can make these enhancements:

- **Database specification** — Specify the database that the dialog accesses
- **Work records** — Supply the names of all work records associated with the dialog
Note: If the dialog is using subschema records, they must belong to the same subschema as the dialog.
- **Premap process** — Supply the name of the premap process associated with the dialog.
- **Response process** — Supply the name of the response process associated with the dialog and a control key and/or response field value unique to that response process.

Recompile the dialog after making the appropriate enhancements.

3.4 Stage III: Refining the Maps and Processes

The final stage of prototype development can focus on refinement of the map design and the map field attributes. Some of the following additions can be made:

- Incorporate additional fields in the maps
- Add or change map field attributes
- Specify automatic editing on selected map fields
- Provide informational messages
- Add error messages

Chapter 4. Designing Maps

- 4.1 Attributes of Successful Maps 4-3
- 4.2 Design Standards for a Dialog Map 4-4
- 4.3 Online Mapping Procedures 4-5
- 4.4 Choosing Menu Maps 4-6
 - 4.4.1 System-Defined Menu Maps 4-6
 - 4.4.2 User-Defined Menu Maps 4-6
- 4.5 Designing Dialog Maps 4-9

4.1 Attributes of Successful Maps

Determining success of application: Maps displayed during the execution of the application interface directly with the user and, therefore, can influence the success of an application. Consequently, the designer must consider the appearance of the menu screens and the layout of the dialog maps.

A successful map design should exhibit the following attributes:

- **Consistency** — Entities (for example, fields, headings, labels, responses, messages, and control keys) should have the same meaning or effect throughout the application. The meaning or effect need not be identical for every map, but should be consistent within the broader confines of the system. In general, there are two special fields on any screen: a message field and a response code field. These areas should appear in a constant location on the screen throughout any application; for maximum effectiveness, they should remain standard for all applications at a site.
- **Convenience** — Features of the system should be designed to associate related entities by using similar constructs, positioning, and responses to produce similar reactions from the system. For example, assign one particular control key to initiate the update function in all the dialogs of a given application.
- **Supportiveness** — The reactions of the system should enable the user to handle normal contingencies conveniently. Tutorial aids should be available when needed. Displayed informational and/or error messages should be meaningful.

The remainder of this chapter discusses the following aspects of map design:

- Standards to consider when designing maps
- Mapping procedures that can be adopted by an installation
- Choices available in the design of menu maps
- Suggestions for designing dialog maps

4.2 Design Standards for a Dialog Map

The user is the intended audience: The developer needs to consider the following standards when designing dialog maps:

- Design the map with the user in mind. For example, a very dense screen is tiring and difficult to use. In general, the screen most pleasing to the eye is about 40 percent full.
- The placement of fields on the screen, the use of high intensity, and the neatness of the format have a great deal of impact on the effectiveness of the system.
- When the screen is sent to the terminal, the cursor should be in the position most likely to be used for data entry. Other frequently used fields should be easily accessible with the tab and return keys.
- The sequence of fields, when tabbed, should match the most common pattern used for data entry.
- Fields requiring special attention should be highlighted and clearly visible.
- The screens should be as uncluttered as possible. The common error of using one screen format for excessive and/or dissimilar functions tends to produce cluttered or busy screens; separate screens with some common fields are more usable.
- Users should be able to initiate processing by typing in the necessary data and pressing a control key. They should not be required to make decisions that could have been incorporated in program logic, nor should they be forced to use control keys or responses needlessly.

4.3 Online Mapping Procedures

The following illustrate the mapping procedures that might be implemented by a specific site:

- Have one individual (for example, the data administrator) responsible for creating and modifying all maps.
- As much as possible, use the features of the online mapping facility to handle editing, error handling, error messages, and modifying field attributes.
- Use a standard map template. Whenever possible, keep data fields in columns and double space rows of data.
- Use the BRIGHT attribute to contrast items on the screen that have different uses (for example, highlight required fields). Be consistent in the use of attributes.
- Use the cursor in a consistent manner. For example, either place the cursor at the first field to be used for data entry or at the field where the user is to enter the next function.
- Use the BRIGHT attribute for redisplaying data fields that are in error.

4.4 Choosing Menu Maps

Two types available: When designing an application, the developer needs to decide if **system- or user-defined menu maps** are to be used. The system-defined menu provides a standard format for the information provided by the developer during the definition of the functions and responses of the application in an ADSA session. If a format other than the standard format is desired (for example, the developer wishes to redefine certain literal fields on the map or wants to supply site-specific headers), the user-defined menu map is used. Both types of maps are discussed separately below.

4.4.1 System-Defined Menu Maps

Designer's options: If the menu map is to be system-defined, the designer has the option of using one of the following menu formats:

- **Short description menu map (ADSOMUR1)** — The menu screen that lists 30 valid menu responses per page; a short (12-byte) textual description is displayed for each response.
- **Long description menu map (ADSOMUR2)** — The menu screen that lists 15 valid menu responses per page; a long (28-byte) textual description is displayed for each response name.
- **Signon menu map (ADSOMSON)** — The menu screen that requires a DC/UCF validation of user id and password before the menu request can be processed. The standard signon menu map can have 12 valid menu response names per page with 28 bytes of descriptive text displayed for each.

If none of the above menus meets the needs of the user, the system-defined menu map can be altered by the user or a new menu (designated as a menu/dialog function) can be formatted. Both methods of creating user-defined maps are discussed below.

4.4.2 User-Defined Menu Maps

Two methods of altering maps: When user-specific modifications to the existing system-defined menu maps are necessary, designers can alter the menu maps by using either of the following techniques:

- Reformatting and regenerating the standard system-defined menu
- Designing a menu/dialog (that is, a menu map that is part of a menu/dialog function)

Each of these methods is discussed below.

Reformatting the system-defined menu: The existing system-designed menu map can be reformatted and regenerated, retaining the same name. This method has the advantage of allowing the developer to use the standard menu function rather than designing and using a menu/dialog function.

Take the following steps to reformat the system menu:

1. Obtain the source for the map being used (that is, ADSOMUR1, ADSOMUR2, or ADSOMSON) from the source data sets created when the distribution tape was installed. The maps are stored as members under their own names.
2. Use the batch mapping compiler to store the source in the dictionary.
3. Use the online mapping facility to modify and regenerate the menu map.

Regenerating the system-defined menu: When regenerating a menu map with the online mapping facility, the following rules must be observed:

- ADSO-APPLICATION-MENU-RECORD is a required map record. Optionally, the menu can map to additional records, but it must always map to the ADSO-APPLICATION-MENU-RECORD.
- The menu must contain the same number of responses per page as the number of responses for the selected map (that is, 30 for ADSOMUR1, 15 for ADSOMUR2, or 12 for ADSOMSON).
- The AMR-RESPONSE field of the ADSO-APPLICATION-MENU-RECORD is a required field. The first response name on the map must map to the first occurrence of AMR-RESPONSE. Each subsequent response name must map to the next corresponding occurrence.
- The AMR-USER-ID and AMR-PASSWORD fields of the ADSO-APPLICATION-MENU-RECORD are required on a signon menu map. The user id data field must map to AMR-USER-ID, and the password data field must map to AMR-PASSWORD.
- All other fields on the ADSO-APPLICATION-MENU-RECORD are optional. The map data fields that are used must be associated with the appropriate fields on the record (for example, heading data must map to AMR-HEADING).

If using the AMR-KEY field, note that this field appears as a single byte (the AID byte) in the ADSO-APPLICATION-MENU-RECORD. The AMR-KEY field is associated with a code table (ADSOAIDM) that translates the AID byte to more easily readable characters (for example, 1 translates to PF1, percentage translates to PA1).

►► For more information on using online mapping facility to regenerate a map, refer to *CA-IDMS Mapping Facility*.

Designing a menu/dialog: The user has the option of designing and generating an entirely new menu with the online mapping facility. This map must be defined as a menu/dialog function of the application. Follow these procedures when defining a menu/dialog function:

1. Design and generate the map using the online mapping facility. Observe the following rules when generating the map:
 - ADSO-APPLICATION-MENU-RECORD must be one of the records associated with the map.

- The AMR-RESPONSE field is required for all menus. The number of required occurrences depends on the number of responses per page (to a maximum of 50) specified on the ADSA Menu Specification screen. The first response name on the map must map to the first occurrence of AMR-RESPONSE; each subsequent occurrence must map to the next corresponding occurrence of AMR-RESPONSE.
 - The AMR-USER-ID and AMR-PASSWORD fields are required for signon maps. The user id data field must map to AMR-USER-ID, and the password data field must map to AMR-PASSWORD.
 - All other fields on the ADSO-APPLICATION-MENU-RECORD are optional. The map data fields used must be associated with the appropriate fields on the record (for example, heading data must map to AMR-HEADING).
2. Add the process source to the dictionary in an IDD session. (The dialog associated with the menu does not have to include any process code, although the choice of a menu/dialog function suggests that some processing is intended.)
 3. Compile the dialog in an ADSC session, associating the map and any processes with the dialog using the ADSC Dialog Definition screen. Note that the dialog must be compiled to include the map before the application can be executed at run time.
 4. Define the dialog as a menu/dialog function for the application.

An installation can develop standard map templates and the associated boilerplate code for site-specific menu/dialogs. When a menu is needed, programmers can obtain a copy of the template/boilerplate, fill in the appropriate fields and the edit/code tables needed for those fields, and submit it to the data administrator for approval.

4.5 Designing Dialog Maps

Answering design questions: Each dialog map is associated with its own dialog and must be designed to reflect the function of the associated dialog. The application specifications developed during the initial design stages can be used to answer design questions such as the following:

- How many of the dialogs specified for this application will require maps?
- What premap and response processes are required for each map?
- What job is performed by each process?
- Will the map be used to pass data between processes and/or between dialogs?
What data will be passed?
- What database and mapping work records are associated with the map?
- What editing criteria should apply to the map fields?

Standardizing formats: Just as site-specific standards can be established for menu/dialogs, an installation can use map templates to standardize the formatting of maps associated with dialog functions. Programmers can obtain a copy of the template; fill in the appropriate fields, indicating the corresponding map record fields; and submit this information to the data administrator. The data administrator can then add the necessary map design, map records, and edit/code tables (if any) to the dictionary.

The following figure illustrates a sample map template that can be provided for programmers. This template designates standard areas for headers, footers, message codes and descriptions, response areas, and the passing data field. The installation using this template has written a routine that divides the message area into four 40-character messages.

Sample template for an application screen

4.5 Designing Dialog Maps

Column	1	1	2	2	3	3	4	4	5	5	6	6	7	7	8	
	5	0	5	0	5	0	5	0	5	0	5	0	5	0	5	0

```
<dialog>      <.n a m e .. o f .. a p p l i c a t i o n.> < date > <tm>  
<functn>      <.. function description ..>  USER: <userid> <md> MODE
```

```
NEXT RESPONSE: <respns>      NEXT KEY: <..... p a s s e d ... d a t a .....>  
<.....m e s s a g e   a r e a.....><... may contain up to four.....>  
<.....><...40 character messages.....>
```

Chapter 5. Designing Dialogs

- 5.1 Overview 5-3
 - 5.1.1 Dialog Level 5-3
 - 5.1.2 Dialog Status 5-4
 - 5.1.3 Dialog Control 5-5
- 5.2 Design Considerations 5-7
 - 5.2.1 Record Buffer Management 5-7
 - 5.2.2 Working Storage Areas 5-9
 - 5.2.3 Global Records 5-10
- 5.3 Dialogs That Issue Navigational DML 5-17
 - 5.3.1 Database Currencies 5-17
 - 5.3.2 Extended Run Units 5-18
 - 5.3.3 Longterm Locks 5-19
 - 5.3.4 Record Buffer Management for Logical Records 5-20

5.1 Overview

What is a dialog?: A dialog is a unit of work within a CA-ADS application that enables interaction with the user. Because dialogs are the basic building blocks of a CA-ADS application, it is important that they be well-designed. This chapter discusses characteristics and design features of dialogs that merit the attention of application developers.

Dialog characteristics: The characteristics of a dialog determine its role within the application; each dialog has an implicit level and status, and can pass and receive control of the processing. The significance of the dialog level and status and the manner in which control is passed are discussed below.

5.1.1 Dialog Level

Developer's role: The level of a dialog refers to its position within the application structure. The application developer can pass processing control to a dialog at the next lower level, the same level, the next higher level, or the top level of the application structure.

Note: The meaning of TOP changes whenever a LINK command is executed. The dialog issuing LINK becomes the current TOP.

Aspects influenced: At runtime, the dialog level affects the following aspects of an application:

- **Availability of data** — When combined with the manner in which processing control is received, the level of a dialog governs the data passed in the record buffer blocks and the currencies that are established, saved, stored, or released.
- **Use of system resources** — The runtime system maintains record buffer blocks, database currency blocks, and variable dialog blocks for dialogs at each level. There is a direct correlation between the number of dialog levels in an application and the size of the storage pool that is needed.
- **Performance** — The number of dialog levels can affect the performance of an application. For example, performance times are affected if a frequently accessed dialog is located three or four levels down in an application structure.

An application can be composed of any number of dialog levels, but the most efficient application uses many levels only when absolutely necessary.

The top-level dialog must be a **mainline** dialog and must be defined as such by the application developer. A mainline dialog is the entry point to the application. An application can have more than one mainline dialog; entry points can also be established at a lower level in the application structure. In addition to defining a task code for the top-level dialog, the developer can identify an alternative entry point by using the Task Definition screen to associate a task code with a lower-level function.

5.1.2 Dialog Status

Two types: A dialog can have an **operative** or a **nonoperative** status within the application thread. A dialog becomes operative when it receives control and begins executing; at a given level, only one dialog can be operative at a time.

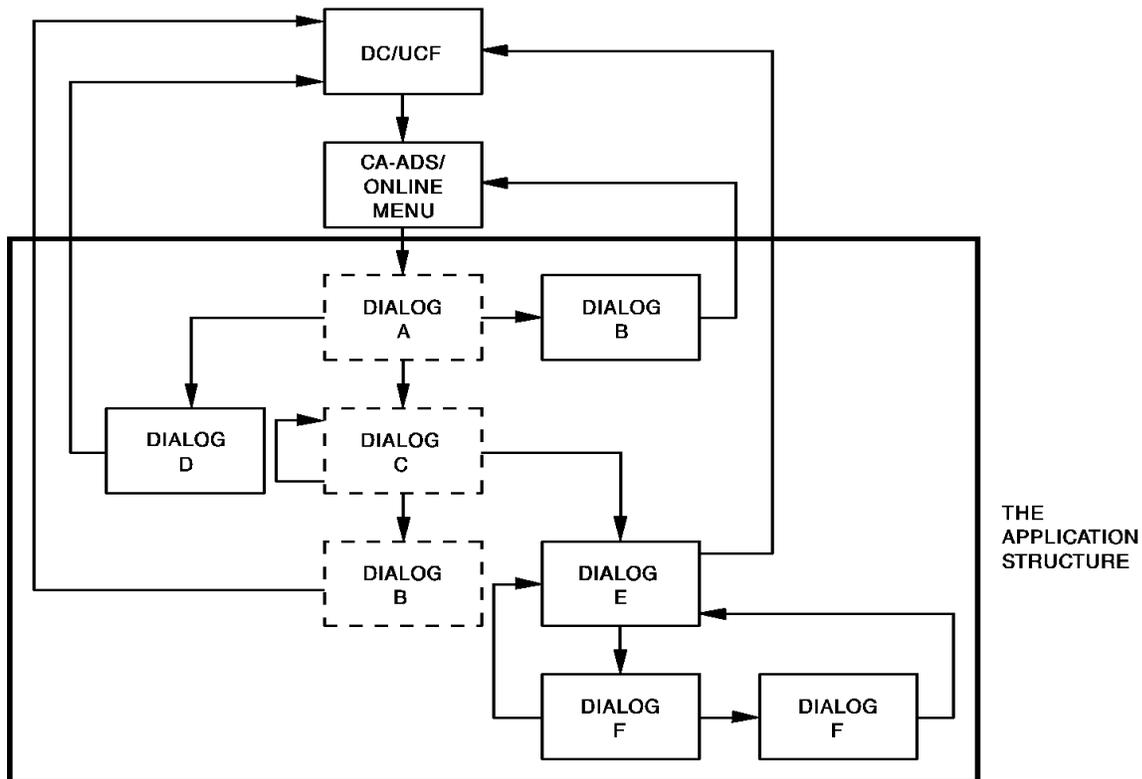
When control passes to a dialog at another level, the issuing dialog can remain operative or can become nonoperative, depending upon the level of the next dialog. For example, when control is passed with the LINK command, the issuing dialog remains operative; when control is passed with the TRANSFER command, the issuing dialog becomes nonoperative.

As long as a dialog is operative, all data that it has acquired is retained. When a dialog becomes nonoperative, its data is released. See the table, later in this chapter, that summarizes the way in which a dialog's status is affected by the successful execution of a control command.

Sequence of dialog execution: Within the application structure, only one dialog executes at a time. The sequence of dialog execution within an application structure is called the **application thread**. The response of the user determines the dialogs that constitute a given application thread. A figure later in this chapter shows an application structure and one application thread.

One dialog can exist in several places within the application structure and be part of the same or different application threads. A dialog can execute more than once within the application thread whether or not it remains operative.

In the next figure, the boxes with dotted lines represent an application thread that includes dialog A, dialog C, and dialog D.



5.1.3 Dialog Control

Passing control to another dialog: A dialog passes control to another dialog based on the execution of a control command and/or the user's selection of processing. The dialog that receives control can be a different dialog, a copy of the executing dialog, or all or part of the executing dialog itself.

The application developer can use specific control commands to perform the following operations:

- Pass processing control from one dialog to another dialog or to a user program
- Display a dialog's map
- Terminate an existing dialog or application
- Exit CA-ADS
- Pass processing control to specified points within a dialog and reinitialize the record buffers associated with a dialog

Most of the control commands used are available to all applications. When designing dialogs that will become part of an application that is compiled in an ADSA session, the developer can also use the EXECUTE NEXT FUNCTION command.

For a discussion of the commands that direct the flow of control within an application, see Appendix A, “Application Concepts” on page A-1. This appendix also contains a diagram and discussion of how the runtime system determines the order in which the functions of an application are executed.

5.2 Design Considerations

The application developer needs to keep the following CA-IDMS/DB, DC/UCF, and CA-ADS system features in mind when designing the dialogs:

- Record buffer management
- Working storage areas
- Global records

5.2.1 Record Buffer Management

What affects record buffer management:

At the beginning of each application thread, the CA-ADS runtime system allocates a primary Record Buffer Block (RBB) and initializes a buffer in the RBB for each record associated with the top-level dialog. All lower-level dialogs can access records in any of the existing buffers, unless one of the following conditions is true:

- The dialog that receives control accesses a record that has been assigned the NEW COPY attribute during dialog generation.
- The dialog that receives control accesses a record not used by a higher-level dialog.
- The dialog that receives control issues navigational DML statements to access a record that uses a subschema not used by a higher-level dialog.

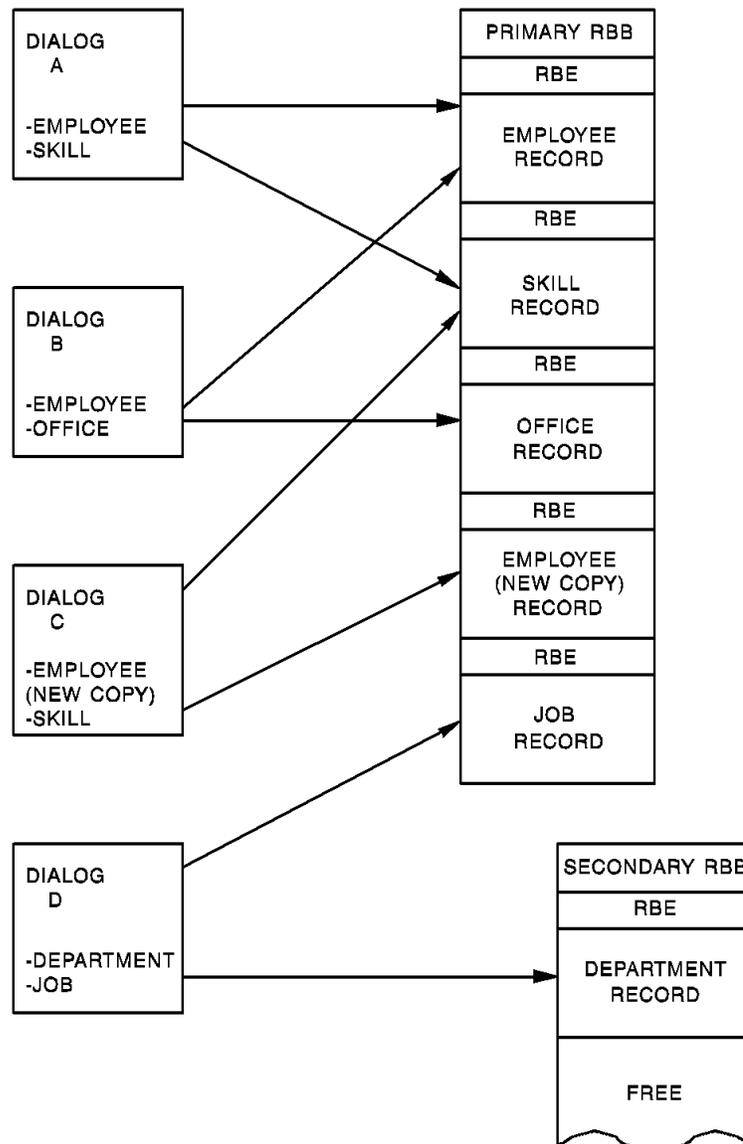
If one or more of these conditions exist, CA-ADS allocates and initializes an additional buffer for the record.

Additional buffers are also allocated and initialized when one of the following situations exists:

- The record is assigned the WORK RECORD attribute during dialog generation.
- The record is associated with the map used by the dialog.
- The record is named explicitly in a database command.

Record buffer allocation: The following example illustrates the sequence in which CA-ADS initializes record buffers as a series of dialogs receives control.

When dialog A begins executing, CA-ADS allocates buffers for the EMPLOYEE and SKILL record types. Dialog B uses the previously allocated EMPLOYEE record buffer, but requires a new buffer for the OFFICE record. Dialog C requests and receives a new copy of the EMPLOYEE record buffer, but uses the previously allocated SKILL record buffer. Dialog D requires new buffers for both the DEPARTMENT and JOB records. CA-ADS allocates a secondary RBB to accommodate the DEPARTMENT record, but uses the remaining space in the primary RBB for the JOB record.



NEW COPY records: Records or tables can be assigned the NEW COPY attribute during the definition of a dialog. The NEW COPY designation signifies that the record in question is to receive newly initialized record buffers when the dialog is executed.

The NEW COPY attribute is used when the programmer wants to obtain another occurrence of a record type without overwriting the data that is in the current buffer. To have the use of a second, temporary buffer for the same record type, the programmer links to a lower-level dialog that has specified NEW COPY for that record. An occurrence of the record type is brought into the new buffer and processed as directed. When control returns to the calling dialog, the record buffer at the upper level contains the same data as before; the data in the lower-level record buffer is no longer available.

Dialogs at a level lower than the dialog with a NEW COPY record will not use the NEW COPY buffer, but will use the first buffer allocated for the record.

5.2.2 Working Storage Areas

Queue, scratch areas: The DC/UCF system queue and scratch areas can be used by the CA-ADS dialogs as working storage areas. The methods by which dialogs can store and use records in the queue and scratch areas are presented below.

Queue records: Queue records can be used as work records that are shared by tasks on all DC/UCF system terminals. Entries are directed to a queue with database commands embedded in the dialogs or batch programs. Queues can transfer data across the entire DC/UCF system and are maintained across system shutdowns and crashes. Currencies and locks are not passed between tasks.

Note: When used in a sysplex environment, the queue area may be shared between multiple DC/UCF systems. For more information on shared queues, please see the *CA-IDMS System Operations* manual.

Queue records have the following characteristics:

- A queue header record is allocated either at system generation or by an application dialog.
- Queue records participate in a set in the dictionary; this set is commonly referred to as a queue.
- Queue records are locked by each task; no other task can use them until the locks are released.

Queues created at system generation with the system QUEUE statement can be accessed by an CA-ADS application. Additionally, an application can create its own queues by requesting storage space with a GET QUEUE statement in the dialog process code.

An application can use queue records to accomplish the following functions:

- **Automatically initiate a task** — The DC/UCF system initiates a task that processes the queue entries when the number of entries in a queue reaches a specified limit or when a specified time interval has passed. For example, an application can write records to a queue and the system will route the records to a printer when the collected records exceed the specified limit.
- **Avoid prime time updating** — Records that need to be updated can be collected on a queue; the queue can be accessed by a batch program at a low-use time.
- **Prevent run-away tasks** — A maximum limit can be established for the number of entries permitted in a queue. The UPPER LIMIT parameter of the QUEUE statement is especially useful in a test environment to prevent a looping program from filling the scratch/queue area.

Scratch records: Scratch records are shared between tasks and saved across the transactions of an CA-ADS application. Used as a temporary storage area, scratch records provide a means of passing data between tasks running on the same terminal; they are not accessible to tasks that execute on other terminals and are not saved across a system shutdown or a system crash.

The following characteristics are associated with scratch records:

- Scratch records are stored in the dictionary.
- Multiple scratch areas are allowed for a task and multiple records can be maintained within a scratch area.
- Currency is maintained for each area and record, and can be passed between tasks.
- The scratch area is allocated dynamically within the storage pool. When all scratch records are deleted, the area will also be deleted.

Scratch records can be used in the following ways within an application:

- To save input acquired from two or more dialogs over the course of the application.
- To allow multiple occurrences of a record to be mapped out at one time. For example, if the names, addresses, and phone numbers of all department employees need to be mapped onto the same screen in multiples of five, the following steps could be taken:
 1. Walk the set of employee records, moving the required data to a work record that contains multiply-occurring fields.
 2. When the work record contains the data on five employees, move the contents of the work record to the scratch area with a PUT SCRATCH command so that, in effect, a screenful of data on five employees is put on each record in the scratch file.
 3. Walk the set of scratch records when the screens of information are to be displayed.
- To pass the contents of the record buffer when a dialog receives control with a TRANSFER command. Data acquired by the dialog issuing a TRANSFER command is not available to the dialog receiving control. However, the dialog receiving control could access buffer data that had been placed in a scratch record.

5.2.3 Global Records

Global records are records that are available to all dialogs, maps, and user programs in an application. Subschema records cannot be defined as global records.

The ADSO-APPLICATION-GLOBAL-RECORD is the system-defined global record that enables communication between the application and the runtime system. To be accessed by a dialog, the ADSO-APPLICATION-GLOBAL-RECORD must either be specified as a dialog work record or be associated with the dialog's map. This record is initialized when an application is first loaded by the runtime system.

All fields in the ADSO-APPLICATION-GLOBAL-RECORD are addressable by dialogs or user programs. Selected fields from the ADSO-APPLICATION-GLOBAL-RECORD are listed below.

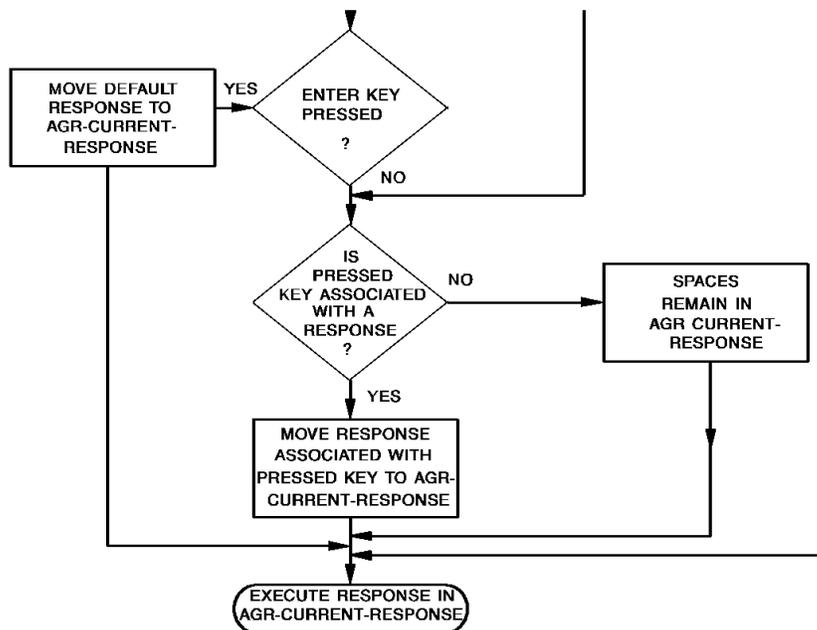
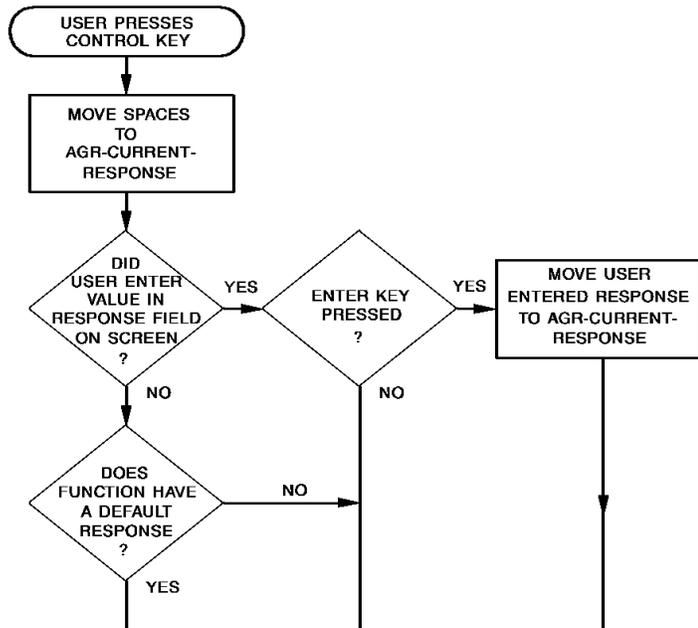
Selected fields

- The AGR-NEXT-FUNCTION field contains the name of the next function that is to be executed. When the dialog associated with the current function ends with an EXECUTE NEXT FUNCTION command, the function named in the AGR-NEXT-FUNCTION field is executed by the runtime system. A dialog or user program can query this field to check what the next function will be. Modification of the AGR-NEXT-FUNCTION field, however, does not change the next function to be executed; a change in the next function can only be accomplished by modification of the AGR-CURRENT-RESPONSE field (see below).
- The AGR-DEFAULT-RESPONSE field contains the default response value specified on the Function Definition screen when an application is generated. When a value is specified and the screen includes a data field for a default response, the user can type in a new value or can space out the value that appears.
- The AGR-CURRENT-RESPONSE field contains the response specified by the user. The process code of a dialog or user program can also move values into this field, overwriting the user response. Note that, if AGR-CURRENT-RESPONSE is modified by a dialog, security is not checked for the response moving into the field, even if security is associated with this response.

When EXECUTE NEXT FUNCTION is encountered within process code, the response named in the AGR-CURRENT-RESPONSE field is executed if it is a valid response for the current function. The AGR-CURRENT-RESPONSE field determines the next function in the application thread (that is, it determines the value moved into the AGR-NEXT-FUNCTION field).

The value in AGR-CURRENT-RESPONSE depends upon whether the AGR-DEFAULT-RESPONSE field contains a value; whether the user enters a new value in the response field; or whether there is a response value associated with the control key (other than ENTER) pressed by the user.

The following flowchart illustrates how the CA-ADS runtime system places a value in the AGR-CURRENT-RESPONSE field of the ADSO-APPLICATION-GLOBAL-RECORD. The runtime system executes the response named in the AGR-CURRENT-RESPONSE field after determining that it is a valid response for the current function.



- The AGR-EXIT-DIALOG field initially contains the name of the exit dialog specified on the Application Definition screen. This field can be used to link to a special routine.

For example, one department of a company might want the employee name specified as John Doe, while another department wants the name specified as Doe, John. The same dialog could be used for both departments by linking to an exit dialog (that is, LINK TO AGR-EXIT-DIALOG) containing a name routine.

- The AGR-PRINT-DESTINATION field initially contains the default name of the printer for the application as specified on the ADSA Application Definition screen. Dialogs and user programs can use this print destination with the WRITE PRINTER DESTINATION command.
- The AGR-USER-ID field can be queried by dialogs and user programs.
- The AGR-PRINT-CLASS field initially contains the default printer class for the application as specified on the ADSA Application Definition screen. The dialog can reference this field with the WRITE PRINTER CLASS command.
- The AGR-SIGNON-SWITCH field can be queried to determine if there has been a valid signon.
- The AGR-SIGNON-REQMTS field indicates whether signon is optional, required, or not used for the signon menu, as specified on the Security screen. This field can be referenced for additional security checking.
- The AGR-MAP-RESPONSE field can be used as a response field, in place of the \$RESPONSE field, in any user-defined nonmenu map. The dialog can initialize this response field before mapout so that the desired default response appears on the map. For input purposes, the AGR-MAP-RESPONSE field works in the same manner as the \$RESPONSE field.
 - ▶▶ For further information on the \$RESPONSE field, refer to *CA-IDMS Mapping Facility*.
- The AGR-MODE field initially contains the value STEP or FAST as specified on the Application Definition screen. Typically, the design of a dialog map includes a field that displays the value of AGR-MODE. The user can change this field at any time.

Two examples: In the following text, two examples of how the AGR-MODE field can be used are presented, with the EXECUTE NEXT FUNCTION command, to implement a STEP/FAST mode for an ADSA application. The logic in the first example assumes that all data field validation is handled by the automatic editing specifications in the dialog's map. The logic in the second example assumes that additional data validation is required in the response process code. In both cases, any data entered by the user is always processed. Note that the first pass flag field has no significance in FAST mode.

Using the AGR-MODE-field (example 1)

```
IF ANY OF (EMPLOYEE-NBR, SKILL-CODE, SKILL-LEVEL)
ARE CHANGED
DO.
    MOVE 'Y' TO FIRST-PASS-FLAG.
    MOVE EMPLOYEE-NBR TO WK-EMPnbr.
    MOVE SKILL-CODE TO WK-SKLCODE.
    MOVE SKILL-LEVEL TO WK-SKLEVEL.
    LINK TO 'CEMDUEMP'.
END.
IF AGR-STEP-MODE
DO.
    IF FIRST-PASS-FLAG='Y'
    DO.
        MOVE 'N' TO FIRST-PASS-FLAG.
        DISPLAY MSG TEXT IS 'EMPLOYEE UPDATED'.
    END.
    MOVE 'Y' TO FIRST-PASS-FLAG.
END.
EXECUTE NEXT FUNCTION.
```

The preceding sample process code illustrates the manner in which a dialog can query the AGR-MODE field of the ADSO-APPLICATION-GLOBAL-RECORD to determine what course to follow. If the dialog is in STEP mode, the dialog redisplay the screen with a confirmation message for the user; if in FAST mode, control is passed immediately to the next function. The initial value of AGR-MODE is supplied by the runtime system; the user can alter the value of AGR-MODE at any time during application execution.

Using the AGR-MODE field (example 2)

```

IF ANY OF (EMPLOYEE-NBR, SKILL-CODE, SKILL-LEVEL)
ARE CHANGED
DO.
  MOVE 'Y' TO FIRST-PASS-FLAG.
  IF EMPLOYEE-NBR GE 2000 AND SKILL-CODE='A'
  DO.
    MOVE 'Y' TO ERROR-FLAG.
    DISPLAY MSG TEXT IS
      'EMPLOYEE NUMBER/SKILL CODE MISMATCH'.
    END.
    MOVE 'N' TO ERROR-FLAG.
    MOVE EMPLOYEE-NBR TO WK-EMPnbr.
    MOVE SKILL-CODE TO WK-SKLCODE.
    MOVE SKILL-LEVEL TO WK-SKLEVEL.
    LINK TO 'CEMDUEMP'.
    CALL EMPDTE25.
  END.
IF ERROR-FLAG='Y'
  DISPLAY MSG TEXT IS
    'EMPLOYEE NUMBER/SKILL CODE MISMATCH'.
CALL EMPDTE25.
!*****
DEFINE EMPDTE25.
!*****
IF AGR-STEP-MODE
DO.
  IF FIRST-PASS-FLAG='Y'
  DO.
    MOVE 'N' TO FIRST-PASS-FLAG.
    DISPLAY MSG TEXT IS 'EMPLOYEE UPDATED'.
  END.
  MOVE 'Y' TO FIRST-PASS-FLAG.
END.
EXECUTE NEXT FUNCTION.

```

The sample code shown in the preceding figure illustrates the use of the AGR-MODE field when data validation needs to be handled by code in the response process. Note that the EXECUTE NEXT FUNCTION command is never encountered while uncorrected validation errors still exist.

Mapping to screens: The following fields from the ADSO-APPLICATION-GLOBAL-RECORD are often mapped to screens associated with user-defined nonmenu maps:

- AGR-DIALOG-NAME
- AGR-APPLICATION-NAME
- AGR-CURRENT-FUNCTION
- AGR-FUNCTION-DESCRIPTION
- AGR-DATE
- AGR-USER-ID
- AGR-MODE
- AGR-PASSED-DATA
- AGR-MAP-RESPONSE

For an illustration of how these fields can be used on maps, refer to Chapter 4, “Designing Maps” on page 4-1.

5.3 Dialogs That Issue Navigational DML

Additional design considerations apply to dialogs that issue navigational DML commands. These considerations are:

- Database currencies
- Extended run units
- Longterm locks
- Record buffer management for logical records

5.3.1 Database Currencies

How currency is maintained: In CA-ADS, currency is maintained automatically for the user. To facilitate this feature, a currency control block is created that maintains currency information. At run time, a currency block is created for each dialog in the application structure that performs database requests.

Database currencies are passed from one dialog to another dialog at a lower level, enabling dialogs to continue database processing from an established position in the database. Currencies are cumulative. The currencies established by each dialog are passed to lower-level dialogs, which, in turn, establish their own currencies; the cumulative currencies are passed to the next lower-level dialog.

Currencies are established, saved, restored, and released as follows:

- **Established** — Currency is established with the dialog's first functional database call. Established currencies are updated when database commands (for example, FIND, OBTAIN, ERASE) are encountered during the run unit. Currency is nulled when a dialog receives control with a RETURN or TRANSFER command.
- **Saved** — When a LINK, DISPLAY, or INVOKE command is issued, the database currencies established with the last database command in the dialog are saved. Saved currencies are available to lower-level dialogs and are restored to the issuing dialog if processing control returns.
- **Restored** — Saved currencies are restored when CA-ADS opens a run unit in the dialog receiving control (that is, saved currencies are restored just prior to the first database call).
- **Released** — When a LEAVE, RETURN, or TRANSFER command is issued, all database currencies at the same and lower levels are released. The dialog receiving control must establish its own currencies or use the currencies passed to it from another higher-level dialog.

The successful execution of control commands can affect the operative or nonoperative status of a dialog, the dialog's acquired data that is retained or released, and the currencies that are saved, restored, or released.

The effects of control commands: The following table illustrates the ways in which the passing and receiving of control affects the contents of the currency block.

Command	New Level Established	Status of Issuing Dialog	Data Available to Receiving Dialog	Currency Action	
				Issuing Dialog	Receiving Dialog
DISPLAY	No	Operative	All data	Saved	N/A
INVOKE	Yes	Operative	All data	Saved	Restored
LEAVE	No	Nonoperative	No data	Released	Must establish
LINK DIALOG PROGRAM	Yes	Operative	All data	Saved	Restored
	No	Operative	All, some, or none (depending on command specification)	Saved	Must establish
RETURN	No	Nonoperative (any operative dialogs between the issuing dialog and the receiving dialog also become nonoperative)	Data previously acquired by the receiving dialog	Released (currencies for any dialogs between the issuing dialog and the receiving dialog are also released)	Restored
TRANSFER	No	Nonoperative	All data except that acquired by the issuing dialog	Released	Can use currencies previously established by higher-level dialogs

5.3.2 Extended Run Units

Definition: Typically, an CA-ADS run unit begins when the dialog issues a command accessing the database (for example, OBTAIN) and ends when the runtime system encounters the next control command issued by the dialog (that is, LINK, INVOKE, DISPLAY, TRANSFER, LEAVE, or RETURN).

An extended run unit is a run unit that is kept open when the runtime system encounters the LINK command under the following circumstances:

- When the LINK is to the premap process of a dialog with no associated subschema
- When the LINK is to the premap process of a dialog with an associated schema and subschema identical to those of the calling dialog
- When the LINK is to a user program

Implications of the extended run unit are as follows:

- Currencies are passed to the lower-level dialog and are restored upon return to the upper-level dialog.
- Currencies are not passed to user programs; currencies are saved and restored to the upper-level dialog when control is returned.
- The lower-level dialog can perform error checking to decide whether to issue a ROLLBACK command.
- Because a FINISH is not issued, record locks held by the upper-level dialog are not released. A COMMIT can be coded in the upper-level dialog if the developer needs to release locks before linking to the lower-level dialog.
- If a COMMIT is issued prior to the LINK command and an abend occurs in the lower-level dialog, the rollback will be incomplete; the rollback will only go to the COMMIT checkpoint and not to the start of the run unit.
- If a lower-level user program opens its own run unit, a deadlock can occur. The possibility of a deadlock condition can be avoided by taking either of the following actions:
 - Issue a COMMIT prior to the LINK.
 - Pass the subschema control block to the user program and let the program use the same run unit. Issue no BINDs or FINISHes in the user program.

5.3.3 Longterm Locks

KEEP LONGTERM is a navigational DML command that sets or releases longterm record locks. Longterm locks are shared or exclusive locks that are maintained across run units. Once the longterm locks are set, all other run units are restricted from updating or accessing the named records until the dialog explicitly releases the locks.

Example: The following example requests the release of all longterm locks associated with the current task:

```
KEEP LONGTERM ALL RELEASE
```

The KEEP LONGTERM command can also be used to monitor the database activity associated with a record, set, or area. When a dialog is updating records that could also be updated by another user, the following code can be included in the premap process of the named dialog:

```
KEEP LONGTERM longterm-id NOTIFY CURRENT record-name
```

This command instructs the CA-ADS runtime system to monitor the database activity associated with the current occurrence of the named record type.

The following code is included in the response process of the same dialog:

```
KEEP LONGTERM longterm-id
TEST RETURN NOTIFICATION INTO return-location-v
```

This command requests notification of any database activity against records that were specified in the KEEP LONGTERM premap process. If appropriate, the dialog can check the return value placed in the specified work record field.

5.3.4 Record Buffer Management for Logical Records

When an application thread contains dialogs that use a combination of database records and logical records, special considerations apply with respect to record buffer management. For each database record component of a logical record, CA-ADS initializes individual, contiguous record buffers. The logical record components are placed in the buffer in the order named in the logical record definition.

For example, consider the EMP-JOB-LR logical record, which consists of four database records: EMPLOYEE, DEPARTMENT, JOB, and OFFICE records. If dialog B accesses EMP-JOB-LR, CA-ADS initializes new record buffers for each of the four records listed above (in that order) *regardless* of whether buffers for one or more of the records were initialized when dialog A, a higher-level dialog, began executing. Therefore, dialog B (and lower-level dialogs accessing the same logical record) does not have access to data established in the record buffer by dialog A. However, dialogs at levels lower than dialog B will use the buffers established by dialog A if those dialogs use the same database records as dialog A.

When using both database records and logical records, the first dialog of the application thread should include an INITIALIZE command for the logical record. This action associates the logical record with the top-level dialog and ensures that the buffer for the entire logical record will be allocated and available to all lower-level dialogs. Lower-level dialogs will use the component record buffers established at the highest level unless the logical record itself is referenced.

Chapter 6. Naming Conventions

- 6.1 Overview 6-3
- 6.2 Naming Application Entities 6-4
- 6.3 Naming Database Information Entities 6-7

6.1 Overview

The establishment of naming conventions reduces the accumulation of redundant data and improves the overall design of an application. Naming convention standards apply to the components of an application as well as to the database entities accessed by the application. Naming conventions for application entities and database information entities are each discussed separately below.

6.2 Naming Application Entities

Naming conventions make it easier to keep track of application components as they are created and maintained. While mnemonic names can work well for less complex applications, mnemonics are inadequate when handling the large volume of complex applications that typically exist at most sites. Adhering to a naming convention eases the construction of component names, eases the reconstruction of component names if one is forgotten, and eases the use and maintenance of an application.

Sample naming conventions: The table below lists the naming convention standards used for the sample application in this manual.

Position	Value	Meaning
1	C	CA product
2-3		Type of application:
	EM	Employee information
	IS	Information system
	FS	Financial system
	MS	Manufacturing system
	SY	System activities
4		Component type:
	D	Dialog
	F	Function
	M	Map
	P	User-defined program
	R	Report
	S	Subschema
	T	Table
	U	Menu
5		Component functions:
	A	Add operation
	C	Encode/decode (column 4 indicates table)
	D	Delete operation
	E	Edit operation (column 4 indicates tables)
	I	Inquiry operation
	M	Modify operation
	U	Update operation
6-8		Component designator
	xxx	Three characters used as unique identifiers

Assigning names: Names in an application can be assigned in the following manner:

- Dialogs, maps, tables, programs, and reports can use the conventions in the previous table, as follows:

Dialog: CEMDILIS
Map: CEMMILIS
Code table: CEMTCLIS
Edit table: CEMTELIS
Menu: CEMUILIS
User program: CEMPILIS
Report: CEMRILIS

- Dialog premap and response process names can be the concatenation of the dialog name and the suffix -PREMAP or -RESPONSE, as in the following examples:

CEMDILIS-PREMAP

CEMDILIS-RESPONSE

If there are multiple response processes, the suffixes can be structured to reflect the function of each response process, as follows:

CEMDILIS-ADDRESP

CEMDILIS-DELRESP

- Names for subroutines included in the premap and response processes can be made up of a meaningful name of up to six characters with a 2-digit suffix, as follows:

PASSDT05

MESSGE97

DBERR99

The numeric suffixes can be assigned and incremented as the subroutines appear in the dialog. This numbering convention makes it easier to locate a subroutine in the dialog listing. For example, MESSGE97 is located near the end of the listing while PASSDT05 is located near the beginning.

6.3 Naming Database Information Entities

The creation of a glossary can be an effective means of establishing naming conventions for database information. The glossary can be stored in the dictionary where it is readily available as a reference tool. Tools such as the glossary also aid in the development of consistent site-specific application coding standards.

Sample glossary of naming tokens:

The following figure illustrates sample entries from one type of glossary. The following example shows one way in which a glossary can be defined; each design team must determine the naming conventions that best suit its needs. Note that the word **WORD** in this example is a user-defined entity defined to the dictionary, as follows:

```

ADD CLASS NAME IS WORD
CLASS TYPE IS ENTITY.
ADD WORD ABEND           ABBREVIATED NEVER
ADD WORD ABSOLUTE       ABBREVIATED NEVER
ADD WORD ACCEPT         ABBREVIATED NEVER
ADD WORD ACCOUNT        ABBREVIATED SOMETIMES ABBR ACCT
ADD WORD ACCRUAL        ABBREVIATED NEVER
ADD WORD ACCUMULATE     ABBREVIATED SOMETIMES ABBR ACCUM
ADD WORD ACKNOWLEDGE    ABBREVIATED SOMETIMES ABBR ACK
ADD WORD ADMINISTRATION ABBREVIATED ALWAYS ABBR ADMIN
ADD WORD ADDRESS        ABBREVIATED ALWAYS ABBR ADDR
.
.
.
.
.
ADD WORD YIELD          ABBREVIATED SOMETIMES ABBR YLD
ADD WORD YTD            ACRONYM 'YEAR TO DATE'
ADD WORD YY             ABBREVIATED NEVER
ADD WORD ZERO          ABBREVIATED NEVER
ADD WORD ZONE          ABBREVIATED NEVER

```

The sample entries from this glossary show one way in which naming conventions can be implemented within an installation. In this glossary, the application designers have determined that certain words are always to be abbreviated and others are never to be abbreviated; the majority of words are to be spelled out completely whenever possible. When stored on the dictionary, the glossary is readily available as a reference guide for programmers and developers.

Information entities can use the following naming conventions:

Available naming conventions

- Database elements can be established using approved names from the glossary and can be further defined with synonyms. Element names should have a maximum of 25 characters. The following example lists an element and three synonyms:

EMPLOYEE-CODE

DB-REC-EMPLOYEE-CODE
MAP-EMPLOYEE-CODE
WORK-EMPLOYEE-CODE

- Database Records can be composed of approved, usable names (for example, EMPLOYEE). Records can be given greater flexibility with the addition of suffixes. The following example lists employee records with identifying suffixes:

EMPLOYEE-0600
EMPLOYEE-2500
EMPLOYEE-6359

SQL: Hyphens are not valid in SQL identifiers referenced in statement syntax. Therefore, SQL entities may not be named using hyphens but may be named using underscores. Hyphens are valid in host variables referenced in SQL statement syntax.

For more information, refer to *CA-IDMS SQL Programming*.

In CA-ADS process source, as well as in COBOL, CA-CULPRIT, and map source, the elements can be referenced by the element name plus the suffix, as follows:

EMPLOYEE-CODE-6359

- Map work records are composed of the map name followed by the suffix -MAP-RECORD, as in the following example:

CEMMILIS-MAP-RECORD

Elements in the map record utilize the prefix MAP- and the element name, as follows:

MAP-OFFICE-CODE

If the map needs more than one work record, a number is added to the word MAP, as follows:

CEMMILIS-MAP2-RECORD (the second map record)

MAP2-OFFICE CODE (a record element from the second record)

- Dialog work records are composed of the dialog name followed by the suffix -WORK-RECORD as in the following example:

CEMDILIS-WORK-RECORD

Elements in the dialog work record utilize the prefix WORK- and the element name, as follows:

WORK-OFFICE-CODE

If the dialog needs more than one work record, a number is added to the word WORK, as follows:

CEMDILIS-WORK2-RECORD (the second dialog work record)

WORK2-OFFICE CODE (a record element from the second record)

- Set names are established by concatenating an abbreviation of the owner record (a seven-character maximum) with that of the member record (a six-character maximum), as follows:

EMPL-SKILL

Chapter 7. Performance Considerations

- 7.1 Overview 7-3
- 7.2 System Generation Parameters 7-4
 - 7.2.1 ADSO Statement Parameters 7-4
 - 7.2.2 PROGRAM Statement Parameters 7-4
 - 7.2.3 TASK Statement Parameters 7-4
 - 7.2.4 Allocating Primary and Secondary Storage Pools 7-5
 - 7.2.5 Setting the Fast Mode Threshold 7-6
 - 7.2.6 Specifying the Number of Internal and External Run Units 7-7
- 7.3 Resource Management 7-8
 - 7.3.1 Monitoring Resource Consumption 7-9
 - 7.3.2 Conserving Resources 7-14

7.1 Overview

The performance of the CA-ADS runtime system is dependent upon a number of factors, such as the size of the DC/UCF system, the number of applications being run concurrently, and the number of users for a given application. Rather than attempting to give definitive instructions for the improvement of performance, this chapter discusses the following aspects of the CA-ADS runtime system:

- Parameters affecting performance
- Resource management

Each of these considerations is discussed separately below.

7.2 System Generation Parameters

The CA-ADS runtime system is generated by submitting ADSO, PROGRAM, and TASK statements to the CA-IDMS system generation compiler. Optionally, the KEYS statement is used to define site-specific control key functions.

►► For detailed syntax and examples of system generation statements, refer to *CA-IDMS System Generation*.

7.2.1 ADSO Statement Parameters

The ADSO statement includes parameters that define the CA-ADS runtime environment, as follows:

- The task code (ADS) that initiates the CA-ADS runtime system
- The mainline dialog that can begin executing immediately
- The maximum number of dialog levels that can be established by each application
- The disposition of record buffers during a pseudo converse
- The size of the primary and secondary record buffers
- The AUTOSTATUS facility that handles errors generated by navigational DML, queue record, and scratch record processing
- The Status Definition Record that associates status codes returned by non-SQL data processing

7.2.2 PROGRAM Statement Parameters

The PROGRAM statement defines the following CA-ADS components as DC/UCF system programs:

- The ADSORUN1, ADSORUN2, and ADSOMAIN runtime system programs
- The system maps (the menu map, runtime message map, and maps for each of the application and dialog compiler screens)
- The application and dialog compiler programs (ADSA and ADSC)
- CA-ADS dialogs (an optional parameter if null Program Definition Elements (PDEs) are defined in the SYSTEM statement)

7.2.3 TASK Statement Parameters

The TASK statement defines the following task codes:

- ADS and ADS2 to initiate the runtime system
- ADSA to initiate the CA-ADS Application Compiler
- ADSC to initiate the CA-ADS Dialog Compiler

- ADSR to initiate the runtime system when returning from a linked user program

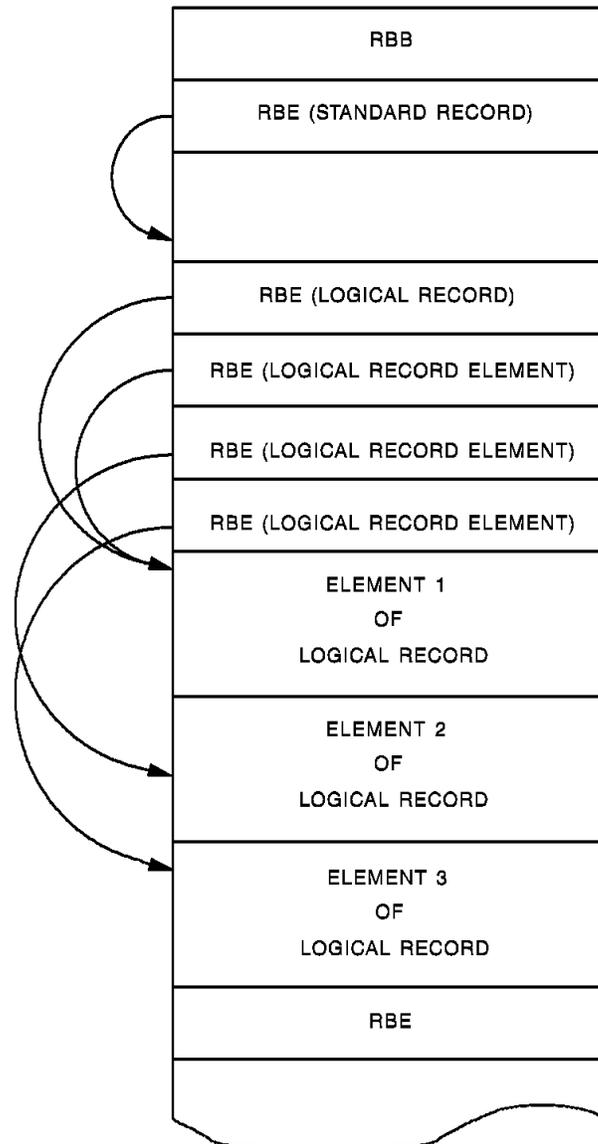
7.2.4 Allocating Primary and Secondary Storage Pools

How storage is managed: The runtime system allocates and initializes record buffers for use by executing dialogs. When an application is initiated, CA-ADS allocates a Record Buffer Block (RBB) from the DC/UCF system's storage pool to hold the records identified in the dialog definitions and accessed by the dialogs in the application thread. The RBB must be large enough to accommodate the largest of these records.

There is one primary RBB for each application. CA-ADS allocates a secondary RBB when the RBB becomes full during execution of the application or does not have enough remaining space to hold a record.

Additional secondary RBBs can be allocated by the CA-ADS runtime system as necessary. The data communications administrator (DCA) specifies the size of the primary and secondary RBBs with the PRIMARY POOL and SECONDARY POOL parameters of the ADSO statement. When allocating the primary and secondary storage pools, the DCA needs to consider the size and number of the records used by the application as well as the header records maintained by the buffers.

Layout of the record buffer block: The following figure diagrams the structure of the Record Buffer Block allocated for a combination of subschema records and logical records:



Size considerations: Each record buffer contains a 24-byte header to keep track of available space. For each record in the pool, CA-ADS maintains a record header (RBE) that requires at least 44 bytes of storage. Each buffer must be large enough to accommodate the largest record used by a dialog in the application.

7.2.5 Setting the Fast Mode Threshold

Affects where record buffers written: The fast mode threshold is used by the CA-ADS runtime system to determine whether record buffers are written to disk or kept in main storage across a pseudo converse. If the total size of all record buffers, in bytes, exceeds the fast mode threshold, the record buffers are written to disk; otherwise, the record buffers are kept in the storage pool.

The size of the threshold is a site-specific determination that is based on the availability of general resources versus the amount of available storage. I/Os for DC/UCF system journaling and CPU cycles for record locking are used when record buffers are written to the scratch/queue areas. Therefore, when buffers exceed the fast mode threshold, the increased use of resources will slow down the transaction response time. On the other hand, if buffers are always under the threshold (that is, if the fast mode threshold is high), more memory is required.

7.2.6 Specifying the Number of Internal and External Run Units

The `MAXIMUM TASKS` and `MAXIMUM ERUS` parameters specify the maximum number of user tasks and external request units that can be active concurrently. The size of these parameters can affect the amount of time spent by the DC/UCF system in searching the queues for tasks that are waiting to be executed.

The numbers that should be specified are a site-specific determination and are dependent upon factors such as the number of tasks processed each hour in a particular environment. When setting the `MAXIMUM TASKS` and `MAXIMUM ERUS` parameters on the `SYSTEM` statement, the following statistics should be considered:

- Increasing the `MAXIMUM TASKS` or `MAXIMUM ERUS` parameters by one (1) causes virtual storage requirements to increase as shown below:

Resource	Size of resource	Total
TCE	736 bytes	736 bytes
STACKSIZE	320 words	1,280 bytes
DCE	64 bytes	64 bytes
ECB * 3	8 bytes	24 bytes
DPE * 20	16 bytes	320 bytes
RCE * 15	24 bytes	360 bytes
RLE * 25	12 bytes	300 bytes
Total increase per task		3,084 bytes

Note that a value larger than the default (420) should be specified for the `STACKSIZE` when using CA-ADS. If the `STACKSIZE` is at 420 and two tasks exceed stacksize and go into abend storage at the same time, the system will abort with an abend code of 3995.

- The following DC/UCF system parameters should be increased as specified for every increment of one (1) in the size of `MAXIMUM TASKS` or `MAXIMUM ERUS`:

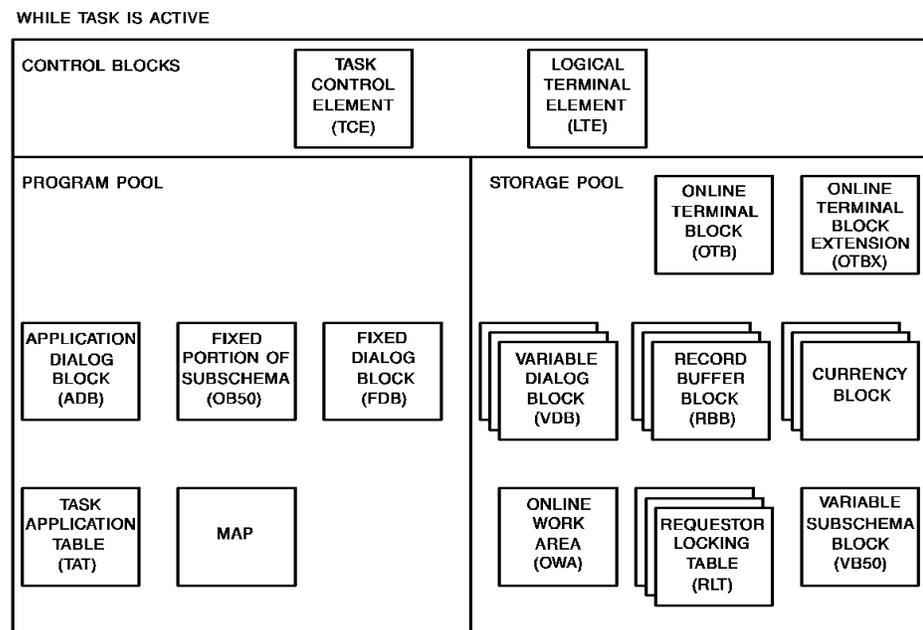
Parameter	Amount increased
ECB LIST	3
DPE COUNT	20
RCE COUNT	15
RLE COUNT	25

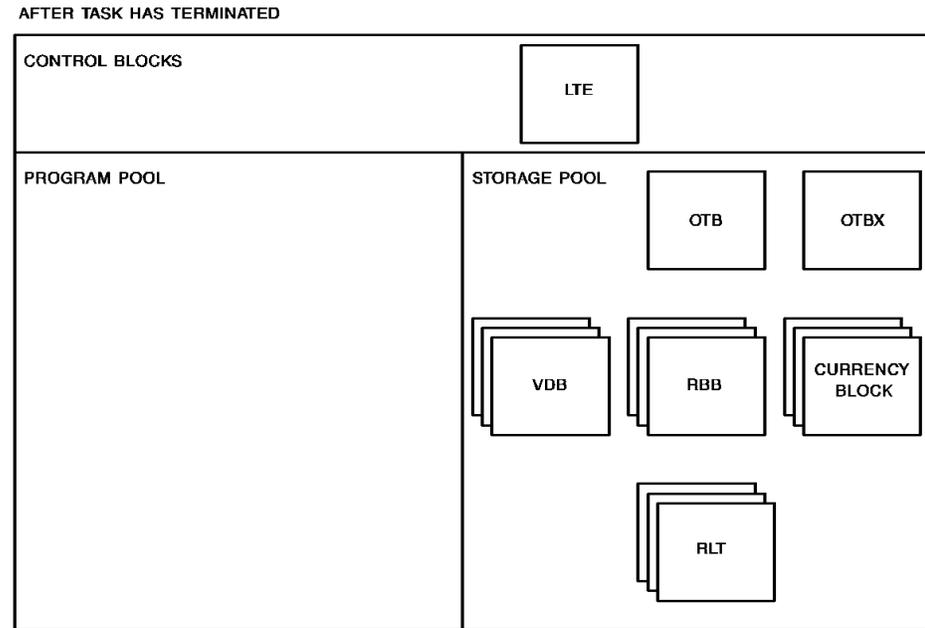
7.3 Resource Management

In designing applications, consideration must be given to the efficient management of system resources. The management of resources such as the database, the storage pool, and the program pool storage affects the performance of online applications because many users may require access to these resources simultaneously.

The following figure illustrates the resources used by an application while a task is active and after the task has terminated.

Application resource use





7.3.1 Monitoring Resource Consumption

The remainder of this chapter discusses methods that can be used to monitor the resource consumption of an application and ways in which to use available resources efficiently.

Tools: As with any task running under the DC/UCF system, the major resources to be monitored in a CA-ADS environment are as follows:

- Task processing support
- Variable storage pool
- Program pool storage
- Database locks
- I/Os (disk and terminal data transmission)
- CPU cycles

Each of these resources can be monitored with dictionary reports and DC/UCF system master terminal functions, as discussed below.

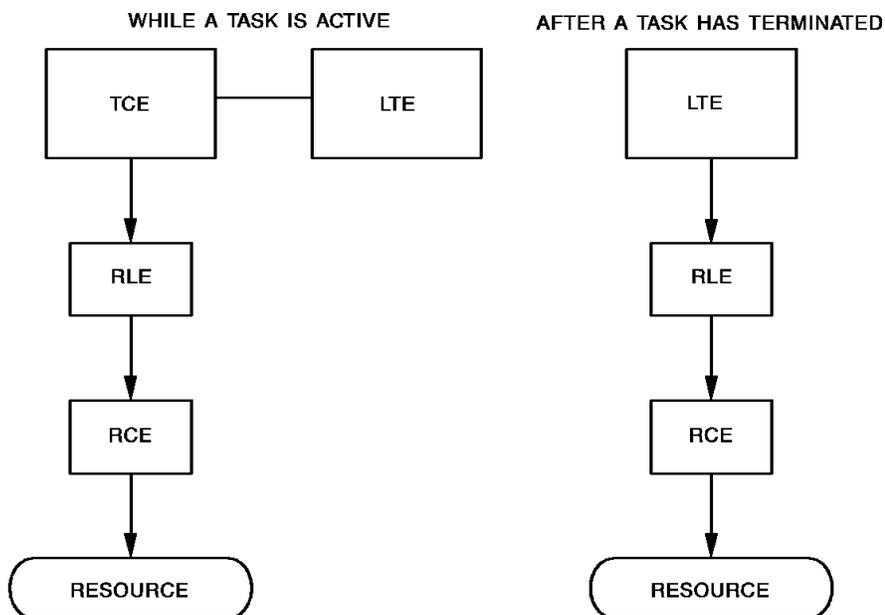
Task processing support:

The next figure shows the resources in use while a task is active and those in use after

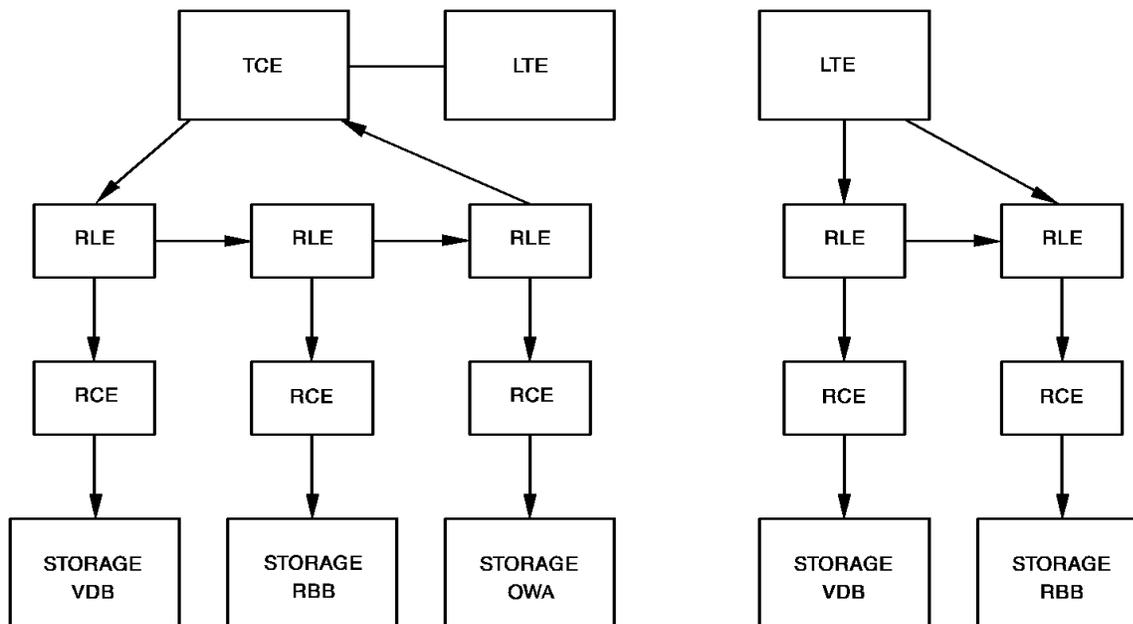
the task terminates. The following DC/UCF system master terminal functions display the internal resources used to support task processing:

- DCMT DISPLAY ACTIVE TASK displays global statistics on active tasks and information on each active task thread.
- DCMT DISPLAY STATISTICS SYSTEM displays information about the system including the peak task control element (TCE) stack; and the maximum number of resource link elements (RLEs), resource control elements (RCEs), and deadlock prevention elements (DPEs) used by the tasks.

Task resource structure



EXAMPLE:



Variable storage pool: The following sysgen reports (CREPORTS) and DCMT functions can be used to monitor the use of the storage pool:

- CREPORT 25 verifies the size of the storage pool and indicates whether storage protection has been enabled for the system.
- DCMT DISPLAY ACTIVE STORAGE shows the current fragmentation of the storage pool.
- DCMT DISPLAY LTERM indicates which terminals are active and own resources.
- DCMT DISPLAY LTERM *logical-terminal-id* RESOURCES displays the specific resources (and the addresses of those resources) owned by the named terminal.
- DCMT DISPLAY MEMORY can be used to display an actual resource as it appears in memory.
- CREPORT 40 supplies the current parameters specified in the ADSO statement, as it this example:

REPORT NO. 40		CA-IDMS/DC ADS REPORT						
		LISTING OF CA-ADS PARAMETERS						
		OBJECT REPORT						
SYSTEM VERSION	AUTODIALOG	PRIMARY TASK CODE	SECONDARY TASK CODE	MAXIMUM LINKS	MENU IS	FAST MODE THRESHOLD	PRIMARY POOL	
90		ADS	ADS2	10	USER	50000	40	
99		ADS	ADS2	10	USER KEEP	OFF	40	

Information from the above displays and reports can be used to calculate the number of users the system can currently support, assuming various storage pool sizes.

The *CA-IDMS System Generation* manual describes CREPORTS; the *CA-IDMS System Operations* manual details the master terminal functions available to monitor system resources.

Program pool storage: The following DCMT commands can be used to provide information on the program pool:

- DCMT DISPLAY ACTIVE PROGRAMS displays the following:
 - Statistics on program pool usage, including the total number of pages and total number of bytes in the pool; the number of loads to the program pool; the number of pages loaded; and the number of load conflicts
 - Information on currently active programs including the program name, type, and version number; count of users currently using the programs; size of the program in K bytes; the number of times the program was called; and the number of times the program was loaded into the program pool
 - The program pool page allocation map that shows which pages are not in use; which pages are in use by one program; and which pages are in used by more than one program
- DCMT DISPLAY ACTIVE REENTRANT PROGRAMS displays the above information for the reentrant program pool and the active reentrant programs. If no reentrant pool is defined, the standard program pool is shown.

Database locks: The DCMT DISPLAY RUN UNIT and OPER WATCH DB RUN UNITS commands can be used to show the number of database locks being requested for a particular run unit. The number of database locks maintained by a CA-IDMS system has considerable impact on CPU usage. These locks are specified at sysgen time by the RULOCKS and SYSLOCKS parameters of the SYSTEM statement.

►► For further information on database locks, refer to *CA-IDMS Database Design*.

►► For further information on factors to consider when preparing the SYSTEM statement, refer to *CA-IDMS System Operations*.

Disk I/O: The following reports can be used for monitoring disk I/O:

- JREPORT 004 shows the average number of I/Os to disk for a given program.
- DCMT DISPLAY RUN UNITS or OPER W DB RU shows if any run units are waiting for a journal buffer (as indicated by a run unit status value of IUH). IUHs occur most frequently when the fast mode threshold is set too low.

►► For information on JREPORTS (journal reports), refer to *CA-IDMS Reports*.

Terminal I/O: The following steps can be taken to monitor terminal I/Os:

1. Run the mapping utility (RHDCMPUT) for a report on a specific map. This report will display a picture of the map and the attributes currently assigned to the map. The report will also indicate whether BACKSCAN is enabled for any mapping fields. If BACKSCAN is in effect and the NEWPAGE option on the ADSO statement has been selected, extraneous data from the previous mapout may be left on the screen when a map is redisplayed. It is advantageous to have NEWPAGE in effect, however, because this option increases runtime efficiency by reducing the number of data fields that need to be transmitted to the terminal.
2. Use DCMT VARY PTERM *physical-terminal-id* TRACE ALLIO to cause the datastream being transmitted to the terminal to be written to the log as well.
3. Use SHOWMAP *map-name* with DCUF USERTRACE to cause the datastream of a particular map to be traced.
4. Use DCMT VARY PTERM *physical-terminal-id* TRACE ALLIO OFF to turn off the trace, suppressing any further transmission of datastreams to the log.
5. Run the PRINT LOG utility to show the actual trace.

Transmission times can be calculated by analyzing the length of the datastream.

CPU usage: To monitor CPU cycles and obtain CPU usage by task, the system can be instructed to collect task statistics. It is advisable not to request task statistics unless there is a demonstrated need as they require considerable overhead and generate a large volume of data. Task statistics are requested by specifying TASK STATISTICS WRITE or TASK STATISTICS COLLECT on the SYSTEM statement. The statistics are written to the DC/UCF system log.

►► For further information on collecting task statistics, refer to *CA-IDMS System Operations*.

7.3.2 Conserving Resources

Storage protection: Storage protection is enabled by specifying PROTECT in the SYSTEM statement at system generation. The benefits of using storage protection are that CPU overhead is reduced because there are shorter chains for the system to walk.

To avoid SVC overhead, it is advisable to enable storage protection (that is, specify PROTECT) on the SYSTEM statement and to disable storage protection (that is, specify NOPROTECT) on the PROGRAM statement.

Buffer sizes in multiples of 4084 bytes: The 4084-byte limit represents a multiple of 4K (4096 bytes) less the 12 bytes for pointer information and task id address, as illustrated below:

RCE address	Storage length	Actual storage	RCE address
4 bytes	4 bytes	4084 bytes	4 bytes

If a 4K page were selected, storage would have to be taken from two contiguous pages. The benefits of placing a **4084-byte limit** on the amount of storage acquired are as follows:

Benefits of storage limit

- Fragmentation of the storage pool is reduced when only one page is requested. Space is allocated in contiguous frames for a particular request. It is easier for the system to find one page rather than two contiguous pages.
- Less CPU overhead is required because partial pages do not have to be calculated or scanned.

Size of subschemas: Subschemas for navigational DML access should be specified to the requirements of the application. The size of the currency block is directly related to the storage requirements of the variable subschema storage block (VB50) used at run time; the runtime system maintains currency tables for every record, set, and area in a subschema, regardless of whether they are accessed by the dialog. Therefore, it is worthwhile to make subschemas as streamlined as possible.

Number of dialog levels: The MAXIMUM LINKS parameter of the ADSO sysgen statement specifies the maximum number of dialog levels that can be established by each respective CA-ADS application; keep this parameter low. A well designed application has as few levels as possible. The number of levels should be limited because, for each level established in the application, kept storage is acquired for the Variable Dialog Block (VDB) and the currency block. Storage established at a particular level is not released until control is passed upward.

To limit the number of levels established, use the TRANSFER command whenever possible; build the application horizontally (that is, pass control laterally) rather than vertically.

Size of the application: The size of dialog premap and response processes, the number of data fields included in a map, and the size of records affect the performance of the CA-ADS runtime system. The actual number of I/Os required to load a complete program is dependent upon the size of a page in the DDLDCLOD area, the amount of overflow that will be encountered to load that record, and the size of the actual program being loaded. Therefore, the following benefits are realized by minimizing the size of programs:

- A reduction in the work required to load a small program as compared to a large program
- A reduction in time spent loading a particular program in the program pool or reentrant pool
- A reduction in time spent waiting for space in the program pool or reentrant pool

Under the DC/UCF system, the term program includes dialogs, edit and code tables, maps, subschemas, and online and batch programs.

Making frequently called programs resident: A frequently called program (such as ADSOMAIN) is virtually a resident in the program pool or the reentrant pool. The program should be made resident because the operating system can page more rapidly than the DC/UCF system can read in a page from the DDLDCLOD area. By making the program resident, the operating system, rather than the DC/UCF system, will be requested to bring the page in core. Additionally, the program and resident pool will be less fragmented when a frequently used program is made resident. A program can be specified as resident on the PROGRAM statement at system generation.

Free the resources of an inactive terminal: The resource timeout facility can be activated on the SYSTEM statement at system generation, specifying the amount of time a terminal is permitted to be inactive (that is, have no task executing) before all resources owned by the terminal are deleted and control is returned to the system. Because longterm storage resources are associated with a terminal even though a program is not active, freeing those resources will free space for other users of the system. This is particularly important in navigational DML if longterm locks are being implemented.

Appendix A. Application Concepts

- A.1 Overview A-3
- A.2 Application Components A-4
 - A.2.1 Functions A-4
 - A.2.2 Responses A-6
- A.3 Dialog Features A-7
 - A.3.1 Dialog Components A-7
 - A.3.2 Dialog Procedures A-8
- A.4 Control Commands A-9
- A.5 The Flow of Control A-11

A.1 Overview

This chapter provides an overview of application terms and concepts within the CA-ADS environment. The following topics are discussed:

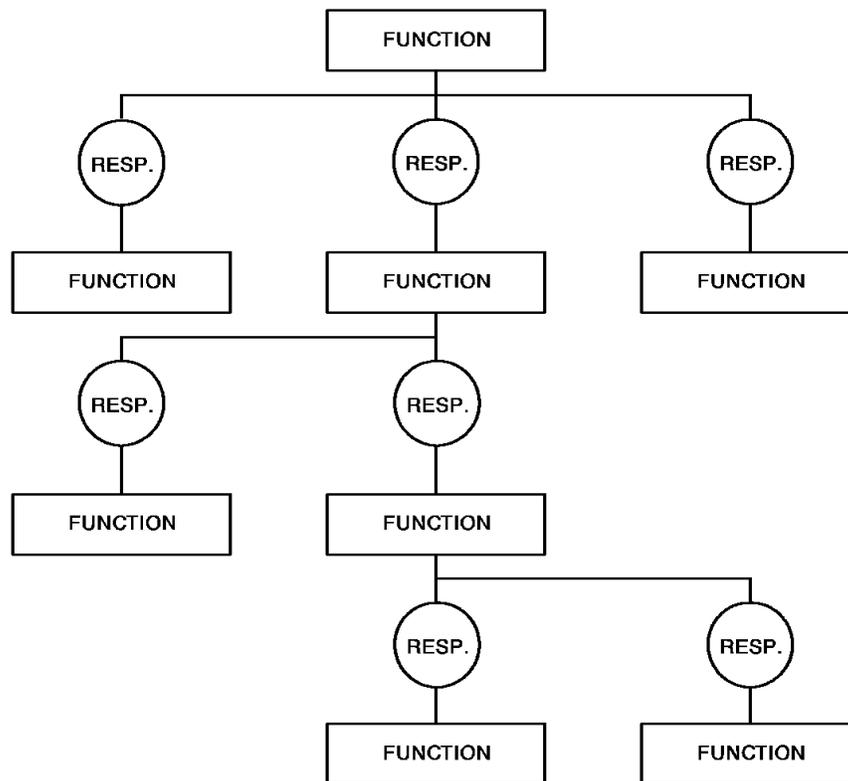
- **Application components** — The two basic parts of a CA-ADS application
- **Dialog features** — The components and procedures that make up a dialog
- **Control Commands** — The commands that can be used to pass control within an application
- **Flow of Control** — How the runtime system determines the way in which an application is executed

A.2 Application Components

An application is composed of **functions** and **responses**.

The following figure illustrates the relationship between functions and responses. Each of these components is described.

The structure of a CA-ADS application



An application is composed of functions and responses. Functions define the activities that can be performed in an application; responses associate the functions with one another and direct the flow of processing. A response can be associated with a control key and/or a code entered by the user.

A.2.1 Functions

Definition: A function is a named procedure or activity within an application.

Available types: Six types of functions are available under CA-ADS, as follows:

- **Menu Functions** are used to direct a user through an application. The menus contain a list of valid responses for the terminal user to use when processing in application. System-defined menus are built automatically by the runtime system.

- **Dialog Functions** are used for data processing, such as database access. A dialog function can have any number of valid responses defined for it during application generation. A response can activate another dialog function or can activate a dialog not defined as a function.
- **Menu/Dialog Functions** are dialogs that display a user-defined menu. When a menu is associated with a dialog, its map is displayed when the executing dialog issues a DISPLAY process command. Menu/dialogs must be used if the menu map is user-defined.
 - ▶▶ For further information on the options available when designing menus for an application, refer to Chapter 4, “Designing Maps” on page 4-1.
- **System Functions** are predefined. Available to all applications, they perform the same action in all applications to which they are assigned. The use of system functions adds flexibility to an application, eliminating the need to write code for a given activity.

Available system functions: The following system functions are available:

- **POP** returns processing control to the previous menu in the application thread.
- **POPTOP** returns processing control to the first menu in the application thread.
- **TOP** returns processing control to the highest function in the current application thread.
- **RETURN** returns processing control to the next higher function in the current application thread.
 - Note:** If a RETURN command is coded into a response process, it is considered a process command, *not* a system function. As a process command, RETURN performs as it would in an the DC/UCF system's environment.
- **HELP** displays a HELP screen at run time. This screen lists all valid responses for the current function.
- **QUIT** terminates processing of the current application. If previously signed on to the application, the user is automatically signed off.
- **SIGNON/SIGNOFF** allows a user to signon or signoff. This is a CA-IDMS/DC signon/signoff function executed from within the application.
- **FORWARD/BACKWARD** allows a user to page forward or backward on menu maps.

System functions can be subdivided as follows:

- QUIT, POPTOP, POP, TOP, and RETURN are generally executed when an EXECUTE NEXT FUNCTION command is encountered.
- SIGNON, SIGNOFF, and HELP are always executed as soon as they are encountered by the runtime system.
- FORWARD and BACKWARD (menu functions only) are executed as soon as they are encountered. If associated with a nonmenu dialog function, the FORWARD and BACKWARD functions are moved into the

ADSO-APPLICATION-GLOBAL-RECORD prior to executing the dialog's response process.

- **User Program Functions** are written in a process language other than CA-ADS. When a user program function is activated, the CA-ADS runtime system relinquishes control to the user program. CA-ADS does not define valid responses for a user program; any responses made by the user must be processed by the executing user program. The runtime system maintains all buffers for the application at the level at which control was relinquished, anticipating return of the processing control.
- **Internal Functions** are associated with the current dialog function. An internal function is assumed to be a response defined for the dialog response process.

The developer might define a response that initiates an internal function as a method of documenting the response process and/or as a method of providing the dialog response process as a valid response choice at run time. Additionally, a security class can be assigned to this type of response, thereby enabling security protection for a dialog's response process.

A.2.2 Responses

A response is a named entity that establishes a relationship between two functions. A response can be a control key or a response value entered in the response field by the user.

Note: It is important to distinguish between a *response* and a *response process*. A *response* is the action taken by the user when pressing a key or entering a response value. A response is defined by the CA-ADS Application Compiler; it can initiate an application function or the dialog's response process.

A *response process* is the dialog component that receives data from the terminal user, processes it accordingly, and passes control to the next activity. A response process is stored as a MODULE-067 record (with the attribute LANGUAGE IS PROCESS) and is associated with a dialog by using the CA-ADS Dialog Compiler in an ADSC session.

Processing control is directed by the valid responses of a function. When a valid response to the current function is selected by the user, a new function (or a reiteration of the current function) is executed.

A.3 Dialog Features

A dialog enables interaction between the user and the application and can be defined in terms of its components and how it accomplishes its job. The components and procedures of a dialog are discussed below.

A.3.1 Dialog Components

Each dialog consists of the following components:

- **Map** — Provides a means of communication between the application and the user. Map definitions in the dictionary maintain a formatted screen layout of literal and variable map fields (that is, data fields). Map data fields are associated with areas in program variable storage and are contained in map records. There can be only one map for each dialog. The application developer defines the map online with the online mapping facility; the resulting map load module is stored in the load area of the data dictionary.
- **Processes** — Perform data retrieval and processing. Processes are instructions written in CA-ADS process code. Each process consists of one or more commands that specify the type of processing to be performed (for example, database accessing, conditional testing, inter- and intra-dialog communication). A dialog is associated with two types of processes: premap and response. Both types are optional. A maximum of one premap process can be associated with a dialog; there is no limit to the number of response processes.

The application developer defines the processes by using the batch or online capabilities of IDD. The batch DDDL compiler stores the source statements as modules in the dictionary.

- **Subschema** — Provides the dialog with a view of the database. Each dialog can be associated with a maximum of one subschema. Subschemas are defined by the database administrator and stored in the dictionary by the subschema compiler. Subschemas are associated with dialogs when a dialog is compiled by ADSC.
- **Records** — Supply data to the dialog for processing. A dialog obtains data from a combination of records, as follows:
 - **Subschema records** are database and logical records that comprise the subschema.
 - **Map records** are subschema or work records.
 - **Dialog work records** are dictionary records used as working storage by a dialog.

These records contain the data elements that are needed by the application. Data elements and records are created with the use of IDD DDDL and are stored in the dictionary. They can have associated values, edit criteria, external and internal pictures, and code tables that are all recognized by the maps and dialogs of an application. For a more detailed discussion on creating the records used in an application, see Chapter 2, “Design Methodology” on page 2-1.

A.3.2 Dialog Procedures

When the CA-ADS runtime system executes a dialog, one or all of the following procedures can take place:

- **Premap processing** — Performs optional processing prior to displaying a map to the user. For example, the dialog can retrieve a record that contains the data to be displayed by the map. The dialog premap procedure is not automatic.
- **Mapout** — Displays a formatted screen (map) for use by the user. The user uses the map to supply data and to specify how this data is to be processed. For example, a dialog can display data from a customer record; the user then updates the record and requests that it be modified in the database. The mapout procedure is automatic when there is no premap processing; otherwise a mapout occurs when the DISPLAY command is issued.
- **Mapin** — Receives data and the requested response from the user. For example, if the user requests that the customer record be modified, the values that the user keys into the map data fields are then moved into variable storage. The dialog mapin procedure is performed automatically when the user presses a control key.
- **Response process selection** — Selects a response process based on the response entered by the user. The runtime system performs this procedure automatically.
- **Response processing** — Processes data as directed by the terminal user's response (for example, modifies the customer record) and specifies the next activity to be executed. Response processing is not performed automatically.

A.4 Control Commands

The application developer can use specific CA-ADS commands to perform the following operations:

- Pass control from one dialog to another dialog or to a user program
- Display a map
- Terminate an existing dialog or application
- Exit the CA-ADS environment
- Direct processing to specified places within a dialog
- Reinitialize the record buffers associated with a dialog
- Establish the status and level of a dialog within the application structure
- Implicitly govern the available data and database currencies maintained for a dialog

Most of the control commands are available to all applications. When designing dialogs that will become part of an application defined by using the CA-ADS Application Compiler, the developer can also use the EXECUTE NEXT FUNCTION command.

The CA-ADS control commands are as follows:

- **DISPLAY** — Requests display of the dialog's map or reexecution of the premap process
- **INVOKE** — Specifies the next lower-level dialog to be executed in the application thread
- **LEAVE** — Terminates the current application, optionally initiating another application, or terminating the CA-ADS session
- **LINK** — Specifies the next lower-level dialog to be executed in the application thread, implicitly establishing a nested application structure, or links to a user program that executes outside the CA-ADS environment
- **RETURN** — Terminates the currently executing dialog, returns control to a higher-level dialog, and, optionally, initializes that dialog's record buffers
- **TRANSFER** — Terminates the currently executing dialog and passes control to a dialog at the same level (which may be the same dialog)
- **EXECUTE NEXT FUNCTION** — Activates fields in the ADSO-APPLICATION-GLOBAL-RECORD that determine the next activity to be executed

►► For further information on the way in which the runtime system moves information to these fields, refer to "Global Records" in Chapter 5, "Designing Dialogs" on page 5-1.

Note: If an EXECUTE NEXT FUNCTION command is encountered in a dialog that has not been defined to an ADSA application, the command is processed as a DISPLAY command and a message is issued indicating that the user should select the next function.

The numerals in the flowchart, above, refer to the four sets of circumstances that determine when the next function will be executed, as discussed in the text.

When the user selects a valid response, the function associated with that response is established as the next function to be executed. This function is not executed until the runtime system satisfies certain criteria. The flowchart illustrates the circumstances that determine when the next function will be executed, as follows:

1. If the response is known to the dialog, the runtime system immediately executes the response process of the dialog. If an EXECUTE NEXT FUNCTION command is encountered and the response is valid for the application function, the function associated with the application response is executed next. If there is no EXECUTE NEXT FUNCTION command, the dialog passes control with an INVOKE, TRANSFER, RETURN, LINK, or DISPLAY control command.

If the response is not valid for the application function, the following error message is displayed when an EXECUTE NEXT FUNCTION command is encountered: PLEASE SELECT NEXT FUNCTION

2. If the response is valid for the function, the system checks to see if the response is associated with one of the following ADSA system functions:
 - HELP
 - SIGNON/SIGNOFF
 - FORWARD/BACKWARD (menus only)

If so, the system function is executed immediately.

If the response is not valid for the dialog, the CA-ADS runtime system determines if the response is known to the application. If not, the following error message is displayed:

UNACCEPTABLE RESPONSE. PLEASE TRY AGAIN.

3. If the response is valid for the application function, but not known to the dialog, and if the response is not an immediately executable ADSA system function, the runtime system checks to see if there is a response process associated with the ENTER key. If there is no such associated response process, the application function is executed immediately.
4. If the status of the response is the same as in situation #3 (that is, valid for the application, not known to the dialog, and not an immediately executable function) and a response process *is* associated with the ENTER key, the ENTER response process is executed first and the application function is executed when an EXECUTE NEXT FUNCTION is encountered. If there is no EXECUTE NEXT FUNCTION command, the dialog passes control with an INVOKE, LINK, TRANSFER, RETURN, or DISPLAY command, as in the first example.

Glossary

A

ADB *See* Application Definition Block.

ADSA The task code that activates the CA-ADS Application Compiler; *also*, the application compiler.

ADSC The task code that activates the CA-ADS Dialog Compiler; *also*, the dialog compiler.

ADSOCDRV The CA-ADS runtime program that initializes and updates the ADSO-APPLICATION-GLOBAL-RECORD; performs system functions (for example, TOP, POPTOP); processes responses entered on the HELP screen; and selects the value for the AGR-CURRENT-RESPONSE field of the system global record.

ADSORUN1 The CA-ADS runtime program that loads the Task Activity Table (TAT), creates an Online Terminal Block Extension (OTBX), if necessary, and loads the Application Definition Block (ADB) for the application being executed.

ADSORUN2 The CA-ADS runtime program that allocates application global records in the Record Buffer Block (RBB); builds menu records prior to mapping out application menus; and builds and maps out the runtime HELP screen.

ADSO-APPLICATION-GLOBAL-RECORD The CA-ADS system-defined global record that is used by the CA-ADS Application Compiler to pass information between functions and the runtime system; fields defined in the record are addressable and can be modified by dialogs and user programs.

ADSO-APPLICATION-MENU-RECORD The system menu record that is included in all menu maps; when the menu map is to be mapped out, the runtime system moves values into the fields of this record.

ADSO-STAT-DEF-REC The predefined status definition record that contains level-88 record element definitions associating condition names with the status codes most commonly tested after database, logical record, and queue and scratch record access; stored on the dictionary, this record can be modified or replaced to meet site-specific needs.

application A named set of functions or dialogs used to accomplish a specific business task (for example, general ledger, shop floor control, inventory control, payroll).

application components The application functions and responses defined during an ADSA session; *see also*, dialog components.

Application Definition Block (ADB) The application load module compiled by ADSA for use by the CA-ADS runtime system; the ADB contains the application information supplied on the definition screens during an ADSA session.

application function The basic structural component of an application; functions can be defined as dialogs, menus, menu/dialogs, user-defined programs, or system functions.

application levels The logical structure of an application; levels are achieved through the use of dialog control commands and are important for the purpose of maintaining currencies and record buffers.

application response *See* response.

application thread The path through the application, as decided by the response of the user at runtime.

automatic editing and error handling A mapping feature of the CA-ADS runtime system that compares input/output data with internal and external pictures, validates data against edit tables, and encodes/decodes data by using code tables.

AUTOSTATUS A runtime facility that handles errors compiled by database, logical record, or queue and scratch record processing; enabled for each dialog during an ADSC session.

B

bill-of-materials structure The database structure, with a variable number of levels, that represents network relationships among record occurrences of the same type; since the relationship is really many-to-many, it is implemented by two or more sets.

BIND The database command that signs on a run unit and notifies the database management system that the

user will be requesting runtime services; this function is automatically performed by the CA-ADS runtime system.

C

CA-ADS Application Compiler (ADSA) A facility of CA-ADS that provides the application development team with a flexible design and prototyping tool; ten definition screens prompt the designer for names of functions, responses, records, task codes, and security and menu specifications; ADSA updates the Task Activity Table (TAT) and compiles an application load module (Application Definition Block (ADB)) that is stored in the dictionary and used at runtime to direct the flow of control in an executing application.

CA-ADS Dialog Compiler (ADSC) A facility of CA-ADS that processes dialog, map, and process definitions, and stores this information in the dictionary; ADSC compiles a dialog load module (Fixed Dialog Block (FDB)) that is used by the runtime system.

CA-ADS A CA software product, running under the DC/UCF system, that enables users to develop and execute online applications for the query and update of the database with more ease than when traditional programming techniques are used.

CA-ADS runtime system A task that runs within the DC/UCF system environment; the runtime system can execute an application as compiled in an ADSA session or can execute a combination of dialogs as compiled in an ADSC session.

CA-CULPRIT A CA software product that is fully integrated with the dictionary and is designed to generate reports from CA-IDMS databases as well as from other databases and conventional files.

CA-IDMS/DB A CA software product that interprets application requests for database services and issues calls for access and update of the database.

CA-IDMS/DC A CA software product that controls the concurrent execution of online applications and provides support facilities for the use of sophisticated terminal devices.

CA-OLQ A CA software product that is fully integrated with the dictionary and provides conversational access to CA-IDMS databases for applications developers and end users.

checkpoint An entry in the journal file that defines a position after which run unit updates to the database can be reversed during recovery.

code table A table used to translate internal codes in a record to a screen display format.

COMMIT The database command that causes a checkpoint to be written to the journal file and releases record locks if they are being maintained; committed updates cannot be rolled back.

compile The process that produces output that is itself in machine-executable code or is suitable for processing in the form of a load module that can be executed at runtime; *also*, to store DDDL descriptions in the dictionary; *also*, the process that creates a load module that is stored in the DDLDCLOD area of the dictionary by ADSA (the ADB), ADSC (the FDB), online mapping facility (the map load module), and SSC (the subschema load module).

control block A logical collection of specific parameter data used by the operating system during runtime.

control commands The DISPLAY, TRANSFER, INVOKE, RETURN, LINK, and LEAVE CA-ADS process commands that instruct the runtime system to pass control from one dialog to another, or to a user program during the execution of an application.

control key A program function (PF) key, program attention (PA) key, ENTER key, or CLEAR key defined to activate an application response at runtime.

currency block The control block that maintains currency information on all subschema records used by the application; maintained by CA-IDMS, a currency block is created for each level in the application that accesses the database.

D

Data Description Language (DDL) The CA-IDMS language used to define the structural components of a database: schema, Device-Media Control Language (DMCL), and subschema.

Data Dictionary Definition Language (DDDL) *See* DDDL.

Dictionary Reporter A CA report compiler that provides standard reports on the contents of the dictionary.

DDDL A medium for describing and maintaining the contents of IDD (the Integrated Data Dictionary).

DDDL compiler An IDD-supplied program that stores DDDL descriptions in the dictionary.

DDL *See* Data Description Language.

DDR *See* Data Dictionary Reporter.

Device-Media Control Language *See* DMCL.

dialog A unit of work in the CA-ADS environment that performs one interaction with a user and all the processing associated with that interaction.

dialog components A dialog comprises not more than one premap process module, zero or more response process modules, and, optionally, one map and one subschema view of the database; components are associated with the dialog during an ADSC session.

dialog function An application function that is defined as a dialog during an ADSA session.

dialog response process *See* response.

dictionary A storage facility that is integrated with other CA products and is used by these products as a central source for information on data descriptions and relationships.

DMCL A database component that controls the mapping of the schema-defined database into physical files; designates which areas of the database are utilized at runtime; and, optionally, describes the files used to journal database activities.

E

ECBLIST *See* Event Control Block List.

edit table A list of single values or ranges of values that are valid for a data field.

Event Control Block List (ECBLIST) The control block used to synchronize events between the DC/UCF system and the host operating system; the list contains an ECB for each task waiting for an operating system event (for example, a disk read).

EXECUTE NEXT FUNCTION The process command that activates the flow of control in an ADSA-defined application at runtime.

extended run unit A feature of the CA-ADS runtime system that keeps the run unit open when a dialog issues a LINK to a lower-level dialog or to a user program under certain conditions.

external picture The format of data as displayed on the terminal screen; defined at record element level via IDD or during an online mapping session.

F

FAST mode An optional mode of execution in ADSC, ADSA, the online mapping facility, and in an application at runtime, in which control is passed directly to the next sequential screen when a transaction is successful; otherwise STEP mode is in effect and the current screen is always redisplayed before control is passed.

FDB *See* Fixed Dialog Block.

field mark The special character used to define the beginning of a map field.

FINISH The database command that releases all resources and completes the run unit; FINISH is performed automatically by the CA-ADS runtime system.

first functional call The first database command passed to CA-IDMS/DB at execution time.

Fixed Dialog Block (FDB) The dialog load module compiled in ADSC for use by the CA-ADS runtime system when an application is executed.

function *See* application function.

G

global record A record, defined in the dictionary prior to compiling an application, that is available to all functions of an ADSA-defined application; *also*, a record that remains in the record buffer for the duration of the application, unaffected by dialog control commands; *also*, a record defined on the Global Records screen during ADSA; *see also*, ADSO-APPLICATION-GLOBAL-RECORD.

global response *See* response.

I

IDD A CA software product used to control and report information that is stored in a central storage facility called a dictionary.

Integrated Data Dictionary *See* IDD.

internal picture The format of data as stored in variable storage or the database; defined in the dictionary through IDD.

internal response *See* response.

K

KEEP The database command that locks a record occurrence against access or update by another run unit.

L

load module A program unit that is suitable for loading into main storage for execution; CA-ADS uses the TAT, ADB, FDB, map, table, and subschema load modules stored in the DDLDCLOD area of the dictionary.

local response *See* response.

logical record One or more database records presented to the application program as a single record, permitting access to fields in multiple database records by a single request.

Logical Record Facility (LRF) The CA software product that simplifies application programming by allowing the DBA to predefine combined database records and the processing sequence necessary to access them.

logical terminal CA-IDMS/DC's view of the events associated with a particular physical terminal; the logical terminal is used by CA-IDMS/DC to communicate with the physical terminal; at runtime, the user's signon information (for example, password, security codes), the executing task, and dynamic storage are associated with the logical terminal; a logical terminal is defined on the LTERM statement at system generation.

Logical Terminal Element (LTE) The control block used by CA-IDMS/DC to manage and maintain the resources associated with a particular terminal; *also*, the control block that ties together the user's longterm resources across a pseudoconverse.

LRF *See* Logical Record Facility.

LTE *See* Logical Terminal Element.

M

mainline dialog A dialog that is designated as an entry point to an CA-ADS application.

map A formatted layout that names the literal and variable fields on a terminal screen, identifies the location of each field on a screen, names the record elements associated with each variable field; and allows transfer of data a full screen at a time.

map load module The load module generated by the DC/UCF system's mapping facility; used by the CA-ADS runtime system.

Map Request Block (MRB) The control block, contained in the Variable Dialog Block (VDB), that is used to perform mapping operations.

mapin The mapping operation in which values keyed by the user into variable map fields are moved into variable storage; in the CA-ADS environment, a task begins with each mapin operation.

mapout The mapping operation in which the map is displayed out to the terminal; literal fields are moved to their assigned positions and contents of the associated data areas in variable storage are moved to the map's data fields.

mapping The method used by CA-ADS to transfer data between the application and the user.

menu *See* menu map.

menu/dialog function A function defined as a menu in an ADSA session; this function is controlled by a user-written dialog that may provide additional processing.

menu function An application function that is defined as a menu during an ADSA session.

menu map A map that contains a list of valid responses for the user to use in the processing of an application; automatically built by the runtime system, the format of the map can be system- or user-defined.

MRB *See* Map Request Block.

O

online mapping facility The online facility for defining and compiling maps used by application programs executing in the CA-IDMS/DC environment.

Online Terminal Block (OTB) The control block used by the CA-ADS runtime system; associated with a logical terminal, this block exists across tasks in user kept storage, anchoring all other CA-ADS control blocks; the OTB contains the name of the current dialog and addresses of the current Variable Dialog Block (VDB) and the Fixed Dialog Block (FDB).

Online Terminal Block Extension (OTBX) An extension of the Online Terminal Block (OTB) that is created when the CA-ADS runtime system executes an application compiled by ADSA; contains pointers to the TAT, and the RBB and ADB for the currently executing application.

Online Work Area (OWA) The work area that exists for the life of an CA-ADS task; the OWA contains fields for communication between ADSORUN2 and ADSOCDRV, the subschema control block, a pointer to the current Map Request Block (MRB), and an internal stack.

operative status The status of a dialog that is still an active part of an application thread.

OTB *See* Online Terminal Block.

OTBX *See* Online Terminal Block Extension.

OWA *See* Online Work Area.

P

PA key *See* program attention key.

PF key *See* program function key.

physical terminal

A physical device such as a CRT (3270-type device), TTY, or printer that exists within a teleprocessing system; in the DC/UCF environment, physical terminal are associated with logical terminals; physical terminals are defined with the PTERM statement at system generation

premap process

An optional component of an CA-ADS dialog that performs any necessary processing before a mapout operation

process code

A modular set of commands used to perform one or more specific functions within a dialog; the set of commands is stored as a module (with LANGUAGE IS PROCESS) in the dictionary

program attention (PA) key

A predefined key that serves as an alternative to typing the corresponding response code; when a PA key is pressed (for example, [PA1]) no data is transmitted to the record buffer

program function (PF) key

A predefined key that serves as an alternative to typing the corresponding response code; when a PF key is pressed (for example, [Clear]) data is transmitted to the record buffer

pseud conversational programming

A programming technique that frees resources being used by a task while the system waits for completion of data entry by the terminal operator; this technique utilizes CA-IDMS/DC's ability to permit a task to terminate after issuing a terminal output request that requires an operator response; the CA-ADS runtime system is pseudconversational

pseudconverse

The interval between mapout and mapin

Q

queue

A disk work area shared by tasks on all CA-IDMS/DC terminals and by batch programs; queue records allow a task or application to pass data to another task or application, or to transfer data from one terminal to another

R

RBB

See Record Buffer Block

RCE

See Resource Control Element

READY

The statement that specifies to CA-IDMS the areas of the database that the application program will access and in which usage mode; the CA-ADS runtime system readies all database areas when the first functional navigational DML database command is encountered; no more than one READY should be coded in an CA-ADS process

Record Buffer Block (RBB)

The storage block dynamically allocated by the CA-ADS runtime system for subschema, database, work, and map records used by a dialog; an application can have one primary RBB and as many secondary RBBs as needed; the size of the RBB is specified by the PRIMARY POOL and SECONDARY POOL parameters of the ADSO system generation statement

Resource Control Element (RCE)

The control block that is created when a task requires the use of a resource and contains pointers to the task id and to the resource being used

Resource Link Element (RLE)

The control block that links all resources in use by a particular terminal

response

See specific responses below

application response

The action taken by the user when pressing a key or entering a response code when the runtime system is executing an application; *also*, the response that can initiate an application function or a dialog's response process; *also*, the global or local response associated with a function an ADSA session

dialog response process

A process module initiated by a uniquely assigned response code or control key, after a mapin operation; this module must contain a control command to pass control to another point either inside or outside of the application; there can be multiple response processes for a single dialog; *also*, the process module associated with a dialog during an CA-ADSC session

global response The type of response that is valid for all functions in a particular application; *also*, a response type that can be defined for an application during an ADSA session

internal response

In ADSA, a response, known only to a dialog, that is assumed to initiate the response process of that dialog

local response

A type of response that is valid only when specifically associated with a function; *also*, a response type that can be defined during an ADSA session

response code

The response field value that is associated with the dialog response process; *also*, the response field value supplied during an CA-ADSC session

response field

The 1- to 32-character map field in which terminal operators can choose to enter the response code that initiates the next activity to be executed by the runtime system; *also*, the mapping \$RESPONSE field or the AGR-MAP-RESPONSE field of the ADSO-APPLICATION-GLOBAL-RECORD

response process

See response, dialog response process

valid response

A global or local response that is defined as valid for a particular application function; there can be more than one valid response for a single function

RLE

See Resource Link Element

run unit

In CA-ADS, that portion of runtime processing that begins with the first functional navigational DML database call and ends when a control command (except for

certain cases of LINK) is encountered; *see also*, extended run unit

S

schema

The part of the database definition that describes the logical structure of the database, including the names and descriptions of all tables, elements, records, sets, and areas. One schema exists per database.

schema compiler

A CA-IDMS-supplied program that converts source schema statements into a description of the database and stores this description in the dictionary.

STEP mode

An optional mode of execution in ADSA, CA-ADSC, the online mapping facility, and in an application at runtime, in which the current screen is redisplayed with error messages (if any) or verification messages (if the transaction is successful) before control is passed to the next sequential screen

subschema

A program view of the database used at run time and consisting of all or a subset of the data elements, record types, set types, and areas defined in the schema

subschema compiler

A CA-IDMS-supplied program that converts source subschema DDL into subschema descriptions, which are stored in the dictionary and in the dictionary load area or in a load (core-image) library for use at run time

system function

A predefined function available to all applications compiled during an ADSA session (that is, POP, POPTOP, TOP, RETURN, HELP, QUIT, SIGNON/SIGNOFF, FORWARD/BACKWARD); a system function is associated with an application when it has been associated with a valid r

T

table The IDD entity type describing the edit and code tables that enable automatic editing, encoding, and decoding of map fields used by the DC/UCF mapping facility

task

The basic unit of work under DC/UCF that consists of the execution of a main program and one or more additional programs; a task is identified to the system by a unique name (such as ADS); an IDD entity type, the task name in the dictionary is usually identical to the task code used by the teleprocessing system

Task Application Table (TAT)

The table that contains names of task codes used to initiate applications and the names of the applications (ADB) thus initiated; the TAT is maintained in the dictionary by the application compiler (ADSA) and is loaded by the runtime system

task code

The unique name, of 1 to 8 characters, that identifies a task to the runtime system; the terminal operator types the task code in response to the DC/UCF ENTER NEXT TASK CODE prompt

Task Control Element (TCE)

The control block that ties together all the resources of an application

TAT

See Task Application Table

TCE

See Task Control Element

U

UCF

The CA software product that can be integrated with CA-IDMS to offer teleprocessing-monitor independence to communication users; this facility enables CA-IDMS-based applications to run without modification under a variety of teleprocessing monitors

Universal Communication Facility (UCF)

See UCF.

usage mode

The manner in which a run unit will access a given database area; the usage mode dictates whether a run unit will perform retrieval or update functions against records in the area and specifies the allowed extent of concurrent usage of these records by other run units

V

valid response

See response

Variable Dialog Block (VDB)

A non-reentrant table used by the CA-ADS runtime

system to obtain user-specified information about a particular dialog; dynamically created for each user dialog when the dialog is initiated, the VDB resides in the storage pool and contains header information, the Map Request Block (MRB) for the dialog (if any), and a Variable Record Element (VRE) for each record used in the dialog

Variable Record Element (VRE)

A control block, one for each record needed by the dialog, that contains variable runtime information on each record

VDB

See Variable Dialog Block

VRE

See Variable Record Element

Index

A

ADSA
 CA-ADS Application Compiler (ADSA) 1-3
 task code 1-3

ADSC
 CA-ADS Dialog Compiler (ADSC) 1-3
 task code 1-3

ADSO sysgen statement 7-4

ADSO-APPLICATION-GLOBAL-RECORD 5-13, A-9
 AGR-CURRENT-RESPONSE field 5-11
 AGR-DEFAULT-RESPONSE field 5-11
 AGR-EXIT-DIALOG field 5-12
 AGR-MAP-RESPONSE field 5-13
 AGR-MODE field 5-13
 AGR-NEXT-FUNCTION field 5-11
 AGR-PRINT-CLASS field 5-13
 AGR-PRINT-DESTINATION field 5-13
 AGR-SIGNON-REQMTS field 5-13
 AGR-SIGNON-SWITCH field 5-13
 AGR-USER-ID field 5-13

ADSORPTS utility 1-9

AGR-CURRENT-RESPONSE field
 See ADSO-APPLICATION-GLOBAL-RECORD

AGR-DEFAULT-RESPONSE field
 See ADSO-APPLICATION-GLOBAL-RECORD

AGR-EXIT-DIALOG field
 See ADSO-APPLICATION-GLOBAL-RECORD

AGR-MAP-RESPONSE field
 See ADSO-APPLICATION-GLOBAL-RECORD

AGR-MODE field
 See ADSO-APPLICATION-GLOBAL-RECORD

AGR-NEXT-FUNCTION field
 See ADSO-APPLICATION-GLOBAL-RECORD

AGR-PRINT-CLASS field
 See ADSO-APPLICATION-GLOBAL-RECORD

AGR-PRINT-DESTINATION field
 See ADSO-APPLICATION-GLOBAL-RECORD

AGR-SIGNON-REQMTS field
 See ADSO-APPLICATION-GLOBAL-RECORD

AGR-SIGNON-SWITCH field
 See ADSO-APPLICATION-GLOBAL-RECORD

AGR-USER ID field
 See ADSO-APPLICATION-GLOBAL-RECORD

application
 compiling 3-4
 design tools 1-4
 guidelines 1-3

application (*continued*)
 size 7-15

Application Definition Block
 See load modules, ADB

application design
 methodology 2-3
 phases 2-3

application thread 5-4

automatic editing, online mapping facility 1-8

C

CA-ADS Application Compiler (ADSA) 3-4, 3-7
 See also CA-ADS components

CA-ADS components
 CA-ADS Application Compiler (ADSA) 1-5
 CA-ADS Dialog Compiler (ADSC) 1-5
 interrelationship 1-6
 runtime system 1-6

CA-ADS Dialog Compiler (ADSC) 3-8
 See also CA-ADS

CA-IDMS Online Mapping Facility
 automatic editing 1-8, 2-11
 error handling 1-8

CA-IDMS/DC Mapping Facility
 checklist 2-9

control commands 2-7, A-8
 DISPLAY 5-17, A-9
 EXECUTE NEXT FUNCTION A-9
 INVOKE 5-17, A-9
 LEAVE 5-17, A-9
 LINK 5-17, A-9
 RETURN 5-17, A-5, A-9
 TRANSFER 5-17, A-9

currency
 See database currencies

currency control block

D

data administrator 1-11

data communications administrator (DCA) 1-11

data definition 2-3, 2-29

Data Dictionary Reporter (DDR)
 See reports

data redundancy 2-28

data resources 2-28

database administrator (DBA) 1-11
database currencies 5-17
DDR
 See Data Dictionary Reporter (DDR)
debugging aids 2-23
dialog
 See also response process
 compiling 3-5
 design 2-26
 levels 2-7, 5-3, 7-14
 premap process 3-7, 3-8
 process code 2-14
 response A-6
 response process 3-7, 3-8
 specifications 2-15
 status 5-4
dialog components
 map A-7
 processes A-7
 work records A-7
dialog function A-4
dialog procedures
 See also response processing
 mapin A-8
 mapout A-8
 premap processing A-7
 response process selection A-8
 response processing A-8
dialog report
 See ADSORPTS utility
dialog work records
 See records, work
dialogs
 definition 5-3
dictionary
 See Integrated Data Dictionary (IDD)

E

end users 1-11
entry point 3-4
error handling, online mapping facility 1-8
EXECUTE NEXT FUNCTION A-9
 See also control commands

F

FAST MODE THRESHOLD 7-5
Fixed Dialog Block
 See load modules, FDB

functions
 definition A-4
 dialog functions 3-4
 menu functions 3-4
 types A-4

G

global records 5-10
 See also ADSO-APPLICATION-GLOBAL-RECORD
 prototype 3-7
glossary 6-7

I

IDD
 See Integrated Data Dictionary (IDD)
Integrated Data Dictionary (IDD) 1-7
internal function A-6

L

load modules
 ADB 3-5
 FDB 3-5
 map 3-5
 TAT 3-5
load modules (figure) 1-7
logical records 5-20
longterm locks
 KEEP LONGTERM command 5-19

M

mainline dialog 5-3
map
 data fields A-7
 definition A-7
 load module A-7
map templates 4-9
mapin
 See dialog procedures
mapout
 See dialog procedures
mapping utility 1-9
maps 2-11
 compiling 3-5
 design 4-3
 design standards 4-4
 format 2-11
menu function A-4

menu maps
 reformatting 4-6
 system-defined 4-6
 user-defined 4-6
menu/dialog
 design 4-7
 function A-5
 generation 4-7
menus
 system-defined 2-10
 user-defined 2-10

N

naming conventions 2-10
 See also glossary
 application components 6-4
 database entities 6-7

O

online mapping facility
 screens for a prototype application 3-5

P

performance
 monitoring tools 7-9
populating the dictionary 2-26, 3-7
premap process A-7
premap processing
 See dialog procedures
process modules 3-8
PROGRAM sysgen statement 7-4
programmers 1-11
programming aids 2-18
project leader 1-11
prototype 2-11
 features 2-12
 first stage 3-4
 second stage 3-7
 third stage 3-9
 uses 2-12, 2-26

Q

QUEUE sysgen statement 5-9
queue/scratch working storage areas 5-9

R

RBB

See Record Buffer Block (RBB)

record buffer 5-7
 allocation 5-7
 management 5-7
 size 7-14

Record Buffer Block (RBB) 7-5

records 2-10, 5-7
 map 3-8, A-7
 queue 5-9
 scratch 5-10
 subschema A-7
 work 3-8, A-7
WORK RECORD 5-7

reports

 ADSORPTS utility 1-9
 CA-CULPRIT 1-9
 CA-IDMS Reports 1-8
 Data Dictionary Reporter (DDR) 2-26
 IDMSRPTS utility 1-9
 mapping utility 1-9
 OnLine Query (CA-OLQ) 1-9
 subschema compiler 1-9

request units 7-7

 external 7-7
 MAXIMUM ERUS parameter 7-7
 MAXIMUM TASKS parameter 7-7

resources

 CPU usage 7-13
 database 7-12
 disk and terminal I/Os 7-13
 internal processing 7-9
 longterm storage 7-15
 management 7-8
 program pool storage 7-12
 storage pool 7-11, 7-14

response

See dialog, response
 response process selection
 See dialog procedures

responses 3-4

 flow of control A-10

run units 5-17

 extended 5-17, 5-18

runtime system

See CA-ADS components

S

security 2-11, 2-26
signon menu function 3-7
subschemas
 size 7-14
system functions 2-7, 3-7, 5-3, A-5
systems analyst 1-11

T

tables
 code 3-8
 edit 3-8
Task Activity Table
 See load modules, TAT
task code
 ADSA 3-4
 ADSC 3-5
 application 3-4
TASK sysgen statement 7-4
testing guidelines 2-24
testing procedures
 acceptance testing 2-25
 integration testing 2-25
 regression testing 2-25
 unit testing 2-24

U

user program function A-6

V

valid responses A-10

