CA Gen

Block Mode Design Guide

Release 8.5

CA Technologies Product References

This document references the following CA Technologies products:

- CA Gen

Contact CA Technologies

Contact CA Support

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

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

Providing Feedback About Product Documentation

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

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

Contents

Chapter 1: Introduction

During system development using CA Gen, designers use the information discovered during analysis as the basis for describing an information system that can satisfy the needs of the business.

The system description can be used to build and test a prototype iteratively until the users are satisfied with the design.

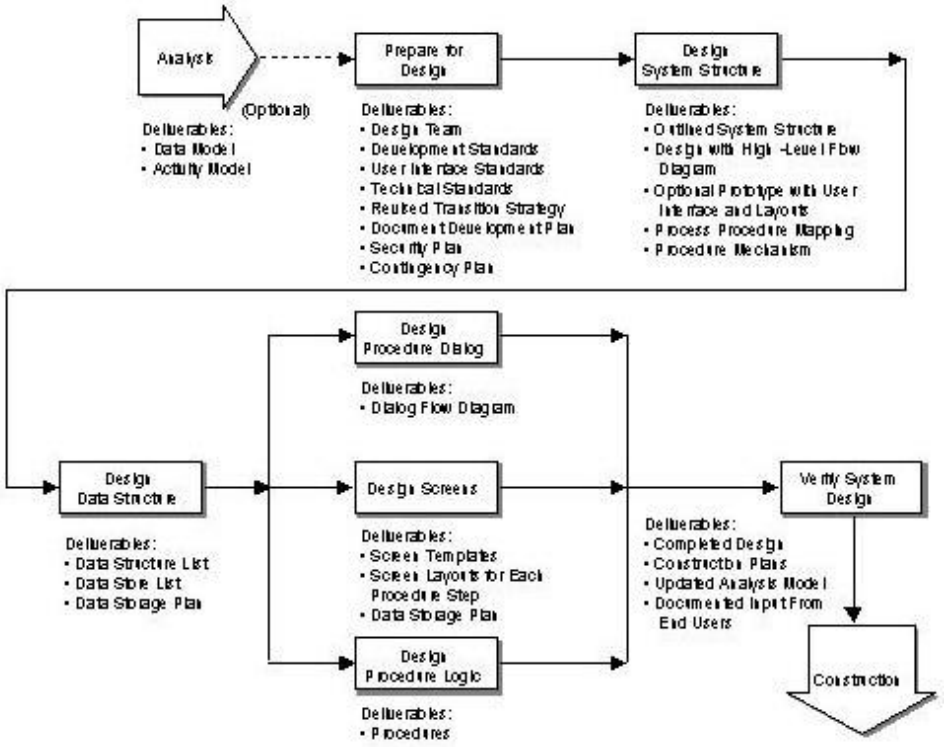
At this point, you can construct and test a system based on this design.

Note: This guide covers block mode design. For information about the techniques and best practices for client/server, see the *Client Server Design Guide*.

Design Process

The block mode design process begins with the results of analysis. The process concludes when you have a system design ready for construction.

The following illustration shows the sequence of activities that make up the block mode design process.



Note: Analysis is optional for block mode design.

You can design the data structure any time after system structure design.

Objectives of Design

Your goals for design activities are designing and confirming the external aspects of the system with end user representatives.

You, the designer, must ensure that the external design of the system is suitable for the proposed business users and locations and broadly suitable in technology for the expected volume and speed of response. You will probably want to defer consideration of detailed technical issues until the users agree on the functionality of the design, its form, and its suitability to meet the business objectives for developing the system.

For example, an analyst may have discovered that the business must keep track of customers. You may have defined an elementary process, called Add Customer, in which the rules for adding a new customer are specified to support this requirement.

To implement the process during design, you must address issues such as shown next:

- Should customers be added online, in batch, or both?
- If online, should a user be allowed to add multiple customers in a single execution?
- How should the user interface (screen or window) look, and how should the user interact with it?
- Should the process Add Customer be combined with other elementary processes, such as Change Customer and Delete Customer, into a single procedure?

There may be other objectives of design or constraints on the form of the system. For example, management may want the new system to exploit and protect investment in other systems wherever possible. This may involve designing procedures to make effective use of current systems and data. You may hear the term *legacy* applied to current applications and data storage structures that will survive into the new system. Management may also want you to design the new system to work with some other system that is currently planned or under parallel development.

CA Gen Tools for Design

Throughout design, you can add detail to the business system model using the tools listed in the following table.

CA Gen Tool	How to Use It
Dialog Design	Defining procedures and designing dialog.
Action Diagram	Designing procedure logic.
Data Structure List and Data Store List	Displaying the physical structure of the database and modifying that physical structure.
Procedure Synthesis	Automating the construction of procedure action diagrams.
Structure Chart	Showing the relationship between implemented action blocks and procedure action diagrams.
Screen Design	Designing screen layouts.

Note: This guide deals primarily with design techniques, not with a detailed description of the use of CA Gen tools.

CA Gen Construction Toolsets can generate code for a wide variety of target operating systems, database management systems, presentation management environments, and teleprocessing monitors. In most cases, you can use the results of design directly for construction.

In other cases, Design Toolset reports can be collected for a system specification to be used in implementing procedures manually.

Chapter 2: Preparing for Design

This section contains the following topics:

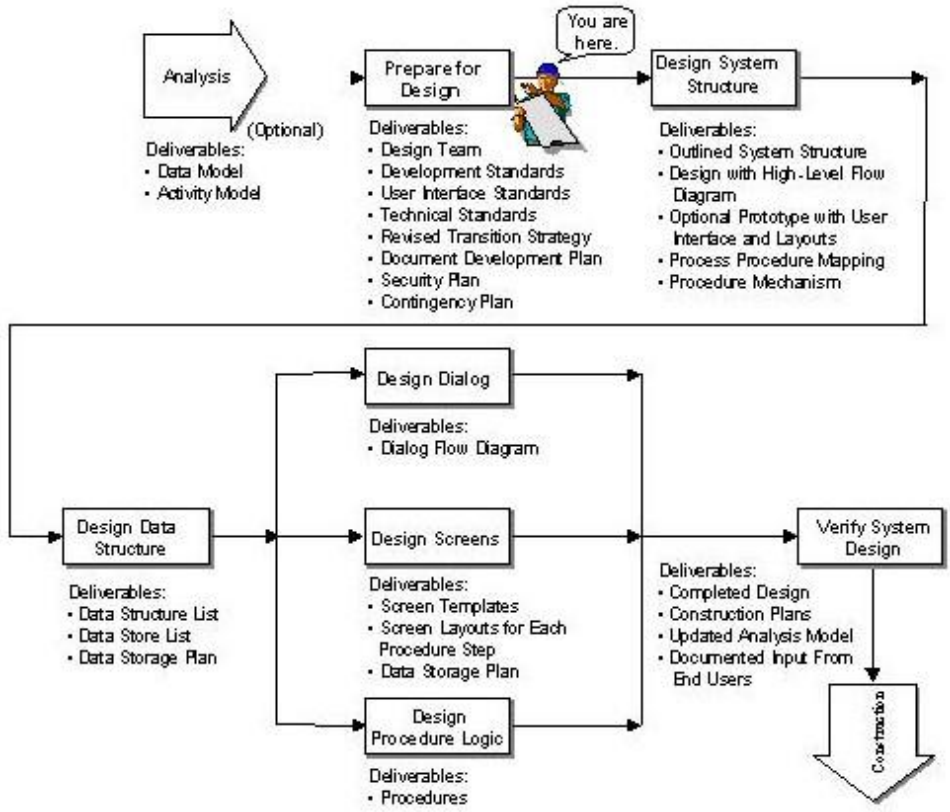
Prerequisites

Completing analysis before beginning design is optional.

Note: For more information about performing analysis, see the *Analysis Guide*.

Design Process

The following illustration lists the deliverables from preparing for design and shows where you are in the overall design process.



Design Team Selection

The membership for the design organization should come from the following functional organizations:

- The Information Management function—Systems designers should be experienced with system development in general and design techniques in particular.
- The user community—User representatives should be able to determine end user needs and help designers translate those needs into the design. Some or all of the user representatives who participated in analysis should also participate in design.

In organizations with a formal quality program in place, the design team also needs access to a quality assurance coordinator from within the organization. This will ensure that the resulting system meets the organization's quality standards.

Understand Analysis Results

Design team members who participated in analysis will be well acquainted with the characteristics of the business to be supported by the system.

Designers new to the development project need time to review and familiarize themselves with the results of analysis. They need to develop a clear picture of both the underlying Information Architecture and the users' work environment. They should therefore take time to understand the data that the business deals with, the activities the business performs, and the interaction between the two.

For all designers, a detailed understanding of these components is essential.

Note: For more information about the tasks, tools, and techniques used during analysis, see the *Analysis Guide*.

Review the Business System Definition

The Business System Definition identifies which business systems will implement which elementary processes.

Note: For information about establishing business system boundaries based on the results of analysis, see the *Analysis Guide*.

You need to review the project scope and implementation plan to determine the following items:

- What business systems will be created to support the business requirements.
- Which elementary processes will be implemented by the business system being developed by the project.
- In what sequence those elementary processes should be implemented.

If this information is not available, you should complete the missing details before beginning any work in design. Use CA Gen's Analysis Toolset to review analysis results. If you have not already done so, define the business systems in CA Gen.

Specify Business Systems to CA Gen

Review the clustered Entity Type/Elementary Process Matrix. After you have addressed all anomalies and firmly defined all clusters, you can define the business systems in CA Gen.

Perform the following tasks to define business systems in CA Gen:

- Give each business system a name.
- If names have not yet been specified for the business systems discovered during cluster analysis, they must be invented now.
- There are no firm rules governing the formulation of business system names. Simply choose words that express the function of the system.
- Select the elementary processes that each named business system will implement.
- Use the clustered matrix to identify the elementary processes belonging to each business system, bypassing any that were not selected for implementation.

Set Development Standards

One aim of design is to create a consistent user interface for the system being constructed. This is especially important in an online environment where ambiguity can lead users to make serious mistakes.

Some standards need to be common to many systems. These may need to be set at a corporate level. Some standards may be set for a specific development project.

Standards may apply to the following functional design areas:

- Procedure style
- User interface
- Reusable logic
- Quality

Choose a Procedure Style

Automated procedures should employ the styles most appropriate for each type of system user. In design, this involves deciding which style should be used for each procedure in a system.

The choice of procedure style is usually limited by organizational standards, which are influenced by:

- Strategic direction—There may be a policy to develop systems to a preferred style. For example, if the organization has standardized use of a database management system supporting distributed data, “remote data access” might be preferred.

- Target production environment—The best style option could involve the use of powerful client machines. If these are not in place, that style option may not be possible.
- Cost—The cost of implementing different styles will vary. Some may be prohibitively expensive.

The frequency of procedures performed and their physical location will also influence the choice of styles for the system. It might be that a number of activities are performed in one location only, and so need not be distributed.

The system might have to reuse a significant amount of code from existing systems or off-the-shelf packages. This may influence the choice of style.

More information:

Designing the System Structure (see page 31)

Set User Interface Standards

User interface standards ensure that users are presented with consistent displays, error messages, and commands in all the systems they use. This requires setting standards that are uniform across the entire organization, while also conforming to industry guidelines for user interfaces, for example, Common User Access (CUA).

Set Reusable Logic Standards

It is essential to document all new procedure logic and reused existing code and follow standards that will allow for its reuse, both within this system and for future systems.

Set Quality Standards

Adhering to a number of quality standards and guidelines during design ensures that the system meets its intended purpose and achieves a certain "quality threshold." These will generally be corporate standards.

System Defaults Standards

Before design work begins, specify standards for system defaults through the System Defaults panels in the Design Toolset. You can establish other standards using individual tools in the Toolsets.

Tool standards relate to the following system defaults:

- Commands
- Function keys
- Screen video properties
- Screen formats
- Clear screen input delimiters
- Field edit properties
- Field prompts
- Common exit state definitions
- Dialect

Commands Standards

A command provides a way for a user to direct the execution of a procedure.

Commands should be consistent across the system so that users are comfortable in their choice of commands, regardless of the procedure being executed. The lack of standards can cause confusion.

For example, you might choose the command D to mean DISPLAY while another designer might choose D to mean DELETE. Such ambiguity can have serious consequences.

More information:

Designing the Procedure Dialog (see page 71)

Function Key Assignments Standards

Function keys (also called PF keys) are available on many types of video display terminals.

You can use function keys to provide a shorthand way for users to communicate commands to procedures in a variety of online and graphical environments. For example, you could specify that pressing Function Key 10 will have the same effect as entering the command ADD.

If a function key represents a particular command, it should represent that same command in all procedures that use it. For example, if Function Key 1 invokes the command CHANGE in one procedure and Function Key 10 invokes it in another, this will result in user confusion and error.

You can define function keys as standard or default:

- Standard-You cannot override a standard function key. For example, no matter what procedure is being executed, Function Key 1 means Help.
- Default-You can override a default function key. For example, procedures that support a Display command will all use Function Key 4 for display, but a procedure with no Display capability can use Function Key 4 to represent some other command.

CA Gen supports the maintenance of both standard and default function keys across the system.

More information:

Designing the Procedure Dialog (see page 71)

Screen Video Properties Standards

Standards for the use of highlighting and colors are important. All data entry fields should be one color or intensity, prompts and literals another, and fields in error yet another.

More information:

Designing Screens (see page 101)

Window and Screen Format Standards

The general format of screens should remain consistent throughout a system. For example, error messages should appear at the same place on each screen, and users should enter commands at the same place on each screen. CA Gen accomplishes this standardization by establishing screen design templates.

More information:

Designing Screens (see page 101)

Clear Screen Input Delimiters Standards

Permissible parameter and string delimiters for clear screen input to a procedure are defined if any procedure in the business system will use Clear Screen Input.

More information:

Designing Screens (see page 101)

Field Edit Pattern Standards

Edit patterns dictate the output format of a field (for example the preferred format for a date field) on a display or report.

Use consistent edit patterns for fields containing the same information. In a system with part numbers, for example, one designer might cause a part number to be displayed as "123456789," while another might format it as 12-34567/89. Such inconsistency could lead to confusion.

CA Gen supports the maintenance of standard edit patterns to help enforce consistency throughout a system.

More information:

Designing Screens (see page 101)

Field Prompt Standards

Where possible, standardize field prompts. Use one common prompt for fields implementing the same attribute.

For example, if the attribute Customer Number appears on two displays, a user will have a better chance of recognizing it as the same value on both displays if a consistent label is used (such as CUSTOMER NUMBER) rather than two different ones (such as CLIENT NUM: on one and CUST NUM: on the other).

CA Gen supports the standardization of prompts by reminding the designer of all previous prompts used to describe a particular attribute. This help becomes available whenever you place a field using the Screen Design tool.

More information:

Designing Screens (see page 101)

Common Exit State Definition Standards

During design, an exit state can be associated with a message that appears on the display.

It is wise to establish standard exit state definitions for outcomes or results common to many procedures, most notably successful outcomes.

For example, if processing was successful, a procedure might set the exit state to REQUESTED OPERATION COMPLETE. However, if a problem is detected, such as an invalid value entered for the command special attribute, the outcome might be INVALID COMMAND.

Exit state definitions are shared across all the business systems and business areas defined in a single model.

More information:

Designing Screens (see page 101)

Specify Multiple Dialects

The generating business system will use the same language as that specified for the CA Gen model. You can specify an additional dialect.

Before the system is generated, anything that affects the language used in the user interface must be translated into the additional dialect. This includes literals on layouts, commands, and messages.

The transition plan may call for the system to be generated in different locations for different dialects. In this case, several translations must be performed on copies of their completed design.

If you need to use several dialects within a system, contact CA Technical Support for information on the options available.

Confirm the Transition Strategy

The overall concern of transition is with changing the business and its systems together to support a new set of business requirements.

Although transition is still some way off at this point, you should consider the transition strategy now. If a strategy has already been produced during analysis, it should be reviewed and revised.

The following transition issues are critical:

- Responsibility-Who will be responsible for transition? What information will they need?
- Location of data-In a distributed environment, where will data be stored? What processing is needed to maintain it and provide it to applications? You need to know the location names to be used whenever a procedure must select a location.
- Loading new databases-This is how the newly implemented entities will be initially populated, either manually or through a conversion program, all at once or portions at a time. If you decide to use a conversion program, someone will need to design and construct it.
- Replacing old systems with new-This is how the newly implemented system will be phased in while the old ones phase out.
  - The new system may replace the old system overnight or the old and new systems may run in parallel.
  - The new system may be implemented all at once or a portion at a time.

Duplicate maintenance may be required to synchronize the old and new databases, and possibly to integrate them, which will also require maintenance. Such duplicate maintenance could be handled dynamically, or synchronization transactions may be batched.

The decisions are largely dependent on the organization, but the intent here is to fire the imagination to consider transition issues.

These considerations may yield a requirement for some special procedures to be designed for transition, or may point out some special transition logic to be addressed during procedure logic design.

Identifying and addressing transition issues at this point helps you avoid later unpleasant surprises for the user.

Prepare a Documentation Plan

You need to establish a documentation plan defining the requirements for:

- Online help
- Training
- Reference and technical guides

Define Online Help Requirements

In an online environment, documentation can be provided directly to users through their workstations or terminals.

An online help system is useful to infrequent users who need constant assistance in performing normal operations, as well as to frequent users who attempt unusual or complex operations.

You should consider including online help before starting system structure design. Providing online help will influence menu design. You will also want to word your definitions so that you can use them in help documentation. This will avoid duplication of effort.

In organizations with an online help system already in place, you should investigate whether the business system can be designed to use this help system.

If a help facility is unavailable, you should consider providing procedures that display help information as part of the design.

Consider the needs of non-English speakers when planning the help system.

Define Training Requirements

Planning for training is an extremely important aspect of design. Good training helps ensure that the users are introduced to a system in the best possible way.

Training for users can be conducted by many different techniques, from formal classroom courses, and self-study tutorials, to informal *learning on the job* scripts.

The delivery medium for training can range from online interactive multimedia tutorials to hand-written demonstration notes.

There are usually several types of users, such as regular user, supervisor, system administrator, and so forth. You need to determine the most appropriate training technique and medium for each type of user.

Define Reference and Technical Guide Requirements

Reference and technical guides should accurately describe the system. If the guides are accurate and complete, they can be used both by users and developers wishing to reuse system components.

Capture Data Requirements Identified During Design

CA Gen automatically transforms the conceptual data model, expressed as a Data Model Diagram during analysis, into a database definition during design.

However, when addressing implementation issues, you may need to use the conceptual model in different ways than were expected during analysis. In some cases, you may need to invent additional types of data needed in a particular implementation.

CA Gen provides the following techniques for capturing data requirements identified during design:

- Special attributes
- Local data views
- Work attribute sets
- Design entity types

Special Attributes

CA Gen supplies variables called special attributes that can be used to communicate with the execution environment.

Special attributes include:

- Current Date
- Error Message
- Transaction Code

More information:

Designing the Procedure Dialog (see page 71)
Designing the Procedure Logic (see page 123)

Local Data Views

The sole purpose of a local data view is to provide a temporary store for attribute values that are to be referenced in action statements.

Unlike import and export data views, a local data view cannot:

- Receive/present data to or from displays.
- Receive/present data to or from called procedure action blocks, except in the context of a USE statement.
- Communicate with the underlying data model, unlike entity action views.

More information:

Designing the Procedure Logic (see page 123)

Work Attribute Sets

When you need to satisfy an implementation-specific data requirement by creating a view of an item not known in the data model, you need to define a new data item. In CA Gen such data items are included in work attribute sets. You use work attribute sets to keep track of execution time information such as totals and counts.

Work attribute sets are composed of attribute definitions but are not referred to as entity types because they do not reflect the permanently stored reality of the business.

Work attribute sets work is similar to entity types in many ways. For example, they can:

- Appear in import, export, and local views
- Be placed on layouts
- Be passed by a USE action in an action diagram

Entity actions (CREATE, READ, UPDATE, and DELETE) are not available for work attributes. This prevents them from finding their way into the database.

Design Entity Types

In some cases, you may need to store implementation-specific data discovered during design. When this is so, and the requirement is clearly not related to the business itself, you may define a design entity type.

CA Gen accepts both business entity types and design data. You can add design data definitions as entity types and attributes to the data model defined previously in analysis.

Design data might be needed to maintain the following information:

- Security information for the business system, such as user IDs and passwords
- Quality and productivity statistics, such as error rates of high volume procedures and keystrokes per hour, for users of the business system
- Information specific to a particular implementation technique, such as job accounting information and printer IDs by user

Prepare a Security Plan

You need to prepare a security plan to ensure:

- Security of design work
- Security within applications
- Privacy

Ensure Design Work Security

Security of the design work depends on the type of system being built.

It might be appropriate to divide the system into interdependent modules and then split the development among different independent development teams.

Regardless of the type of security, make sure the working procedures for security are in place before development work starts.

Ensure Security Within Applications

Security within a completed application involves allowing users access to the appropriate data and processes.

Access to data can be by:

- Type of data
- Data values
- Associations with other data

- Organizational unit
- Location

Access to processes can be:

- Available to some users
- Available in a limited way to other users
- Not available at all to some users

The organization's standard security package should control access. You may want to augment the standard package with other security procedures.

Ensure Privacy Within Systems

With the increase of legislation about data protection, it is essential to incorporate privacy standards into the system.

You should also be aware that if the system will operate internationally, data protection legislation may vary between different countries. This can affect international flows of data.

Prepare a Contingency Plan

Integrity controls can never be completely effective. Even if it were possible to identify every potential risk, the cost of securing against all of them would probably be prohibitive. For this reason, you must establish a contingency plan.

A contingency plan is a set of provisions for events that interrupt or destroy information processing capability.

Contingency handling should be designed into the system. It is too late to start contingency planning when things begin to go wrong.

A contingency plan should contain the following procedures:

- Fallback-Lets business activities continue while the normal computer system is unavailable.
- Back-up-Devised to take regular copies of data and transactions to be used in case of loss or corruption of the database.
- Recovery-Can be performed to enable a return to using routine computing procedures after a failure. This can involve:
  - Roll back, the reversal of changes made by the current transaction.

- Restore, the resetting of stored data to the state immediately following the most recent back-up.
- Roll forward, the rerun of the current transaction after a roll back.

Consider Coordination and Integration Issues

Design is rarely undertaken in isolation. There are likely to be needs for coordination and integration between associated design and analysis work.

Your team needs to be aware of other development projects and plan appropriate procedures to ensure coordination.

Chapter 3: Designing the System Structure

System structure design is driven by user requirements and the business results of analysis. The business processes defined in analysis can be mapped to procedures and then organized so that users can perform those procedures in the most effective and natural way.

As design work proceeds, the basic system structure that supports business processes continues to evolve. Additional procedures needed to provide reports, business and operational controls, conversion of current data, inter-working with current and packaged systems, bridging between new procedures and current data stores augment the structure.

After you have identified the need for a procedure, whatever its purpose, you use the same techniques for that procedure's detailed definition and construction.

Prerequisites

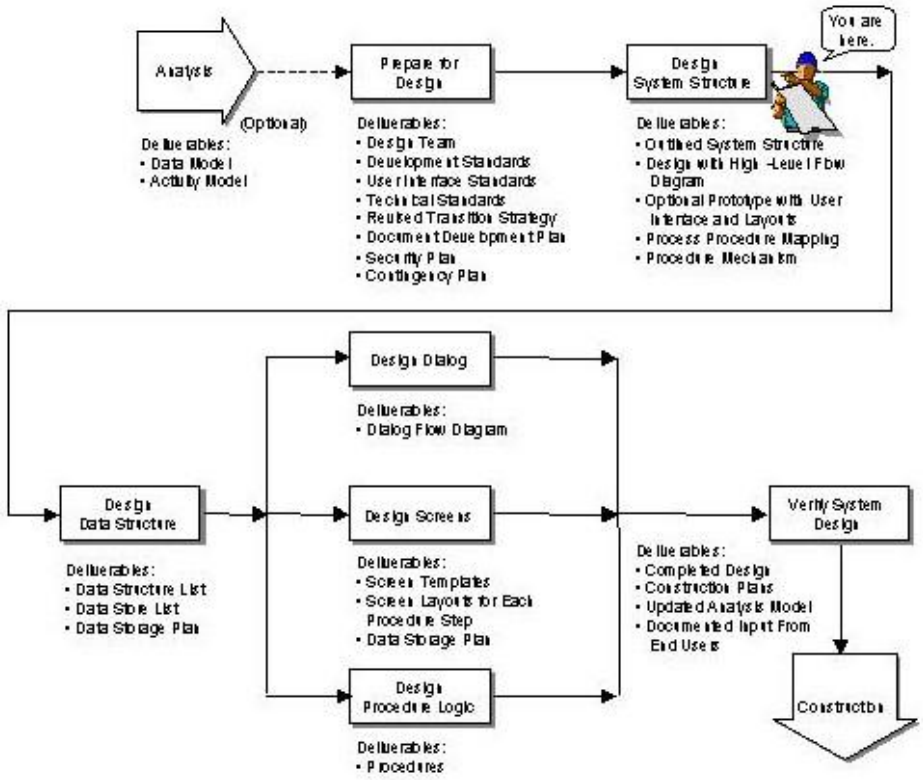
You should complete the tasks in preparing for design before beginning system structure design.

More information:

Preparing for Design (see page 15)

Design Process

The following illustration lists the deliverables from system structure design and shows where you are in the overall design process.



Influences on System Structure Design

User requirements should provide the primary drive for the design of any system. Therefore, consider the user interface before any other components.

The user interface determines the appearance of the system desired by the user. It is composed of a number of displays, including screens, windows and dialog boxes.

Linking the displays using dialog flows will determine the behavior of the system.

After the users confirm the appearance and behavior of the user interface, the procedure logic can be fully developed.

In system structure design, you are only interested in those aspects of the user interface that influence the other components of the system. Aspects of the user interface that display only for the user can be left until layout design.

Consider the characteristics of all the intended users. The design of an online dialog should address all possible interactions users might have with the business system.

The following factors influence the final appearance of the dialog:

- Volatility in the work environment
- Users' roles in the business
- Frequency of dialog use
- Geographical location of users
- Linguistic differences

Volatility in the Work Environment

Work environments fall into the following broad categories based on their degree of volatility:

- Constant-Deviation from a predictable, established pattern of work is unlikely. For example, a job that involves tabulating the results of questionnaires every day tends to be constant.
- Dynamic-The work pattern is so variable that a sequence of operations is unlikely to be repeated frequently. For example, a customer service role, in which the workflow can be dramatically affected by a phone call, is dynamic.

The dialogs designed to support a particular work environment should reflect its degree of volatility.

A constant work environment requires a highly structured dialog in which the system guides the user through the work pattern.

A dynamic work environment requires a loosely structured dialog in which the user directs the system based on shifting priorities.

A highly structured dialog tends to exhibit the following characteristics:

- Allows the user limited control because the system determines the sequence of actions
- Uses few menus

A loosely structured dialog has the following characteristics:

- Provides the user great flexibility in switching between procedures because the sequence of actions is largely unpredictable.
- May use a number of menus to simplify navigation among procedures.
- May use function keys and short command synonyms (usually single characters) to provide quick access to many procedures.

User's Role in the Business

You should consider the position and responsibilities of individuals using dialogs. The dialog design may vary based on a user's role, level of authority, or frequency of use of a particular procedure.

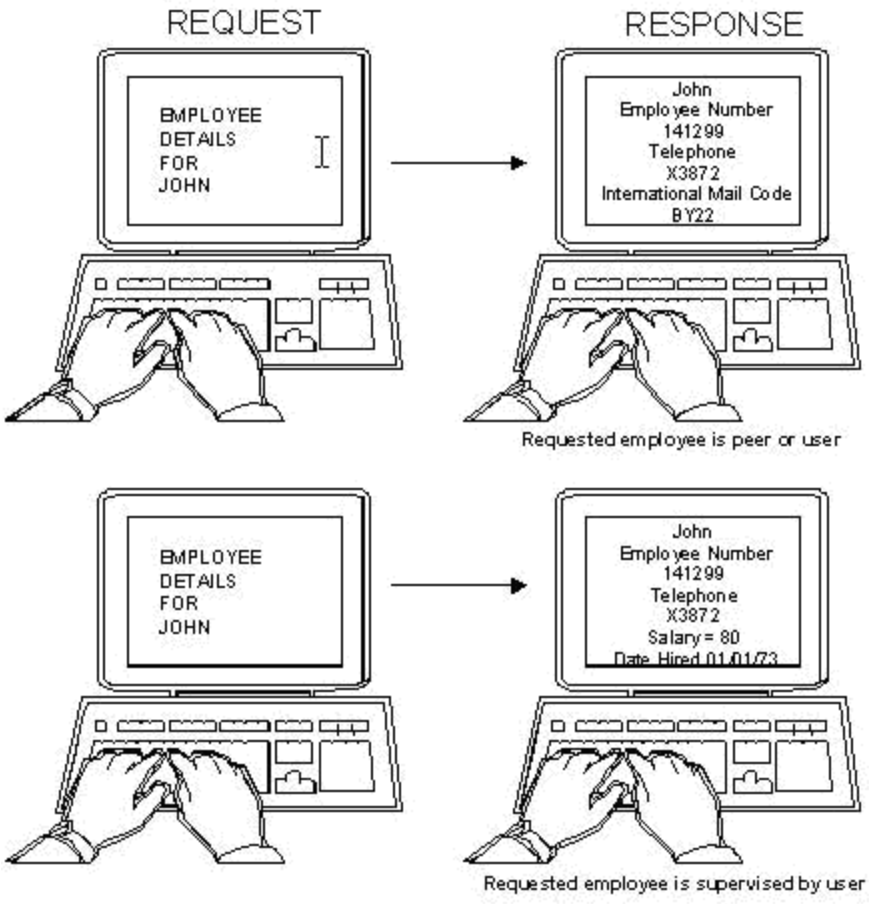
Dialog Variation Based on Role

The following table illustrates how a difference in role can result in different dialogs.

Order Entry System	Order Entry System
Data Entry Menu	Data Entry Menu
Maintain Customer Details	Allocate Work
Enter Orders	Review Productivity
Exit System	Check Quantity
	Data Entry Menu
	Exit System

The menu used by a data entry clerk (left) identifies the type of work the clerk performs. Since supervisors in the data entry department have broader responsibilities, their menu (right) supports additional functions as well as access to the same data entry functions as the clerks they supervise.

The following illustration shows how two users with different levels of authority might require different responses based on a common request.



In this example a peer and a supervisor have requested detail about an employee. The response the peer receives is less revealing than the response to the same request made by the supervisor. The response the supervisor receives includes Salary and Date Hired. The supervisor does not need the Internal Mail Code. The design would need to identify the user's level of authority. Perhaps a sign-on procedure could be constructed to recognize levels of authority. Layouts could then reflect that authority.

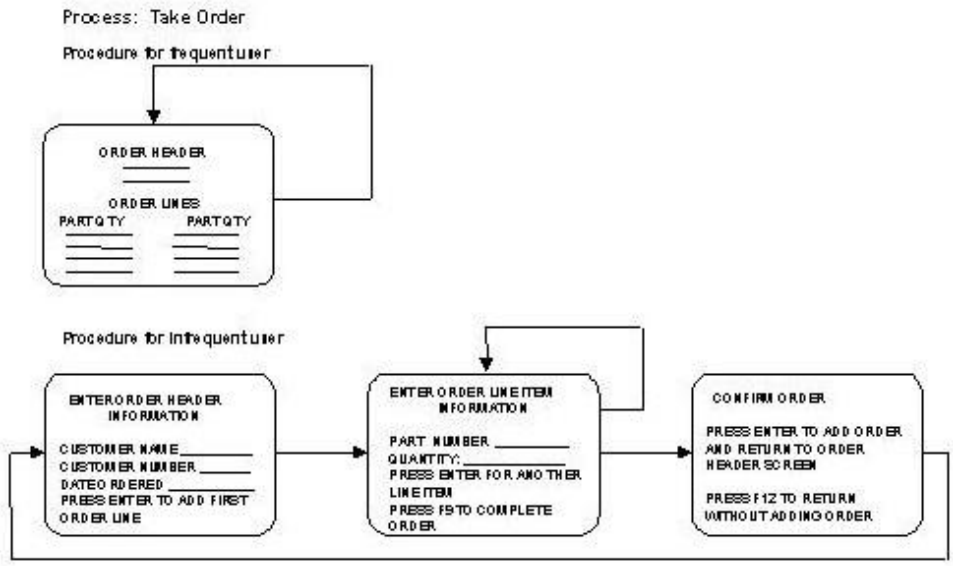
Frequency of Dialog Use

A difference in frequency of use can affect a dialog. A frequent user of a system, after becoming familiar with the system's operation, wants to complete each task with the minimum number of interactions possible. This results in fewer displays, more data presented in each display, and less help offered by the system.

The infrequent user needs a great deal more assistance from the system to complete the same work as the frequent user. As a result, the infrequent user will not object to a greater number of interactions.

The following illustration shows two separate sets of interactions that implement the same elementary process. Both sets of interaction implement Take Order. Each set uses a different dialog design and different procedures. One set accommodates frequent users, the other infrequent users.

The illustration uses screens as an example, but the same approach is true for windows.



Geographical Location of Users

If the application is to be used in more than one country or region, consider the following cultural differences:

- Different character sets and keyboards
- Relevance of examples
- Different terminology
- Different idioms

General policy or by consulting representative, users should identify these differences. This may require a number of country or region-specific user interfaces.

Linguistic Differences

Linguistic differences influence the design. CA Gen provides support for a number of different dialects. You can apply these to the user interface.

Note: For more information, see the *Toolset Help*.

Guidelines for System Structure Design

Consider the following key principles for designing a system structure:

- Adopt the user's perspective-Observe or imagine the user carrying out a certain task. The user interface should closely support that task.
- Give the user control-The user should be in control of the application and always able to move on to, cancel or switch to, another part of the system.
- Keep the user interface natural-The navigation of the system should be intuitive and easy to understand.
- Ensure consistency throughout the applications-Consistency helps users to transfer familiar skills to new situations. Applying well-formulated corporate standards for user interface design also ensures consistency.
- Keep the context of the system-If users are able to fill in six displays in any order to complete a task, they should know at any point what they have completed and what remains unfinished. This may be done by marking changes on a display, or providing a command to display all changes or the last change performed. This aids navigation through the system, and lets the user resume a task after interruption.
- Use real-world metaphors-A good user interface lets the user transfer skills from real world experiences. This makes it easier for the user to infer how to use an application. Base the interface design on things with which users are familiar, such as the manual files, documents or forms they use, or their engagement diary.

More information:

Preparing for Design (see page 15)

How to Analyze User Tasks

A user task is an identifiable activity, either manual or computerized, that the user carries out to achieve a particular result or goal. User tasks represent the human activities involved in carrying out the work of elementary processes.

User task analysis involves modeling the procedures performed by roles or persons in the organization in support of elementary processes. Current tasks may be analyzed, and an improved set of tasks may be specified.

If you have not already done so, perform task analysis for those tasks to be supported by the application. You may omit user task analysis where user involvement is minimal, or if a security subsystem is being built.

User task analysis consists of the following procedures:

- Gather information
- Identify external, temporal, and internal events
- Analyze how the user manipulates data
- Identify user tasks
- Define user tasks
- Define user task structures
- Map user tasks to elementary processes

Gather Information

Familiarize yourself with the elementary processes and roles defined during analysis.

To help identify the tasks that a user performs, gather information about the data (files, documents and data stores, described in analysis documentation) and activities currently involved in the tasks being examined.

Identify Events

Since events require a response, they are useful pointers to the tasks that might be performed in order to respond.

The following list details the types of events:

- External-An external event is something that happens outside the system being designed. Mail arriving is an example of an external event.
- Temporal-A temporal event is the arrival of a time point of interest to the system, for example, end of month.
- Internal-An internal event is something that occurs within the business activities that are included in the scope of the development project. They often arise as the result of human activities that are not specified as part of formal procedures. For example, making a decision or initiating an analysis of data is an internal event.

The event analysis technique defines external and temporal events, and the planned responses by business processes. This technique is described in the *Analysis Guide*. Analysts should have recorded the frequency of event, and possibly the required response time. This detail is useful for assessing the performance requirements of user tasks and system procedures.

External and temporal events can cause user roles to initiate a system procedure in order to respond. A batch procedure may be initiated by some automatic mechanism, such as the arrival of a time of day, or the existence of data that has been transmitted through a network.

Internal events are now of interest to you because they may indicate the need for additional procedures (such as reporting or browsing data), as well as providing an additional reason for executing formal, process-implementing procedures.

Analyze Data Manipulation

You need to examine how users manipulate data by identifying the actions they perform. For example, a user may complete, authorize, amend, or cancel an order form.

Identify User Tasks

A user task is an identifiable activity, either manual or computerized, that is carried out by the user to achieve a particular result (goal).

The task is typically triggered by an event and may consist of several sub-tasks. It is essential for the user to be able to tell when the task has been completed so the goal should be measurable.

Understanding how data is manipulated enables you to list the user tasks. At this stage, a simple list is sufficient. You can add detail later.

The identified data and actions may be a useful starting point, so are the business processes and the events that initiate them. However, these should be checked carefully with the user.

It may be useful to observe the user carrying out a task and to record observations for later discussion with the user. For example, a user may often need to switch to a higher priority task before returning to the original task. You need to ensure that the user can easily resume the original task.

Define User Tasks

You are now ready to define user tasks.

You need to record the following information:

- Flows between tasks
- Sub-tasks (components of each task)
- Data and documents used to complete the task
- Flows between sub-tasks
- Dependencies between or within tasks
- Time constraints on the completion of a single task, or of a group of related tasks
- Frequency of task performance
- How and when the user switches between tasks

Document tasks using structured English or a diagram such as a flowchart or a data flow diagram.

Define User Task Structures

Define the structure of each user task by observation or examination of task descriptions and the data used.

After you have defined the task structure, verify the design with the users. A *walk through* is an effective technique for conducting this type of review. It may be necessary to go back and review the tasks with the user to clarify any issues.

Example of a Task Structure

This is an example of a formal task structure, with its sub-tasks. User task-Take order from a customer over the telephone.

Note: The user can be interrupted in this task.

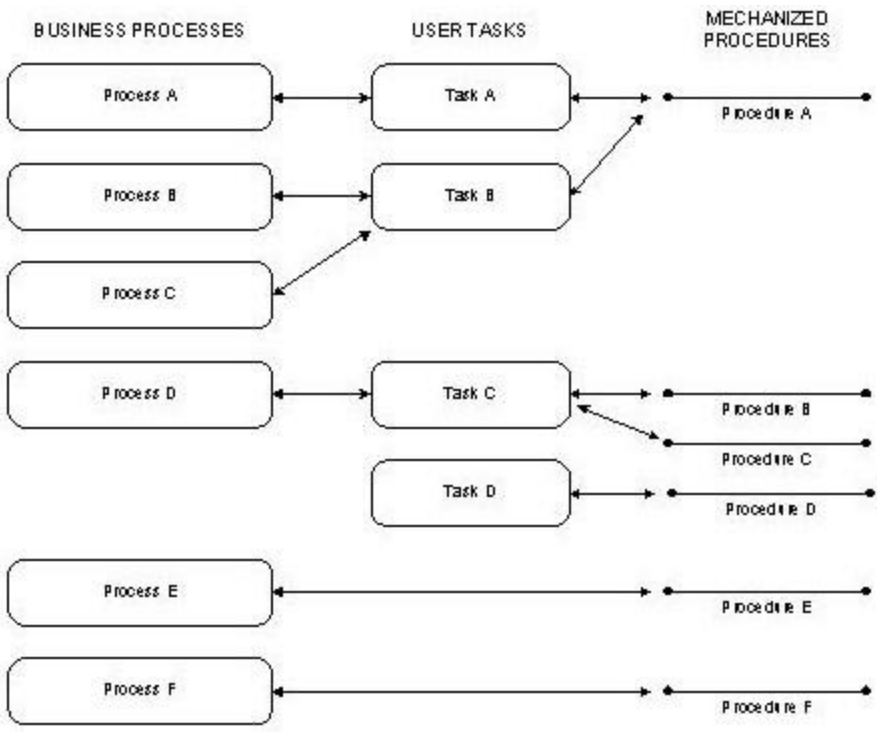
Initiating event-Customer telephones and indicates desire to place an order.

Sub-Task	Description
1	All customers must be registered before an order can be taken, so the user first checks to see if this customer is registered: If the customer is already registered, user continues with the task. If the customer is *not* registered, user must register the customer. (Registering the customer is a different but associated task.)
2	If the customer's registration details are correct, user completes customer details on the order form.
3	User asks what products the customer would like to order and in what quantity. User checks the availability of the products and the current price and confirms the order with the customer.
4	User repeats sub-task 3 until the customer is satisfied with the products ordered, the quantity, price, and any limitations on delivery time.
5	The completed order form is finally confirmed with the customer before it is formally recorded and sent off. The customer's record is amended to reflect the new order, and the product stock levels are adjusted accordingly. Copies of the order are sent to the appropriate departments.

Map User Tasks to Elementary Processes

You can now map the sub-tasks to the elementary processes you defined in analysis. This mapping may not be directly one-to-one. It depends on the granularity of the user task structure.

The following illustration shows some possibilities available to you:



Notes:

- A complete user task (such as Task A) may map to an elementary process in one instance, or its sub-tasks (as in Task B) may map to several elementary processes.
- A task (such as Task C) may need to be implemented as several procedures, each of which may support a specific sub-task.
- If a task or sub-task (such as Task D) does not map to an elementary process, this could indicate that the user task or sub-task should be implemented as a designer-added procedure, or that a change to the Analysis model needs to be negotiated.
- Other designer-added procedures (such as Procedure F) may not directly support any user tasks. Such a procedure may instead support business logic (such as calculate tax) defined in a common action block, rather than a process.
- Alternatively the procedure may be a system control procedure; if neither of these is the case, you should check carefully for missing user tasks.

- It is possible that the user task as performed is unnecessary. This should have been confirmed during analysis, or during business process re-engineering. Discuss any doubts with appropriate users.
- Sometimes a process, such as Process E, has no single directly correlating user task. This may indicate that not all user tasks have yet been identified, or the process may be concerned with maintaining data that will be used by many tasks or sub-tasks. Such a process may typically be mapped to a server procedure, which will be used by client procedures that do directly support user tasks.

There may still be elementary processes, such as Process F, that are not mapped to any user tasks. You should discover the cause and take corrective action along the following lines:

- The scope of the business system is incorrect, and the elementary processes should be included in a business system.
- An insufficient or inappropriate cross-section of the user community has been interviewed. Identified and interviewed those users who do perform the corresponding user tasks.
- A new elementary process was identified as desirable during analysis, or in business process re-engineering, but is not yet performed; this requires that user tasks be designed to support it.
- An elementary process does not apply to the user community to be supported.
- An elementary process identified during analysis no longer needs to be performed. This situation requires that the analysis model and requirements for the system be amended and reconfirmed.

Set Usability Criteria

Use measurable usability criteria for the user interface to assess the likely success of the system during testing or user trials.

Consider the following usability criteria:

- Ease of learning-How easy do users find it to learn to use the interface?
- Efficiency once learned-How efficient are users after they are familiar with the interface? Their productivity should increase.
- Ability of infrequent users to re-learn-How easy do occasional users find it to re-learn how to use the system after a break?
- Good design features, such as keeping the number of menu levels low, should help. Keeping a record of recurring problems will also be helpful, possibly even using the system itself to record problems.

- Frequency and seriousness of errors-How often do errors occur and how serious are they? Do any errors occur repeatedly?

  Measure these criteria by counting and classifying the errors to give the number of errors as a percentage of the total number of times a procedure executed. This will help to balance error ratios for often and infrequently used procedures.

- User satisfaction-How satisfied are the users?

  This can be subjective, but it does help to gauge user acceptance of the interface. However, the speed with which new users take up the new system or facilities may provide some measurable indication of satisfaction.

Design the User Interface Structure

This activity involves designing the user interface in detail and constructing a prototype to obtain comments and acceptance from the user.

Map User Tasks to a User Interface

The system structure is built using a prototype of the user tasks and sub-tasks.

Examine the opportunities and technical limitations of the products being used. In particular, decide how to subdivide the system into procedures by considering issues such as the following issues:

- A data maintenance task should be supported by one or more system procedures, depending on data integrity and other business rules. For more information about procedures, see the chapter “Designing the Procedure Logic.”
- Part of a user task, such as a look-up display, is a part of many user tasks, and so needs to be supported by a separate reusable procedure.
- Parts of the user task require different levels of security.

Map User Tasks to Procedures

User tasks are mapped to procedures to ensure that the user tasks provide the main framework on which the user interface is based.

The following list shows the categories of procedures:

- Process-implementing procedures
- Designer-added procedures

More information:

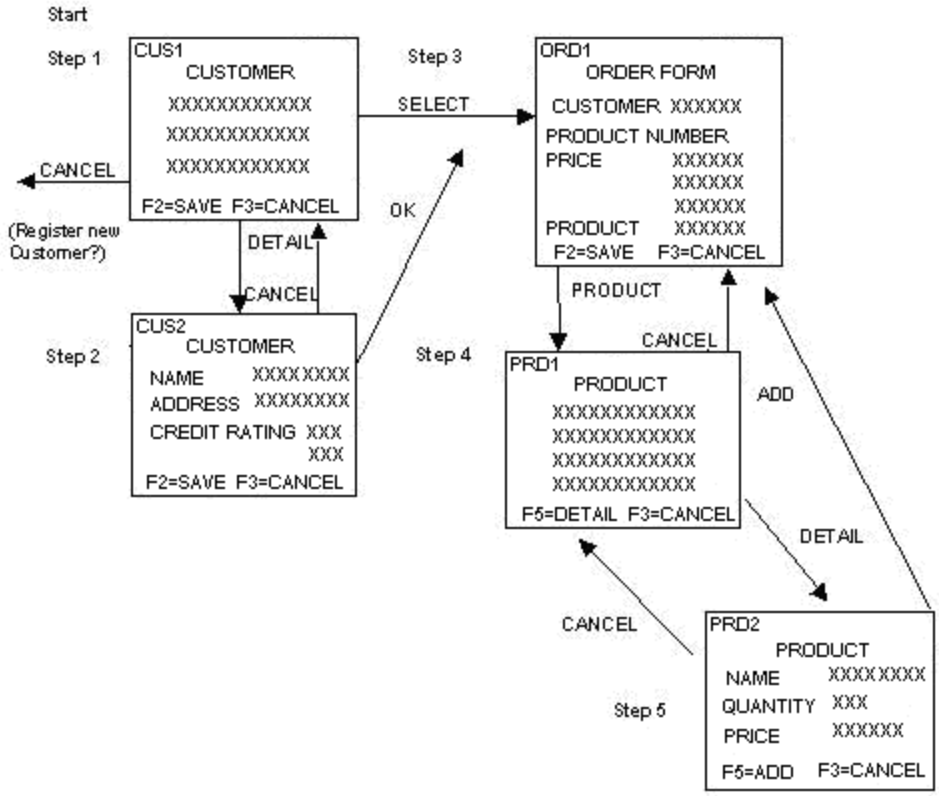
Designing the Procedure Logic (see page 123)

User Interface Prototyping

Perform prototyping to check the completeness and usability of support for user tasks, and to refine the design of the user interface.

A useful method for verifying the design with business persons is to convert the user interface structure into a computer-based prototype running on a workstation.

The following illustration shows this method:



The prototype should demonstrate the screen interface, screens, menus, and command lines.

The data usage identified earlier provides a useful basis for the design of the user interface. For example, you can base form layouts in the user interface on forms currently in use. An on-screen product catalog can mimic a paper-based one.

It should be possible to move between the displays and dialogs so that the usability of the dialog flow can be checked. You can collect initial comments from the potential users of the interface.

You may need to produce several prototypes, iterating where necessary, before the design can finally be confirmed with the user.

It may be beneficial to review the usability criteria, amending them as necessary.

A prototype may still prompt questions about the user interface and the movement between the different displays. For example, consider the prototype shown in "Prototype for a User Task." Would users wish to display Order Forms after displaying Customer Details, or after displaying Product Details in addition to displaying Product? What selection criteria should be assumed?

During an analysis of user tasks and prototyping a user interface structure, it might become obvious that the structure, sequence, and interaction of the activities within the user task are not the most efficient way of performing the overall business task. Although it is not the purpose of system structure design to radically re-engineer the user task, any suggestions for improvements in workflow should be presented to the users. Ultimately, it is the users' system and therefore should work effectively for them.

Choose a Procedure Style

Choose a procedure style for implementation. The basic decision is whether to use batch or online.

Online Procedure Style

For many systems that are driven by the arrival of events or work to be performed by users, the default procedure style will be online.

Users may be able to perform some online procedures entirely on a workstation or personal computer.

Batch Procedure Style

Consider batch procedures when the following statements are true:

- A large volume of transactions must be processed together, and the processing can proceed without interim user decisions.
- The response time achievable for the processing volume would interrupt the user's workflow.

A batch procedure might not be the solution for all executions of a user task. For example, creating a new manufacturing plan for a week might be requested by a user as a batch procedure. Optimizing a plan for a single shift period might be best performed online.

Workstation or Personal Computer Procedures

Users can perform some tasks entirely using a personal computing application in isolation from data used by any other users. This might require Custom-designed procedure executing as an application.

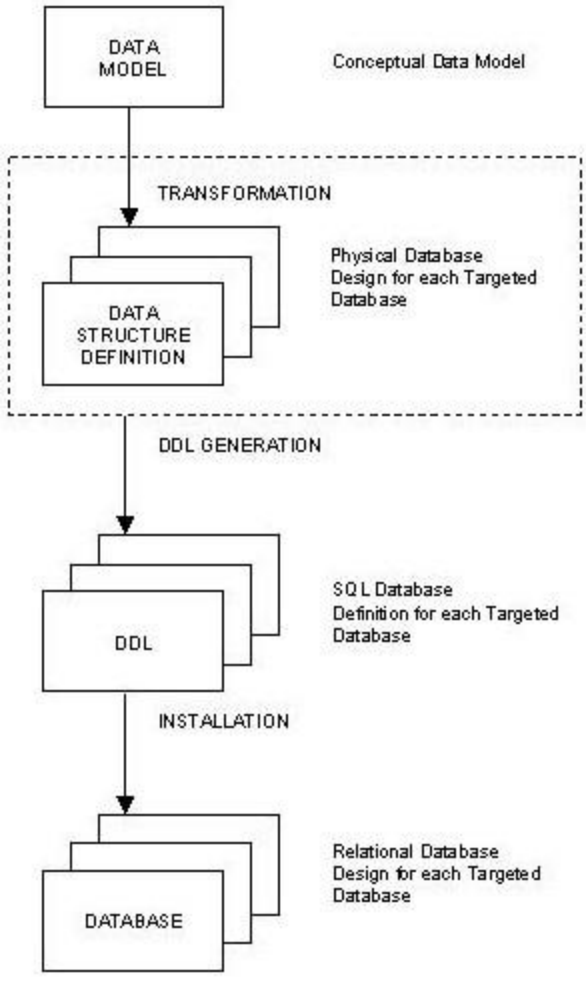
Where possible, a standard packaged application, for example, writing a letter to a customer or supplier using a word processing application, or planning a warehouse construction project using a project management application.

Chapter 4: Designing the Data Structure

Data structure design is the design of a relational database using CA Gen. The data structure is the physical version of the data model. This version does not violate the intent of the data model but does allow customization for performance and specifically targeted databases.

Create a Relational Database Definition

The following illustration shows the overall approach to creating a relational database definition.



Each database defined using CA Gen contains its own Data Structure List and Data Store List. These two lists represent the physical database definition that forms the basis for the generation of the Data Definition Language (DDL). DDL generation yields the Structured Query Language (SQL) statements required to define a database to a relational database management system (DBMS).

Also consider the means of converting data to the new database (single location or distributed). Some further conversion procedures or external action blocks to interface to database management systems not supported by CA Gen Toolsets may be needed. Early planning for these transition and associated implementation issues makes the entire process much easier.

Prerequisites

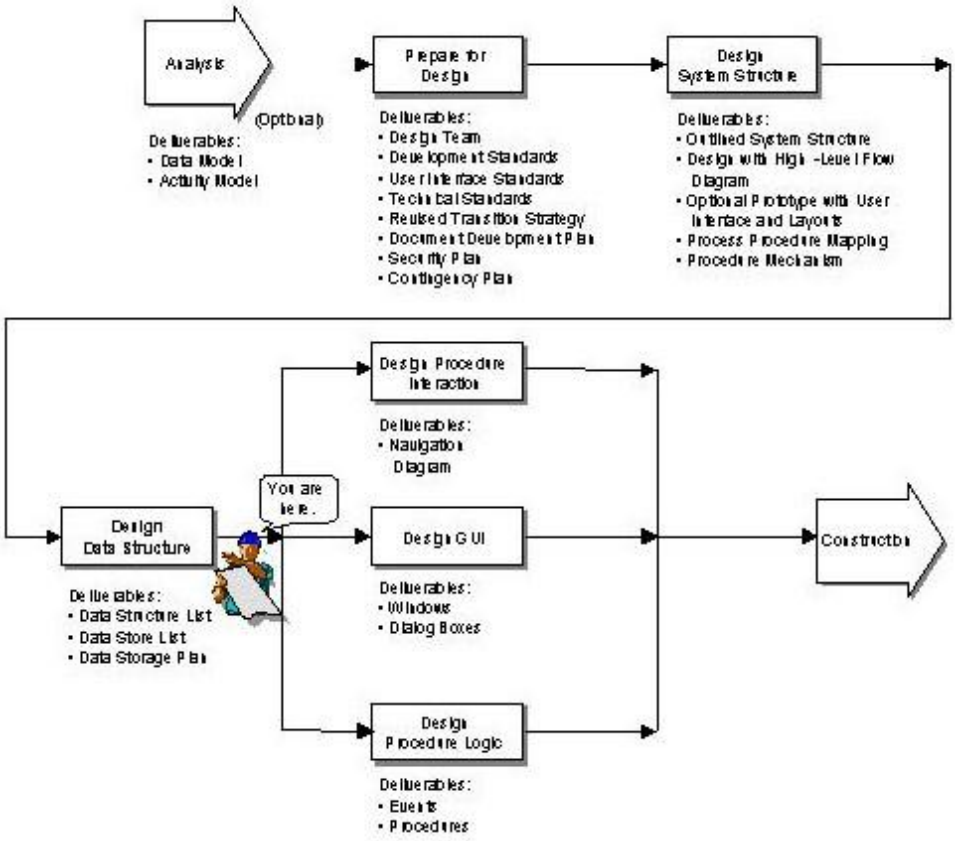
You need an outlined system structure design before beginning data structure design.

More information:

Designing the System Structure (see page 31)

Design Process

The following illustration shows the deliverables from data structure design and shows where you are in the overall design process:



Note: Although this illustration shows data structure design preceding Procedure Interaction, GUI, and Procedure Logic design, data structure design can be done in parallel with or after these activities.

Data Model Transformation

You begin database design with the data model defined in analysis.

Note: For more information, see the *Analysis Guide*.

The data model is only a conceptual definition of the database. When the data model is complete, CA Gen can automatically generate the initial database definition that reflects the data model. This activity is called transformation. Further definition is needed for specific database implementations.

More information:

Data Model Retransformation (see page 66)

Transformation Implications

Before starting transformation, it is worth considering the implications of transformation, which are the differences between the following definitions:

- Conceptual (business level) definition represented as a data model
- Logical database definition represented by the Data Structure List
- Physical definition represented by the Data Store List

When you transform the data model, a physical database definition is created. There is no dynamic link from this definition back to the data model.

The following list details the implications of transformation:

- You can make changes to the data model with no impact on the data structure.
- You can use the CA Gen retransformation capability when changes need to be reflected in a Data Structure List or Data Store List.
- You can make changes to a Data Structure List without affecting the data model.
- You cannot change the Data Structure List to violate the rules of the data model. For example, a text field cannot become a numeric field.
- If you use transformation or retransformation again, the changes you made in the Data Structure List and Data Store List will be lost.
- If you remove a foreign key index from a data record and then retransform, you will typically see the foreign key index added back in. In some cases, it may be better to rename unneeded indexes. For example, you could prefix each unneeded index with ZAP. You could then delete them near the end of testing for a project.

You must use meticulous change management to maintain the business logic embodied in the data model while allowing physical changes to be made to the Data Structure List. Any modifications to these lists need to be documented thoroughly to ensure consistency between different versions of the lists.

How to Set Technical Design Properties

Before you transform the data model, you need to establish the technical design properties. Technical design properties for each targeted DBMS provide control over the database definition created during the transformation process.

The technical design properties include the following information:

- Reserved word checking
- Referential integrity (RI) enforced by either the DBMS or CA Gen
- Permitted value default enforcement
- Data structure defaults

Set Reserved Word Checking

The technical design property options for reserved word checking are listed in the following table.

Option	When to Use
DBMS	If you are going to let CA Gen name the tables and you do not want any reserved words from the DBMS used.
None	If you are planning to name the tables such that none of the reserved words for the DBMS are being used. This is much more typical for those applications that are using coded table names or a code in the table name that would make it unique.
All	If you are going to let CA Gen name the tables for you, and you would like the tables in each of the targeted DBMSs to be named exactly the same.

Set Referential Integrity Enforcement

The technical design property options for referential integrity enforcement are listed in the following table.

Option	When to Use
CA Gen-enforced RI	If you want CA Gen to enforce RI. CA Gen generates pieces of code to handle all of the referential integrity.
DBMS-enforced RI	If you want the DBMS to handle as much RI as possible. This keeps the amount of code generated as small as possible and helps system performance. CA Gen allows the DBMS to handle all of the referential integrity it can and only generate pieces of code for the referential integrity not handled by the DBMS.

Set Permitted Value Default Enforcement

The technical design property options for permitted value default enforcement are listed in the following table.

Option	When to Use
Reads - Enabled	If you want the permitted value enforcement for read statements for all of the tables in the databases to be handled by the DBMS. By enabling the DBMS to control the permitted value enforcement, you are requiring all applications (CA Gen or not) that will be reading tables in the generated database to adhere to the same permitted value rules. If you allow the database to handle the permitted value violations, there will be no action taken for the reads within the CA Gen application.
Reads - Disabled	If you want the permitted value enforcement to be handled at the database level and you want CA Gen to do the enforcement. This is a good selection if there are no permitted values in the database, or if you do not want the DBMS to enforce the permitted values for READ statements.
Creates/Updates - DBMS	If you want the permitted value enforcement for row creates and updates for all tables in the databases to be handled by the DBMS. By enabling the DBMS to control the permitted value enforcement, you are requiring all applications (CA Gen or not) that will be creating or updating rows in any of the tables in the generated database to adhere to the same permitted values. If you select this option, an SQL return code will be returned to the CA Gen application within the create statement (when permitted value violation exception) and you will handle it in the procedure logic. See the chapter "Designing the Procedure Logic."
Creates/Updates - CA Gen	If you want CA Gen to handle the permitted value enforcement. This is a good selection if there are no permitted values in the table, or if you do not want the DBMS to enforce the permitted values for create and update statements.

Set Data Structure Defaults

Generally, data structure defaults do not need to be adjusted. Most of the defaults relate to specifics of the DBMS being used.

Review data structure defaults with a database expert before implementation in the other environments. However, the default values for the transformation produce the DDL necessary for SQL generation and installation.

Perform Transformation

Transformation uses the data model to derive a relational database design conforming to the technical design properties, described in How to Set Technical Design Properties.

Before performing transformation, CA Gen runs a consistency check against the data model.

If any part of the data model (every entity type, attribute, and relationship) is incomplete or inconsistent, the transformation is aborted. You need to correct any errors and severe warnings (but not warnings) in the data model before continuing.

You will need to complete a transformation for each DBMS to be used.

Results of Transformation

Transformation creates a logical and physical database definition for all of the DBMSs, even if you are only going to target a single DBMS. This way, each of the implementations matches at transformation time.

Transformation creates the following diagrams for each DBMS targeted:

- Data Structure List-logical table/column definition
- Data Store List-physical storage method definition

These lists serve as a database definition for DDL generation.

When the system design is complete, you may want to modify the Data Store List or the Data Structure List, or both to conform to local standards or optimize performance.

More information:

How to Complete the Data Structure Design (see page 68)

Data Structure List

A Data Structure List is created for each DBMS for which you completed the transformation. Each Data Structure List shows how the data in the data model is logically mapped to a database.

After transformation, each Data Structure List shows how the conceptual data model represented by the data model should be translated into a physical model containing tables, columns, RI constraints, and indexes.

Conversely, the Data Structure List reflects any changes made in the Data Store List.

Use the Data Structure List tool to perform the following tasks:

- Refine the data structure organization.
- Define how relationships are implemented.

More information:

Data Store List (see page 65)

Data Structure Diagramming Terminology

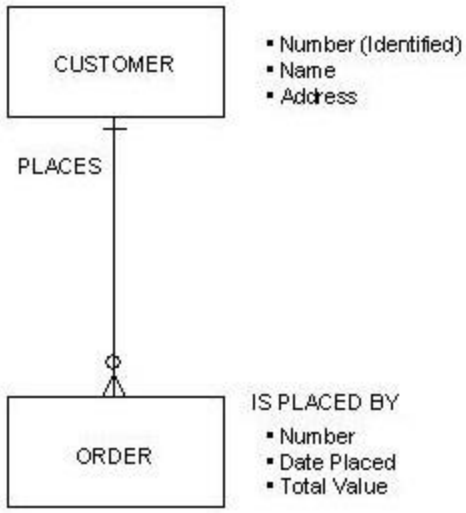
The equivalent data modeling, relational, and production terms are shown in the following table.

Data Structure Terms

CA Gen Model	Relational Term	Production
Data Model	Data structure (Physical Design)	Database Implementation by the DBMS
Entity Type and its Subtypes	One table	Table
Attribute - Basic	Column	Column
Attribute - Designed	Column	Column
Attribute - Derived	Not implemented physically	Not implemented physically
Attribute - Auto Number	Column	Column
Relationship (1:1 or 1:M)	Foreign key	RI constraint
Relationship (M:N)	Table	Table
Identifier	Index	Index (unique)
N/A (added to the Data Structure List)	Denormalized column	Column
N/A (added to the Data Structure List)	Index	Index (typically non-unique)

One-to-Many Relationships

The following illustration shows a one-to-many (1:M) relationship:



This illustration is a data model fragment showing the following relationships:

- Each CUSTOMER entity sometimes places (relationship membership) one or more ORDER entities
- Each ORDER entity is placed by (relationship membership) exactly one CUSTOMER.

The identifier of CUSTOMER is an attribute called CUSTOMER NUMBER.

The following illustration is a Data Structure List showing the implementation of the customer number as a foreign key.

Type	Macro Name	Format	Length	Optional
Table	CUSTOMER			
Column	NUMBER	Integer	4	Not Null
Column	NAME	Char	20	Not Null
Column	ADDRESS	Char	78	Not Null
Index (U)				
Column	PKEY (Primary) NUMBER	Integer	4	Not Null
Table	ORDER			
Column	NUMBER	Integer	4	Not Null
Column	DATE_PLACED	Date	8	Not Null
Column	TOTAL_VALUE	Float	5,2	Not Null
FK Column	FK_CUSTOMER MNUMBER	Integer	4	Null
RI Constraint	<No Name> Customer			
Index	FKEY			
Column	FK_CUSTOMER MNUMBER	Integer	4	Null
Index (U)	PKEY (Primary)	Integer	4	Not Null
Column	NUMBER			

The ORDER table includes a Foreign Key (FK) column called FK_CUSTOMERNUMBER. For any ORDER, the number of the CUSTOMER that places it is stored in that column. This will allow your application user to access from the ORDER table the CUSTOMER who placed an ORDER or the ORDERS a CUSTOMER has placed.

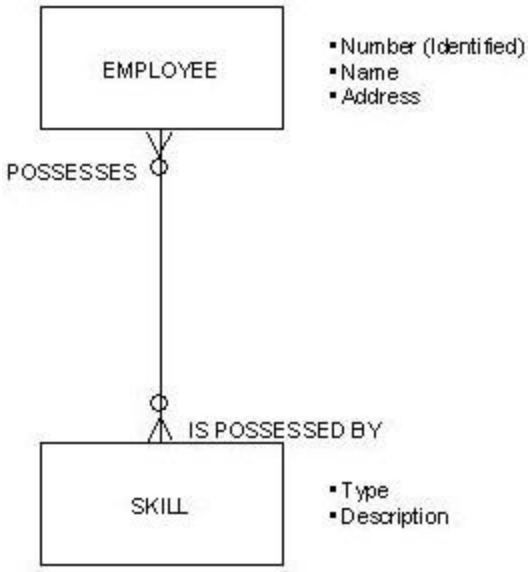
Many-to-Many Relationships

Many-to-many (M:M) relationships should have been resolved in analysis by defining an additional entity type.

Most M:M relationships have some extra information that needs to be kept on an associative entity type, as described in the Associative Entity Type section. Even if you do not find this during the initial implementation of an application, it will often happen later.

Resolving the M:M into an associative entity type results in a more stable model and requires fewer action diagramming changes in additional intersection data is found after the initial version of an application has been implemented. If this has not been done, any remaining M:M relationships are implemented with another table.

The following illustration shows a many-to-many relationship:



This illustration is a data model fragment showing the following relationships:

- Each EMPLOYEE possesses one or more SKILLS.
- Each SKILL is possessed by one or more EMPLOYEES.

The identifier of EMPLOYEE is the attribute Number.

The identifier of SKILL is the attribute Type.

The following illustration is a Data Structure List showing the implementation of the M:M relationship.

```
Type            Macro Name             Format     Length Optional

Table             EMPLOYEE
Column            NUMBER                Integer      4   Not Null
Column            NAME                  Char        30   Not Null
Column            ADDRESS               Char        78   Not Null
Index (U)         PKEY (Primary)
Column            NUMBER                Integer      4   Not Null

Table             IS_POSSESSED_BY
FK Column         FK_SKILL_TYPE         Char        15   Not Null
FK Column         FK_EMPLOYEE_NUMBER    Integer      4   Null
RI Contrating     <No Name> EMPLOYEE
RI Constraint     <No Name> SKILL
Index (U)         FKEY
Column            FK_EMPLOYEE_NUMBER     Integer      4  Not Null
Column            FK_SKILL_TYPE          Char        15  Not Null

Index (U)         PKEY (Primary
Column            FK_SKILL_TYPE          Char        15  Not Null
Column            FK_EMPLOYMENT_NUMBER   Integer      4  Not Null

Table             SKILL                  Char        15  Not Null
Column            TYPE                   Char        78  Not Null
Column            DESCRIPTION
Index (U)         PKEY (Primary)         Char        15  Not Null
Column            TYPE
```

Note: A table joins the EMPLOYEE and SKILL tables and its name is IS POSSESSED BY (the name of one of the relationship memberships). The link record contains both the key of the EMPLOYEE table and the key of the skill table. This way, if the database is queried by EMPLOYEE number, it can find all the SKILLS that the EMPLOYEE possesses, and given a SKILLTYPE, it can find all the EMPLOYEES who possess it.

Data Structure List Terminology

The following terms are used in the Data Structure List:

- Tables
- Indexes
- Referential integrity (RI) constraints

Tables

A table is an implementation of an entity type appearing in the data model. A table is associated with the data model entity type that it implements.

Each table is attached to at least one index, the contents of which uniquely identify an occurrence of the table in the database.

Each table may be attached to RI constraints, which are implementations of relationships from the data model.

A table will be placed in the Data Structure List for each of the M:N relationships. The table will be named by the M:N relationship itself. This type of table is relatively rare. Such tables contain foreign key columns to both participants in the relationship.

Each table created by a M:M relationship is granted two unique entry points. Each entry point contains the two foreign key columns in each of the two possible sequences. The only columns appearing on this type of a table are the foreign keys to the two participants in the relationship.

Properties of Tables

Each table in the database has the properties listed in the following list.

- Entity Type—The name of the entity type in the data model that this table logically represents. This is read only in the Data Structure List.
- Database—This is the database for which this table is implemented.

  You cannot change which database implements this table in the Data Structure List. (If you need to change this, go to the Data Store List.)
  The database name is a maximum of 8 characters.
- Generic Name—A name that is consistent for the implemented entity type throughout all of the technical designs. It is called generic because it is not DBMS specific.

  The generic name can be no more than 32 characters long. Change this name if you want the name of this table to be consistent throughout all of the technical designs, regardless of DBMS.
- DBMS Name—Used by the specific DBMS generation to identify a table.
  CA Gen uses the table name as the default table name, unless otherwise specified here or in the data model. If this name is changed in the data model, each time a transformation is made you would not have to change the name.

  The DBMS table name can be no more than 32 characters long in the CA Gen model. Certain databases, however, only allow a maximum of 18 characters. Do not exceed 32 characters unless you are certain your database supports it.
- Owner—Specifies your name or ID. The DDL uses this owner during processing.

Permitted Value Default Enforcement - Reads

The following list details the options you can set to handle the permitted value enforcement for reads:

- Defaulted—Select this option if you want the permitted value enforcement to be handled at the database level.
- Enable—Select this option if you want the permitted value enforcement for table reads to be handled at the table level and you want the DBMS to do the enforcement.

  By enabling the DBMS to control the permitted value enforcement, you are requiring all application (CA Gen or not) that will be reading this table to adhere to the same permitted value rules.
- Disable—Select this option if you want the permitted value enforcement to be handled at the table level and you want CA Gen to do the enforcement.

  This is a good selection if there are no permitted values in the table, or if you do not want the DBMS to enforce the permitted values for READ statements.

Permitted Value Default Enforcement - Creates/Updates

The following list details the options you can set to handle the permitted value enforcement for creates or updates:

- Defaulted—Select this option if you want the permitted value enforcement to be handled at the database level.
- DBMS—Select this option if you want the permitted value enforcement for row creates and updates to be handled at the table level and you want the DBMS to do the enforcement.

  By enabling the DBMS to control the permitted value enforcement, you are requiring all application (CA Gen or not) that will be creating or updating rows in this table to adhere to the same permitted value rules.
- CA Gen—Select this option if you want the permitted value enforcement to be handled at the table level and you want CA Gen to do the enforcement. This is a good selection if there are no permitted values in the table, or if you do not want the DBMS to enforce the permitted values for create or and update statements.
- Description—Freeform text documentation about the table.

  Keep information about changes that have been made to this table in technical design. Although CA Gen knows these changes, you may want to know what changes were made in case the table is retransformed.

Note: Remember that this description will be eliminated when this table is transformed again (either by transformation or retransformation).

Other than these properties, each table contains a number of details that are specific to target DBMS implementations. The default settings are usually sufficient for unit testing; they need not be considered until volume or integration testing begins.

Indexes

In the data structure, an index describes an arrangement of columns used as an index into a table. Indexes are used for the following purposes:

- Ensure uniqueness of entities represented in the database
- When an index is specified as being unique, it allows no two records in the same table to have the same index value. For example, consider the entity type CUSTOMER with an identifying attribute, Number. When customer is implemented as a table in the Data Structure List, it is given a unique index of Number. When the database is subsequently generated and installed, no two customers are allowed to have the same Number.
- Enhance performance of the generated database
- The use of indexes to improve performance depends on the access characteristics of the procedures that use the data.

Imagine that the CUSTOMER entity type has a non-identifying attribute called Name, and that no index is defined for Name. Thus, for a procedure that lists customers in alphabetical order by Name, at execution time the DBMS would have to execute these steps:

1. Look at all of the CUSTOMER records
2. Sort them into Name sequence
3. Present the list to the procedure

When an index is defined for Name, the DBMS automatically keeps track of customers in Name sequence, eliminating the need for sorting the records. CA Gen automatically creates an index for each identifier during transformation. In addition, it creates an index for foreign keys implementing relationships.

Each index is associated with exactly one record in a table and may have the properties shown in the following table.

Properties of Indexes

Each index in the database has the properties listed in the following list.

- Generic Name—A name for the index that is consistent throughout all of the technical designs. It is called generic because it is not DBMS specific.

  The generic name can be no more than 32 characters long.
  There is no reason to change this name in the Data Structure List.

- Name—During DDL generation and installation, this is the name of the index that will be created using the properties of this entry point.

  The transformation process creates a unique default name for each index in the Data Structure List. Unlike columns and tables, an index is not based on a single object from the conceptual mode, so CA Gen has little basis on which to form a meaningful name. As a result, the names given to entry points created during transformation appear as the letter "I" followed by a string of digits. Although they work properly that way, you may choose to change the name to something more meaningful.

- Unique—Indicates that the index will not permit duplicate key values. If the index is derived from a data model identifier, then it must remain unique.

  It is unusual to have more than one unique index for a table so review carefully before creating a second (or third) unique index.

- Cluster—If you want the table itself to be sorted into the same order as the index, so that performance may be improved for certain types of access, you would add a cluster index. Review the material about clustering for your DBMS.

  In several cases, the DBMSs attempt to put rows in the clustering order by only actually organizing every row in a clustering order after a reorganization.

- Primary—Marks the index as the primary key.

  The referential integrity process sets the primary key during transformation, retransformation, or a separate (RI) process.

  Each table has only one primary key, which is used as the foreign key in relationship implementation.

- Description—A freeform text field used to document the purpose of the index.

RI Constraints

A referential integrity (RI) constraint is the implementation of a relationship in the data model. It defines how the foreign key is used to get from one table to another, as described in Transforming the Data Model.

For a M:M relationship, two RI constraints and a table are required.

In the unlikely event an entity type has more than one identifier, the table that implements it is given a unique index to support each of them. CA Gen uses the primary index of the related table to implement the foreign key. Choosing another identifier may change CA Gen's selection.

Properties of RI Constraints

The following list details the RI constraint properties that can be changed:

- RI Constraint Generic Name—The generic name of the RI constraint is a name that is consistent for the implemented relationship throughout all of the technical designs. It is called generic because it is not DBMS specific.

  The generic name can be no more than 32 characters long.

  There is no reason to change this name in the Data Structure List.

- RI Constraint DBMS Name—If the DBMS detects a referential integrity error, this RI constraint name appears in the error message. Therefore, you should name the RI constraint something meaningful.

- Referential Integrity Options Enforced by—You have the option on each RI constraint to have the DBMS handle the constraint (if the DBMS contains the capability to handle the particular constraint) or let CA Gen generate the code to handle the constraint.

  Typically better performance is gained by selecting the DBMS to handle anything it can and let CA Gen generate the constraints not handled by the DBMS.

- Description—A free text field used to document the purpose of this RI constraint.

Other Data Structure List Details

A number of additional DBMS-specific details can be modified using the Data Structure List, for instance the placement of tables in tablespaces, the names of the database and indexspaces, and so on.

As with the DBMS-specific details of tables and indexes, CA Gen creates defaults that enable database generation to create a usable database. Their values should be adequate for the operational environment used for procedure testing, and need be changed only to satisfy installation standards or performance needs.

Data Store List

A Data Store List is created for each of the DBMS when you complete transformation. After transformation, each Data Store List shows how the data in the data model will be stored in a physical database.

CA Gen's transformation facility uses a rule-based approach to examine data model objects and determine appropriate implementation. CA Gen generates the objects, associations, and properties. Each Data Store List suggests an initial storage strategy that you can modify to fit the needs of your DBMS.

The Data Structure List for that specific DBMS reflects any changes made in the Data Store List and vice versa. For more information, see Data Structure List.

The database administrator uses the Data Store List tool to perform the following tasks:

- Add and define databases
- Add data devices
- Add log devices
- Add data files
- Assign tablespaces and indexes to the appropriate databases
- Assign any tablespace or indexspace to storage parameters
- Define tablespace partitioning

Data Model Retransformation

The CA Gen Data Model tool automatically ensures that changes in the Data Structure List do not invalidate the properties specified in the data model.

The reverse, however, is not true. To provide you with maximum flexibility, changes to the data model are not immediately reflected in the Data Structure List. You must choose when you want the Data Structure List to be updated.

In nearly all cases, data model changes should be immediately followed by retransformation. Failure to do so could result in application code that does not work.

Important! Transformation, which is the wholesale recreation of the Data Structure List and the Data Store List from the data model, results in the loss of a customization that has been done and should be used with extreme caution. It is advisable to delay customizing the Data Structure List until the data model is accurate and stable.

It may be necessary to change the data model after either list has been modified. For instance, you may need to enhance an existing CA Gen-generated system. If so, a selected part of the data model can be transformed using the retransformation technique.

Basics of Retransformation

CA Gen retains complete knowledge of how each component of the data model is implemented in the data structure. Each object on the list (except for indexes) is the direct implementation of an object or objects in the data model. CA Gen can easily keep track of what has or has not been implemented. CA Gen-executed consistency check refreshes the Data Structure List and the Data Store List to eliminate the errors.

Note: A change to the DSD name for an implemented entity type or attribute does not cause a consistency check error and so will *never* result in a change to the name of the corresponding table or column.

As system development progresses and the data model changes, you may need to transform specific pieces of the data model. These changes may need to be implemented into all of the possible DBMSs, only a few, or only one. This is achieved through incremental retransformation.

These situations may require incremental retransformation:

- Adding a new entity type, relationship, or attribute to the data model
- Changing an existing entity type, relationship, or attribute

When new objects are added to the data model, their implementations must be added to the Data Structure List for the targeted DBMS as well.

When an existing data model object is changed, its implementation on the Data Structure List must be deleted. CA Gen then puts the object on the list of unimplemented data model objects, and it may then be added as if it were a new data model object.

The following list details the rules of retransformation for multiple target DBMSs:

- If you want to make sure the data model is transformed for all of the targeted databases, use the synchronization of the data model to the technical design.
- If you need to update only a single DBMS with changes from the data model, use specialize technical design for current DBMS.
- If you have only a single target DBMS, use synchronization of the data model to the technical design. If there is only a concern about a single DBMS, this will always keep all of the DBMSs synchronized.

An object deleted from the data model requires no retransformation action as the object and its dependent objects are immediately deleted from the Data Structure List.

For example, deleting an identifying attribute in the data model will cause the following to be deleted:

- Corresponding column in the table
- Column in the unique column
- Foreign key columns in dependent tables to be deleted
- Foreign key columns in the indexes to be deleted

Identify Changes to the Data Model

Whenever the data model is changed, and a Data Structure List exists, a consistency check should be used to report the following details:

- Which entity types, relationships, and attributes in the data model are not yet implemented in the Data Structure List.
- Which entity types, relationships, and attributes in the data model are not implemented properly in the Data Structure List.

The affected entity types, relationships, and attributes may then be retransformed.

How to Complete the Data Structure Design

The activities in this section should be completed after the system is unit tested. The longer changes are deferred, the less likely they are to be lost in retransformation.

Completing the data structure design consists of the following tasks:

- Change database names to conform to installation standards
- Optimize the data structure design
- Rearrange the database structure

Change Database Names

Most installations have naming conventions for databases installed in shared environments.

Changes to database, tablespace, indexspace, index, and column names can be made without affecting the names of related objects in the data model or Data Structure List.

Optimize the Data Structure Design

In most cases, CA Gen supports tuning database performance without changing the data model. Changes should be reviewed with a specialist for the target database.

The following list details some of the optimizations that can be performed in the Data Structure List:

- Adding and deleting indexes-Adding new indexes will result in new index definitions in the DDL.

  Analyzing the access paths used in procedures will determine a set of entry points that allow the DBMS to traverse the database as efficiently as possible.

  After the system has been designed, it is possible to remove any unused indexes from the Data Structure List.

- Denormalizing-CA Gen also supports a technique called denormalization, which involves replicating fields from one record into another, related record.

  Logic generated by CA Gen automatically maintains the correct values of data replicated in this way.

- Specifying locations for tables-Some DBMSs support specified location of tables on pages and different storage devices. The locations of tables can be specified in CA Gen.

- Making changes specific to the DBMS-The changes that can be made to optimize the design depends on the target DBMS. For example, DB2 allows the clustering of records from different tables to the same tablespace to improve access to related records.

Some optimization needs to be performed before procedure design begins, but some can only be completed after the detailed design has been completed. The DBMS expert will need to make a least two "passes" over the Data Structure List and Data Store List before the physical definition of the database is converted into DDL statements.

The optimizations should be performed to increase performance for the predominate processing pattern, since for every gain in one area, losses may be sustained in others.

Rearrange the Database Structure

Physical storage characteristics of the database can be adjusted for performance reasons.

The detailed reasons for changing these characteristics and the techniques for doing so are beyond the scope of this guide, but they might include:

- Moving tables between tablespaces
- Combining multiple indexes into a single indexspace

- Defining multiple databases
- Clustering a table around an index and partitioning tablespaces

Chapter 5: Designing the Procedure Dialog

Dialog design deals with the movement or flow, between procedures and procedure steps. Screens, dialog, and procedure logic are best designed in parallel. You can then refine the screens and dialogs. Prototyping is an effective technique for refining screens and dialogs. When the initial design is stable, you can complete detailed procedure logic and finalize the screens and dialog flows.

Together, the screen designs, dialog, and procedure logic yield fully detailed displays, a completed dialog flow diagram, and action diagrams that are the basis for generating a completed system.

Prerequisites

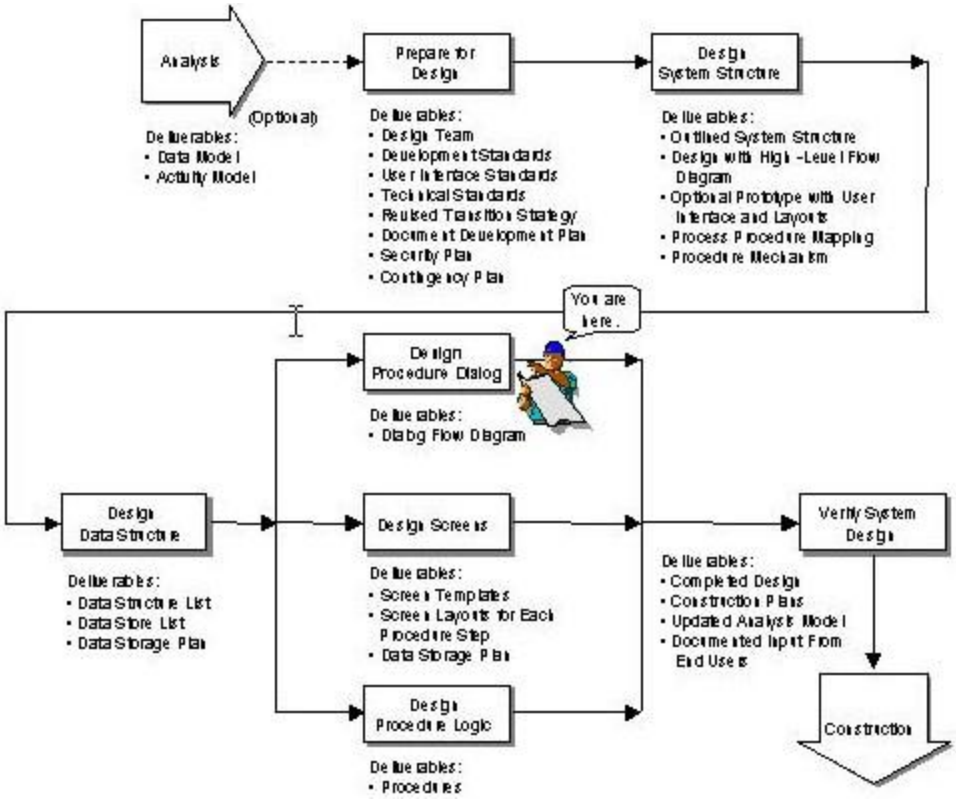
You need an outlined system structure design before beginning procedure interaction design.

More information:

Designing the System Structure (see page 31)

Design Process

The following illustration shows the deliverables from designing the procedure dialog and shows where you are in the overall design process.



Principles of Dialog Design

A dialog describes the movement or flow, between procedures and procedure steps within and between business systems.

A procedure, and therefore all of its procedure steps, can be classified into one of the following categories:

- Online
- Batch

Online Procedures

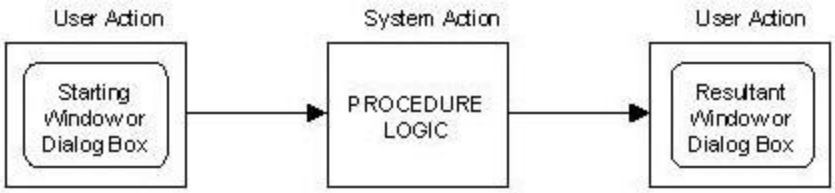
In an online environment, each procedure or procedure step may be associated with a procedure step action diagram and a display. You define the possible series of interactions between a user and the procedures in a business system through these associated displays. The display for block mode will be a screen.

An interaction in a dialog refers to a single instance in which a user requests an action of the system and the system responds.

From the user's perspective, an interaction follows this scenario:

1. The user enters data on a display and indicates that data entry is complete by entering a command.
2. The system acts and responds with another display. The display may be in the same format as the one used earlier to enter data, or it may be in a completely different format.
3. After the system responds, the user may be prompted by the response to begin the cycle again by entering more data, or to select another procedure.

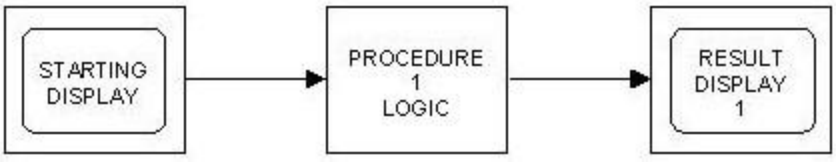
The following illustration shows a very simple online interaction that begins with a user entering data on the starting display.



From the designer's perspective, an interaction involves an indeterminate number of procedure steps. Remember that a procedure step is the unit at which a single screen is defined.

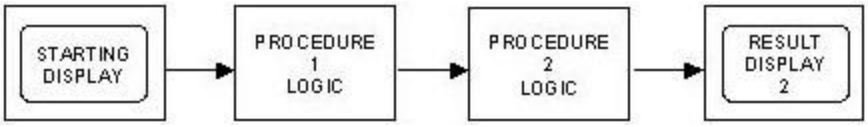
You can design the system action in one of the following ways:

- The system initiates a procedure step to process the data, as shown in the following illustration.



If the user's request can be satisfied without displaying further data, the procedure step can be defined to respond with a display (Result Display 1) in the same format as the starting display on which the user originally entered data. If a different display is needed to satisfy the request, the possibilities depend on the target environment.

- In an online environment, another procedure step is initiated, which can respond with a display (Result Display 2) in a different format from that on which the user originally entered data. This is illustrated in the following illustration.



More information:

Designing Screens (see page 101)

Batch Procedures

In a batch environment, you define the steps of each batch procedure and the flows within each procedure.

The system structure provides a basic outline for the dialog design. At this point you need to detail the basic outline with specific procedure steps and flows.

More information:

Designing the System Structure (see page 31)

CA Gen Commands

A command is a special attribute in system design using CA Gen. It lets you specify how users can influence online procedure execution. Command is defined here because its concept is important to the discussion of procedure definition. Commonly used commands should be standardized across the system.

More information:

Preparing for Design (see page 15)

When to Use Commands

The special attribute Command is used in several places during design. You can place it on a screen, set or interrogate it in a Procedure Action Diagram, and send it along a flow on the Dialog Flow Diagram. A command may have synonyms and be associated with a function key.

Reserved Commands

CA Gen's Dialog Manager (the execution-time component that governs procedure execution) reserves commands for its own use. You should be aware of the commands listed in the following sections and use them appropriately.

CA Gen Reserved Commands

CA Gen contains the following reserved commands:

- HELP—Causes the dialog manager to invoke its HELP exit.
- RESET—Causes the dialog manager to go back to the initial display for the business system.

Commands to Initiate Dialog Flows

Another consideration that has a bearing on dialog design is the use of commands to initiate dialog flows.

When designing dialogs, particularly loosely structured ones, a number of instances arise in which a command is needed merely to initiate a dialog flow. When this is the case, you should adopt some standard. These standards will allow the users to easily distinguish between commands that directly perform "real work" (such as adding a customer, canceling an order, or displaying a report) and commands that indirectly invoke another procedure that performs the work.

An example standard is to prefix with an X all commands that cause a flow. For example, ADD, CANCEL and DISPLAY are appropriate for adding customers, canceling orders, and displaying reports. The commands XADD, XCANCEL and XMENU can mean "transfer to the Add Customer procedure," "transfer to the Cancel Order procedure," and "transfer to the MENU procedure" respectively. This distinction eases maintenance.

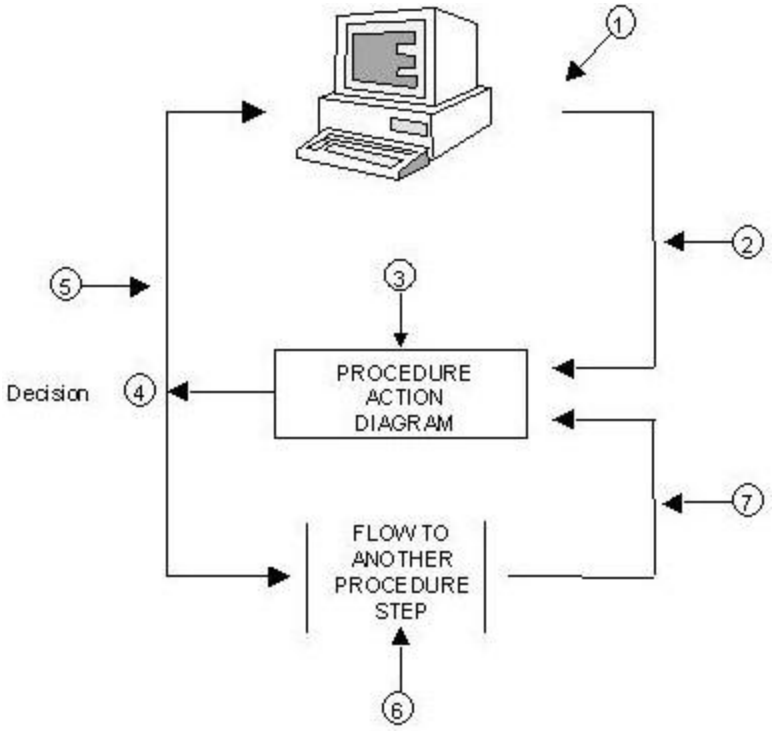
Procedure Step Execution

To understand building dialog flows between procedures, you need to know the relationship between procedure steps, Procedure Action Diagrams, displays, and dialog flows. Each makes its own contribution to implementing a business system.

- Dialog flows provide the means by which one procedure step, based on the results of its execution, can pass control and data to another procedure step.
- A procedure step, whether online or batch, controls the use of Procedure Action Diagrams.
- An online procedure step also controls the use of displays.

- A source procedure step can stop executing at any point in its logic, and so initiate a flow or a display.
- A destination procedure step can execute only from the beginning of its logic; in other terms, a flow is always to the beginning of a procedure step.

The following illustration describes how a display, Procedure Action Diagram, and dialog flow work together during the execution of an online procedure step in the CA Gen environment.



For an explanation of the components of this illustration, see the following table.

Key	Description
1	Procedure step execution usually begins when a user enters data on a display and presses either the Enter key or a function key.
2	The data captured on the display is mapped into the import data view for the procedure step. The chapter “Designing Screens” discusses the connection between fields on the display and a procedure step's data views.

Key	Description
3	The import data view, whether from a display or another procedure step, is processed by the Procedure Action Diagram that supports the procedure step being executed. At its conclusion, the procedure step populates its export data view. For information about procedures, see the chapter "Designing the Procedure Logic."
4	Based on a condition set by the Procedure Action Diagram, the procedure step either shows a display or flows to another procedure step. Such a condition, called an exit state, is described in How Flows Are Initiated.
5	If the decision in key 4 is to show a display, the data from the procedure step's export data view is mapped into the associated display and the display is shown again.
6	If the decision in key 4 is to flow to another procedure step, the export view from the current procedure step is matched to the import view of the procedure step to which the flow will take place. Data view matching is discussed in the chapter "Designing the Procedure Logic."
7	If the dialog flow has the Execute First property, the import data view for the new procedure is passed to its Procedure Action Diagram. Otherwise, the contents of the import data view are placed on the display. The Execute First property is described in Choosing a Flow Action.

In one special case, it is possible to shortcut this cycle by avoiding the execution of the Procedure Action Diagram (step 3), as described in Autoflows.

This description of procedure step execution reveals that the Procedure Action Diagram is essentially independent of the display and dialog flow.

Note:

No explicit actions are required to accept and display displays.

- No explicit actions are required to initiate dialog flows.
- No special logic is required to detect whether the Procedure Action Diagram was initiated from a display or a dialog flow, or whether the result of its execution is to show a display or initiate a dialog flow.

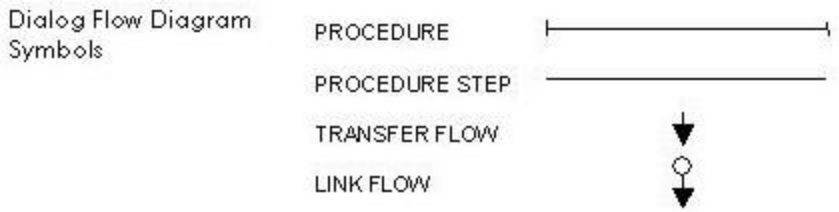
With most database management systems, any requests for update are saved in some type of temporary storage, and the database is not physically updated until a commit point is reached. Each procedure step execution should be seen as a commit unit, sometimes known as a "success unit." When execution finishes, stored data is updated to reflect the create, modify and delete actions initiated during the execution just completed. Where several procedure steps must execute to bring all stored data to a consistent state, you specify procedure step logic to ensure that required data consistency is maintained.

Flows

A flow can be used to communicate between procedures and procedure steps. The CA Gen Dialog Flow Diagram can be used for building flows.

Dialog Flow Diagramming Conventions

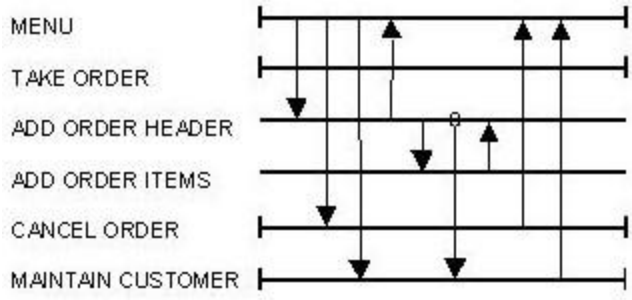
The following illustrations show the conventions for the Dialog Flow Diagram:



The following list details the keys to the illustration:

- The bars represent procedures and procedure steps and are always accompanied by the name of the procedure or step to the left of the bar.
- The symbol for procedure step appears only if a procedure has more than one step.
- If multiple procedure steps support a procedure, they may be contracted on the diagram into a single procedure bar.
- Transfers and links, discussed in the next section, represent flows between procedures and steps. The arrowhead indicates the direction of the flow.
- A dialog flow (that is, a transfer or link) to a multi-step procedure must always flow to the first step in that procedure.

The following illustration shows a CA Gen Dialog Flow Diagram:



The following list details the keys to the illustration:

- MENU is a single-step procedure, so has no procedure steps directly beneath it.
- At execution time, when certain exit state conditions occur as a result of menu selection by the user, explained in How Flows Are Initiated, the MENU procedure can initiate transfers to the procedures TAKE ORDER, CANCEL ORDER, and MAINTAIN CUSTOMER.
- TAKE ORDER is a procedure with two procedure steps: ADD ORDER HEADER and ADD ORDER ITEMS.
- Under certain conditions, the ADD ORDER HEADER procedure step can initiate a transfer to its companion procedure step, ADD ORDER ITEMS. Under other conditions, it can initiate a link to a different procedure, MAINTAIN CUSTOMER.
- Under certain conditions, the ADD ORDER ITEMS procedure step can initiate a transfer to ADD ORDER HEADER.
- Note that no flows enter ADD ORDER ITEMS from outside the TAKE ORDER procedure because it is not the final step of a procedure.
- CANCEL ORDER is a single-step procedure.
- MAINTAIN CUSTOMER is a multi-step procedure with its procedure steps and the flows between them contracted.
- Note the ellipsis (**...**) beneath the procedure label. This indicates that MAINTAIN CUSTOMER has undisplayed procedure steps.
- Two flows enter Maintain Customer. If Maintain Customer were expanded to show its included procedure steps, the flows would be shown as entering the first procedure step of MAINTAIN CUSTOMER, just as the flows in Take Order are shown as entering Add Order Header.

Types of Flows

You can describe the following types of flows with the Dialog Flow Diagram:

- Transfer

- Link

Transfer Flows

A transfer flow indicates that the source procedure step passes control. It can also pass data to the destination procedure step if appropriate conditions are met. In the illustration Dialog Flow Diagram, for example, the transfer from Add Order Header to Add Order Items shows that after Add Order Header completes, Add Order Items is initiated. So, Add Order Header can send data to Add Order Items using the dialog flow.

The following list details the properties of transfers:

- Flows On exit states.
- Flow Actions (Execute First, Display First) for destination procedure step.
- A command triggers execution of the destination procedure step.
- Data is sent from the source to the destination procedure step.

Link Flows

A link flow is more complex than a transfer flow. As with a transfer flow, a link flow indicates that the source procedure step passes control and, optionally, data to the destination procedure step. However, the link flow also allows the destination procedure step to return control and data to the source procedure step. In addition, all export data known to the source procedure step when it completes its original execution is saved and made available after the destination procedure step returns.

For example, the illustration Dialog Flow Diagram shows a link between the procedure steps Add Order Header and Maintain Customer. Assume that this link only takes place when the user takes an order from a customer not yet known to the system. By linking to Maintain Customer, the system lets the user add the customer without losing any of the information already entered into the Add Order Header display. After the user adds the customer, the Add Order Header procedure step begins again with all the information from its original export view available in its import view, augmented by data returned by Maintain Customer.

A link lets a user acquire additional information from another procedure step when the initial procedure step needs it. After the information is acquired, the initial procedure step executes again from the beginning with all the information from its initial execution intact, plus any new data or commands returned from the destination procedure.

If multiple links execute consecutively, all information from each linked procedure step is saved in anticipation of its restart.

For example, suppose that MAINTAIN CUSTOMER flows to another procedure step called CHECK CREDIT STATUS under certain circumstances.

- If TAKE ORDER links to MAINTAIN CUSTOMER, all information required for TAKE ORDER is saved.
- If MAINTAIN CUSTOMER then links to CHECK CREDIT STATUS, all information required by MAINTAIN CUSTOMER is saved.
- When CHECK CREDIT STATUS ends, the data saved for MAINTAIN CUSTOMER is restored, and MAINTAIN CUSTOMER executes again from its beginning.
- When MAINTAIN CUSTOMER ends, the data saved for TAKE ORDER is restored, and TAKE ORDER executes again from its beginning.

If a transfer flow took place at any time during this process, all the information saved during the links would be discarded.

The following list details the properties of link flows:

- Flows On exit states.
- Flow Actions (Execute First, Display First) for the destination procedure step.
- A command triggers execution of the destination procedure step.
- Data sent from the source to the destination procedure step.
- Returns On exit states.
- Flow Actions (Execute First, Display First) for the source procedure step on return.
- A command triggers execution of the source procedure step on return.
- Data is returned from the destination to the source procedure step.

How Flows Are Initiated

Each dialog flow is initiated by the source procedure step setting an appropriate exit state. Each transfer on a Dialog Flow Diagram must be associated with one or more exit state definitions that will trigger the transfer. These are called Flows On Exit States. Each link flow on a Dialog Flow Diagram must have at least one Flows On exit state.

Each link flow must also have at least one Returns On Exit State. After the destination procedure step finishes executing, the Returns On exit state causes control to be returned to the source procedure step.

For a given procedure step, only one flow can be associated with each exit state value.

When the logic of a procedure step concludes, the value of Exit State is evaluated with one of these results:

- If this value matches that associated with any dialog flow from the procedure step, the dialog flow is executed. The flow is executed whether it is a transfer, link, or return from a link.
- If this value does not match any exit state values associated with any dialog flows, you must specify some corrective action.
- Corrective action is appropriate to the procedure implementation.
- In a batch procedure, the particular transaction would have to be aborted or reversed, and the message included as the reason on some reporting file. In extreme cases execution would have to be halted, and the message passed to the initiator of the batch procedure execution. See also Designing for Restart.
- The screen associated with an online procedure step is displayed again with the exit state message appearing in the CA Gen-supplied Error Message field.
- These are the types of message displayed, each of which has been assigned different video properties, as listed in the following table.

Message Type	Description	Color Display	Monochrome Display
None		Normal (the default field color as defined in the video properties)	Normal
Informational	Message gives information only; no action is required.	White High Intensity	High Intensity
Warning	Message advises that user action may be required.	Yellow High Intensity	High Intensity
Error	Message advises that an error has occurred and action is required to correct it.	Red Reverse Video	Reverse Video

Choose a Flow Action

Whenever a flow takes place in either direction, the procedure step the flow initiates will begin in one of the following ways:

- It can show the display associated with the destination procedure step and wait for user input to begin the logic of the procedure step.
- It can execute the logic of the procedure step, which results in either the procedure step's display being shown or a dialog flow being executed.

This distinction relies on the definition of a flow action associated with each procedure step.

The guidelines for determining flow action are quite simple:

- If the destination procedure step requires information from the user before it can execute, use the Display First option.
- If the source procedure step provides all the information required by the destination procedure step, use the Execute First option.

Autoflows

In many cases, the only processing associated with a value of Command is the setting of Exit State to a particular value.

For example, consider the example in Example From MENU Procedure. The MENU procedure does nothing but inspect the incoming command and set an exit state value based on its value in anticipation of flowing to a different procedure step.

In cases like this, it is possible to associate a particular value of Command with a particular value of Exit State. A command associated with an exit state value for a procedure step is called an "autoflow" because you can use it to cause a dialog flow to take place without any action diagram logic to support it.

At execution time, autoflows are processed as follows:

1. The CA Gen Dialog or Window Manager evaluates the Command special attribute. If the value it contains is an autoflow, the execution of the procedure's action diagram is bypassed. Otherwise, the value of Command is passed along to the action diagram as usual.
2. Exit State is set to the exit state value associated with the autoflow.
3. If the value of Exit State is associated with a dialog flow, the flow happens just *as if* Exit State had been set with the EXIT STATE IS action in an action diagram.

In the menu example (see Example From MENU Procedure), it would have been possible to simplify the construction of the Procedure Action Diagram shown in the illustration Procedure Action Diagram for MENU Procedure even further by specifying the autoflows in the following table.

Autoflow Command	Exit State Value
1	TAKE ORDER REQUESTED
2	CANCEL ORDER REQUESTED
3	MAINTAIN CUSTOMER REQUESTED

Autoflows handle all valid exit states. Therefore, the Menu Procedure Action Diagram has to set only the Invalid Request condition as shown in the following illustration. If it ever executes, it means that an invalid value was placed in Command.

Revised MENU Procedure Action Diagram

```
Procedure Step          MENU

MENU
EXIT STATE IS
invalid_request
```

The combination of these autoflow definitions, the Dialog Flow Diagram shown in the illustration Annotated Dialog Flow Diagram in the Example From MENU Procedure, and the action diagram shown in the illustration Revised MENU Procedure Action Diagram give the same execution time result as the MENU Procedure example.

Execute a Command Using a Flow

Most online procedure steps rely on the value in the Command special attribute to direct their processing. When a procedure step initiates from a display, the user selects a pushbutton, enters a value in the Command field, or presses a function key to influence procedure step execution. The same kind of value is required when a procedure step is initiated by a dialog flow from another procedure step.

You can specify a command value for each direction of a dialog flow. When the destination procedure step of the flow executes, its underlying Procedure Action Diagram treats the value exactly as if it were entered on a display. You can choose to send a specific command value, the current value of Command, or, in the case of a return from a link, the previous value of Command.

The following list details the current value and previous value implications:

- The *current* value of command, you will be sending its value at the conclusion of the source procedure step's execution.
- The *previous* value of command, you will be sending the value in the Command field when the procedure step that initiated the link finished executing.

For a transfer, you can specify the command value that triggers the destination procedure step.

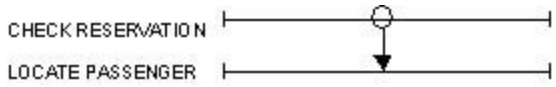
For a link, you can specify both the command value that triggers the destination procedure step and the command value that triggers the return to the source procedure step.

Pass Data Through a Flow

It is possible and often useful to send data between procedure steps along a dialog flow. For example, consider the online Take Order procedure in the illustration Annotated Dialog Flow Diagram in Example From MENU Procedure.

The illustration shows two procedure steps, Add Order Header and Add Order Items. Assume that the Order Number is assigned in the Add Header procedure step. The ability to pass its value from Add Header to Add Lines eliminates the need for the user to retype it when Add Lines starts.

The Dialog Flow Diagram in the following illustration represents a slightly more complex example, Airline Reservation System Fragment.



The Check Reservation online procedure is intended to show the flight on which a particular passenger is booked.

The Locate Passenger procedure, given a passenger name, such as Smith, attempts to find it, and list all names like Smiff, Smithe, or Smythe with which it might be confused.

If Check Reservation fails because it cannot find the passenger's name, Locate Passenger is invoked to resolve a possible misspelling.

In this case, sending the passenger's name along the dialog flow from Check Reservation to Locate Passenger frees the user from having to retype the name when Locate Passenger is invoked.

Likewise, after Locate Passenger identifies the correct passenger name, sending it back to Check Reservation over the link's return executes Check Reservation again without additional user intervention. It is not necessary for the course procedure step to store data values for later use by the destination procedure step.

These examples demonstrate that passing data between procedure steps can improve system flow, simplify dialog interactions, and reduce the chance for error.

Data can pass over both transfers and links from the source of the flow to its destination. Any data in the source procedure step's export data view can be passed to the destination procedure's import data view, as long as the views are compatible.

Links can also return data from the destination of the link to its source. Any data in the destination's export data view can be passed to the source procedure step's import data view, as long as the views are compatible. Remember that during a return from a link, the source procedure step's import view is populated from the export view of its previous execution. However, any data returned from the destination to the source procedure step overlays the corresponding elements of that step's import view.

For example, consider the illustration Airline Reservation System Fragment again. For this example, assume that the following points:

- The import data view for Check Reservation has two components: Passenger Name and Reservation Date.
- The export data view for Check Reservation includes Passenger Name and a list of flights.
- Check Reservation is unable to locate a passenger by the name of John Doe.

When Check Reservation links to Locate Passenger, the following takes place, assuming an Execute First dialog flow:

1. The export view from Check Reservation is saved, including the name John Doe.
2. The appropriate elements of the export view are placed in the import view of Locate Passenger. In this case the import view receives the passenger's name, John Doe.
3. Locate Passenger begins execution, looking for names that are similar to John Doe. In its export view, it returns Shawn Toe, Jon Deaux, and John Dough.

The user may now select from these possibilities to identify the correct passenger. If John Dough is the correct choice, the following takes place when Locate Passenger returns to Check Reservation (assuming that the return has a flow action of Execute First):

1. The import view of Check Reservation is populated from its saved export view. John Doe appears as the passenger name.
2. The appropriate elements of the export view of Locate Passenger are then used to overlay the import view of Check Reservation. John Dough replaces John Doe as the passenger name.
3. Check Reservation executes. This time, the procedure finds the passenger John Dough and displays the list of flights reserved for him.

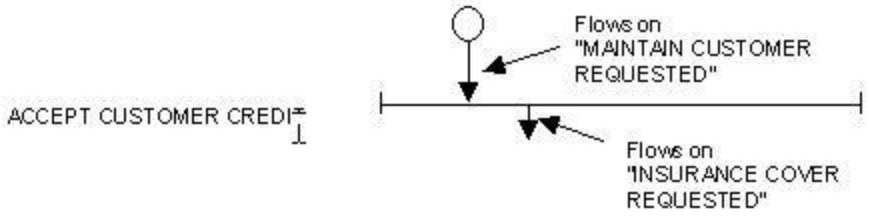
Flows Between CA Gen Business Systems

The subject of a CA Gen Dialog Flow Diagram is a single business system.

When it is necessary to flow between two business systems both specified using CA Gen, you can add external flows and links in or out of the system currently being designed.

The flow is to or from a specific procedure in the other CA Gen-generated system. As with any flow between procedures, the destination must be the first step of a procedure.

Suppose in our example Ordering dialog in the illustration Annotated Dialog Flow Diagram that a high value order needs an extension of customer credit, which is supported by a second Credit Control system. Also suppose that very large credit exposure requires an extension of commercial risks insurance, which is supported by a third Insurance Management system. Then part of the Dialog Flow Diagram for Credit Control would show flows between systems as arrows without a source or destination procedure, as in the following illustration, External Dialog Flows.



Flows Between CA Gen and Non-CA Gen Procedures

It is sometimes desirable to have a flow between online procedures generated by CA Gen and transactions built using some other method. These flows can be represented on the Dialog Flow Diagram.

For online procedures, these flows can be implemented using other dialog design features: Clear Screen Input and the special attribute NEXTTRAN.

Configure Flows to CA Gen Procedures

To flow from a CA Gen-generated transaction to one not generated by CA Gen, the appropriate input message must be set in the NEXTTRAN special attribute. NEXTTRAN is either placed directly on the screen or set in an action diagram.

In either case, whenever the CA Gen dialog manager detects a value in NEXTTRAN, it flows to the transaction in question using the native facilities of the teleprocessing monitor, such as IMS/DC or CICS.

Configure Flows to Non-CA Gen Procedures

To flow to a CA Gen-generated transaction from one not generated by CA Gen, the non-CA Gen-generated transaction is responsible for creating a message to invoke a CA Gen-generated transaction.

Typically, an online procedure uses the facilities of the teleprocessing monitor to transfer control to the CA Gen-generated transaction. The message it creates must be in the same format as that used for Clear Screen Input.

Define Exit States

For each exit state, define the properties described in the following list.

- Value-Set by procedure logic
- Message type-None, Error, Warning, or Informational

  Determines how the message is displayed.
- Message (optional) text-Displayed when the Exit State is recognized when a procedure stops execution
- Termination action-Normal, Rollback, or Abort

  Specifies the action to be taken when the Exit State occurs.

The following list shows examples of meaningful exit state names:

- Customer Was Not Found
- Order Taken Successfully
- Duplicate Item Number

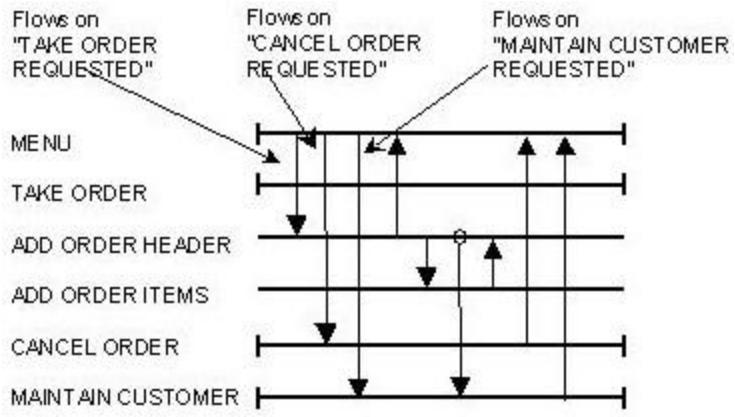
You specify the circumstances in which a procedure step places each possible exit state value in the exit state special attribute by using the EXIT STATE IS action statement. You can also specify when this exit state value is used by invoking the autoflow feature, which is described in Autoflows.

More information:

Designing the Procedure Logic (see page 123)

Example From MENU Procedure

The following illustration, Annotated Dialog Flow Diagram, shows a Dialog Flow Diagram with exit state values for each dialog flow from the MENU procedure.



The following sample code, Procedure Action Diagram for MENU Procedure, shows the Procedure Action Diagram for the MENU procedure.

```
CASE OF COMMAND

     CASE 1
     EXIT STATE IS take_order_requested
     CASE 2
     EXIT STATE IS cancel_order_requested
     CASE 3
     EXIT STATE IS maintain_customer_requested
OTHERWISE
EXIT STATE IS invalid_requested
```

The exit state value and exit state message for each exit state definition are required by the MENU procedure.

The exit state definitions used in the MENU Procedure are shown in the following table.

Exit State Value	Exit State Message	Exit State Termination Action
TAKE ORDER REQUESTED		Normal
CANCEL ORDER REQUESTED		Normal
MAINTAIN CUSTOMER REQUESTED		Normal

Exit State Value	Exit State Message	Exit State Termination Action
INVALID REQUEST	You have made an invalid request. Please make another selection	Normal

The following list shows the possible values of Exit State at the conclusion of the execution of the MENU procedure:

- Take Order Requested—The Take Order procedure will be initiated. (Actually, its first procedure step, Add Header, will be initiated.)
- Cancel Order Requested—The Cancel Order procedure will be initiated.
- Maintain Customer Requested—The Maintain Customer procedure will be initiated.
- Invalid Request—The following message is displayed:

  `You have made an invalid request. Please make another selection.`

  An online procedure will redisplay the menu display along with the error message.

  A batch procedure will halt.

More information:

Designing the Procedure Logic (see page 123)

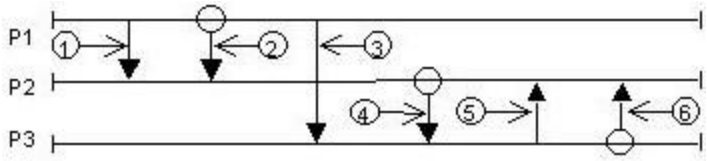
Rules for Assigning Exit State Definitions

When an exit state is set during a procedure step execution, the results must be unambiguous.

The following rules govern the assignment of exit state definitions to dialog flows:

- An exit state value may appear on the Flows On exit state list of only one dialog flow from a given procedure step.
- No exit state value that appears on the Returns On Exit State list of a link to a given procedure step may appear on a Flows On exit state list of dialog flows from the same procedure step.

The following illustration, Exit State Rule Violation, shows violations of both rules using an annotated Dialog Flow Diagram.



① Flows on Exit State 1
② Flows on Exit State 2 and Returns on Exit State 3
③ Flows on Exit State 1
④ Flows on Exit State 1 and Returns on Exit State 3
⑤ Flows on Exit State 3
⑥ Flows on Exit State 4 and Returns on Exit State 3

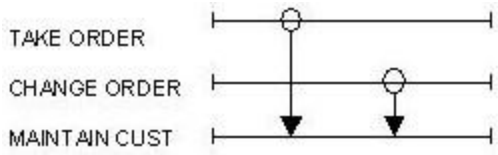
The illustration shows three procedures, P1, P2 and P3, and six dialog flows, each of which has been assigned a number. The Flows On and Returns On Exit States associated with each flow are also listed.

Despite the confused use of exit states in this diagram, there are actually only two violations:

- For procedure P1, flows 1 and 3 violate the first rule because they both use ES1 as a Flows On exit state.
- For procedure P3, flows 5 and 6 violate the second rule because flow 5 uses ES3 as a Flows On exit state and flow 6 uses ES3 as a Returns On exit state.

This discussion reveals an interesting property of Returns On exit states; they depend entirely on the context of a particular link.

The following illustration, Context Sensitivity of Returns on Exit States, shows that both Take Order and Change Order can link to Maintain Customer. (Assume that the Returns On Exit State for both links is Request Successful.)



According to this illustration, there are three possible results if MAINTAIN CUSTOMER sets Request Successful during its execution:

- If MAINTAIN CUSTOMER was linked to from TAKE ORDER, control is returned to TAKE ORDER.

- If MAINTAIN CUSTOMER was linked to from CHANGE ORDER, control is returned to Change Order.
- If MAINTAIN CUSTOMER was not the destination of a link, the Maintain Customer display is redisplayed along with the exit state message associated with the exit state value Request Successful.

The result depends on how MAINTAIN CUSTOMER was initiated.

Design Online Dialogs

When designing online procedures, try to minimize the number of terminal or host interactions to maximize the use that the user can make of each screen displayed.

Besides the description given in the previous section, some aspects of online design need special attention. These include the way commands and function keys are used and the way dialog flows are executed.

Screens for Data Maintenance

You can design one procedure to combine all maintenance functions. On the screen, attributes provide different field protections for the different Add, Change, Delete and Display functions. Whenever detail field changes are needed in the design, only one screen needs to be modified.

Some field level protections that an online application might use are listed in the following table.

Function	Attribute Type	Protection	Comment
Add	Identifier Non-identifier	Unprotected Unprotected	Sometimes the identifier is automatically generated, in which case the identifier would be protected.
Delete	Identifier Non-identifier	Protected Protected	The only enterable field is usually a Y/N to confirm delete.
Update	Identifier Non-identifier	Protected Unprotected	Anything except the identifier can be changed, providing the result of the change still conforms to business and data integrity rules.
Display	Identifier Non-identifier	Unprotected Protected	Only the identifier can be entered to select an occurrence.

Commands

The commands described in the following list receive special attention from the CA Gen Dialog Manager.

- NEXT—For a procedure step using the automatic scrolling feature, this causes the dialog manager to scroll forward based on the value of the Scroll Amt special attribute.

  If NEXT is requested after the last occurrence of a repeating group has been displayed, the NEXT action is passed to the Procedure Action Diagram.

  For a procedure step not using the automatic scrolling feature, NEXT has no special meaning.

- PREV—For a procedure step using the automatic scrolling feature, this causes the dialog manager to scroll backward based on the value of the Scroll Amt special attribute.

  If PREV is requested after the first occurrence of a repeating group has been displayed, the PREV action is passed to the Procedure Action Diagram.

  For a procedure step not using the automatic scrolling feature, PREV has no special meaning.

- TOP—For a procedure step using the automatic scrolling feature, causes the dialog manager to scroll to the first item of the repeating group.

  For a procedure step not using the automatic scrolling feature, TOP has no special meaning.

- BOTTOM—For a procedure step using the automatic scrolling feature, causes the dialog manager to scroll to the last item of the repeating group.

  For a procedure step not using the automatic scrolling feature, BOTTOM has no special meaning.

- RESTART—Causes the dialog manager to redisplay the last display the user was shown for the business system.

More information:

Designing the Procedure Logic (see page 123)
Designing Screens (see page 101)

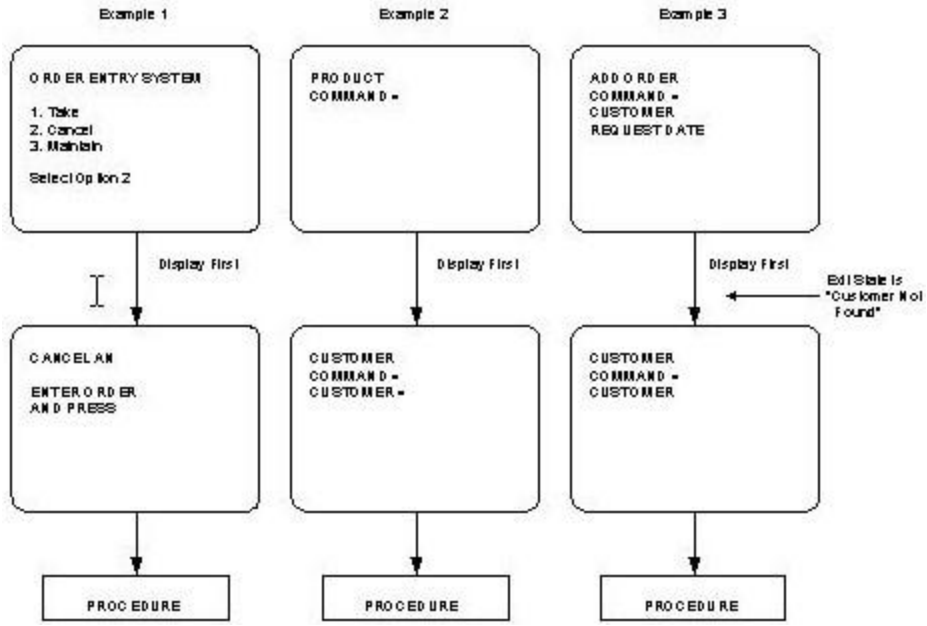
Function Keys

The chapter “Preparing for Design” covers the use of standard function keys.

All commands for a host procedure step should be accessible by function key, provided that the function key standards are not violated, and that the keyboards in use support enough function keys.

Function keys to navigate online procedures are more convenient for the frequent user than entering commands.

The following illustration, Flow Actions for Online Procedures, shows three online examples that call for a Display First flow action.

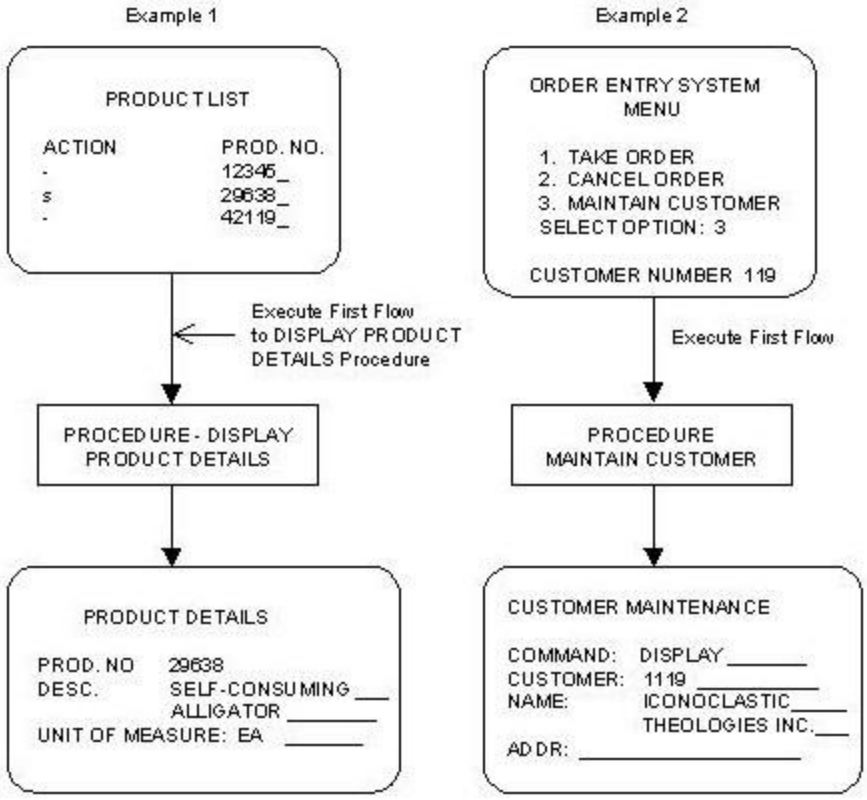


The following list details the keys to the illustration:

- Example 1 shows that Display First is most often used in flows from menus.
- Example 2 shows that flows in loosely structured dialogs between dissimilar procedure steps exist purely for user convenience.
- Example 3 shows that there are occasions when the destination procedure step is required to create entities.

Note: This example, the Command field and Customer Number field are prepopulated with pertinent information, even though Display First was specified.

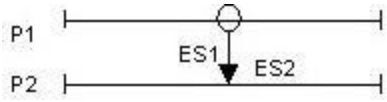
The following illustration, Execute First Flow Actions, shows two examples of dialogs using Execute First flow actions.



Execute First is most often used when the destination procedure step is to present data identified in the source procedure step or in menus that allow the capture of key information relevant to the destination procedure step.

For dialog flows that are transfer flows, a flow action is specified for the destination procedure step. For dialog flows that are link flows, two flow actions are specified-one for the destination procedure step and another for the source procedure step on return.

An interesting aspect of the Execute First flow is that its execution, when followed by any other flow, can cause user interaction with procedure steps to be bypassed. Consider the following illustration.



Assume that the link between procedures P1 and P2 specifies Execute First in both directions, that is, for P2 when the link is initiated and for P1 when the link completes. Also assume that the link flows on ES1 and returns on ES2.

The following scenario is then possible:

1. The user fills in the display for P1 and presses Enter.
2. P1 executes and sets Exit State to ES1 at its conclusion.
3. Based on the value of the Exit State, P1 links to P2.
4. Because the flow action is Execute First, P2 executes. At its conclusion, it sets Exit State to ES2.
5. Based on the value of the Exit State, P2 returns to Pl.
6. Because the flow action for return is Execute First, P1 executes. At its conclusion, it sets Exit State to ES3.
7. No dialog flow is associated with exit state value ES3, so the display for P1 is displayed.

From the user's perspective, the execution of procedure P2 is completely hidden. The display for P1 is used for input and subsequently redisplayed, just as if P2 had never been invoked, although P2 contributed to the successful execution of the procedure.

Sometimes a procedure step does not need to display a display. In this case, it is defined as non-display.

When commands are passed along a dialog flow using a Display First action, the value of Command specified on the flow appears on the display of the destination procedure step. Example 3 in the illustration, Display First Flow Actions, shows this display.

Prototype Online Dialogs

CA Gen includes a prototyping feature that enables you to demonstrate online dialog flows without building action diagrams or generating code or databases.

Define the following items to prototype online dialog flows:

- Procedure steps to be prototyped (except for the action diagram)
- Display for each procedure step
- Dialog flows connecting the procedure steps
- Autoflows and associated exit states for each dialog flow to be prototyped
- Assignments for each command for which an autoflow specified

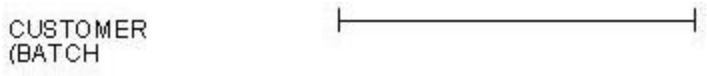
To initiate prototyping, select Prototyping from the Design pull-down menu in the main CA Gen window. The displays are displayed in the same manner as when using the Display command from Display design; the prototype user cannot enter any variable data into the display.

The function keys associated with the autoflows can be used to move from display to display based on dialog flows. This lets you review the behavior of an online conversation without building detailed procedure logic.

Design Batch Dialogs

The design of batch dialogs is also accomplished using the Dialog Flow Diagram. However, a number of features available for online dialog design are not applicable in a batch dialog, so the options available are more limited.

Batch procedures on the Dialog Flow Diagram are annotated BATCH beneath the procedure's label as shown in the following illustration.



Otherwise, the diagramming symbols for batch procedures are the same as those for online procedures. For more information, see Dialog Flow Diagramming Conventions.

Flow Restrictions for Batch Dialogs

The following restrictions apply to dialogs involving batch procedures:

- Flows between procedures are not permitted-It is possible only to create flows between procedure steps in the same batch procedure. Remember that a batch procedure is analogous to a batch job and a batch procedure step is analogous to a batch job step in the MVS world.
- Only flows forward are permitted-Flows forward are represented by arrows pointing downward on the Dialog Flow Diagram. The one exception to this rule is the involuted transfer, which is explained in Establishing Checkpoints.
- Display First has no meaning in a batch procedure-All flows in a batch procedure are processed as Execute First.

Design for Restart

One consideration unique to batch procedures is the need to restart them. A procedure step is restartable when, if it abnormally terminates midway through its execution, it can be restarted from the point it terminated.

Designing for restart has two aspects:

- Establishing checkpoints, which is analogous to leaving a trail of bread crumbs in one's path through the enchanted forest

- Restarting from the last checkpoint, which is something like following the trail of bread crumbs back into the forest after being mysteriously transported back to your initial starting point

Establish Checkpoints

A checkpoint is a point during processing you establish to be certain that everything in the processing environment is synchronized and the procedure can restart.

To establish a checkpoint, it is necessary to commit all outstanding database updates and to take a process snapshot.

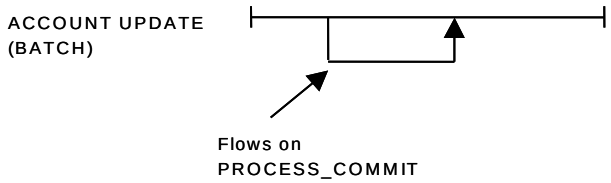
With most database management systems, any requests for update are saved in some type of temporary storage. The physical database is not actually updated until a commit point is reached.

In the absence of an explicit request for a database commit, updates are not committed until the end of a procedure step's execution.

In a typical restartable procedure step, a commit is explicitly requested by the procedure step every so often during processing, usually after a certain number of operations take place. This means that if there is a restart, you lose only the updates made since the last commit.

A database commit can be initiated by calling an external action block which is hand coded to explicitly execute the commit; so using an external action block is a simple and efficient means of achieving a commit.

In a CA Gen-generated system, a database commit can also be requested by using an involuted transfer flow on the Dialog Flow Diagram, as shown in the following illustration.



An involuted transfer flow causes the dialog manager to commit all outstanding database requests and reinvoke the procedure logic for the procedure step.

The illustration shows a single-step batch procedure called ACCOUNT UPDATE. Whenever the execution of the ACCOUNT UPDATE Procedure Action Diagram concludes and the value of exit state is PROCESS COMMIT, the CA Gen dialog manager commits all outstanding updates and returns to the beginning of the ACCOUNT UPDATE procedure.

Process snapshot refers to any information that might be required to return the procedure step to its state at the last commit point if there is a restart. Otherwise, duplicate database updates may result. Enough information to allow the procedure step to return to a commit point must therefore be saved each time a commit is issued.

Restart at the Last Checkpoint

To take advantage of the checkpoints established by conscientiously issuing database commits and taking process snapshots, you should include logic in the procedure step to interpret the snapshot and restore things to the state that existed at the last checkpoint. This usually involves repositioning sequential files and, possibly, resetting the values of some local views.

Chapter 6: Designing Screens

In an online environment, the primary visual layouts are screens; in a batch environment, they are reports. Many of the comments about export views apply to reports as well.

Screens, dialogs, and procedure logic are best designed in parallel. You can then refine the screens and dialogs. Prototyping is an effective technique for refining screens and dialogs. When the initial design is stable, detailed procedure logic can be completed, and the screens and dialog flows can be finalized.

Together, the screen designs, dialog, and procedure logic yield fully detailed displays, a completed dialog flow diagram, and action diagrams that are the basis for generating a completed system.

CA Gen Design Toolset provides a screen design tool that uses definitions of views as a source of the data to be laid out. These views depend in turn on a well-defined data model.

If you wish to prototype dialogs as an aid to developing data and view definitions, you may decide to begin with paper prototypes, or screens produced using word processing or graphic tools. You could also develop the screens in CA Gen using areas of literals to simulate the layout of data.

Prerequisites

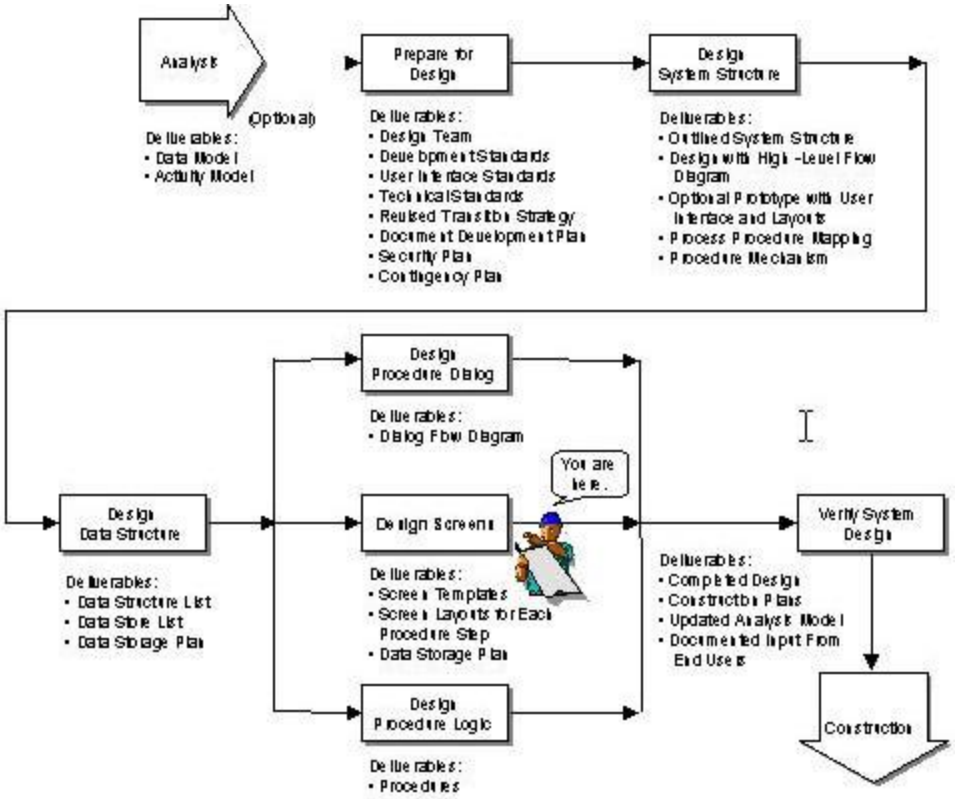
You need an outlined system structure design before beginning screen design.

More information:

Designing the System Structure (see page 31)

Design Process

The following illustration lists the deliverables from designing the application's screens and shows where you are in the overall design process.



Principles of Screen Design

During screen design, you arrange the fields that implement import and export data views and specify explanatory text to support them. An understanding of the user's environment and a set of meaningful standards help to produce good screen designs.

The following list shows critical principles for user-oriented screen design:

- Adopt the user's perspective—The user has to perform a certain task, and the interface has to support this task as closely as possible.

  Different user roles have different requirements. For instance, the frequent user of a system tends to want the greatest amount of information possible on the screen, commands and function keys to streamline navigation, minimal descriptive and help text, and minimal interruption.

  The infrequent user of a system tends to need descriptive headings and labels, longer messages, a low volume of data per screen, and comprehensive help facilities.

  Keep the interface natural. The navigation should feel right to the user.

  Give the user control. The user should be able to go on, cancel or switch to another part of the application.

  Maintain consistency throughout the application. The user will always know what to expect.

- Design for frequent use—Place the most frequently used fields where the user will see them first.

- Take account of physical documents—Arrange fields on data entry screens to closely resemble any corresponding planned or current source document. In the case of a planned document, the document and the layout designs may influence each other.

- Take account of workflow within the layout—Arrange fields in the layout, and in any new document designs, according to the sequence in which users access the information.

  Segregate data functionally. All the information the user needs to complete a particular activity should be on one display.

- Group related objects together—Do not leave large spaces between unrelated objects; this forces the user's eye to travel too far.

- Keep the layout simple—A cluttered and busy display distracts from the task at hand. Balance this against the inconvenience of users having to go through multiple displays to complete a user task.

  Arrange fields on a screen in priority sequence, top-to-bottom, left-to-right. Linguistic differences may change this priority sequence.

  Set standards for colors, highlighting, and general format. This ensures that the look and feel of all components of the system is the same.

By the time screen design is finalized, you should already have defined the data views of the procedure steps involved.

More information:

Designing the System Structure (see page 31)
Preparing for Design (see page 15)

Screens and Templates Design

The following main types of objects are defined during screen design:

- Screens
- Screen templates

Screens

Screens embody the user's view of the system. Each online procedure step is associated with a screen.

A screen can contain fields, special fields, prompts and literals. It can also include one or more screen templates.

Besides providing anchor points for fields, prompts, literals, and special fields, screens have some properties of their own. Since each screen describes exactly one online procedure step, some screen properties are borrowed from the procedure step definition. For instance, screens do not have names of their own. They are identified by the name of the procedure step they support. They do not have their own descriptions, but assume the same description as the procedure step.

Only the properties shown in the following list are unique to a screen:

- Scrolling—This property, which may have either the value yes or no, indicates whether the user can scroll back and forth through a repeating group appearing on the screen using CA Gen's automatic scrolling feature.
- Help identifier—Online systems should be supported by help data.

  Many organizations have systems in place that support the capture and display of help text.

  The Help ID is used to tell the installation help system where to locate help text for the screen.
- Protection of unused occurrences—For screens that support repeating groups, this property is used to specify whether occurrences of the repeating group on the screen that were not populated in the procedure step's export view should be automatically protected.

- Positioning on a Dialog Flow Return—For screens that support repeating groups, this property is used to specify the occurrence of the repeating group that will be displayed at the top of the group. This happens after a return from a link type of dialog flow. There are two options:
  - Restore the position of the group to its position when the link was requested. For instance, if the 10th occurrence of the repeating group was displayed on the top of the screen when the link was requested, it will still be on the top of the screen after the corresponding return.
  - Reset the position of the group so that the first repeating group occurrence is displayed on the first line of the repeating group on the screen. For instance, if the tenth occurrence of the repeating group was displayed on the top of the screen when the link was requested, the first occurrence will be displayed on the top of the screen after the corresponding return. In practice, the second option is rarely used.

More information:

Repeating Groups and Automatic Scrolling (see page 115)
Preparing for Design (see page 15)

Templates

A screen template is a named pattern of fields, prompts, and literals that can be used to build many screens. The template is the layout of a screen.

Fields, which implement import and export data views, are specific to a particular procedure step. Because templates can be shared by many screens, and therefore many procedure steps, they cannot support fields.

Templates have only the properties listed:

- Name-Used to reference the template when it is included on a screen
- Description-a place to record in detail the intended use of the template

Screen and Template Components

The following list shows basic components of screens and screen templates:

- Fields
- Literals

  (For constant information)

- Prompts

  (Labels for fields and special fields)

- Special fields

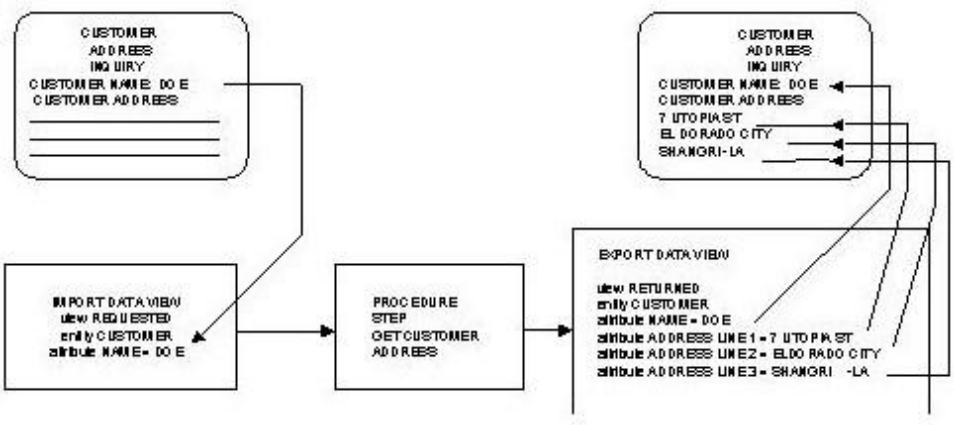
  (The set of special attributes that are appropriate for placement on a screen)

Define Fields

In screen design, a field implements an attribute for which an export data view, and optionally an import data view, has been defined for the procedure step. The attribute may belong either to an entity type or it may be a work attribute.

You need to understand the relationship between screen fields and import and export data views before addressing the properties of fields on screens.

The single screen associated with each procedure step is responsible for providing data to its import view and displaying data from its export view. You can use each field on the screen to implement both an import and an export view of an attribute. The following illustration shows this point.



The following list details the keys to the illustration:

- The procedure step Customer Address Inquiry requires an import view of CUSTOMER NAME. It produces an export view composed of the following information:
  - Customer name
  - Customer Address Line 1
  - Customer Address Line 2
  - Customer Address Line 3

- The screen for Customer Address Inquiry is shown in two states:

  After the terminal user has specified the data required in the import view

  After the system has formatted the export view in response

- The field labeled Customer Name: provides input to the import view Requested Customer Name before procedure step initiation. The field displays the contents of the export view Returned Customer Name at the procedure step's conclusion.
- The fields identified as Customer Address: only display the contents of the export view's Returned Customer Address Lines 1, 2, and 3.
- In general, all fields for which an import view is defined should have a corresponding export view.

The following list shows the properties of fields:

- Import Data View
- Export Data View
- Display Length
- Hidden Field Indicator
- Edit Pattern
- Field Video Properties
- Error Video Properties
- Field Prompt
- Prompt Video Properties
- Help Identifier

Import Data View

This view receives data input to the field, if any.

Export Data View

This view provides data to the field, if any.

Display Length

This is the number of character positions required to display the field on the screen.

Many times a field's display length will be the same as that of the attribute it implements, but in some cases it will not. The most frequent reason for a difference in display length and attribute length is the use of an edit pattern.

For example, consider an attribute of Employee called Date Hired that contains a 4-digit year, a 2-digit month and a 2-digit day, 8 digits in all. When placing a field representing Date Hired on a screen, you determine that it should be displayed in the format YYYY-MM-DD. The insertion of two additional characters (the dashes) requires a display length of 10 characters. So, the attribute length is 8 while the display length is 10.

In some cases, you may choose to truncate a field value to make it fit on a screen. For instance, an 80-character description may not fit on a densely packed screen. You can squeeze it in by reducing the display length. Avoid this approach wherever possible because of the potential loss of meaning resulting from the partial display.

Number of Decimals

For numeric views, this property indicates the number of decimal places the field will accept or display.

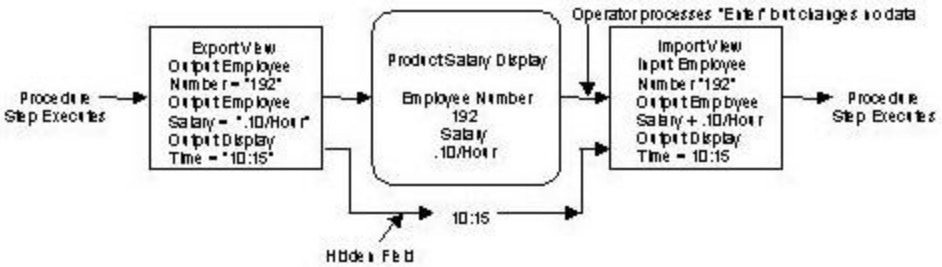
Hidden Field Indicator

In an online system, information sometimes needs to be preserved between procedure step executions. Of course, every field that both comes from an export data view and goes to an import data view can be considered "saved" because it passes between procedure steps without a user retyping it.

The following illustration shows this.

However in some cases, information needs to be preserved but not displayed to the user. You will most often do this by using the hidden field property.

A hidden field is not displayed, but is still passed from export to import view between transaction executions. The following illustration shows an example of a procedure step using a hidden field to pass data between executions.



In this example, a secure procedure step has been designed to require a user password. Whenever the screen is displayed, the Output Display Time is saved as a hidden field. When the user presses Enter to start the next procedure step execution, the value of Output Display Time saved in the hidden field is compared with the current time. If the user lets too much time to elapse, the procedure step can prompt for a password.

Traditionally, hidden fields are implemented as dark, protected fields on the screen.

At execution time, the CA Gen-generated system automatically saves hidden fields and restores them without requiring space on the screen.

CA Gen imposes the following requirements:

- Any field identified as a hidden field must be associated with both an import and an export data view. Otherwise it cannot pass data between procedure step executions.
- Each component of a procedure step's import and export data views must either be implemented by a field placed on the screen or as a hidden field.

More information:

Field Video Properties (see page 110)

Edit Pattern

Edit patterns make the values in views more readable when they appear on a screen. For example, a United States Social Security number is nine digits long. If the contents of an export view of Social Security Number were displayed, it might appear as shown next:

888888888

However, by applying the edit pattern XXX-XX-XXXX before displaying it, the result is shown next:

888-88-8888

which is the format in which Social Security numbers commonly appear.

There are two categories of edit patterns:

- Standard edit patterns
- Local edit patterns

You should name edit patterns for the domain of the data they represent. For instance, to ensure that all currency values are displayed consistently on every screen that uses them, one might establish a standard edit pattern called Currency.

One striking advantage of using standard edit patterns is how easily you can change the appearance of fields across the system. For example, imagine that a standard edit pattern called Date has the value MM/DD/YY. Assume that all fields implementing attributes of type Date have been assigned the edit pattern Date. If you later wish to change the format to DD.MM.YY, it is possible to change every field in the system that uses the Date edit pattern simply by changing the edit pattern definition.

Local edit patterns are specific to a particular field. Designers should avoid them in favor of standard edit patterns wherever possible, but they are available when required.

The characteristics of edit patterns vary by the attribute type of the view implemented by the field. For instance, YYYY-MM-DD is valid for a field implementing a date attribute but not for one implementing a time attribute.

For a complete definition of the edit patterns supported by CA Gen, see the *Toolset Help*.

More information:

Preparing for Design (see page 15)

Field Video Properties

Video properties control a number of terminal-dependent aspects of field display.

Video properties should be standardized across the system. However, in certain cases, you may need to deviate from the standard by specifying local video properties.

You should avoid using local video properties to specify intensity, color or highlighting unless absolutely necessary, since it represents a deviation from the system standard.

The following lists details the video properties, either standard or local, that can be defined for a field.

- Cursor Position—Indicates whether the cursor is to be placed in the field when the screen is displayed.

  The cursor identifies where the next character entered will appear on the screen.

  The possible values of this property are yes and no.

  If no fields on a particular screen have a value of yes specified for this property, the cursor will appear in the first field on the screen.

  If multiple fields have a value of yes specified, the cursor will appear in the first one on the screen (beginning in the upper left corner and working left-to-right through each row on the screen).
- Protection—Indicates whether data can be entered into a field.

- Intensity—On a terminal that supports different intensities, lets fields appear at different levels of brightness (high intensity, normal intensity, or dark, which does not appear at all).
- Color—Fields appear in any of the colors supported by the terminal.
- Highlight—Fields are displayed with any highlighting characteristics supported by the terminal, such as reverse video or blink.
- Justification—Controls the alignment of data at either the right or the left of the field. Normally, text is left-justified and numbers are right-justified.
- Blank when zero—Causes a numeric field to be displayed as blank, regardless of its edit pattern, if its value is zero.
- Fill character—Is inserted in remaining character positions after justification. For example, a value of 999 right-justified in a field five digits long with a fill character of 0 would appear as 00999.

More information:

Preparing for Design (see page 15)

Error Video Properties

Error video properties are similar to field video properties except that they are used to highlight a field in error. For instance, on a data entry screen, data that does not conform to an attribute's permitted value list can be redisplayed to the user in reverse video to highlight the error.

You can specify that a field be displayed using its error video properties by using the MAKE ... ERROR action statement.

The list of error video properties is a subset of the list of field video properties that includes cursor position, protection, intensity, color and highlight. As with field video properties, it is always best to default to the system standard. Use of a local error video property is likely to confuse a user accustomed to reacting to the standard error video property.

More information:

Designing the Procedure Dialog (see page 71)

Field Prompt

A prompt is a label for either a field or a special field. It appears on the screen near the field itself.

In the following example, Customer Name and Hair Color are labels, or prompts, for the fields with which they are associated.

```
                    Customer Hair Color Inquiry

                    Customer Name = John Doe_______

                    Hair Color    = Brown__________
```

CA Gen keeps a list of each prompt used with fields to implement each attribute across the system, and lets you choose among them. This improves the opportunity to standardize field labels across the system. You can use literals to create the same effect as prompts. This chapter will take up the topic of literals.

For example, in the example Field Prompts, assume the field preceded by the prompt Customer Name: implements an attribute called Customer Name.

Suppose another designer builds a screen that requires a field implementing Customer Name. When the designer is positioning the field, CA Gen will issue a reminder that Customer Name was previously used as a prompt for the attribute and will encourage the designer to use it again rather than inventing a new one such as Name of Cust Is or Client Name Is ===>>.

Prompt Video Properties

These properties allow you to specify terminal-dependent characteristics of the prompt display.

As with field properties, you should establish a standard for prompt properties and override it only in extreme situations.

Since prompts contain no variable data, the only video properties available for them are intensity, color, and highlight.

Help Identifier

The value of this property can be communicated to an installation's Help Facility if it supports field-level help.

If you are experienced in building screens using other techniques, you may notice that no mention is made here of the Modified Data Tag (MDT) used on IBM Model 3270 and compatible terminals to indicate that a field on the screen has been changed. CA Gen deals with all Modified Data Tag handling when generating the system. You should assume that all data exported from the procedure step that is not modified will be available in the import view of the next execution. This is assuming all Modified Data Tags had been set.

Define Literals

Literals contain only constant data. As a result, literals have only two components:

- Constant data to be displayed
- Video property of the constant data

Where possible, use the system standard video properties and avoid literal-specific local video properties.

You may specify the following video properties for a literal:

- Intensity
- Color
- Highlight

More information:

Define Special Fields (see page 113)

Define Prompts

Although prompts appear separately on the screen, each prompt is intimately connected with a field.

More information:

Define Special Fields (see page 113)

Define Special Fields

Special fields implement special attributes that are appropriate for placement on a screen.

The following lists details the special fields that are available when using the Screen Design tool:

- Current Date—The system date at the time the procedure associated with the screen is executed.
- Current Time—The system time at which the procedure associated with the screen is executed.

- Transaction Code—The field required by transaction-oriented teleprocessing monitors (such as IMS and CICS) to identify the programs required to execute the procedure step.

  CA Gen's screen generation, a construction facility, requires that Transaction Code appear as the first item (that is, in the upper-left) on each screen.

- Command Area—Where a terminal user may specify a value for the Command special attribute.

  System Error Message—Where messages associated with exit states are displayed.

- Program Function Keys—Where a list of the function keys available for the procedure step can be displayed on a single line as a reminder to the user.

- Terminal ID—Displays the system terminal identification of the terminal on which the screen is displayed.

- User ID—Displays the User ID by which the terminal user is known to the teleprocessing monitor.

- Printer ID—lets a user specify the system terminal identification of a printer on which the screen will print, should the user request it.

- Local System ID—Displays the identity given to the processor and possibly the service under which the procedure step is executing (for instance, the kind of teleprocessing monitor, for example, IMS, CICS, TSO).

- Panel ID—Displays the mapname associated with the screen. Mapnames are assigned during construction. For more information, see the *Toolset Help*.

- Scroll Indicator Message—Used for automatically scrolled repeating groups. If there are more items in the repeating group than appear on the screen, it displays as MORE: followed by a minus sign ("-") to indicate there are items preceding the first occurrence displayed, a plus sign ("+") to indicate that there are items following the last occurrence displayed, or both.

- Scroll Amount Message—Prompts the user to specify the scrolling interval desired. Options are:

  - CURS

    (for cursor, or single line scrolling)

  - HALF

  - PAGE

- Scroll Location Message—Used for automatically scrolled repeating groups. It displays a message in the form:
  LINES *aaa* to *bbb* of *ccc*

  where:

  - *aaa* is the occurrence number, within the repeating group view, of the first occurrence displayed on the screen.
  - *bbb* is the occurrence number of the last occurrence displayed on the screen.
  - *ccc* is the occurrence number of the last populated occurrence in the repeating group view.
- Next Transaction—Where the terminal user can specify a transaction code and formatted input to invoke a transaction. This is usually one not generated by CA Gen. The information placed in this field is processed as though the user had cleared the screen and entered Clear Screen Input to invoke the transaction.

The properties of special fields are the same as for fields, with the following exceptions:

- Do not require import or export data views.
- May not be implemented as hidden fields.
- May not have edit patterns.

Repeating Groups and Automatic Scrolling

Some procedure steps deal with repeating import or export data views. For example, consider a procedure step called List Customers, which provides a list of customer names and phone numbers beginning with a certain value.

Such a procedure step might have import and export views that look like those in the following sample code.

Repeating Export Data View

```
Procedure Step LIST_CUSTOMER
   LIST_CUSTOMERS
      IMPORTS
            Entity ViewCustomer (mandatory, persistem
                  Attributes
                        Name
      EXPORTS
            Group ViewListed
                  Cardinality MIN: 0  Max: 100  Avg: 50
            Entity ViewCustomer (transitive, export only)
                  Attributes
                        Name
                        Phone_Number
```

Note: The group view Listed is a repeating group view, as indicated by its cardinality. The maximum cardinality, shown as 100, indicates that at most 100 values for Customer Name and Phone_Number will be returned as the result of executing the procedure step List Customers.

The implementation of such a procedure step in an online environment will require a screen similar to the following screens.

Screen Containing a Repeating Group 1

```
                          List Customer
   Starting Custmer Name ===> XXXXXXXXXXXXXXXXXX

   Customer Name           Phone Number
   XXXXXXXXXXXXXXXXXXXX  (XXX) XXX-XXXX
   XXXXXXXXXXXXXXXXXXXX  (XXX) XXX-XXXX
   XXXXXXXXXXXXXXXXXXXX  (XXX) XXX-XXXX
   XXXXXXXXXXXXXXXXXXXX  (XXX) XXX-XXXX
   XXXXXXXXXXXXXXXXXXXX  (XXX) XXX-XXXX
   XXXXXXXXXXXXXXXXXXXX  (XXX) XXX-XXXX
   XXXXXXXXXXXXXXXXXXXX  (XXX) XXX-XXXX
```

Screen Containing a Repeating Group 2

```
                          List Customer
Starting Custmer Name ===> XXXXXXXXXXXXXXXXXXXX

Customer Name            Phone Number
XXXXXXXXXXXXXXXXXXXXXX  (XXX) XXX-XXXX (XXX) XXX-XXXX (XXX) XXX-XXXX
XXXXXXXXXXXXXXXXXXXXXX  (XXX) XXX-XXXX (XXX) XXX-XXXX (XXX) XXX-XXXX
XXXXXXXXXXXXXXXXXXXXXX  (XXX) XXX-XXXX (XXX) XXX-XXXX (XXX) XXX-XXXX
XXXXXXXXXXXXXXXXXXXXXX  (XXX) XXX-XXXX (XXX) XXX-XXXX (XXX) XXX-XXXX
XXXXXXXXXXXXXXXXXXXXXX  (XXX) XXX-XXXX (XXX) XXX-XXXX (XXX) XXX-XXXX
XXXXXXXXXXXXXXXXXXXXXX  (XXX) XXX-XXXX (XXX) XXX-XXXX (XXX) XXX-XXXX
XXXXXXXXXXXXXXXXXXXXXX  (XXX) XXX-XXXX (XXX) XXX-XXXX (XXX) XXX-XXXX
```

The multiple occurrences of Customer Name and Phone_Number shown on this screen are referred to collectively as a repeating group.

Notice that only ten occurrences of the repeating group appear on the screen, yet the maximum cardinality of the group view is 100. In a case like this, you can use scrolling to provide a window on the repeating group. Through this window, a user can see ten values for Customer Name and Phone_Number at a time.

You implement the scrolling feature using the Scroll Indicator property of a screen. When the CA Gen-generated procedure step executes, it automatically includes support for moving backward and forward ten occurrences at a time through the 100 occurrences of Customer Name and Phone_Number.

You can use automatic scrolling on one repeating group per screen. In the case of nested repeating groups, only the highest level repeating group is scrolled. For example, the view definitions in the following sample code reflect a modified version of List Customers, called List Customers 2, in which each customer can have six phone numbers.

Nested Repeating Group Views

```
Procedure Step LIST_CUSTOMER_2
LIST_CUSTOMER_2
     IMPORTS
         Entity ViewStarting_Customer
               Attributes
                       Name
     EXPORTS
        Group ViewListed
               Cardinality Min: 0  Max: 100  Avg: 50
        Entity ViewReturned_Customer
               Attributes
                       Name
```

The screen for the nested repeating group views is illustrated next.

Screen With a Nested Partial Repeating Group

```
                              List Customer
     Starting Customer Name ===> XXXXXXXXXXXX

     Customer Name         Phone Number
     XXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX  (XXX) XXX-XXXX  (XXX) XXX-XXXX
     XXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX  (XXX) XXX-XXXX  (XXX) XXX-XXXX
     XXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX  (XXX) XXX-XXXX  (XXX) XXX-XXXX
     XXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX  (XXX) XXX-XXXX  (XXX) XXX-XXXX
     XXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX  (XXX) XXX-XXXX  (XXX) XXX-XXXX
     XXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX  (XXX) XXX-XXXX  (XXX) XXX-XXXX
     XXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX  (XXX) XXX-XXXX  (XXX) XXX-XXXX
```

Automatic Scrolling Command Values

If the Scroll Indicator property for a screen is set to Yes, the following values have special meaning to the Dialog Manager when placed in the Command field at execution time:

- NEXT
- PREV
- TOP
- BOTTOM

The Screen With a Nested Partial Repeating Group shows one way to represent this data on a screen. In this case, the screen shows ten out of a possible 100 customers, and for each customer, three out of a possible six phone numbers. Automatic scrolling can not be used both to scroll through the values of Customer Name and the values of Customer Phone_Number. It becomes the designer's responsibility to provide the scrolling capability with procedure action logic.

More information:

Designing the Procedure Dialog (see page 71)

Special Fields for Automatic Scrolling

The special fields Scroll Location Message, Scroll Indicator Message and Scroll Amount Message, explained in Defining Special Fields, are automatically maintained during execution for screens with a Scroll Indicator property value of Yes.

When to Use Automatic Scrolling

Depending on the specific circumstances, automatic scrolling might not be the best approach. In some cases it is more straightforward, both for the user and you, to handle scrolling explicitly in the procedure logic.

Automatic scrolling is most effective when the following conditions are met:

- A single, non-nested repeating group is to be scrolled.
- The number of occurrences that satisfy the user's selection criteria is likely to be so small that all occurrences will fit in the repeating group view.

For example, consider a procedure step in which an ORDER and the ORDER_ITEMs it contains are displayed. If you are confident that there will never be over one hundred ORDER_ITEMs per ORDER, automatic scrolling is an excellent alternative. You should still include logic to handle the case where there are more than one hundred ORDER_ITEMs for a particular ORDER.

You select automatic scrolling by specifying the Scroll Indicator property for the screen to Yes.

When to Use Logic for Scrolling

Consider a procedure step that displays a list of employees in alphabetical order starting with a particular employee name. The number of occurrences to be returned is indeterminate; depending on the starting employee name and the size of the organization, the number of occurrences satisfying the selection criteria might range anywhere from zero to tens of thousands. In such cases, explicit scrolling logic yields a more efficient and effective design.

More information:

Designing the Procedure Logic (see page 123)

Line Item Actions

If a repeating group implements both an import and an export repeating data view, you may need to allow the user to specify the activity to be performed on each occurrence of the group.

You can accomplish this by specifying a line item action using a work attribute set.

For example, consider a procedure step similar to List Customers that lets a user change or delete a customer's phone number. The following sample code lists the import and export views for this procedure step, named Maintain Customer Phone Numbers.

Data Views for Maintain Customer Phone Numbers

```
Procedure Step  MAINTAIN_CUSTOMER_LIST
   Import View
          ViewSTARTING of entity CUSTOMER
                Attributes:
                              NAME
          Group ViewLISTED_FOR_INPUT
                   Cardinality   Min: 0  Max: 100  Avg: 50
          ViewINBOUND of work group DESIGNER_ADDED
                 Attributes:
                              ACTION_CODE
          ViewINBOUND of entity CUSTOMER
                 Attributes:
                              NAME
                              PHONE_NUMBER
   Export Views
          Group ViewLISTED_FOR_INPUT
                   Cardinality   Min: 0  Max: 100  Avg: 50
          ViewOUTBOUND of work group DESIGNER_ADDED
                 Attributes:
                              NAME
                              PHONE_NUMBER
```

Note: The appearance of the work attribute Action Code in the repeating group view Listed. You can easily add logic in the Procedure Action Diagram to respond to the value of this work attribute.

The following sample code shows a screen that can be used to represent the data views from the sample code Data Views for Maintain Customer Phone Numbers.

Screen for Maintain Customer Phone Numbers

```
                     Customer Phone Number Maintenance
     Starting Customer Name ===> XXXXXXXXXXXXXXXX

     Action       Customer Name            Phone Number
       X          XXXXXXXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX
       X          XXXXXXXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX
       X          XXXXXXXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX
       X          XXXXXXXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX
       X          XXXXXXXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX
       X          XXXXXXXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX
       X          XXXXXXXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX
       X          XXXXXXXXXXXXXXXXXXXXXXXXX (XXX) XXX-XXXX
```

More information:

Preparing for Design (see page 15)

Clear Screen Input

In general, import views of online procedure steps are populated from screens. However, in many cases it is desirable to provide a shortcut initial entry into a procedure by using Clear Screen Input. To illustrate, consider the Screen Containing a Repeating Group.

Ordinarily, a terminal user would need to complete the following steps in order to invoke the List Customers procedure at execution time:

1. The transaction code, assigned during Construction, is entered on the screen and the user presses Enter.
2. The screen associated with List Customers is displayed with no values in its fields.
3. The user keys in a value in the starting customer name field and presses Enter.
4. The screen associated with List Customers is displayed again. This time it contains customer names and phone numbers.

However, it would be more convenient for frequent users of List Customers to be able to enter the Trancode, followed by a value for Starting Customer Name on a clear screen. The screen associated with List Customers is displayed filled in with customer names and phone numbers. This is achieved using the Clear Screen Input feature.

The Clear Screen Input feature also allows the population of a procedure's import views if it is “flowed to” from a non-CA Gen generated transaction. By using the native facilities of the teleprocessing monitor, a non-CA Gen-generated transaction can send a message invoking a CA Gen-generated transaction, with parameters that map to the procedure step's import views.

You can only specify Clear Screen Input for the first procedure step in a procedure. A complete description of the specification of the Clear Screen Input feature is available in the *Toolset Help*.

Chapter 7: Designing the Procedure Logic

Procedure logic design involves taking the outlined system structure and fully detailing its procedure components. The techniques of detailing a procedure and defining procedure logic use the Action Diagramming tool.

The term Procedure Action Diagram refers to both Procedure Action Diagrams and Procedure Step Action Diagrams. For complete information about Action Diagramming, see the *Toolset Help*.

Note: You will need a working knowledge of analysis techniques. CA provides other guides that describe the tasks, tools and techniques to be used, in particular the *Analysis Guide*.

You should design the screens, dialog, and procedure logic in parallel. You can then refine your screens and dialogs, perhaps by prototyping. When the initial design is stable, you can complete detailed procedure logic and finalize the screens and dialog flows.

Together, the screen designs, dialog, and procedure logic yield fully detailed displays, a completed dialog flow diagram, and action diagrams that are the basis for generating a completed system.

Prerequisites

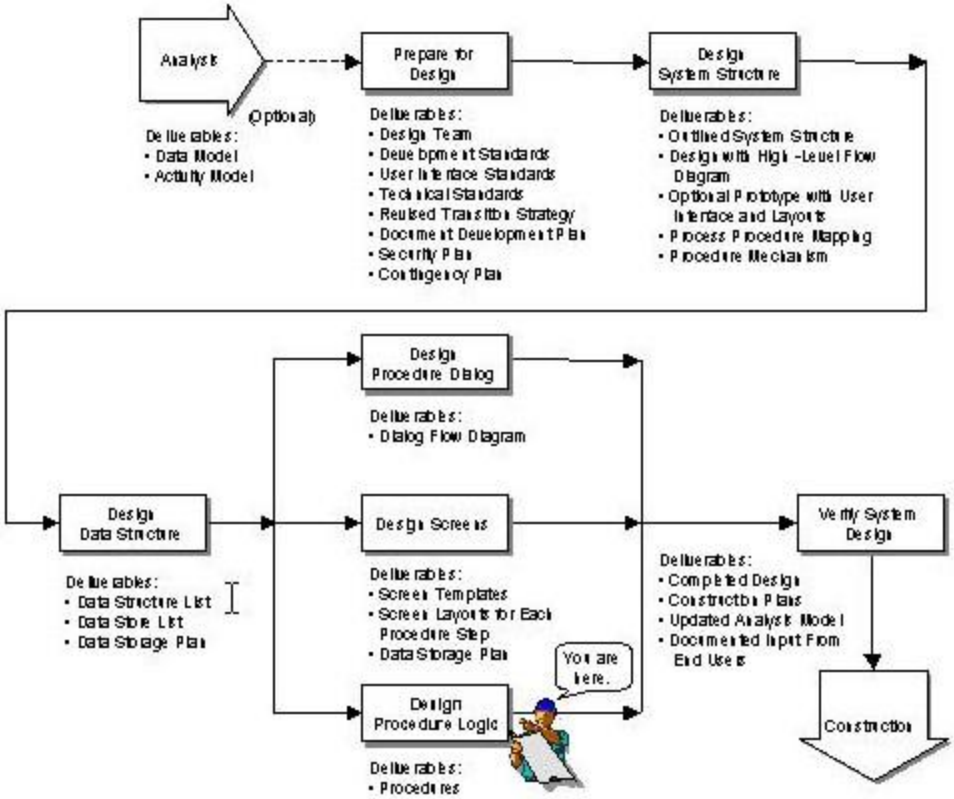
You need an outlined system structure design before beginning procedure logic design.

More information:

Designing the System Structure (see page 31)

Design Process

The following illustration shows the deliverables from procedure logic design and shows where you are in the overall design process.



Types of Procedures

In analysis, processes describe business requirements. During design, you describe the method by which each elementary process is carried out in procedures.

Note: For more information about the elementary processes, see the *Analysis Guide*.

The important point to recall for design is that elementary processes are the objects for which Process Action Diagrams are built.

All activities are described in procedures and components of procedures.

A procedure is a precise set of instructions that facilitates the completion of one or more specific processes.

All procedures support the execution of elementary processes in some way. However, you may consider them in two broad categories based on their level of support:

- Process-implementing procedures-These procedures directly implement elementary processes defined during analysis. For example, the designer may directly implement an elementary process that describes the adding of an order using the online procedure Take Order. Process-implementing procedures generally USE process action blocks.
- Designer-added procedures-These procedures are implementation-specific activities that improve or simplify some characteristics of the overall implementation. For instance, you might add a menu procedure to allow users to select between multiple process-implementing procedures.

Some designer-added procedures are specified during design. However, the majority of procedures defined are process-implementing procedures.

Another way to categorize procedures held as a *property* in the CA Gen design model:

- Online procedures with display—These procedures execute (directly or indirectly) through one or more display interactions with the user.
- Online procedures without display—These procedures execute indirectly without interaction with the user.
- Batch procedures—These procedures execute under the control of the operating system with no interaction with the user, for instance, as a batch job under JES in an MVS environment.

  CA Gen provides 100 percent code generation for most batch processing needs. It supports, but does not generate code for, sequential file processing or report creation.

Note: Sequential file access can be accomplished through external action blocks, and the reports are generated either through a user-written program, a separate reporting package.

Action Diagram Terminology

Before beginning to use action statements that describe the detailed process or procedure logic, you should become familiar with some terms used in action diagramming.

These terms are defined in the following list:

- Process Action Diagram—A collection of action statements that directly support an elementary process. These statements represent business logic and may be defined during analysis, or deferred until design.
- Action Block—(Process Action Block) A named collection of action statements not directly associated with an elementary process.

  An action block may define:

  - A derivation algorithm for a derived attribute
  - A default algorithm for a basic or designed attribute
  - A business algorithm used to calculate an attribute value in a SET ... USING action
  - A business algorithm invoked through a use action
  - Logic for design-specific procedures such as security, operational control, data bridging and conversion, and so forth
- Common Action Block—An action block that is used by more than one process or a process action diagram that is transformed as more than one procedure.
- Procedure Action Diagram—A collection of action statements that directly support a procedure step.
- Business Algorithm—An action block that is invoked through a SET ... USING action or through a USE action.
- Action Group—Any bracketed collection of actions in a Process Action Diagram or Process Action Block.

  All statements appearing within a bracket participate in the action group enclosed by the bracket.

Refine Action Blocks Developed During Analysis

CA Gen lets you create action blocks in the Analysis toolset. However, many developers defer creating action blocks until design so all logic can be specified at the same time.

Any action blocks developed during analysis to support elementary processes or as business algorithms are available for implementation during design.

If the analysis-developed action diagram needs to be changed, remember that all changes made by the designer are visible to the analyst. You are modifying the same specification originally created by the analyst.

Consider the following guidelines for action diagram modification:

- The intent of the analyst should not be violated. You should limit modifications to those addressing implementation issues.
- An analyst should make any modifications that reflect a change in the business model. If an analyst cannot make the change, the analyst should at least approve it.

Refine Synthesized Procedure Action Diagrams

CA Gen automatically creates synthesized Procedure Action Diagrams during process-to-procedure transformation. These are action blocks that can be considered a starting point for designing the detailed logic of the procedure.

You may make changes to the Procedure Action Diagram to address these kinds of processing issues:

- Handling termination conditions
- Dynamically modifying video properties
- Providing for restart
- Designing for ease of use

More information:

Procedure Synthesis to Build an Action Diagram (see page 151)

Handle Termination Conditions

You should address all possible termination conditions for the procedure step's synthesized Procedure Action Diagram.

The following sample code is an example of a procedure step's action diagram immediately after transformation.

Note: Exit State is never set, even for the condition of an invalid command.

Procedure Action Diagram for MAINTAIN CUSTOMER

```
MAINTAIN CUSTOMER
          IMPORTS: Entity Viewimport customer
          EXPORTS: Entity Viewexport customer
     CASE OF COMMAND
     CASE delete
     USE delete_customer
          WHICH IMPORTS: Entity Viewimport customer
          WHICH EXPORTS: Entity Viewexport customer
     CASE add
     USE add_new_customer
          WHICH IMPORTS: Entity Viewimport customer
          WHICH EXPORTS: Entity Viewexport customer
     OTHERWISE
```

The following sample code shows the same Procedure Action Diagram after the designer has addressed termination conditions.

Procedure Action Diagram With Exit State Is Actions

```
MAINTAIN CUSTOMER
     IMPORTS: Entity Viewimport customer
     EXPORTS: Entity Viewexport customer
EXIT STATE IS requested_operation_complete
     CASE OF COMMAND
     CASE delete
     USE delete_customer
          WHICH IMPORTS: Entity Viewimport customer
          WHICH EXPORTS: Entity Viewexport customer
     CASE add
     USE add_new_customer
          WHICH IMPORTS: Entity Viewimport customer
          WHICH EXPORTS: Entity Viewexport customer
OTHERWISE
     EXIT STATE IS invalid_command
```

This example raises the following two issues:

- Exit State is set to Requested Operation Complete before any other logic is executed, as is shown in line 4.

  In this example, the designer has assumed that positive conditions are not addressed in the used action blocks, and has therefore initialized Exit State to the positive condition here, to avoid handling exception conditions throughout the remainder of the procedure step logic.

  If Exit State is never set to a different value as the result of an exception, the positive condition will remain set. This design approach may not be safe if you cannot guarantee correct handling of all exception conditions that may arise. For example, can you be sure that all exceptions will be handled correctly after the action blocks are subsequently modified.

  The EXIT STATE IS action added at line 15 is a typical case of setting Exit State for an exception.

- Exit State is *not* set after the USEd action blocks are invoked (lines 7 and 11). Again this is because the designer has assumed that exception conditions are addressed in the action blocks themselves.

It is essential that all action diagrams are compatible with the exception handling of any USEd action blocks, and that exception conditions are handled completely and safely. You should modify all parts of synthesized logic so that they remain compatible with any system components with which they interact.

Dynamically Modifying Video Properties in Online Procedures

Using the MAKE action statement, which modifies video properties, is restricted to Procedure Action Diagrams that directly support procedure steps. Any special handling of video attributes in a process-implementing procedure must take place in the synthesized Procedure Action Diagram.

The procedure logic can inspect the setting of Exit State on a return from a transformed procedure action block to determine whether video properties should be dynamically modified.

An example of this technique appears in the following sample code. The designer has added a condition followed by a make action to highlight the name of a duplicate customer. See lines 14 through 16.

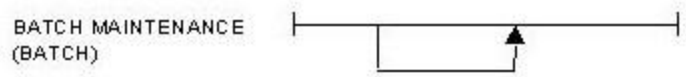
Procedure Action Diagram With a MAKE Statement

```
MAINTAIN_CUSTOMER
          IMPORTS: Entity Viewimport customer
          EXPORTS: Entity Viewexport customer
EXIT STATE IS requested_operation_complete
     CASE OF COMMAND
     CASE delete
     USE delete_customer
          WHICH IMPORTS: Entity Viewimport customer
          WHICH EXPORTS: Entity Viewexport customer
     CASE add
     USE add_new_customer
          WHICH IMPORTS: Entity Viewimport customer
          WHICH EXPORTS: Entity Viewexport customer
          IF EXIT STATE IS EQUAL TO customer already exists
          MAKE export customer name "ERROR"
     EXIT STATE IS invalid_command
```

Restart in Batch Procedures

The following illustration represents the Dialog Flow Diagram for a batch process-implementing procedure named Batch Maintenance. This procedure handles checkpointing and restart at last checkpoint.

Dialog Flow Diagram - Batch Maintenance



A fully commented Procedure Action Diagram for the procedure Batch Maintenance is represented in the following sample code.

Procedure Action Diagram - Batch Maintenance (Part 1)

```
BATCH_MAINTENANCE
     IMPORTS:   Work View  returned_from_transfer work
     EXPORTS:   Work View  sent_through_transfer work
     LOCALS:    Work View  restart work
                Work View  target work
                Work View  returned work
                Work View  returned customer
                Work View  commit ief_supplied
                Work View  message work
EXIT STATE IS requested_operation_complete
NOTE
     ... FIRST, check to see if this is a RESTARTED procedure. If the number
         written to the restart file by WRITE RESTART RECORD in a previous
         execution is greater than zero, it means that BATCH MAINTENANCE
         previously failed in mid-execution.
IF returned from transfer work restart_count IS EQUAL TO 0
USE get_restart_number
     WHICH EXPORTS: Work View target_work
NOTE
     Get RESTART NUMBER is an external action block which reads the restart
     file (written by WRITE RESTART NUMBER at the end of the action block). It
     passes back a value of zero if the restart file is empty.
NOTE
     GET TRANSACTION is an external action block that reads a sequential file
     of CUSTOMERS. It returns CUSTOMER details (in RETURNED CUSTOMER) and a
     request code (in RETURNED WORK).
     Possible values of REQUEST TYPE are:
     - A=Add (CREATE) this customer
     - C=Change (UPDATE) this customer
     - D=Delete this customer
     - E=End of input file reached
USE get_transaction
     WHICH EXPORTS: Entity View returned customer
                    Work View returned work
NOTE
     ... NEXT. If we are restarted, spin through the transaction
         file until you get to the first one that was not processed
         properly the first time.
```

Procedure Action Diagram for Batch Maintenance (Part 2)

```
WHILE restart work restart_count IS LESS THAN target work

     restart count

AND returned work request_type IS NOT EQUAL TO E

SET restart work restart_count TO restart work

     restart_count + 1

USE get transaction

     WHICH EXPORTS: Work View returned work

                    Entity View returned customer

NOTE

     ... NOW, you must be positioned at the first customer transaction not yet
         processed. If this is not a restarted execution, that will be the
         first transaction on the input file; if it IS a restart, it will be
         the transaction right after the last one for which a commit took
         place.

WHILE commit_ief_supplied count IS LESS THAN 100

AND returned work request_type IS NOT EQUAL TO E

EXIT STATE IS requested_operation_complete

SET message work result TO OPERATION COMPLETED

NOTE

     CREATE CUSTOMER, UPDATE CUSTOMER, and DELETE CUSTOMER are Elementary
     Processes for which Process Action Diagrams were created during BAA.

CASE OF returned work request_type
CASE A

USE create_customer

     WHICH IMPORTS: Entity View returned customer
          IF EXIT STATE IS NOT EQUAL TO
               requested_operation_complete
          SET message work result TO NOT ADDED

CASE C

USE update customer

     WHICH IMPORTS: Entity View returned customer
```

```
          IF EXIT STATE IS NOT EQUAL TO

               requested_operation_complete
          SET message work result TO NOT CHANGED
```

Procedure Action Diagram for Batch Maintenance (Part 3)

```
CASE D
USE delete customer
     WHICH IMPORTS: Entity View returned customer
          IF EXIT STATE IS NOT EQUAL TO
               requested_operation_complete
          SET message work result TO "NOT DELETED"
NOTE
     WRITE MESSAGE is an external action block that writes a result message to
     a message file it forms the message based on REQUEST TYPE (in RETURNED
     WORK), the details in RETURNED CUSTOMER and RESULT (in MESSAGE WORK).
USE write message
     WHICH IMPORTS: Entity View returned customer
                    Work View message work
                    Work View returned work

SET commit_ief_supplied count TO commit_ief_supplied_count +1
USE get_transaction
     WHICH EXPORTS: Entity View returned customer
                    Work View returned work
     IF returned work request type IS EQUAL TO e
     SET sent_through_transfer work restart_count TO 0
     ELSE
     SET sent_through_transfer_work restart_count TO
          returned_from_transfer work restart_count +
          commit_ief_supplied_count + restart work
          restart_count
EXIT STATE IS process_commit
SET target work restart_count TO sent_through_transfer
     work restart_count
NOTE
     WRITE RESTART NUMBER is an external action block that records the number
     of input transactions for which database updates have been committed.
USE write_restart_number
     WHICH IMPORTS: Work View target work
...
NOTE
     WRITE RESTART NUMBER is an external action block that records the number
     of input transactions for which database updates have been committed.
USE write_restart_number
     WHICH IMPORTS: Work View target work
```

The views used by Batch Maintenance are presented in the following sample code. Note that the technique for using external action blocks for sequential file processing is also illustrated in these samples.

View Definitions for Batch Maintenance

```
Import View
------------------------------------------------------------------------------
View RETURNED_FROM_TRANSFER of work group WORK Attributes:
     RESTART_COUNT
------------------------------------------------------------------------------
Export Views
------------------------------------------------------------------------------
View SENT_THROUGH_TRANSFER of work group WORK Attributes:
     RESTART_COUNT
------------------------------------------------------------------------------

Local Views
------------------------------------------------------------------------------
View RESTART of work group WORK Attributes:
     RESTART_COUNT

View Target of work group WORK Attributes:
     RESTART_COUNT

View RETURNED of work group WORK Attributes:
     REQUEST_TYPE

View RETURNED of entity CUSTOMER Attributes:
     NUMBER
     NAME
     STATUS
     CREDIT_RATING
     BILL_TO_ADDRESS_LINE_1
     BILL_TO_ADDRESS_LINE_2
     BILL_TO_ADDRESS_LINE_3
     BILL_TO_CITY
     BILL_TO_STATE
     BILL_TO_ZIP

View COMMIT of work group COMPOSER_SUPPLIED Attributes:
     COUNT

View MESSAGE of work group WORK Attributes
     RESULT
```

More information:

Designing the Data Structure (see page 49)

Design for Ease of Use

Depending on the characteristics of the implementation, you may supplement the action diagram significantly. Any logic is required to provide a reasonable implementation of elementary processes is permissible.

Suppose that while considering the implementation of Maintain Customer and the requirements of the user community, you conclude the following points:

- A user should not be allowed to delete a customer without first displaying it.
- If a delete fails, the Command special field should contain the value “Display” in anticipation of the user's next request.
- A user can delete the customer just added.

The following sample code presents one possible solution to these requirements.

Procedure Action Diagram for Maintain Customer (version 2)

```
MAINTAIN_CUSTOMER_VERSION_2
     IMPORTS:        Entity View hidden_import_customer
                     Entity View import_customer
     EXPORTS:        Entity View hidden_export_customer
                     Entity View export_customer
     LOCALS:         Entity View temporary customer
     ENTITY ACTIONS: Entity View customer
EXIT STATE IS requested_operation_complete

CASE OF COMMAND
CASE display
     READ customer
          WHERE DESIRED customer number IS EQUAL TO import customer number
     WHEN successful
     MOVE customer TO export customer
     SET hidden export customer number TO customer number
     WHEN not found
     EXIT STATE IS customer_not_found

CASE delete
     IF hidden_import customer number IS EQUAL TO import customer number
     USE delete_customer
          WHICH IMPORTS: Entity View import customer
          WHICH EXPORTS: Entity View export customer
     ELSE
     EXIT STATE IS read_required_before_delete
     COMMAND IS display

CASE add
     USE add_new_customer
          WHICH IMPORTS: Entity View import customer
          WHICH EXPORTS: Entity View export customer
               IF EXIT STATE IS EQUAL TO requested_operation_complete
               MOVE temporary customer TO export customer
               SET hidden_export customer number TO customer number
               MAKE export customer name "ERROR"
     ELSEIF EXIT STATE IS EQUAL TO customer_already_exists
     OTHERWISE
     EXIT STATE IS invalid_command
```

This procedure step includes several new data views:

- The requirement for a Display action implies the need for a read statement, which requires a new entity action view of CUSTOMER. Note line 7.
- The requirement that a Delete be preceded by a Display or an Add implies the need to save data after a Display or an Add. One can test the saved data before a Delete to verify that the correct occurrence was selected for deletion.
- In this case, the Customer Number field is exported as a hidden field through the view Hidden Export Customer (line 4) and imported as a hidden field through the view Hidden Import Customer (line 2).

This procedure step includes a new CASE option to the CASE OF COMMAND action.

If the command is Display, the requested customer is read and displayed. Note lines 10 through 19.

A move action is required to populate the export view from the entity action view. See note line 15.

If the read statement is successful, the Customer Number is saved in the hidden field Hidden Export Customer Number.

In the Add case, modeled in lines 30 through 40, the user may delete the customer just added. The newly added customer's number should be saved as a hidden field just as in the Display case. See line 16.

To allow the user to continue with the next transaction, it is useful to clear the display of the customer data just added, so in line 35 the local view Temporary Customer (defined in line 6, containing attributes that remain unpopulated) is moved to Export Customer.

More information:

Designing Screens (see page 101)

Design Designer-Added Procedures

Designer-added procedures are implementation-specific procedures that support user tasks or otherwise improve or simplify some characteristics of the system implementation.

Note: Take care when creating a designer-added procedure to avoid altering the results of analysis. You may be tempted to add a procedure that does not implement an elementary process but is still relevant to the business rather than to the user task or system. Resist this temptation.

The situation should be remedied by further analysis, which may result in the addition of an elementary process, and any consequent modification to the data model. The additional procedure can then be transformed from the elementary process in the same way as other procedures.

When to Use a Designer-Added Procedure

The following examples detail when you can use a designer-added procedure:

- To add a menu procedure in an online environment
- A menu procedure gives the user a way to navigate through the process-implementing procedures in the business system
- For example, the menu screen in the following screen allows selection among three procedures, each of which implements an elementary process

```
                Order Entry Menu

                1.  Take An Order

                2.  Delete An Order

                3.  Correct An Order

                Select Option ===> _________________
```

- The menu window allows the same choices. The menu procedure that supports the display has no direct relevance to the business and so would not have been identified during analysis. In this case, the designer has added the procedure simply to improve the system flow
- To create a report or display, in which data is presented to the user, but is not manipulated for any single business purpose

When Not to Use a Design-Added Procedure

In general, a designer-added procedure should not perform CREATE, UPDATE or DELETE actions on entity types identified during analysis.

The following list details the exceptions to this guideline:

- Maintaining business or designer-added entities in support of user tasks; for instance, contained in personal work-in-progress, or in a work queue.
- Maintaining replicated data by procedure logic instead of by the DBMS.

- This is the recommended method, but may not be possible where more than one DBMS must be used, or data is related to data in a current data store not maintained by the application.
- Transfer or deletion by archive procedures of entities that are no longer of interest or that can legally and safely be discarded.

More information:

Build Designer-Added Procedures (see page 148)

Procedures and Procedure Steps

In a CA Gen design, each procedure contains at least one procedure step.

Online and batch procedures can be single-step procedures or multi-step procedures.

Although it is not immediately apparent when looking at the Dialog Flow Diagram, the procedure step (not the procedure) is the object for which a Procedure Action Diagram and a screen are defined.

Since most procedures are single-step procedures, the existence of procedure steps is sometimes not obvious to the designer. On the Dialog Flow Diagram, where both procedures and procedure steps appear, procedure steps are not added to the diagram unless more than one is required to support the parent procedure. Based on whether the procedure is defined as online or batch, the meaning of the terms procedure and procedure step differ slightly.

Online Procedures

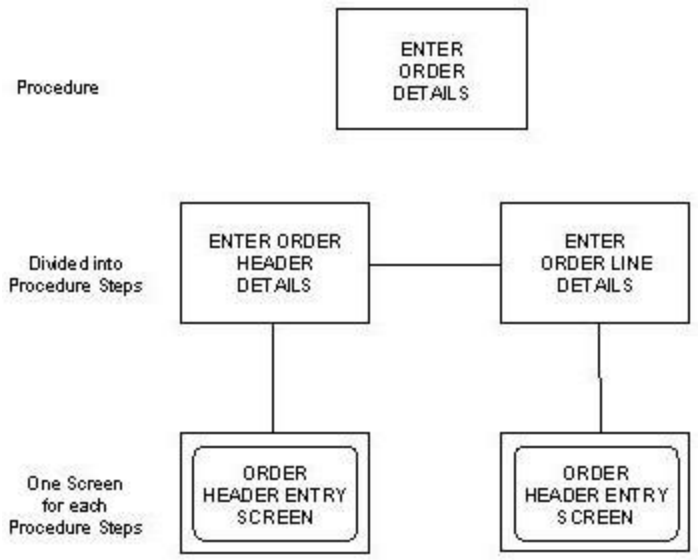
An online procedure step is the portion of a procedure associated with a display and one procedure action diagram.

Each online process-implementing procedure is responsible for implementing complete elementary processes.

Multiple Online Procedure Steps

Any procedure requiring multiple screens to support the elementary processes it implements will need a procedure step for each screen.

Consider the example in the following illustration, Procedure With Two Procedure Steps.



The example shows a single procedure called Enter Order Details. Assume that this procedure implements an elementary process called Take Order. The designer determines that the user needs to use two displays when taking an order: one to enter the base order information and another to enter order item details. As a result, two procedures are created to support the Enter Order Details procedure. These new procedure steps are Enter Order Header Details and Enter Order Item Details.

Whenever possible, you should implement online procedures in a single procedure step. Multi-step procedures can require more effort to develop and maintain than single-step ones, and splitting the results of transformation into multiple steps is largely a manual process.

Multi-step procedures may be appropriate when one of the following conditions occurs:

- The contents of the procedure's data views are too large to fit on a single display.
- Implementing the procedure in one procedure step is awkward and might lead to user confusion.

In all these situations the designer should carefully manipulate any action blocks developed during analysis, to ensure that the business requirement for the elementary process being implemented is not compromised. CA Gen does not automatically maintain the integrity of the elementary process once a procedure is divided into procedure steps.

It is important to ensure the integrity of the business across multiple procedure steps. We strongly recommended that you execute entity actions that will cause database modifications in the final procedure step. These entity actions are CREATE, UPDATE, and DELETE. In this way, interruption of a procedure between procedure steps will not leave the data partly updated, and therefore possibly in an inconsistent state.

Divide Online Procedures Into Multiple Steps

Consider the elementary process Add New Customer, which has been used in the examples in this guide. Rather than combining Add New Customer and Delete Customer into a single Maintain Customer procedure, suppose that the designer opts to define Add New Customer in its own procedure.

Assume that the designer divides Add New Customer into two procedure steps as shown next:

- Add First Part is responsible for collecting the following attributes:

  Name

  Street Address

  City

  State

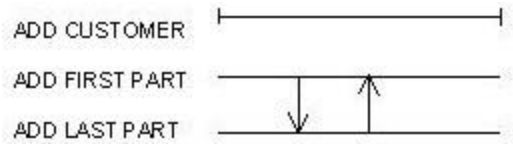
- Add Last Part is responsible for collecting the following attributes:

  Phone Number

  Credit Limit

The elementary process Add New Customer should be implemented in two procedure steps. You should implement all update type entity actions, such as CREATE, UPDATE, and DELETE, in the final procedure step. This will avoid violating the integrity of the data base. By observing this guideline, the designer guarantees that the view of the business represented on the data base remains in a consistent state if the system fails between procedure steps.

Given this constraint, the sample Add Customer procedure can be implemented as shown in the following illustration, Dialog Flow Diagram With Two Procedure Steps.



Assume the following details of the dialog flows shown in the illustration:

- Add First Part transfers to Add Last Part when Exit State is Requested Operation Complete.
- All data in the export view of Add First Part is sent to the import view of Add Last Part during the transfer.
- Add Last Part transfers back to Add First Part when Exit State is Requested Operation Complete.
- Both transfers have flow actions of Display First.

This division of procedures into steps requires no change to the views in the original Add Customer Process Action Diagram as shown in the following sample code.

Data Views for Add First Part

```
Procedure Step Definition
Procedure Step     ADD_FIRST_PART

Import Views
   ViewIMPORT of entity CUSTOMER
      Attributes:
                 NAME
                 STREET_ADDRESS
                 CITY
                 STATE
Export Views
   ViewEXPORT of entity CUSTOMER
      Attributes:
                 NAME
                 STREET_ADDRESS
                 CITY
                 STATE
End Of Report
```

The following notes will help clarify the example:

- Add First Part should validate all data for which it is responsible, even though the update is deferred until the final procedure step. Otherwise, the user may fill in several screens of data only to learn on the final screen that an item has failed validation.

```
ADD_FIRST_PART
    IMPORTS:         Entity View import customer
    EXPORTS:         Entity View export customer
    ENTITY ACTIONS:  Entity View requested customer

EXIT STATE IS requested_operation_complete
    READ requested customer
       WHERE DESIRED customer name IS EQUAL TO import customer name
    WHEN successful
    EXIT STATE IS customer_already_exists
    WHEN not found
    MOVE import customer TO export customer
```

- In this case, the READ action (line 7) that tests for duplicate customer name appears in the Procedure Step Action Diagram for Add First Part. Note that this is the same logic specified in the transformed action block, Add New Customer.

- An experienced designer of online procedures may notice a flaw in this portion of the example, should two users concurrently attempt to add the same customer. The first procedure step may succeed where the second fails. This could happen if, between the execution of the first and second, another user adds the same customer. In this example, such a possibility is assumed to be remote. Even if it should occur, the worst that would happen is user inconvenience. There is no loss of system integrity because no real update takes place until Add Last Part completes.

- Those components of the import and export views of Add Last Part that are not displayed on the screen (everything but Phone Number and Credit Limit) are implemented as hidden fields. For more information about hidden fields, see the chapter "Designing Screens." This way, all information captured in Add First Part is available through multiple executions of Add Last Part.

```
Procedure Step ADD_LAST_PART
Import Views
    ViewIMPORT of entity CUSTOMER
      Attributes:
        NAME
        STREET_ADDRESS
        CITY
        STATE
        PHONE_NUMBER
        CREDIT_LIMIT
        DATE_ADDED

Export Views
    ViewEXPORT of entity CUSTOMER
      Attributes:
        NAME
        STREET_ADDRESS
        CITY
        STATE
        PHONE_NUMBER
        CREDIT_LIMIT
        DATE_ADDED
```

- All attributes of the import view of Add Last Part appear in its export view as well, even though some are not returned by the Add New Customer action block. By moving the contents of the import view to the export view, all information placed on the initial screen for Add Last Part is available through multiple executions.

```
ADD_LAST_PART
    IMPORTS:          Entity View import customer
    EXPORTS:          Entity View export customer

EXIT STATE IS requested_operation_complete
MOVE import customer TO export customer

USE add_new_customer
    WHICH IMPORTS:    Entity View import customer
    WHICH EXPORTS:    Entity View export customer
```

This example satisfies the guidelines mentioned previously. The update takes place in Add Last Part using an unmodified Process Action Diagram, Add New Customer as shown in the following sample code.

Action Diagram for Add New Customer

```
ADD_NEW_CUSTOMER
     IMPORTS:         Entity View candidate_customer
     EXPORTS:         Entity View newly_created_customer
     ENTITY ACTION:   Entity View brand_new_customer
READ brand_new_customer
     WHERE DESIRED customer name IS EQUAL TO candidate customer name
     WHEN successful
     EXIT STATE IS customer_already_exists
WHEN not found
     CREATE brand_new_customer
     SET name TO candidate_customer_name
     SET number USING calculate_customer_number
          WHICH IMPORTS: Entity View candidate customer
     SET street address TO candidate customer street_address
     SET city TO candidate customer city
     SET state TO candidate customer state
     SET zip TO candidate zip
     SET phone_number TO candidate customer phone_number
     SET credit_limit TO 0
     SET date_added TO CURRENT DATE
MOVE brand_new_customer TO newly created customer
EXIT STATE IS requested_operation_complete
```

If, for any reason, you do not follow the guidelines, you should ensure that the business requirement is satisfied and that the integrity of the data is not compromised.

Batch Procedures

Batch procedures are implemented as jobs.

Batch procedure steps are implemented as job steps.

Dividing a batch procedure into multiple procedure steps is somewhat different from doing so for online procedures. Since batch procedure steps have no displays, the case where the procedure's data views are too large to fit on a single display does not apply. Likewise, since there is no interaction with a user, the notion that a procedure be split into steps to reduce confusion does not hold.

Most batch procedure steps that deal with elementary processes at all implement at least an entire elementary process (and maybe multiple ones).

The only reason for defining multi-step batch procedures is to arbitrarily combine different process-implementing procedure steps so that they execute as a single batch job for the sake of convenience.

Build Procedures

You use the results of analysis and the system structure design to build process-implementing and designer-added procedures.

Process-Implementing Procedures

The purpose of a process-implementing procedure is to implement directly one or more elementary processes specified during analysis.

The technique for converting analysis objects into design objects is called transformation.

Any elementary process that has been fully defined can be implemented by one or more procedures as shown in the following illustration.

Relationship Between Processes and Procedures

CA Gen automatically transforms the necessary detail for this purpose and makes any process action diagrams available as process action blocks.

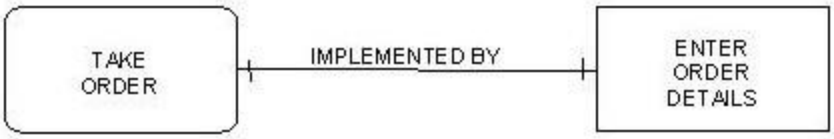
These are the possible relationships between processes and procedures:

- One process to one procedure
- This relationship is the easiest to implement
- One process to many procedures
- Many processes to one procedure
- Many processes to many procedures

One Process to One Procedure

Typically, one procedure implements one elementary process, as shown in the following illustration.

The one-to-one case is the easiest to implement because it requires less complex logic and is simpler to maintain than many-to-many.

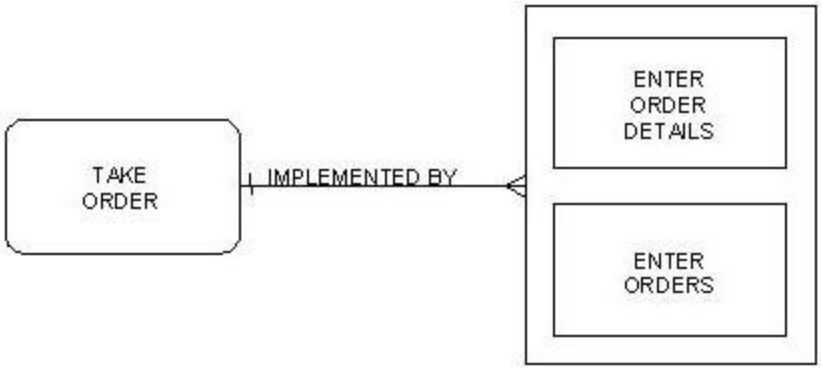


One Process to Many Procedures

This relationship is used for alternative implementations of the same elementary process, for instance, where different users have differing access and update authorities when performing the same process.

For example, consider a single elementary process named Take Order. Two groups of users execute this elementary process. One group, the expert group, takes orders extremely often. To satisfy their needs, a single procedure must be able to take many orders. The other group, a casual group of users, takes orders occasionally and, therefore, needs more reference material on the screen. To satisfy their needs, a single procedure must take only one order. In both cases, the elementary process Take Order is being implemented. Only the details of how Take Order is implemented vary.

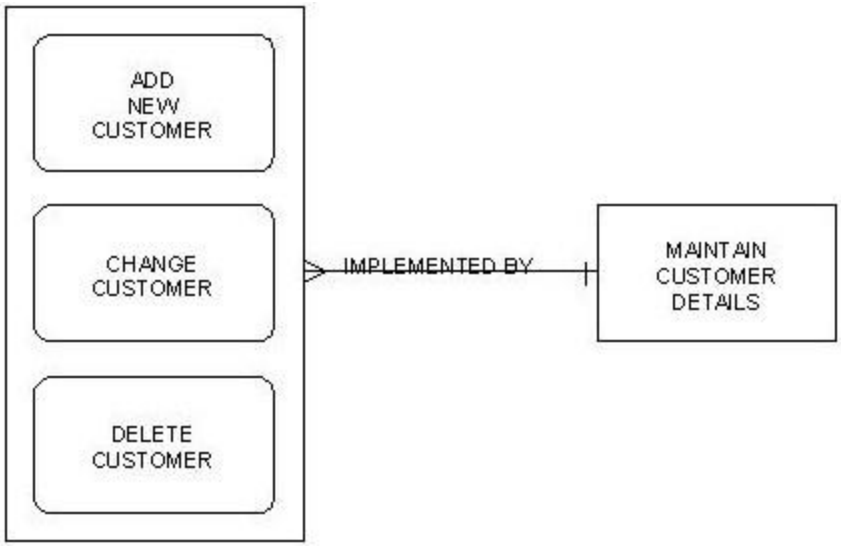
In this instance, one elementary process is implemented in multiple procedures, as shown in the following illustration.



Many Processes to One Procedure

This relationship is used when several elementary processes have similar information views and are required by a common set of users.

An example of this case is shown in the following illustration.



In this example, three separate elementary processes are implemented by the same procedure. This is possible if the following two conditions are met:

- The information required by the elementary processes is very similar. In the illustration, all three elementary processes probably require Customer Details.
- The same set of users performs all the elementary processes under consideration. In other words, if one person is responsible for all aspects of Customer Maintenance, the arrangement depicted in the illustration is acceptable. However, if different people are responsible for adding, changing, and deleting customers, you should implement the three processes separately.

Many Processes to Many Procedures

This relationship is a combination of the one procedure to many processes and many procedures to one process cases.

Build Designer-Added Procedures

Designer-added procedures are implementation-specific procedures that improve or simplify some characteristic of the overall implementation.

Designer-added procedures should not alter the results of analysis. To avoid compromising the results of analysis, designer-added procedures should never modify business information.

Note: Do not use the entity actions CREATE, UPDATE, or DELETE, except for design entity types.

Such a requirement identified during Design generally points out a deficiency in analysis that should be corrected by adding or modifying a process.

There is another class of designer-added procedure that is less frequently used. This class of designer-added procedure is responsible for displaying and maintaining design entity types. Imagine that you should determine the failure rate of each user who adds customers using the Maintain Customer procedure step. This involves:

- Adding a design entity type that has attributes of Terminal ID (to identify the user), number of executions of Add Customer, and number of failures of Add Customer
- Modifying the Maintain Customer action diagram to add to the number of executions each time Add Customer is invoked and to add to the number of failures each time it fails

The mechanism for displaying this information on a terminal is a designer-added procedure. The following illustration is a designer-added procedure that supports this example.

Procedure Action Diagram for Designer-Added Procedure

```
DISPLAY_FAILURE_RATE

     IMPORTS:        Entity View inbound add_customer_failure
Procedure Step:      DISPLAY FAILURE RATE
     EXPORTS:        Work View outbound Composer_supplied
                     Entity View outbound add_customer_failure
     ENTITY ACTIONS: Entity View add_customer_failure

EXIT STATE IS requested_operation_complete
     READ add_customer_failure
          WHERE DESIRED add_customer_failure terminal_id
          IS EQUAL TO inbound add_customer_failure_terminal_id
     WHEN successful
     MOVE add_customer_failure TO outbound_add_customer_failure
     SET outbound_Composer_supplied percentage TO (add_customer_failure
          number_of_failure/add_customer_failure
          number_of_attempts - add_customer_failure
          number of attempts) 100
     SET outbound Composer_supplied total_integer TO add_customer_failure
          number_of_attempts - add_customer_failure
          number_of_failures
WHEN not found
EXIT STATE IS terminal_id_not_found
```

This example references two views of the work attribute set supplied by CA Gen.

Percentage is used to show the percentage of failures to attempts. Total Integer shows the user's total number of successes, calculated as the number of attempts less the number of failures.

More information:

Preparing for Design (see page 15)

Build Action Diagrams

In both analysis and design CA Gen can be used to generate Action Diagrams automatically. This facility can save considerable time. It can ensure that standard, efficient action statements are written for all types of diagrams.

These are the basic ways to build an action diagram:

- Let CA Gen synthesize a complete process or procedure, and, if necessary, the action blocks within.
- Copy an existing procedure with substitution of entity types.
- Manually adjust either of the above.
- Write an action diagram from scratch using the Action Diagram editor.

For the first two methods of building an action diagram, CA Gen offers the following tools:

- Process synthesis
- Procedure synthesis
- Copy with substitution

More information:

Process Synthesis to Build an Action Diagram (see page 150)
Procedure Synthesis to Build an Action Diagram (see page 151)
Copy With Substitution to Build an Action Diagram (see page 154)

Process Synthesis to Build an Action Diagram

The CA Gen process synthesis tool may already have been used during analysis to generate elementary processes that create, read, update, and delete, and an entity type. In certain circumstances, tool may also list entity types. The resulting action diagrams will already have entity actions, import and export views.

In creating the process action diagram, CA Gen lets the developer select actions to be performed on related entity types for the create, update and delete entity actions. For instance, if a mandatory relationship exists between two entity types, one of which is to be created, selecting this option will allow you to choose whether to read or to create the other entity in the relationship.

The developer may also request automatic attribute view selection. This directs CA Gen to create views and view matching without designer intervention. This is used when all attributes are required. Otherwise you will find it more convenient to select the required attributes when prompted by CA Gen during synthesis.

Procedure Synthesis to Build an Action Diagram

The use of the CA Gen procedure synthesis tool dovetails well with process synthesis. It generates new procedure action diagrams that implement existing action blocks, and entire procedures with enhanced features. Procedure synthesis can build:

- A menu procedure that will allow flows to other procedure steps based on commands.
- An entity maintenance action diagram that will USE process action blocks to create, read, delete, and update an entity type for which process action blocks already exist. These process action blocks are often created by process synthesis.
- An action diagram that USEs existing action blocks.
- An action diagram that reads a group of entities and then lists them (selection list). In this case, an action block to perform the read is also generated.
- In addition, from an existing selection list procedure CA Gen can synthesize a further subject list or neighbor list procedure and a flow between the two.

The following list shows procedure synthesis options:

- Entity Maintenance
- Selection List
- Implements Action Blocks
- Menu
- Subject List
- Neighbor List

Entity Maintenance

This is used to synthesize a procedure that uses action diagrams for the entity actions read, update, delete and create. CA Gen can add extra logic to the action diagram if required, for instance:

- Two-stage commits for create, update and delete entity actions, that require the user to confirm a selected entity action for a selected entity. This reduces the risk of unintentional updates, at the cost of increased user interaction
- Associating commands on the associated layout to the synthesized actions for entity maintenance, clear screen, and refresh screen
- Clear screen logic
- Validation logic

The following sample code shows the type and sequence of logic that CA Gen will synthesize.

Synthesized Procedure Action Diagram

```
SYNTHESIZED PROCEDURE
     IMPORTS:  POPULATED
     EXPORTS:  POPULATED
SET INITIAL EXIT STATE
CLEAR SCREEN LOGIC
REFRESH SCREEN IF COMMAND IS EQUAL TO
CLEAR
IS EQUAL TO CLEAR
MOVE IMPORTS TO EXPORTS
SECOND STAGE UPDATE/DELETE LOGIC
VALIDATION LOGIC
MAIN CASE OF COMMAND
USE ACTION BLOCKS
```

The synthesized groups of actions form the starting point for designing the detailed logic. For more information, see Refining Synthesized Procedure Action Diagrams.

Selection List

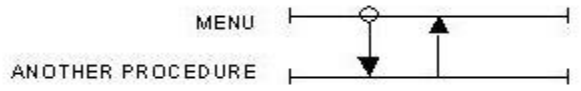
This option builds a procedure that lists occurrences of an entity type. CA Gen gives the designer the opportunity to choose the sort attribute, the attributes that are to be listed, and a sequence for the list.

Implements Action Blocks

This option lets the designer select action blocks to be used in a procedure, and to associate the execution of each action block with a designated command that will be selected by a user. CA Gen then builds the appropriate CASE OF COMMAND and action group to USE the action block.

Menu

With this option CA Gen synthesizes a menu procedure and the flows to other procedures. It assigns commands and associated exit states to the flows. This option is shown in the following illustration:



CA Gen adds a transfer and a link flow to each destination procedure selected.

The link flow is from the menu to the procedure and the transfer flow is from the selected procedure back to the menu.

The transfer flow may be used when the selected procedure is flowed to from one or more other procedures.

Subject List

Synthesizing a subject list flow lets the designer to add a flow between a synthesized selection list procedure step and a maintenance procedure step. The selection list procedure step must have the same subject entity as the entity maintenance procedure, and may have been synthesized earlier.

This procedure synthesis automatically creates a transfer and a link:

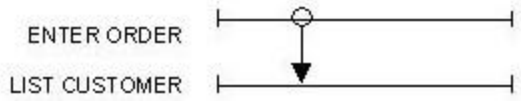
- A link is created between the selection list procedure step and the entity maintenance procedure step. The link flow is from the selection list step to the maintenance procedure step.
- A transfer is created between the entity maintenance procedure step and the selection list procedure step. The transfer flow may be used when the entity maintenance procedure is flowed to from one or more other procedures.

Neighbor List

Some procedures require the user to enter data about two or more related entity types. Often the information to be entered is the identifier of an entity occurrence that is to be associated with some central entity being created or updated in the procedure. For example, in the data model fragment shown in the following illustration, CUSTOMER is a neighbor of ORDER.

When a procedure creates an order, the identifier of the customer must be supplied so that the order can be associated with one customer. In this example the user does not know the customer identifier.

One solution, shown in the following illustration, is to design a list procedure that the user may link to, select the customer entity, and return from after obtaining the identifier of the chosen customer entity occurrence.



Synthesizing a neighbor selection list flow lets you add a flow from a current procedure step to a selection list procedure step. For example, you could add Enter Order. The selection list procedure step has any subject entity type, in this case customer, which could be that of the current procedure step.

Copy With Substitution to Build an Action Diagram

Copy with Substitution lets you do the following tasks:

- Select an existing action block that references a single entity type
- Replace the reference to that entity type and its attributes with another entity type
- Create a new action block

For example, consider an action block named Add Employee. Its logic may be quite similar to an existing action block, Add Customer. Using copy with substitution, the designer can request that an Add Employee action block be built from Add Customer, substituting references to Customer with references to Employee. You may use it on procedure steps or action blocks.

Copy with Substitution is presented as an alternative only when both of these situations occur:

- The action block to be copied references entities of only one type.
- Only those relationships that are involved with the entity type are referenced in the action block.

Copy with Substitution has to make assumptions about the use of attributes. For this reason, the designer should check that the action block produced by Copy with Substitution really does what it should.

Copy with Substitution for an online procedure step action block invokes the Auto Layout option of the Screen Design tool. The result is both the action diagram and the screen for the procedure step.

Action Diagram Editor

CA Gen action diagram editor lets you add, modify, and delete actions in an action diagram. Its menu-driven interface enables you to build action diagrams by picking from the lists that include all options available in a particular context. Thus, there can be no syntax errors. Details of actions that can be used in the editor are described in the Toolset documentation.

The actions relevant to block mode design are described in the following list:

- Entity-Retrieve and manipulate stored information about entities.
- Relationship-Manipulate pairings of stored entities.
- Assignment-Assign values to attribute views.
- Repeating-Manipulate components of a repeating group view.
- Conditional-Change the flow of actions based on some condition.
- Control-Change the flow of the process unconditionally.

Structure of Completed Action Diagrams

Completion and maintenance of action diagrams are easier if they follow a predictable pattern.

The following illustrates a possible standard action diagram structure for an online procedure step.

Action Diagram Structure for Online Procedure

Note: This suggested structure includes all possible import validation before the main processing takes place.

- View definitions

  Import views

  Export views

  Local views

- Initial processing

  Setting a default Exit State

  Checking for clear screen

  Moving import view contents to export views

  Setting initial values

  Validating imported data and making export data indicated as error

  Testing for commands to control subsequent processing

- Processing

  Application processing

  Moving data to export views

  (Optionally) setting field properties

  Setting Exit States

  Exit

Advanced Action Diagram Features

This section deals with some of the more advanced concepts in action diagramming, starting with topics that will help you to more fully understand some of the action diagramming features frequently used during design:

- Action diagram expressions
- Notes on set using, Default and Derivation Algorithms
- Using Exit States
- Using special attributes

There are two more topics, which deal with action diagramming constructs that you will use less frequently. The topics provide an additional level of flexibility for handling more complex detailed logic:

- Action diagram functions
- Advanced repeating group handling

The final topic in this section deals with the use of external action blocks.

Action Diagram Expressions

Expressions are used in conditions of all kinds and in SET statements. Their purpose is to provide a value consistent with the primitive domain of the attribute view or special attribute to which they are being assigned (in SET statements and the FOR loop) or compared (in conditions). As a result, the construction of an expression is different depending on whether it is to return a Text, Number, Date or Time value.

Text Expressions

Expressions that evaluate to a text value are described in the following list.

- Character View-An attribute view on the primitive domain Text, like CUSTOMER Name
- Character String-A literal value enclosed in quotes, like "Davy Crockett"
- Character Function-An action diagram function that evaluates to a character value, like SUBSTR (CUSTOMER Name,3,2)

In online procedures, a text expression can also refer to one of the following special attributes: TERMINAL ID, USER ID, PRINTER TERMINAL ID, or TRANCODE.

Number Expressions

Expressions that evaluate to a number can include parentheses for grouping and the arithmetic operators -, +, /, * and **, as well as the components described in the following list.

- Numeric View—An attribute view on the primitive domain Number, like ORDER_LINE Quantity
- Number—A numeric constant, like 153
- Numeric Function—An action diagram function that evaluates to a number, like JULDATE (CURRENT_DATE)
- Subscript—The current value of a subscript for an explicitly subscripted repeating group view

- Last—The occurrence number of the last populated entry of an explicitly subscripted repeating group view
- Max—The maximum cardinality of an explicitly subscripted repeating group view

Date Expressions

Expressions that evaluate to a date must begin with one of the components described in the following list.

- Date View-An attribute view on the primitive domain Date, like ORDER Date_Shipped.
- Date Function-An action diagram function that evaluates to a date, for example:

  ```
  DATEJUL(Working JULIAN_DATE)
  Current Date-The special attribute which contains current date.
  ```

After they begin with one of the specified components, they can include a numeric modifier that adds a number of years, days, or months to the initial date component or subtracts years, days, or months from it. For example, the following expression returns the date exactly one week before the current date:

```
current_date - 7 days
```

The following expression returns the date three months after an order was placed:

```
Order date_placed + 3 months
```

Time Expressions

Expressions that evaluate to a time are similar to those which evaluate to a date.

A time expression must begin with one of the components described in the following list.

- Time View-An attribute view on the primitive domain Time, like FLIGHT Time_of_Arrival
- Time Function-An Action Diagram Function that evaluates to a time, for example:

  ```
  TIMENUM(IMPORT APPOINTMENT NUMERIC_TIME)
  ```

- Current Time-The special attribute which contains the current time

Time expressions can include a numeric modifier to adjust the time component by a number of hours, minutes or seconds.

Action Blocks Used in SET ... USING Actions

SET ... USING invokes an action block that establishes the value of the attribute view being set. The used action block may have any number of import, local, or entity action views. However, it can have only one export attribute view, and it must be a view of the attribute being set in the using action block.

For example, assume that the action block Create Customer includes the following SET ... USING statement:

```
SET customer number USING customer_number_calculation
```

The action block Customer Number Calculation must contain only a single export attribute view, and it must be a view of CUSTOMER Number.

Derivation Algorithms

An attribute's derivation algorithm executes whenever an entity of the type to which the attribute belongs is READ, provided the entity action view includes a view of that attribute.

A derivation algorithm must return a single export attribute view of the attribute being derived. Additionally, it can have only a single import view, and that is a view of the entity type to which the derived attribute belongs. All non-derived attributes for the entity occurrence have already been populated before the derivation algorithm is invoked.

The import view of a derivation algorithm has a special property: it can be referenced as the equivalent of a CURRENT view (a distinction normally reserved for entity action views) in READ statements appearing in the derivation algorithm.

The derivation algorithm in the following sample code calculates a value for the attribute Total of the entity type ORDER, which is the sum of the value of all its ORDER ITEMs.

Example Derivation Algorithm

```
DERIVE_ORDER_TOTAL
     IMPORTS:         Entity View provided order
     EXPORTS:         Entity View derived order
     ENTITY ACTIONS:  Entity View product
                      Entity View order_item

READ EACH order_line
     product
     WHERE DESIRED order_item is on CURRENT provided order
     AND DESIRED product appears on DESIRED order_item
SET derived order total TO derived order total +
     (product price * order_item_quantity)
```

Composite attributes can be simulated by using a derivation algorithm that includes string functions. For example, assume that a composite attribute Phone Number is built from the elementary attributes Area Code, Exchange, and Station. This is a reasonable need for a phone company.

To simulate the composite attribute, Phone Number can be defined as a derived attribute with the derivation algorithm shown in the following sample code.

Simulating a Composite Attribute

```
DERIVE_PHONE_NUMBER
     IMPORTS:    Entity View provided customer
     EXPORTS:    Entity View derived customer
SET derived customer phone_number TO CONCAT
     (provided customer exchange, provided customer station)
```

Exit States

Exit State is a special attribute whose value controls what happens at the conclusion of a procedure step.

Depending on the value of Exit State, the procedure step may react in the following way:

- Display a message in the system message field.
- Initiate a dialog flow.
- Roll back database updates resulting from the execution of the procedure step.
- Abort the procedure step. Roll back all database updates and abnormally terminate.

The special attribute Exit State is assigned a value using the exit state is action. The only exit state values that can be assigned are those defined using the Exit State definition facility.

Exit state definitions can be built using System Defaults or from any tool that can reference Exit States.

Each Exit State definition includes the following components:

- Name—The name appears as the subject of the EXIT STATE IS action.

  For example, an Exit State customer NOT FOUND may be set if a particular CUSTOMER does not exist on the database. In that case, the EXIT STATE IS action in the action diagram would read:

  ```
  EXIT STATE IS CUSTOMER_NOT_FOUND
  ```

- Message—The message appears in the system message field if its associated exit state value is set when the procedure step concludes and the exit state value does not trigger a dialog flow or an abnormal termination.

  Messages are displayed according to the message type. The message types are Informational, Warning and Error.

  In online procedure steps, each type of message is displayed on the screen with different video properties.

  For example, the CUSTOMER_NOT_FOUND Exit State may have an associated message: Requested customer not on the database. If CUSTOMER_NOT_FOUND is set in the WHEN NOT FOUND condition of a READ statement, the message requested customer not on the database appears in the system message field of a screen.

- Termination Action—The termination action indicates the type of processing that is performed at run time when the associated exit state value is detected. The possible termination actions are described in the following table.

More information:

Designing the Procedure Dialog (see page 71)

Termination Actions

The following list describes three types of termination actions:

- Normal—The exit state message will be displayed or a dialog flow will take place.
- Rollback—All database updates (that is, the results of create, update, and delete actions) performed by the procedure since the previous checkpoint will be backed out.
- Abort—The procedure step will abnormally terminate and will roll back all database updates

Set Exit States

In an action diagram, exit states are set according to the action performed, and its outcome.

The outcome of an action may be signaled by the:

- Return condition of a data action (for example, WHEN NOT FOUND)
- Value of a return code attribute exported from a USEd Action Block; (for example, “processing successful”)

These are options for ensuring that an appropriate exit state is always set:

- Set an exit state only after each action is successfully achieved or each exception condition is discovered.
- You must ensure that every possible exception condition is identified.
- Set a positive default exit state at the beginning of the procedure logic, then test for each positive outcome and every exception that can be foreseen, and reset the Exit State accordingly. (This is currently the most widely adopted option.)
- If no other exit state value is set, then the default value is used.
- You must ensure that every possible exception condition is identified to avoid giving a false “positive” result.
- Set a negative exit state at the beginning of the procedure logic, then test for each successful outcome that is part of the procedure specification (and as before, every exception that can be foreseen), and reset the Exit State accordingly.
- If no positive or known negative exit state value is set, the default value is used. This ensures that any unexpected exception condition will not be assumed to be a successful outcome.

For ease of logic design and maintenance, each organization should try to adopt a single, consistent option for all online and common action block logic.

Where a mixture of these options is already adopted, you must be aware of which option is employed by existing logic with which their procedures must inter work.

Special Attributes

The following special attributes are used in online environments:

- CURRENT DATE
- CURRENT TIME
- TERMINAL ID
- USER ID

- TRANCODE
- NEXT TRANSACTION ID

CURRENT DATE

The data type of CURRENT DATE is Date. You cannot modify it, but you can compare it with any view of a date attribute and use it as the source for a SET statement whose target is the view of a date attribute.

This is an example of how CURRENT DATE is used:

```
if current date is equal to client_checkup_date
  exit state is send_client_reminder
  set customer date_added to current date
```

CURRENT TIME

The data type of CURRENT TIME is Time. You cannot modify it, but you can compare it with any view of a time attribute and use it as the source for a SET statement whose target is the view of a time attribute. (The examples shown for CURRENT DATE also apply to CURRENT TIME, with appropriate substitutions.)

TERMINAL ID

The data type of TERMINAL ID is Text. You cannot modify it, but you can compare it with any view of a text attribute or literal, and use it as the source for a SET statement whose target is the view of a text attribute.

This is an example of how TERMINAL ID is used:

```
IF TERMINAL ID IS NOT EQUAL TO user usual_terminal_id
  EXIT STATE IS security_violation
  SET user last_terminal_used to terminal id
```

USER ID

The data type of the USER ID special attribute is Text. You cannot modify it, but you can compare it with any view of a text attribute or literal, and use it as the source of a SET statement whose target is the view of a text attribute.

This is an example of how USER ID is used:

```
IF USER ID IS EQUAL TO "DAACJCN"
  EXIT STATE IS warn the system administrator
```

TRANCODE

The data type of the TRANCODE special attribute is Text. You cannot modify it, but you can compare it with any view of a text attribute or literal, and use it as the source for a set statement whose target is the view of a text attribute.

This is an example of how TRANCODE is used:

```
IF TRANCODE IS EQUAL TO "DX01"
AND TERMINAL ID IS NOT EQUAL TO valid dx01 terminal
  EXIT STATE IS security violation
```

NEXT TRANSACTION ID

The data type of the NEXT TRANSACTION ID special attribute is Text. You can modify it with the SET NEXTTRAN statement, but you cannot inspect its contents.

If, at the conclusion of procedure step execution, the Next Transaction ID special attribute contains a non-blank value, the Dialog Manager will attempt to cause the transaction specified to be invoked using the facilities of the teleprocessing monitor (for instance, IMS, CICS and TSO). This feature is used to implement dialog flows to non-CA Gen-generated transactions.

This is an example of how NEXT TRANSACTION ID is used:

```
SET NEXTTRAN TO
    CONCAT("DM01",CONCAT(" ",customer name))
```

Action Diagram Functions

Action diagram functions are special-purpose subroutines invoked as part of an expression. Categories of available functions are shown in the following sections.

Action Diagram Functions

The following list describes the action diagram functions:

- String manipulation-Manipulates and inspects text attributes and portions of text attributes.

  For example, string manipulation functions can extract an employee's last name from the attribute EMPLOYEE Name.

- Domain conversion-Converts a value from one primitive domain to another.

  For example, a domain conversion function can convert a date value to a numeric value.

- Date format conversion-Converts date values from one form to another.
  For example, date format conversion can convert a Julian date (YYYYDDD) to a Gregorian date.
- Date/Time extraction-Extracts a portion of a date or time value.
  For example, a function of this type can extract the DAY portion of CURRENT_DATE.

Diagram functions always take the form:

```
FUNCTION_NAME(parameter_list)
When the action diagram function executes, the value it returns is substituted for
it in the expression.
```

A different set of functions is available, depending on the primitive domain of the expression in which it is included. There are text functions, numeric functions, date functions, and time functions.

Some functions are not available in READ actions. The CA Gen Toolsets clearly identify which functions are not available in READ actions.

Functions can themselves be parameters of functions.

Function Example

Assume that an employee's name is stored as three separate attributes: First Name, Middle Name, and Last Name. The following actions set the value of EMPLOYEE Full Name to Last Name followed by a comma and a space, First Name followed by a space, and the first character of Middle Name followed by a period.

```
SET employee full_name TO CONCAT(TRIM(employee last_name),
    CONCAT(', ',employee first_name))
SET employee full_name TO CONCAT(TRIM(employee full_name),
    CONCAT(' ',CONCAT(SUBSTR(employee middle_name,1,1 ),'.')))
```

- CONCAT is an action diagram function that combines two character strings.
- TRIM is an action diagram function that removes trailing blanks from a character string.
- SUBSTR is an action diagram function that extracts a portion from a character string. It requires a starting position within the string and the length of the string to extract.

If the values of EMPLOYEE First Name, Middle Name and Last Name are Charles, Taze, and Russell, respectively, the value of EMPLOYEE Full Name after the execution of the example statements would be Russell, Charles T.

Advanced Repeating Group Handling

Each repeating group view is defined as either implicitly or explicitly subscripted. This section discusses the distinction in detail.

Implicit Subscripting

If a repeating group view is implicitly subscripted, CA Gen keeps track of position within the group view, so that no logic is needed to do this explicitly.

For instance, when the FOR EACH action is used, the CA Gen-generated procedure automatically jumps to the next occurrence of the subject repeating group view of the FOR EACH after each iteration.

When the TARGETING clause is used and the current occurrence is modified, the generate procedure automatically jumps to the next occurrence of the targeted repeating group view after each iteration.

The following constructs support implicit subscripting and apply only to implicitly subscripted repeating group views:

- For each
- The targeting clause on For Each, Read Each, While, and Repeat actions.
- The conditions are: is full, is not full, is empty, and is not empty.

In most cases, implicit subscripting is sufficient for the handling of repeating group views, particularly during analysis.

Some cases exist where the processing requirement for repeating group views exceeds the capabilities of implicit subscripting. For instance, it is impossible to iterate through two different repeating group views simultaneously using this technique. Likewise, the procedure logic cannot reverse direction or select arbitrary occurrences from the group. In cases where such processing is required, explicit subscripting is needed instead.

Explicit Subscripting

When explicit subscripting is used, the procedure logic must ensure that the proper occurrence of the repeating group view is accessed at any given time.

The current occurrence is identified by its subscript. The value of a repeating group view's subscript is a number that corresponds to its position in the repeating view.

For example, consider a circus application in which the names and relative sizes of the circus's elephants are stored in a repeating group view. It might look like this:

Elephant Name	Size
DUMBO	puny
HUMONGO	large
KING FORTINBRAS THE BRAVE	huge
LEON	medium

If the repeating group view's subscript is set to 1, the occurrence for dumbo is current. If it is set to 4, the occurrence for LEON is correct.

Subscript values are set using the SUBSCRIPT option of the SET action. Each explicitly indexed repeating group view has a single subscript called SUBSCRIPT OF repeating-group-view. Assuming that the elephant list is stored in the repeating group view Elephants, the following action points to the occurrence for KING FORTINBRAS THE BRAVE:

```
SET SUBSCRIPT OF Elephants TO 3
```

When iterating through an explicitly subscripted repeating group view, the procedure logic must increment the subscript explicitly and must test for the end of the group explicitly. In the example shown in the following sample code causes the size of each elephant in the repeating group view to be set to elephantine.

Explicit Subscripting

```
SET SUBSCRIPT OF elephants TO 1
WHILE
     SUBSCRIPT OF elephants IS LESS THAN OR EQUAL TO LAST OF elephants
          SET elephant size TO elephantine
          SET SUBSCRIPT OF elephants TO SUBSCRIPT OF elephants +1
```

Had the repeating group view Elephants been defined as implicitly subscripted, the statement in the following sample code would accomplish the same result.

Implicit Subscripting

```
CHECK HOLIDAY
     IMPORTS:     Entity View incoming customer
     EXPORTS:     Entity View work status
     LOCALS:      Entity View work date
                  Entity View order_item
SET work date request TO HOLIDAY TEST
USE date_services WHICH IMPORTS incoming customer
                               work date
                  WHICH EXPORTS work status
     IF work status flag IS EQUAL TO H
          FOR EACH elephants
     EXIT STATE IS shipment_requested_on_a_holiday
     SET elephant_size TO "elephantine"
```

Note: Using implicit subscripting eliminates the need to initialize, maintain, and test the subscript value.

Certain action diagram constructs apply only to explicitly indexed repeating group views:

- FOR action
- SUBSCRIPT OF repeating-group-view special attribute
- LAST OF repeating-group-view special attribute
- It identifies the highest subscript value for which a repeating group view occurrence exists.
- MAX OF repeating-group-view special attribute
- It identifies the maximum cardinality of the repeating group view.

The special attributes SUBSCRIPT OF, LAST OF, and MAX OF are used in numeric expressions.

SUBSCRIPT OF and LAST OF can be the target of a SET statement.

External Action Blocks

A procedure may need sometimes to make use of logic that is not defined using CA Gen. For example, you may wish to use an installation standard date manipulation routine that takes into account its own holidays and work schedules.

In another case, you may need to access data bases and files that are not yet supported by CA Gen. Access to logic not originally specified using CA Gen is achieved by using External Action Blocks.

An external action block is an action block that has only three components:

- Import Views
- Export Views
- Designation that it is external

An external action block contains no action statements of its own.

The designation external on the action block definition indicates that a non-CA Gen-generated program will be used to obtain the data for the action block's export view.

Action blocks of this sort can be the object of a USE statement in normal Procedure Action Diagrams or Blocks.

During construction, CA Gen ensures that the program referenced by the external action block is properly referenced by CA Gen-generated programs.

The following sample code illustrates a CA Gen-defined Procedure Action Diagram that references an external date manipulation routine called Date Services:

Procedure Action Block Referencing an External Action

```
WHILE restart work restart_count IS LESS THAN target work restart count
AND returned work request_type IS NOT EQUAL TO E
SET restart work restart_count TO restart work restart_count + 1
USE get_transaction
     WHICH EXPORTS:  Work View returned work
                     Entity View returned customer
NOTE
     ... NOW, you must be positioned at the first customer transaction not
     yet processed. If this is not a restarted execution, that will be the
     first transaction on the input file; if it IS a restart, it will be the
     transaction right after the last one for which a commit took place.

WHILE commit_ief_supplied count IS LESS THAN 100
AND returned work request_type IS NOT EQUAL TO E
EXIT STATE IS requested_operation_complete
SET message work result TO OPERATION COMPLETED
NOTE
     CREATE CUSTOMER, UPDATE CUSTOMER, and DELETE CUSTOMER are
     Elementary Processes for which Process Action
     Diagrams were created during BAA.
CASE OF returned work request_type
CASE A
USE create_customer
     WHICH IMPORTS: Entity View returned customer
     IF EXIT STATE IS NOT EQUAL TO requested_operation_complete
     SET message work result TO NOT ADDED
CASE C
USE update_customer
     WHICH IMPORTS: Entity View returned customer
     IF EXIT STATE IS NOT EQUAL TO requested_operation_complete
     SET message work result TO NOT CHANGED
CASE D
...
```

The following sample code shows the external action block definition of the Date Services routine:

External Action Block for Date Services

```
DATE_SERVICES
     IMPORTS: Entity View incoming work date
              Entity View incoming work request
     EXPORTS: Entity View exported work status
EXTERNAL
```

Design Block Mode Presentation Logic

The following list shows the main elements of handling a block mode dialog:

- Using Clear Screen Input
- Clearing the screen for the next transaction
- Validating input from a screen
- Handling scrolling of repeating group views that are too large to display on a single screen.

Clear Screen Input

Where it can save screen displays and so be efficient for the user dialog, it is possible to provide some import data directly from the TRANCODE or other field of a previously displayed screen. In the case of an external procedure, some output can be matched with the import view.

This can be done using Clear Screen Input, but only for the first step of a procedure.

Note: The Clear Screen Input feature of CA Gen is described fully in the *Toolset Help*.

The CA Gen Dialog Manager automatically populates the import view for the destination procedure, but a procedure that can be initiated in this way may need to test the command to determine whether this situation has occurred.

Before the last display of the screen during the execution of a transaction, a procedure step can populate the TRANCODE field with the command and required data value. The required data value is usually a key that will steer the execution of the following process. This will give the user the option to transfer directly to the procedure of another screen. The user then needs only to press the Enter key to proceed.

More information:

Designing the Procedure Dialog (see page 71)

Clear the Screen

When a transaction is complete or if the user requests the screen to be cleared, to allow the user to continue with the next transaction, there should be logic to clear the display of the customer data just added, by moving an empty local view to the export view.

If the next screen needed by the user is different from the current screen, then the procedure must set an exit state that ensures that a clear screen will be displayed as a result of initiating a flow to the appropriate procedure.

Validate Input From a Screen

Design of a host terminal dialog seeks to minimize interactions between the host and the terminal, so the logic is specified to perform all possible validation of the data imported from the screen.

Any field found to be in error is indicated using the MAKE ... ERROR statement.

When the screen is redisplayed, the cursor can be automatically placed at the start of the first field indicated as an error.

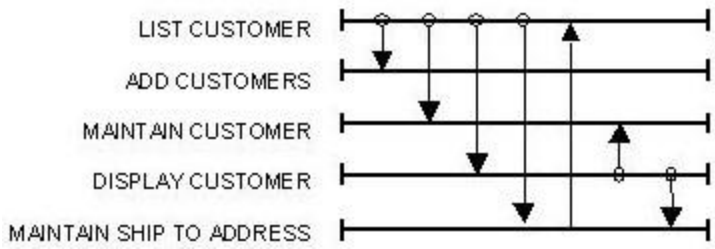
Handle Scrolling on a Screen

Scrolling through the display of a single repeating group can be handled automatically, providing that the contents of the repeating group will never be too large for the generated transaction to handle.

For a large group, or for more than one group on a screen, a procedure can handle scrolling explicitly by tracking the values of key fields used in the last display of the screen, and explicitly indexing and retrieving data as required by scrolling commands entered by the user.

The following illustration is an example of a procedure called List Customers. This procedure produces a list of CUSTOMERS.

The following illustration shows how the procedure, based on the value of an action code next to each, flows to a process-implementing procedure.



The following screen illustrates the screen layout for the procedure List Customers:

```
TRANCODE                                                          MMDDYY
                              CUTOMER LIST                        HH:MM:SS
                                                                  <USERID>

STARTING CUSTOMER NUMBER: 9999999999  LIST SEQUENCE (NAME OR NUMBER) XXXXXX
STARTING CUSTOMER NAME: XXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ACT Number       Name                          Status     Credit Rating
X 9999999999     XXXXXXXXXXXXXXXXXXXXXXXXX       XX           9
X 9999999999     XXXXXXXXXXXXXXXXXXXXXXXXX       XX           9
X 9999999999     XXXXXXXXXXXXXXXXXXXXXXXXX       XX           9
X 9999999999     XXXXXXXXXXXXXXXXXXXXXXXXX       XX           9
X 9999999999     XXXXXXXXXXXXXXXXXXXXXXXXX       XX           9
X 9999999999     XXXXXXXXXXXXXXXXXXXXXXXXX       XX           9
X 9999999999     XXXXXXXXXXXXXXXXXXXXXXXXX       XX           9

ACT  X-Full Screen Display  M-Maintain Customer Details  S-Maintain Ship To Info
Command ===>
F3=EXIT    F7=PREVIOUS    F8=NEXT
```

The user can choose between displaying the customers in NAME or NUMBER sequence.

The user can request a DISPLAY by entering DISPLAY in the Command field or by pressing the function key associated with the command value "display." This causes LIST CUSTOMERS to build a list starting with the customer specified in the STARTING CUSTOMER fields.

The user can requests a NEXT operation by entering a command or pressing the appropriate function key. This will cause the last populated item in the incoming repeating group view to be used as the starting point of the list. Assuming that the example screen does not use the automatic scrolling feature, this will result in the last entry on the screen before the NEXT operation appearing at the top of the list after the NEXT operation.

The NEXT action in this example works whether or not the automatic scrolling feature is used.

If automatic scrolling were specified for the screen, the CA Gen Dialog Manager would handle all NEXT operations until the user displayed the final item of the repeating group.

If the user were to enter NEXT again, the Dialog Manager would pass the next command to the Procedure Action Diagram for processing.

The following sample code, View Definitions for List Customers, shows the views for the procedure List Customers:

```
Import Views

------------------------------------------------------------------------------
Group View IN_REPEATING
     Cardinality Min: 1  Max: 10  Avg: 10
     Optional Import
View IMPORT_REPEATING of entity CUSTOMER
     Optional_Import
     Attributes:
          NUMBER
          NAME
          STATUS
          CREDIT_RATING
View IMPORT_REPEATING of work group LINE_ITEM
     Attributes:
          LINE ACTION
View IMPORT_STARTING of entity CUSTOMER
     Attributes:
          opt NUMBER
          opt NAME
View IMPORT_STARTING of work group LINE_ITEM
     Attributes:
          opt SEQUENCE
------------------------------------------------------------------------------
Export Views
------------------------------------------------------------------------------Gro
up View OUT_REPEATING
     Cardinality Min: 1  Max: 10  Avg: 10
View EXPORT_REPEATING of work group LINE_ITEM
     Attributes:
          LINE ACTION
View EXPORT_REPEATING of entity CUSTOMER
     Attributes:
          NUMBER
          NAME
          STATUS
          CREDIT_RATING
View EXPORT_STARTING of work group LINE_ITEM
     Attributes:
          NUMBER
          NAME
------------------------------------------------------------------------------
Local Views
------------------------------------------------------------------------------Vie
w LOCAL_STARTING of entity CUSTOMER
     Attributes:
          NUMBER
```

```
          NAME
--------------------------------------------------------------------------------
Entity Action Views
--------------------------------------------------------------------------------
View of entity CUSTOMER
      Attributes:
           NUMBER
           NAME
           STATUS
           CREDIT RATING
```

Procedure Action Diagram for List Customers (Part 1)

```
LIST_CUSTOMER
     IMPORTS:        Group View in_repeating
                     Entity View import starting customer
                     Work View import_starting line_item
     EXPORTS:        Group View out_repeating
                     Work View export_starting line_item
                     Entity View export_starting customer
     LOCALS:         Entity View customer
     ENTITY ACTIONS: Entity View customer
     EXIT STATE IS requested_operation_complete
     SET export_starting line_item sequence TO import_starting line_item sequence
     MOVE import_starting customer TO local_starting customer
          IF COMMAND IS EQUAL TO next
               FOR EACH in_repeating
                    IF import_repeating customer number IS NOT EQUAL TO 0
                    MOVE import_repeating customer TO local_starting customer
                    MOVE import_repeating customer TO export_starting customer
          IF import_starting line_item sequence IS EQUAL TO NAME
               READ EACH CUSTOMER
                    TARGETING out_repeating FROM THE BEGINNING UNTIL FULL
                    SORT BY ASCENDING customer_name
                    WHERE DESIRED customer_name
                         IS GREATER OR EQUAL TO local_starting customer_name
                    MOVE customer TO export_repeating customer
          IF out_repeating IS EMPTY
          EXIT STATE IS no_more_customers
          ELSE
               READ EACH customer
                    TARGETING out_repeating FROM THE BEGINNING UNTIL FULL
                    SORT BY ASCENDING customer_name
                    WHERE DESIRED customer_number IS GREATER OR EQUAL TO
                         export starting customer number
                    MOVE customer TO export_repeating customer
...
```

If the user requests a PROCESS operation, the first item in the list that has a valid line item action is used to trigger a process-implementing procedure (based on the setting of EXIT STATE in the action diagram in Procedure Action Diagram for List Customers (Part 1) and the flows shown in Dialog Flow Diagram for List Customers.

Assume that EXPORT_STARTING_CUSTOMER is specified as Data Sent on each of the flows from List Customers. Each of the flowed-to procedures will have their import views populated with the number of the selected customer when they begin execution if the flow action is Execute first. They will have their screens displayed if the flow action is Display First. This is illustrated in the following sample code.

Procedure Action Diagram for List Customers (Part 2)

```
...
               IF out_repeating IS EMPTY
               EXIT STATE IS no_more_customer

          ELSEIF COMMAND IS EQUAL TO process
               FOR EACH in_repeating
                    TARGETING out_repeating FROM THE BEGINNING UNTIL FULL
               MOVE import_repeating line_item TO export_repeating line_item
               IF import_repeating line_item line_action IS NOT EQUAL TO SPACES
               MOVE import_repeating customer TO export_starting customer
               SET export_repeating line_item line_action TO SPACES

               CASE OF import_repeating line_item line_action
               CASE M
               EXIT STATE IS link_to maintain_customer
               CASE S
               EXIT STATE IS link_to maintain_ship_to
               CASE X
               EXIT STATE IS link_to display_customer
               OTHERWISE
               EXIT STATE IS unknown_line_action
               SET export_repeating line_item line_action TO ?
               MAKE export_repeating line_item line_action ERROR

          ELSE
          EXIT STATE IS unknown_command
```

In the sample code, Dialog Flow Diagram for List Customers, notice that all flows are implemented as links.

Notice also that the logic in the Procedure Action Diagram sets the EXPORT LINE_ITEM_ACTION of the selected line item to spaces (line 66). Assuming that the flow action for the return of each link is Execute first, List Customers will be re-executed whenever one of the process-implementing procedures returns to it. If List Customers finds another valid line item action, it will trigger another flow.

To a user entering multiple valid line actions, it will appear that all those actions are processed without any intervening displays of the screen for List Customers.

More information:

Designing the Procedure Dialog (see page 71)
Designing Screens (see page 101)

Chapter 8: Verifying the Design

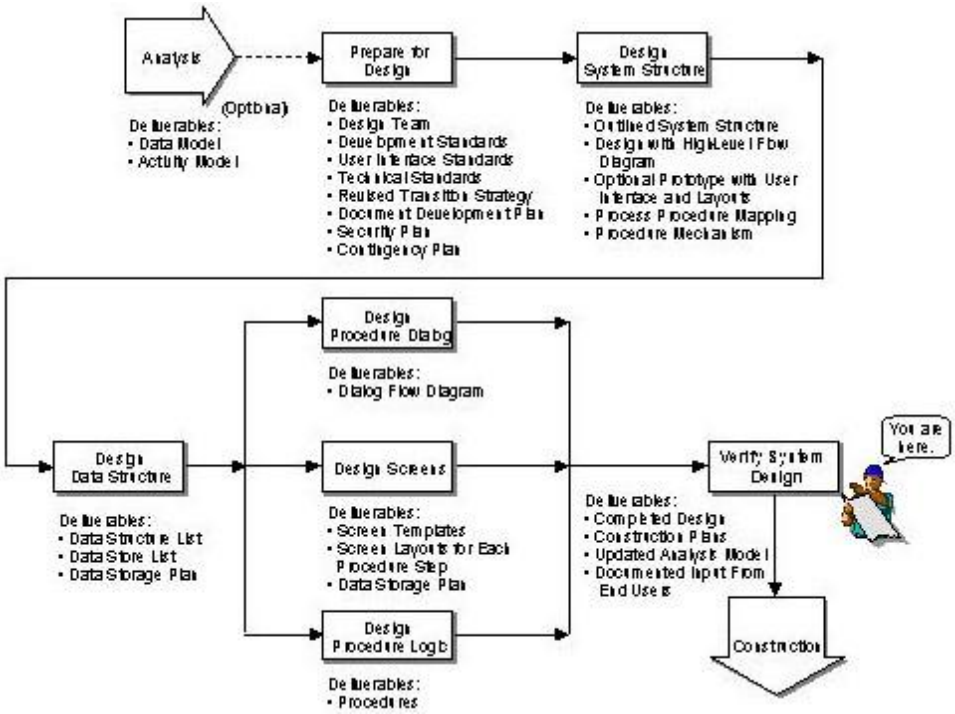
Examine the design model at intervals throughout design to verify the correctness and completeness of the work so far performed. Toward the end of design, it is especially important to make sure that the model is ready for continued refinement during the next phase of system development.

Apply verification techniques in a systematic manner before proceeding with construction activities. CA Gen will not allow inconsistent objects to be used in construction.

CA Gen provides the basic information to support these activities, but verification can never be completely automated. Designers and user team members must use their own judgment as required.

Design Process

The following illustration lists the deliverables of verifying the system design and shows where you are in the block mode design process.



Prototyping

Prototyping is a technique that can support a variety of analysis and design activities. Prototyping performance is a form of simulation. It can also be used to explore technical design and performance issues.

In analysis, prototyping can be used to explore a business requirement by simulating the design of a system that might support that requirement.

In design, prototyping is commonly used to check and improve the user interface and dialog structure.

Prototyping using the CA Gen Construction Toolset can be:

- Simply displaying layouts for user acceptance.
- Simulating the execution of a dialog.
- Constructing of all or part of a working system with which users can interact and evaluate.

The system can be executed in a test environment. If some aspect of system performance is to be verified, it can be executed in a technical environment that replicates some part of the planned operational environment.

CA Gen is a prototyping tool because of the speed and ease with which a design can be modified and regenerated to meet the need for a change or to explore a design alternative. CA Gen lets you make changes that can be related to the business requirements defined during analysis. It is not necessary to use the full operational environment to execute parts of the system.

Check for Completeness

Verifying the design also involves inspecting the design results for completeness. This verification assures that each elementary process has been faithfully implemented by a procedure.

CA Gen's Consistency Check facility handles a great deal of the required checking automatically. Consistency checking ensures that:

- Each procedure has been fully defined.
- A Procedure Action Diagram details each procedure.
- Each host/terminal procedure has a screen.
- The Consistency Check facility automatically checks these rules.

- Each elementary process has been implemented by a procedure.
- This can be achieved by checking a Procedure Definition report or a Where implemented encyclopedia report.

Check for Correctness

Perform correctness checking to ensure that:

- Each process implementing procedure accurately reflects the requirements of each of its client processes.
- Compare the process definition, expected effects, and Procedure Action Diagrams to ensure that procedures faithfully reflect the processes they implement. Someone other than the designer who built the procedure using the techniques of facilitated sessions, walkthroughs, and inspections should perform this comparison. These techniques are described in Facilitated Sessions, Walkthroughs, and Inspections.
- Each design object is properly formed. These design objects include screens, action diagrams, and dialog flows.
- The Consistency Check facility is used to check that:
  - Each dialog flow is initiated by an exit state.
  - Each import and export view of an online procedure step is placed on a screen or defined as hidden.
  - Each Procedure Action Diagram has at least one action statement.

Check for Consistency

Consistency checking verifies that the design model conforms to CA Gen's rules and conventions. Where a model does conform to these rules and conventions, it is said to be in a consistent state.

CA Gen automatically checks the consistency of the model at the data, activity and interaction levels. CA Gen can selectively perform those checks that are applicable to design. However, a model that is in a consistent state may still not fully and accurately represent the required design.

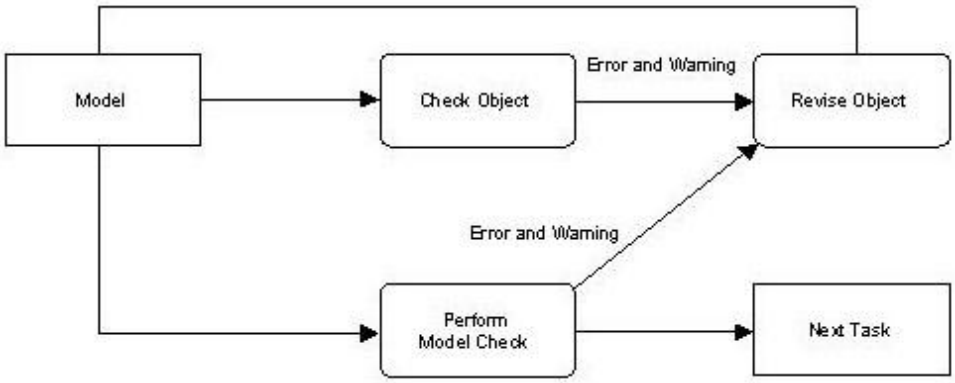
When to Perform Consistency Checks

It is useful to perform CA Gen consistency checks on the design objects at regular intervals during design, and especially before completing an aspect of design and beginning a new design activity.

Check for consistency before performing the following tasks:

- Transforming data
- Completing the dialog flow diagram
- Completing a screen design
- Completing a procedure step action diagram

The following illustration shows consistency checking.



Consequences and Levels of Inconsistency

The Design Toolset automates consistency checking at the level of a model and for objects you select. It reports the rules to which elements of the model do not appear to conform.

Each problem encountered is classified according to likely severity, and this level of severity affects whether system development is able to proceed.

Levels of Severity

The following list describes the individual levels of severity:

- Warning-Conditions indicating that context is important.

  For example, if X is required before Y, a warning on Y appears, but it may still be possible to perform transformation and system and database generation.

- Severe Warning-Conditions in the model that have a greater degree of severity than warnings.

  In most instances, severe warnings cause errors during system implementation.

- Error-Conditions that must be corrected before the model moves to the next stage of system development.

Resolve Inconsistencies

When CA Gen reports inconsistencies at all levels of the design model, it lists the current anomalies at a given point.

An inconsistency at the object level may have a cascade effect and generate other consequential inconsistencies in different parts of the model. It is therefore sensible to approach the resolution of inconsistencies in a structured and ordered manner, concentrating first on the object being checked, its immediately associated objects, then all the objects in the Dialog Flow Diagram.

Following this approach maximizes the likelihood of resolving consequential inconsistencies, and minimizes iterations of the checking and revision cycle.

Facilitated Sessions, Walkthroughs, and Inspections

Facilitated sessions are meetings of business staff and developers to explore and verify the structure of the system, or some aspect of system design. These sessions are widely referred to as Joint Application Design or JAD sessions.

You can incorporate many general group working techniques into your facilitated sessions. CA Gen can be used to record design decisions and to prototype parts of the system.

As a final test of the correctness of the design, a series of structured walkthroughs is recommended.

A structured walkthrough is a formal, step-by-step inspection of design results.

The following list details the levels of walkthroughs:

- First level—You can present the detailed design results to the analysts who built the business model on which it is based.
- Second level—You can present a subset of the design to the most knowledgeable intended users of the system, using presentation or prototyping techniques.

  Such walkthroughs not only verify the completeness and correctness of the design, but serve to point out potential problems in usability as well.
- Third level—Where technical performance has been identified as an issue, you can present the results to technical specialists who may identify performance problems or advise on use of the planned technical facilities.

You should address the problems identified during design verification before considering the design complete. You can then use the verified design in system construction and transition.

Consolidate the System Structure

Throughout system design, the different aspects of design interact closely. For example, changes to a dialog cause changes to layout designs, or a decision on the distribution of data may require a new control procedure.

Using CA Gen helps ensure that system components remain compatible. When the developers and business staff are happy with a design that is consistent, the design is complete. The design documentation can then be finalized, and a compatible set of system development plans, technical facility development, and performance requirements refined to reflect the design.

Refine the Project Plan

You should ensure that the design results are reflected as needed in the system development plan after completing any major aspect of design. This check is especially important when finally verifying a design model.

In the case of a large system with many procedures, you may need to establish the sequence in which procedures and data are to be implemented.

The implementation sequence is determined by the following criteria:

- Business priority of the processes supported by each procedure.
- Availability of the data used by procedures but created by other procedures.
- These data structures and data-creating procedures need to be implemented first.

Results of Design Verification

The results of design verification are:

- Verified design is a defined system that can immediately be constructed.
- Plans and preparations to commission that system with minimum business and technical risk.
- Changes to the design model to reflect user requests or issues identified during prototyping.
- If needed, changes to the analysis model to reflect new requirements identified during prototyping.
- (Optional) Minutes of facilitated sessions, prototyping sessions, and walkthroughs.
- (Optional) Sequence in which procedures are to be implemented within the business system

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W