CA Gen

Client Server 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

This chapter describes the design process and the tools CA Gen provides for design. 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 users are happy with the design.

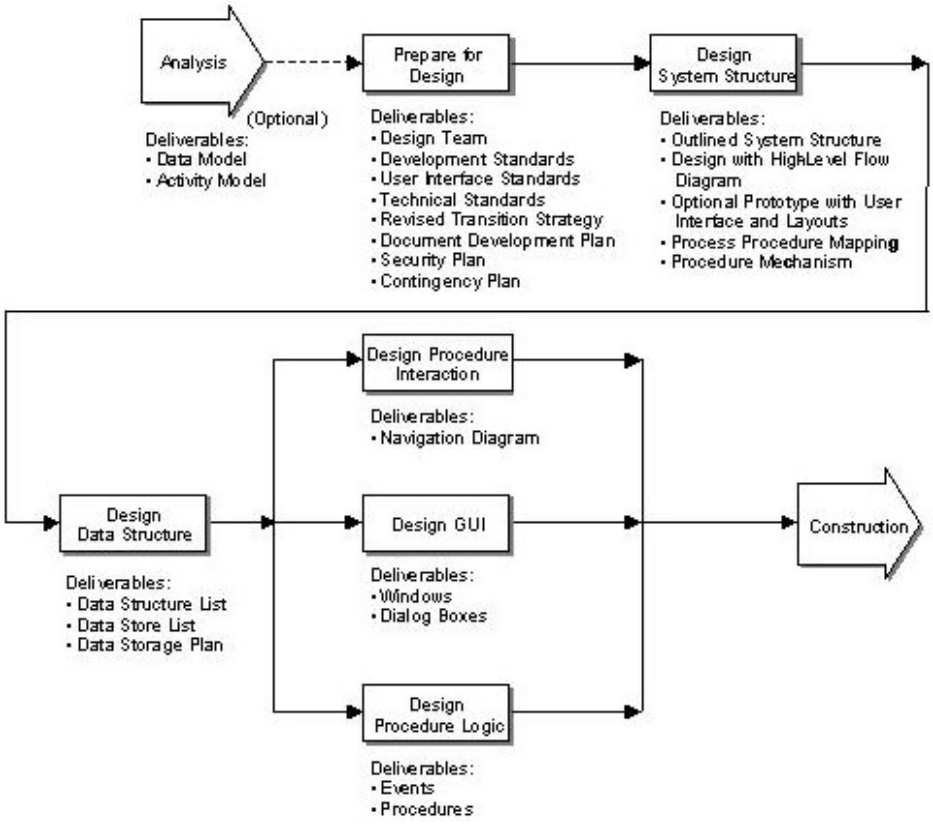
The design is then constructed and tested.

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

Design Process

The client/server design process may begin with the results of analysis and ends with a system design ready for construction.

The following illustration shows the sequence of activities that make up the client/server design process.



Analysis is optional for client/server design.

Data structure design may be done any time after system structure design.

Objectives of Design

Design activities are aimed at designing and confirming the external aspects of the system with end user representatives.

You must ensure that the external design of the system is suitable for the proposed business users, locations, and broadly suitable in technology for the expected volume and speed of response. Detailed technical issues are usually deferred until the users agree with 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 needs to keep track of customers. The analyst may have defined an elementary process, called Add Customer, in which the rules for adding a new customer are specified to support this requirement.

You must consider the following questions during design:

- 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?
- Should a client/server environment be used, and if so, which system components should be implemented in a client or server role?

Other objectives of design (constraints on the form of the system) include ensuring that the development of the new system, where possible, exploits and protect investment in other systems. This requirement may involve designing procedures to make the best use of current (sometimes called legacy) systems and data. It may also be necessary to design the system to work with some other system that is currently planned or under parallel development.

CA Gen Design Tools

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	Defines procedures and designing dialog.
Action Diagram	Designs procedure logic.
Data Structure List and Data Store List	Displays the physical structure of the database and modifying that physical structure.
Procedure Synthesis	Automates the construction of procedure action diagrams.
Structure Chart	Shows the relationship between implemented action blocks and procedure action diagrams.
Screen Design	Designs 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 chapter explains how to assemble the design team, review analysis results, and establish system standards. Preparing for design involves dealing with the technical and non-technical issues to be considered before beginning design work.

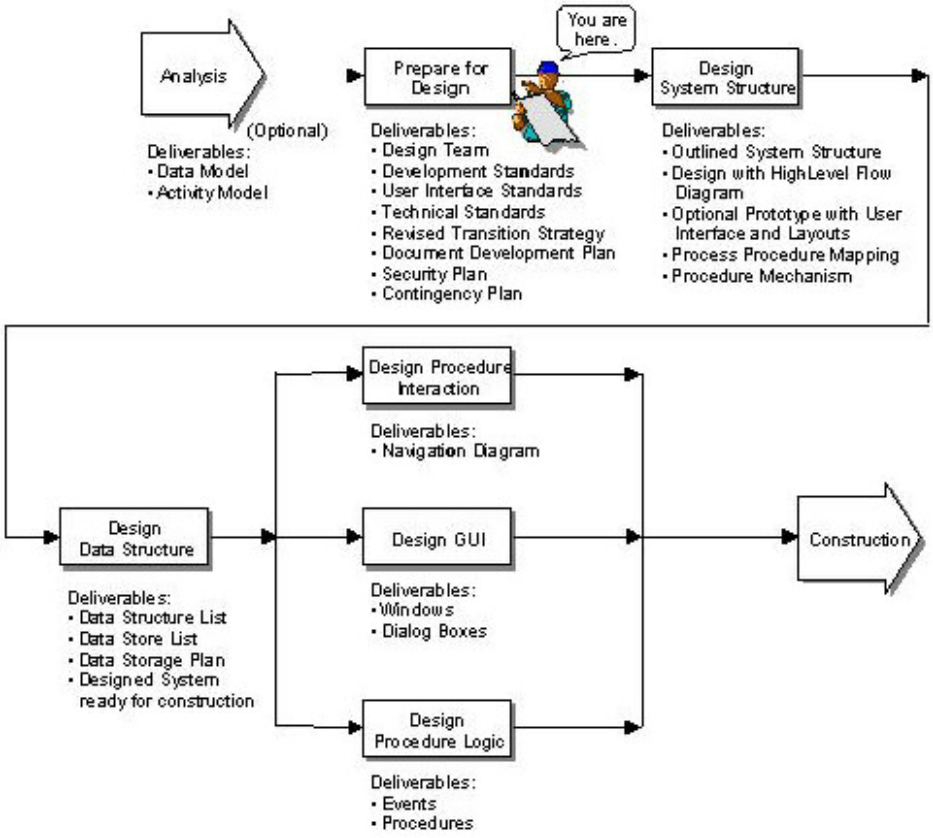
Prerequisites

Completing analysis before beginning design is optional.

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

Design Process

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



Organizing for Design

The membership for the design organization should come from the:

- Information Management function-Systems designers should be experienced with system development in general and design techniques in particular.
- 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 access will help ensure that the resulting system meets the organization's quality standards.

Understanding 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: CA provides other guides that describe the tasks, tools, and techniques used during analysis, in particular the *Analysis Guide*.

Reviewing the Business System Definition

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

Note: For a description of 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:

- What business systems to create to support the business requirements

- Which elementary processes to implement 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 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.

Specifying 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.

Tasks for defining business systems in CA Gen include:

- 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.

Developers specifying business systems to be implemented in a client/server environment may find that:

- Processes that are closely clustered by their creation and maintenance of data that is widely used by many other processes are candidates for support by business systems that include server procedures that can be used by many client procedures.
- Processes that need many entity types (possibly created by several different clusters) are candidates for support by business systems composed of client procedures that reuse data maintained by a combination of current systems and provided by common server procedures.

Setting Development Standards

One aim of design is to create a consistent user interface for the system being constructed. This objective 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:

- Procedure style
- User interface
- Reusable logic
- Quality

Choosing 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 33)

Setting 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).

For a graphical user interface (GUI), guidelines need to be set within the project for the modality of the interface. If the interface is to be very open, and therefore "modeless," standards need to be set for the point at which a modal interface should be enforced and for what user actions.

Standards can be set only for delete actions, or delete and create actions.

More information:

Designing the Graphical User Interface (see page 117)

Setting 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.

Setting 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 controls:

- Commands
- Function keys
- Accelerator keys
- Windows and dialog box formats

- Field edit properties
- Field prompts
- Common exit state definitions
- Dialect

Command 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.

You can also specify standard synonyms for menu items in windows. For example, you might assign MODIFY the synonym M to reduce the actions required to invoke it. Whether the user enters M or selects MODIFY, the logic associated with MODIFY will execute.

More information:

Designing the Procedure Interaction (see page 89)

Function Key Assignment Standards

Function keys (also referred to as "Product Function keys," and "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 F10 will have the same effect as entering the ADD command.

If a function key represents a particular command, it should represent that same command in all procedures that use it. For example, if F1 invokes the CHANGE command in one procedure and F10 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, F1 means Help.
- Default—You can override a default function key. For example, procedures that support a Display command will all use F4 for display, but a procedure with no Display capability can use F4 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 Interaction (see page 89)

Accelerator Key Standards

In a graphical user interface, the same principle that applies to standards for function keys should be applied to accelerator keys (also known as "hot keys") for menu items and pushbuttons. The accelerator key (for example, Ctrl + D for Detail) that is assigned to a command should be used wherever that command is used.

More information:

Designing the Graphical User Interface (see page 117)

Window and Dialog Box Layout Standards

The general format of windows and associated dialog boxes should remain consistent throughout a system. Windows should have a common look and feel.

CA Gen supports default font and color usage for windows and window components (also called "controls"). These defaults apply unless overridden by the designer for a specific control or window.

More information:

Designing the Graphical User Interface (see page 117)

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 will lead to confusion.

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

Field Prompt Standards

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

For example, if the Customer Number attribute appears on two displays, a user has a better chance of recognizing it as the same value if they see a consistent label (such as CUSTOMER NUMBER) rather than two different labels (such as CLIENT NUM: on one and CUST NUM: on another).

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 Window Design functionality.

More information:

Designing the Graphical User Interface (see page 117)

Common Exit State Definition Standards

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

We recommend that you establish standard exit state definitions for outcomes or results common to many procedures, especially successful outcomes.

For example, if processing completed successfully, 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 the Procedure Interaction (see page 89)

Specifying 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, consult CA support for information on the options available.

Confirming 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:

- Who will be responsible for transition? What information will they need?
- 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. This is especially critical where client/server technology is to be used.

- How will the newly implemented entities be populated initially in new databases, 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.
- How will the newly implemented system 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. The 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.

Preparing a Documentation Plan

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

- Online help
- Training
- Reference and technical guides

Defining 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 for the help system.

Defining 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.

Defining 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.

Capturing 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

Using Special Attributes

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

Special attributes include:

- Current Date
- Error Message
- Transaction Code

More information:

Designing the Procedure Interaction (see page 89)
Designing the Procedure Logic (see page 163)

Using 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 163)

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

Using 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:

- 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.

Preparing a Security Plan

You need to prepare a security plan to ensure:

- Security of design work
- Security within applications
- Privacy

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

More information:

Security (see page 229)

Ensuring 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.

Ensuring Security within Applications

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

Access to data can depend on the following criteria:

- Type of data
- Data values
- Associations with other data
- Organizational unit
- Location

Access to processes can depend on the following criteria:

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

Ensuring 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.

Preparing 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, which 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 procedures listed in the following:

- Fallback—Allows business activities to 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 may 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

Considering 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.

Results of Preparing for Design

The results of preparing for design are:

- Established design team
- Application development standards
- Procedure style
- User interface
- Reusable logic

- Quality
- Technical standards
- Transition strategy (revised)
- Document development plan
- Security plan
- Contingency plan

Chapter 3: Designing the System Structure

This chapter describes how to determine the procedures and outline dialog for the design, based on processes and user tasks detailed using analysis techniques.

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.

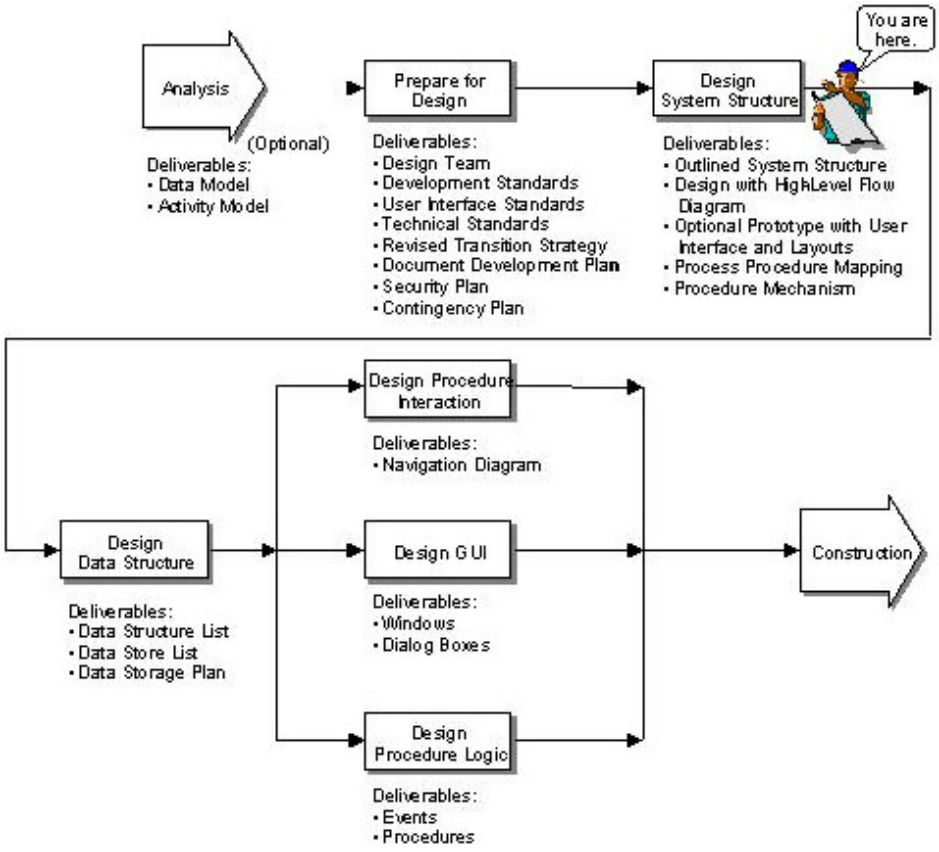
After you have identified the need for a procedure, whatever its purpose, you use the same techniques for that procedure's detailed definition and construction.

Prerequisites

You need to have completed the tasks in preparing for design before beginning system structure design.

Design Process

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



Audience Analysis

You need to create a profile of the potential users of the application to be developed.

The user profile describes these characteristics:

- User experience
- Frequency of task execution
- Geographical location
- Volatility in the work environment

User Experience

How experienced is the user with the job to be performed? How well developed is the user's information technology skill set?

Often in a single environment, users will range from raw recruits to seasoned veterans, and from the technology-impaired to computer genius. Clearly, this yields several possible user permutations.

Ensure that the application is equally usable by users of all levels of experience and knowledge. This may mean providing more than one version of the user interface to cater to different users' needs.

Frequency of Task Execution

If some users perform a certain task only occasionally, they may forget how to use the application from one execution to the next. In such cases, special guidance must be made available each time the application is run.

Frequent users, on the other hand, will undoubtedly become very comfortable with the interface once the initial learning phase is complete. Often, such users will greatly benefit from a fast path through the dialog.

Geographical Location

If the application is to be used in more than one country or region, cultural differences will have to be kept in mind.

Cultural differences can manifest themselves in many ways, including different character sets and keyboards, the relevance of examples, different terminology, and different idioms, to name a few.

From this, it can be seen that different classes of user will have different requirements. For example, some users may require a different interface or extra help.

It is a good idea to interview several different users when performing user task analysis to ensure that you accommodate all relevant issues of experience, usage and culture.

Volatility in the Work Environment

Work environments fall into two 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 work flow 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:

- It allows the user limited control because the system determines the sequence of actions.
- It uses few menus.

A loosely structured dialog has the following characteristics:

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

User Task Support

A user task is some identifiable activity, often part manual and part computer-supported, that is carried out by the user to achieve a specific result. User tasks represent the human activities involved in carrying out the work of elementary processes.

A user task is typically triggered by a business event and may consist of several subtasks.

A key feature of a task specification is that the user can tell when the task has been completed. That is, each task has a clear beginning and a clear end. It may, however, be subject to interruption.

A user task typically requires several inputs. It may involve a complex arrangement of steps, not just a linear sequence. For example, assume that an order processing task requires an order form as input, and requires the use of a price list during the task. While this task sounds clear, in practice it can be fraught with interruptions and exceptions, such as:

- Poor handwriting or missing information on the form may require contact with the purchaser to clarify details.
- Some purchaser requests may be open.
  ("Let me know when you can deliver the goods.").
- Some purchaser requests may be impossible and require resolution.
  ("Deliver before 7 a.m.")
- The total price may exceed the authority of the person processing the order, requiring that person to seek higher authorization.

Objectives of User Task Support

User task support is a project-level objective to help ensure that users' interactions with an application are effective and comfortable and they can complete their tasks as easily as possible.

User task support is achieved using two techniques:

- User task analysis
- User task design

These techniques result in a user interface design that focuses more on the work the user does and the way the user does it rather than on the computer processing to be undertaken.

Rationale for User Task Support

User task support focuses the development on a client-level application that supports the work flows of end users. To that end, user task analysis is not limited to the man/machine interactions. It includes looking at the work flow of the user, under both normal and abnormal conditions, to determine the required behavior of an application that supports that work flow.

User task analysis helps you to understand:

- Tasks the user performs.
- Ways the user performs the tasks.
- Appropriate application behavior to optimize the user's performance of the tasks.

User task design helps the developer to create the user interface design required to produce the desired behavior.

The combination of user task analysis and design yields the following benefits:

- The user interface should accurately reflect the user's tasks.
- The application design should meet the business requirements.

Principles of User Task Support

During user task analysis, the application developer looks at application requirements from the standpoint of the user's objectives and working conditions, not that of the information architecture. Once the tasks have been identified and documented in detail, the client application, including its user interface, can be designed to reflect them.

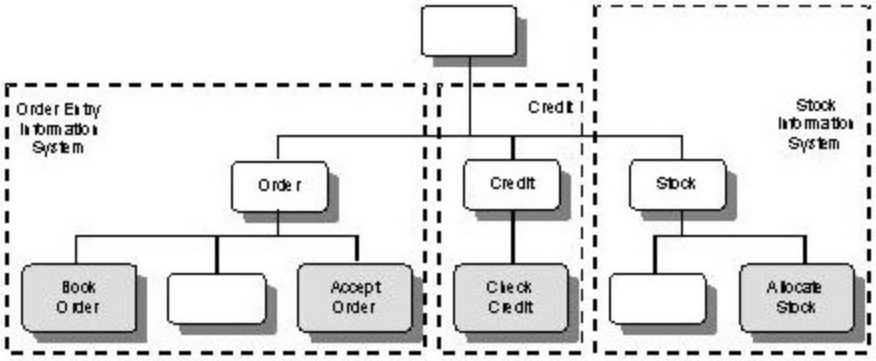
User task analysis is used to answer the following questions:

- Who is the user?
- What tasks does the user perform?
- What is the relationship between the tasks?
  Can one task interrupt another?
- Is there any sequence to the tasks?
- How are interruptions, exceptions and problems handled?
- What business event triggers each task?
- What objects does the user manipulate to carry out each task?
- How can the usability of the final user interface be measured?

Business Impact of User Task Support

It is important to remember that the user's view of a business process may differ significantly from the formal organization structure and current systems that tend to support that structure. To satisfy his customer, the user may need to cross many organizational boundaries, involve several organizational units and make use of multiple functional systems.

This difference in viewpoints is illustrated in the following illustration.



This illustration shows how the business is organized into departments that traditionally have been supported by separate information systems:

- The Order Processing Department has an Order system.
- The Accounting Department has a Credit system.
- Warehousing has a Stock system.

These systems often will have been identified by a top-down analysis of the business processes such as information engineering. While this approach is still valid today, it does not yield the optimum solution on its own.

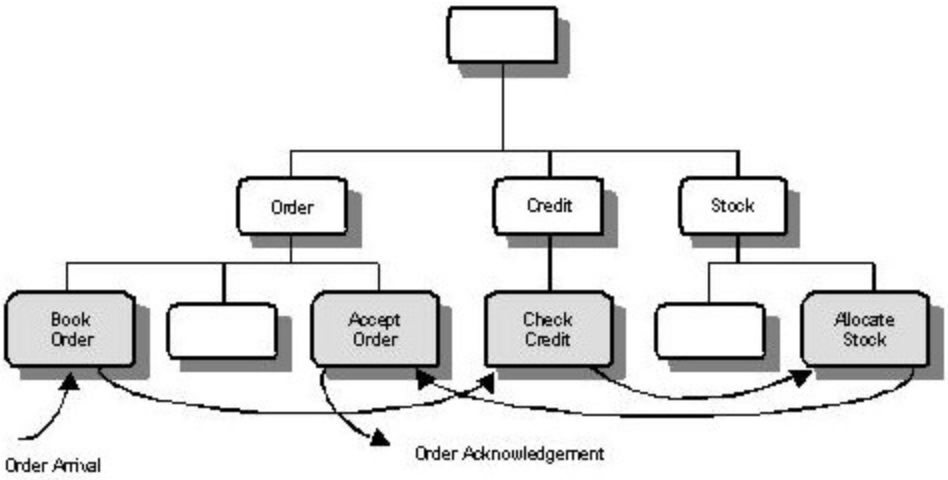
To completely satisfy the customer request, "Please supply me some goods," a variety of actions must be taken by each of these departments. While the separate information systems support the work of the individual department, none of them focus on the end-to-end process of responding as effectively and as quickly as possible to the customer. Only one department, Order Processing, may have contact with the customer. Often this contact is indirect through a sales person or the mail. The apparent "customer" of the Accounting Department might be perceived as the internal Order Processing Department.

One of the key business benefits of client/server computing is better support for the external customer by focusing on the overall, end-to-end process. We all know that as customers ourselves, we prefer to deal with a business that is focused on us, and we often show our appreciation by submitting repeat orders and by recommendation. Where opportunities exist for a business to implement improved customer-oriented support, significant benefits can accrue.

How does this objective of improved customer-oriented support translate into the analysis and the eventual information development? The answer is through business event analysis.

For example, using business event analysis, we can determine that the event Order Arrival leads to a response of Sending an Order Acknowledgment. The information processes required to handle the Order Arrival are found by process decomposition and involve the three business areas: Order, Credit and Stock. Business event analysis focuses attention on the flow of control from start to finish "Order Arrival." Any interruption in that flow can lead to the delay or loss of a response to the customer.

This example is illustrated in the following illustration.

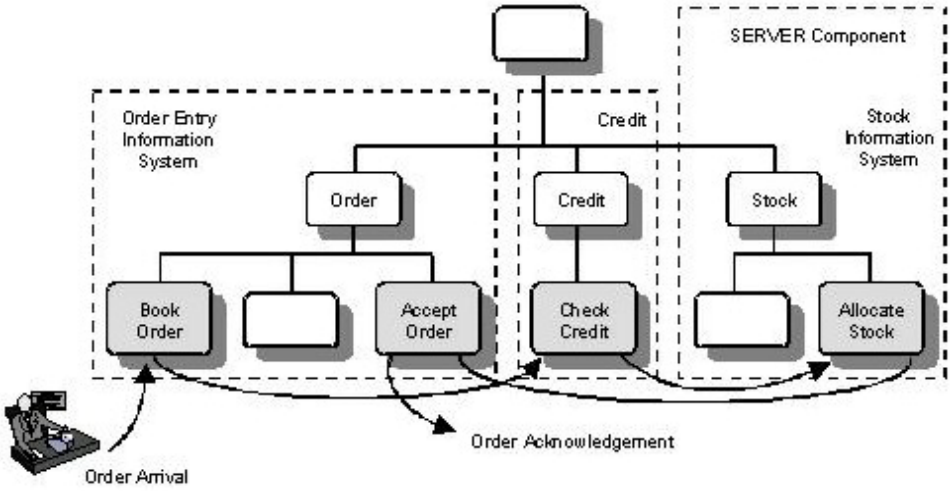


Event analysis identifies the complexity of the relationships between the various departments, the dependencies of the existing business processes.

Note: If this event were the subject of a Business Re-engineering study, the question might be asked whether the correct response to an Order Arrival is Order Delivery rather than Sending an Order Acknowledgment.

The analysis identifies that although the individual processes Order, Credit and Stock are supported by separate information systems, there is no system to ensure timely completion of the end-to-end process. Each information system (even if linked electronically) hands off to the next system.

An overview of the services currently provided in this example is shown in the following illustration.



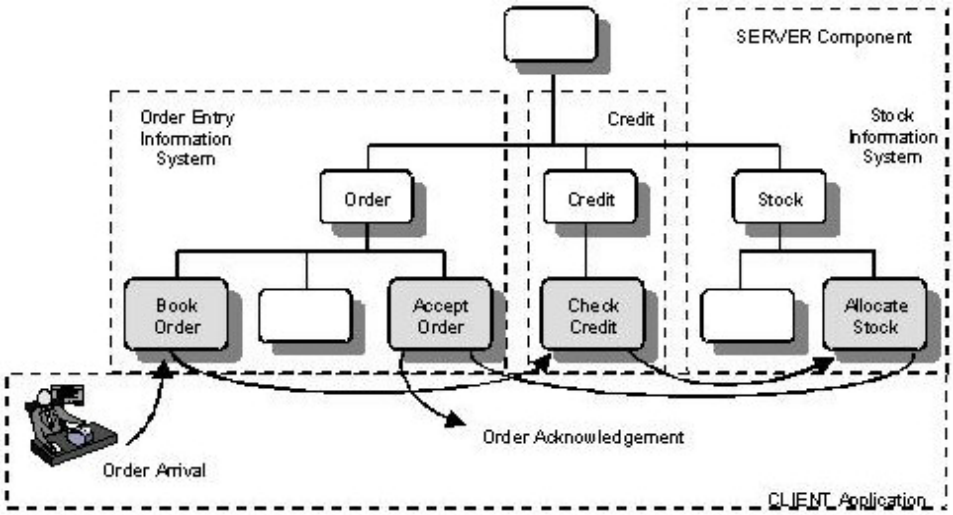
If we were merely setting out to rewrite our systems for client/server, then these existing information systems would be redesigned as components located on one or more servers.

The previous illustration shows which parts of the business event handling lack support. Any breakdowns in communication in these areas would mean delays in Sending an Order Acknowledgment.

Business users have to track new Orders and chase delays in response by various departments (Is there an answer yet from the Stock department to the Order from ABC company for the supply of 10,000 left boots?). Traditional computing technologies did not handle inter-departmental interactions easily. Typically, coordination was left to manual mail procedures to ensure that action followed action correctly and without delay.

What is needed on top of the traditional information systems (now located as components on the server) is a layer of computerized support with work queues, inter-departmental messages, time-out alarms for late response, and so on.

This is illustrated in the following illustration.



This illustration shows the business user (and perhaps most importantly, the business customer) supported by a client-based application. The user and this client application are the main focus of attention.

CA Gen enables the client application with its links to server components.

The prime purpose of the client application is user task support. The application's effectiveness can be validated by its ability to handle business events.

The client application makes use of one or more server components, each of which is a conventional information system (excluding any presentation logic). These server components should provide a complete information handling and integrity service incorporating both the business data and the business rules.

If these components are to be reusable, they should also incorporate data validation even though any well-designed client applications would already apply validation.

This approach ensures that the enterprise data is protected and processed according to agreed-upon business rules no matter how many different client applications (of whatever quality) make use and reuse of these components.

Obviously, there are many ways for a business to organize, including the following ways:

- Maintain the separate departmental structure and institute computer-supported group workflow integrated with the Order Handling client application on the workstations of their staff.
- Combine the many tasks previously carried out by several separate departments into a single integrated task that completes, or at least monitors, completion of a business process from start to finish.

- To provide an effective environment for this type of organization, the separate tasks should be integrated into one desktop design.
- If tasks are organized around a single individual who is responsible for carrying out all the activities for any one customer transaction from start to finish, that person is often referred to as a caseworker.
- The supervisor of this type of organization is often called a process owner.
- Both the case worker and the process owner will have on their workstations client applications that access many information systems on servers.

User Task Analysis and Design

User task analysis and design include the activities shown in the following tables.

User Task Analysis

User Task Support	BSI Stage	RAP Stage
Analyze user's tasks	Detailed Analysis	Conceptualization
Set usability criteria	Detailed Analysis	Conceptualization

User Task Design

User Task Design	BSI Stage	RAP Stage
Build interface structure	System Structure Design	Visualization
Prototype interface	System Structure Design	Visualization

These tasks are carried out by conducting interviews, prototyping solutions, and observing users. Observations should be documented in a project notebook.

While these tasks are included within each of the recommended life cycles, we have brought them together in this section because of their heightened significance in client/server applications where much better support can be provided.

Task Analysis

Users naturally center tasks around business events and entry objects, although other types of objects are usually used as a task progresses. For example, taking an order primarily involves filling in an order form (an object). However, it may also involve looking up a price in a price list (another object) or even using a telephone directory (another object) to help phone a supplier (yet another object).

In most cases, simply talking through the task with the user will enable you to identify all the required objects.

Users will be accustomed to dealing with these objects in the real world. They understand how they are integrated to complete a task. Special attention should be given to the user interface design to ensure that this integration is reflected.

Analyzing User Tasks

User task analysis involves modeling the procedures performed by roles or persons in the organization in support of elementary processes. The current tasks may be analyzed, and additionally 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.

User task analysis consists of:

- Gathering information.
- Identifying external, temporal, and internal events.
- Analyzing how the user manipulates data.
- Identifying user tasks.
- Defining user tasks.
- Defining user task structures.

Gathering Information

Familiarize yourself with the fundamental business rules defined during analysis.

To help identify the tasks that a user performs, you need to gather information about the data (files, documents, data storage, and so forth) and activities currently involved in the tasks being examined.

Identifying Events

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

An event may be:

- 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.

Note: For more information, see 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.

Analyzing 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.

Identifying 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.

Defining User Tasks

You are now ready to define by recording:

- Flows between the 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 with which the tasks are performed
- 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.

Defining User Task Structures

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

Once 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 Formal Task Structure with Sub-Tasks

This is an example formal task structure, with its sub-tasks, is shown in the following:

- User task—Take order from a customer over the telephone. Note that the user can be interrupted in this task.
- Initiating event—Customer telephones and indicates desire to place an order.

Task Structure

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.

Setting Usability Criteria

Measurable usability criteria for the user interface should be used to assess the likely success of the system during testing or user trials.

Usability criteria to consider:

- How easy do users find it to learn to use the interface?
- How efficient are users once they are familiar with the interface? (Productivity should increase.)
- 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.

- How often do errors occur and how serious are they? Do any errors occur repeatedly? These criteria can be measured by counting and classifying the errors to give the number of errors as a percentage of the total number of times that a procedure was used. This will help to balance error ratios for often and infrequently used procedures.
- 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.

Designing the User Interface Structure

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

Mapping User Tasks to a User Interface

You build a system structure 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:

- A data maintenance task should be supported by one or more system procedures, depending on data integrity and other business rules.
- 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.

For graphical user interfaces, decide whether:

- A window should remain open while the user accesses other windows.
- Several windows of the same type need to be open at the same time for decision-making purposes.

Mapping 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.

More information:

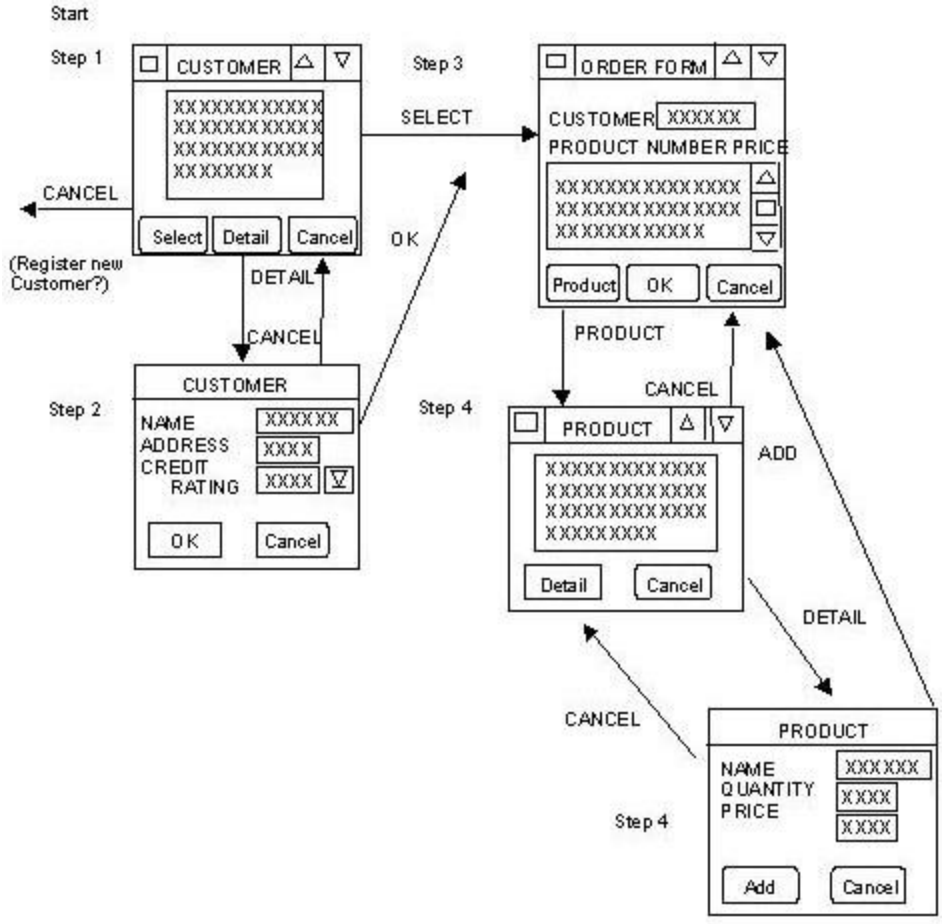
Designing the Procedure Logic (see page 163)

Prototyping the User Interface

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.

The following illustration shows this method.



The prototype should demonstrate the graphical user interface, windows, dialog boxes, pull-down menus, pushbuttons, radio buttons, check boxes, and so forth.

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.

For client/server environments, it is important to look for opportunities for front-end integration. For example, if the user regularly needs to switch between two tasks or use of data, the graphical user interface should make the switch easy. If, for instance, users can be logged on to more than one information system at a time, where previously they had to log on and off each one in turn, this could lead to substantial time-saving.

It should be possible to move between the displays and dialogs so that the usability of the flow of the system 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 the figure "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?

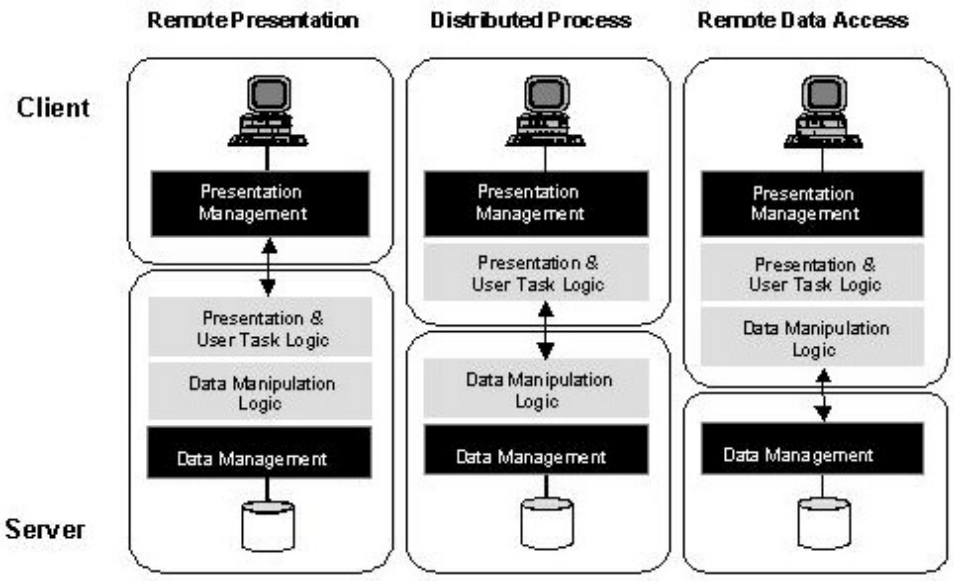
During an analysis of user tasks and prototyping a user interface structure, it might become obvious that the structure, sequence, and interaction of the activities within the user task are not the most efficient way to perform the overall business task. Although it is not the purpose of system structure 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. It should, therefore, work effectively for them.

Choose a Client/Server Style

The following list shows available styles for client/server applications:

- Remote presentation
- Distributed processing
- Remote data access

These styles are represented in the following illustration.



Logic Types

For purposes of distribution, it is possible to divide application software into the layers of logic shown in the following list.

Logic Types Defined

User task support—The part of the application that deals with the supporting the many daily user tasks and workflow between users.

Typical support of user tasks are listed next:

- Providing a checklist of procedures the user needs to follow
- Allowing the user to telephone the current customer on screen
- Providing a diary of partly completed tasks

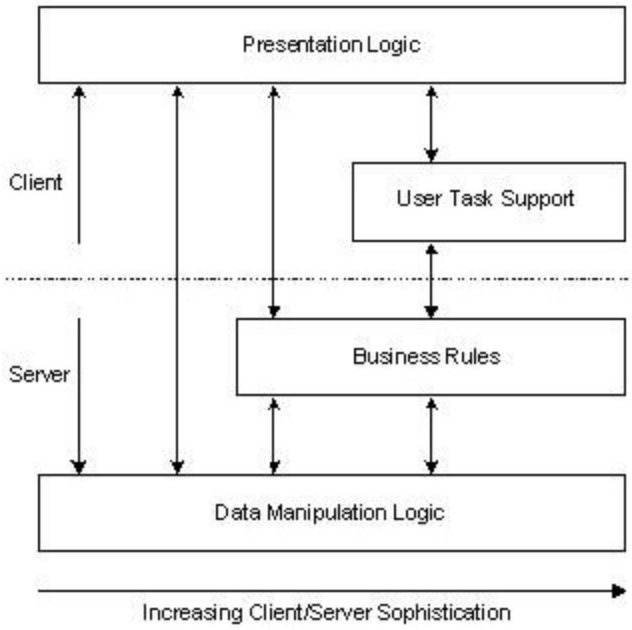
  The same layer provides workflow support, routing tasks or transactions between successive operations by different users. Finally this layer makes use of the underlying business rules layer.

Business rule—The part of the application that deals with business rules and policies.

While most business rules have something to do with persistent data, which is clearly the domain of data manipulation logic, business rule logic holds the fundamental definition that binds data accesses together. It defines such elements of an application as the sequence in which steps must be performed and the conditions that determine which steps are performed.
This layer protects the underlying data manipulation layer from inadvertent actions that might lead to data inconsistency.

Data manipulation—The part of the application that deals with data.
Data manipulation logic determines what persistent data is actually read or written. It requests services of a data management component, the DBMS, or file system responsible for the physical storage of the data.

In CA Gen methodology, business rule logic and data manipulation logic are generally found in elementary processes, as illustrated in the following illustration.



In some application programs, there is no business logic apart from that directly governing data. There may be little or no business rule logic.

Many current systems contain little or no user task support or workflow (because the technology of host-terminal was not very convenient for these purposes). This relationship is depicted on the left side of the previous illustration.

Most applications will, however, contain significant business logic. Even simple information retrieval may include business rules for allocating or grouping transactions.

As client/server applications become more sophisticated, they tend to include all four types of logic shown on the right-hand side of the previous illustration (presentation, user task support, business rule, and data manipulation).

The four types of logic in client/server applications can be divided into the following components targeting different machines:

- Client for presentation and user task support
- Server for business rules and data manipulation

A basic information retrieval application may be composed of only a client presentation workstation accessing a database server.

The result is described by Paul Winsberg as a ham and cheese sandwich (Paul Winsberg, "Designing for Client/Server" *Database Programming and Design,* July 1993, 29-31):

- The bread is the off-the-shelf software components that provide the functions of presentation management and data management.
- The ham and cheese is the interior presentation, user task support, business rule logic, and data manipulation, which are built by the developer.

Remote Presentation

In remote presentation, most of the real work is done on the server. The presentation, user task support, business rule, and data manipulation logic all reside on the server side. Only the presentation management component resides on the workstation.

This alternative includes the technique known as screen-scraping, which is a means of putting a new user interface on an existing mainframe transaction with little rework.

It also includes the more sophisticated mechanism for presentation distribution employed in UNIX and Linux environments with X-terminals.

Distributed Processing

With the distributed process alternative, the presentation logic and user task support logic reside on the workstation with the presentation management component. Business logic and data manipulation logic reside on the server with the data management component.

Any but the most simple of client/server transactions use some form of distributed processing. Some DBMS vendors have developed a mechanism called the stored procedure, which provides a degree of logic distribution. While this limited form has value, it is not as useful as the more flexible approaches that provide greater choice and support wider distribution.

These alternatives form a good conceptual foundation for simple client/server transactions, but business processes may require significantly more complex configurations. For example, an application may require data to be present both at the client and at the server. This may result in data manipulation logic being required at both locations.

When to Use Distributed Processing

Every factor involves a delicate balance between network load, server processing load and server database access that can be altered by locating parts of the logic on either the client or the server.

Distributed processing offers the ability to vary the location of logic and data between client and server to arrive at an acceptable compromise. CA Gen allows such design changes to be made swiftly and with a minimum of effort.

You need to test the effect of the design decisions about locating logic to ensure that the delivered application performs as required.

Use distributed processing when the following factors are true:

- Transaction rates for the application are high—Where transaction rates are high, the processing load on the server can be minimized by locating as much processing as possible to the client. Since every user has an individual workstation, the effect of high transaction rates is dispersed.
- Data volumes are high—The aim here is often to minimize network traffic.

  If a transaction involves retrieving many items of data and manipulating them by some algorithm before selecting a small amount of data, then such manipulation should be located on the server rather than the client if possible.
- The user community is geographically dispersed—Wide Area Networks (WANs) are typically between towns [as compared with Local Area Networks (LANs) within one building or complex] and tend to be much slower than LANs. Therefore, minimizing network traffic may become an important design consideration. In this case distribute the logic so as to minimize the data flow between client and server.

- Integrity and security are critical—Any one server module can be accessed in principle not by just one but by many different client applications. If integrity and security are to be maintained, every usage must be "safe." If the server module exposes data as in the remote data style, each client application must implement the identical high quality logic to ensure safety. This can be very challenging organizationally.

  However, if the server module is built using distributed processing, the safety logic (integrity and security) and indeed any other common business rules can be located on the server with the data. Access to the data is then allowed only through the server logic.

  Client applications should also ensure integrity and security. But with the recommended approach, there is no reliance on the same high quality and identical logic in multiple client applications.

- Most enterprise—shared databases are best catered for by the distributed process style, for which the remote data style is not usually appropriate.

- Multiple legacy screens need to be associated with one GUI window—The distributed processing style lends itself very nicely to this interworking. Legacy and Current systems residing on hosts or servers contain some logic. Further new logic to support user tasks can be located on the client application that accesses these older modules.

Where to Locate Logic with Distributed Processing Style

The distributed processing style often is the best solution for user task support, flexibility, and performance.

Try to design the end-to-end logic of the application so that it runs in one or more modules on the client.

When access is required to shared databases, locate the databases together with the business logic, integrity rules, and security rules on the server. Access to the data is always through the server logic layer not direct to the database. In practice this often means that the information processes reside on the server while the procedural logic is on the client.

You can consider accesses to the server rather like traditional accesses to sub-routines. The overall logic should be visible and testable on the client with calls to the server modules as necessary (by CA Gen dialog link flows). By adopting this style, you should find that your server modules can be reused by new applications without change.

Also, by locating such rules in one place (next to the data), changes to business rules can be more easily implemented and with minor disturbance to the using client applications.

Remote Data Access

In this alternative, the presentation, user task support, business and data logic reside on the workstation with the presentation management component. Only the data management component resides on the server.

The DBMS performs the remote data access.

Guidelines for Selecting a Client/Server Style

Some simple criteria are presented shortly to assist in the task of selecting styles. However, following this single guideline will greatly simplify the whole process:

Always design applications to be modular and flexible.

By maintaining a clear distinction between the four elements of presentation logic, user task support, business rules, and data manipulation logic, you will be in a good position to change between client/server styles as needed. This can be very useful because it is difficult to fully anticipate the behavior of an application before it is actually executing. If you designed an application to run as a remote data access application and discover after implementation that the distributed process style would perform better, it is relatively easy (with CA Gen) to move the data manipulation logic from client to server.

In the CA Gen environment, the appropriate level of modularization is generally achieved by ensuring that the procedure step logic (usually associated with presentation) is held separate from elementary process logic.

Use remote data whenever you can, for example, when you are:

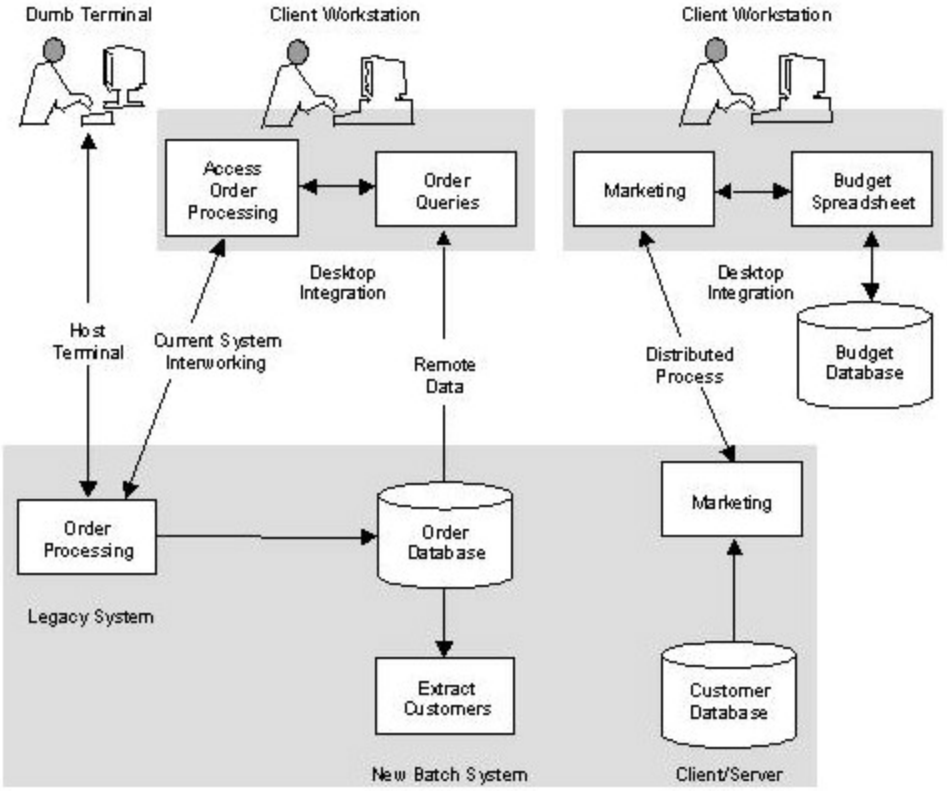
- Putting a GUI front-end on a legacy system
- Building UNIX and Linux applications that will run using X-terminals

This is the easiest style to use because it relies on the DBMS to perform the remote access. There is no division of application components involved.

However, with this style, performance may not be optimum, and data integrity can be compromised.

Combine Client/Server Styles

It is not unusual to find a variety of application styles employed by a given organization. The following figure illustrates how applications using different client/server styles can be combined with legacy systems and workstation application packages.



Client/Server Workstations

Most client/server computing uses an intelligent workstation for the client. This gives added processing power for users to run desktop applications, as well as the CA Gen-generated application.

The two opportunities for improving the visualization display of data accessed on a server follow:

- Front-end integration—To enable users to take full advantage of their desktop applications, it should be possible to integrate the CA Gen-generated application into those other tools. Object Linking and Embedding (OLE) technique can be used.

  If this raises issues about data security and concurrence, these need to be described to ensure data integrity.

  It may be useful to combine some technology analysis with user task analysis (described in a previous section) to determine which personal productivity tools (word-processing, spreadsheets, and so forth) need to be available to users, and what user interfaces are available.

- Current global check system and data integration—Users can improve the visualization display of data accessed on a server, directly or through current systems, and can combine data accessed through different systems and on different servers.

Considerations for System Structure Design

You can use the results of elementary process/entity type usage, user task analysis, distribution analysis, and current systems analysis to decide:

- Whether data that is used by users at the same type of location can be stored on client workstations to support distributed process procedures.
- Whether data should be stored on a shared server because it is used by a wide range of organizational units or users.
- Whether current systems logic can be reused for new distributed presentation procedures.
- Whether new logic should be placed either on a server for remote presentation, or on a client for combining local with remote data.
- How many of the user tasks should be automated on the client.

In general, the functions of client and server procedures should be:

- Client procedures deal with user task support, the user interface, simple field validation, work-flow logic and interworking with other workstation-based facilities (such as for managing documents and spreadsheets).
- Server procedures manage business rule and data integrity logic, and therefore logic that provides access to shared stored data.

Results of System Structure Design

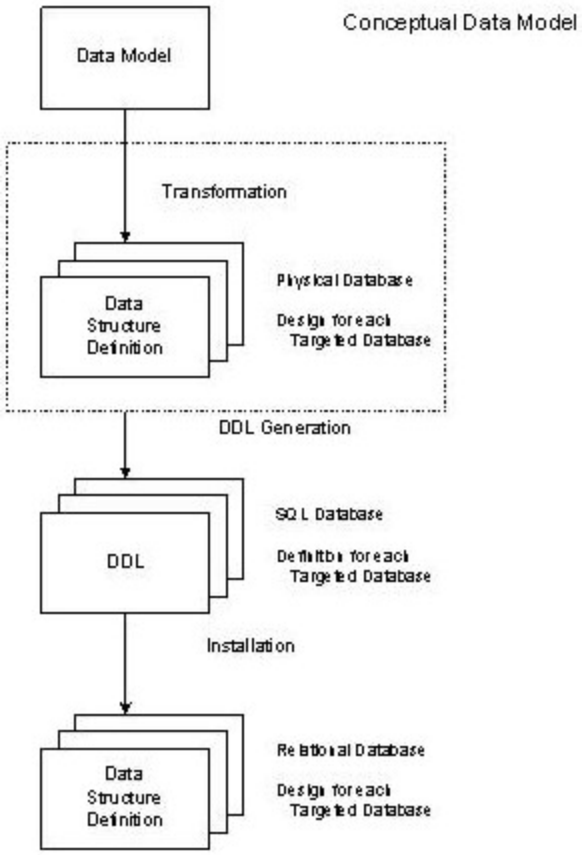
The results of system structure design are listed next:

- Understanding of users' experience levels, frequencies of task execution, geographic locations, and work environments volatility
- Objects of interest to users
- Detailed tasks and subtasks the users need to perform
- Graphical user interface structure of the system
- Client/server style

Chapter 4: Designing the Data Structure

This chapter describes how to transform the conceptual data model into an implementation representation of the database. 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.

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).

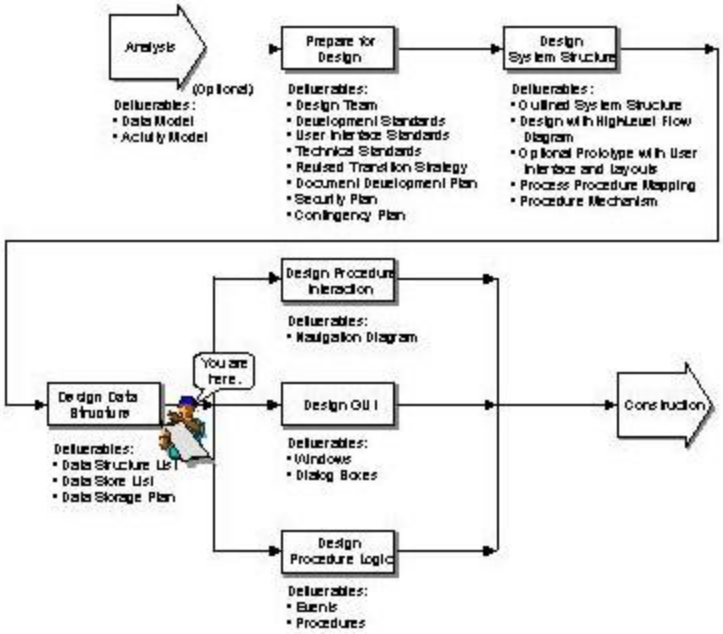
Also consider the means of converting data to the new database(s) (single location or distributed). Some further conversion procedures or external action blocks to interface to database management systems not supported by CA Gen's toolsets may be needed. Early planning for these transition and associated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

Prerequisites

You need an outlined system structure design before beginning data structure design. See the chapter titled "Designing the System Structure."

Design Process

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



Note: Although the illustration, Designing the Data Structure, shows data structure design preceding Procedure Interaction, GUI, and Procedure Logic design, data structure design may be done in parallel with or after these activities.

Transforming the Data Model

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

Note: The detailed content of the data model is described in 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:

Retransforming the Data Model (see page 78)

Transformation Implications

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

- 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 implications of transformation are:

- 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.

Setting 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 information in the technical design properties include the following:

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

Setting Reserved Word Checking

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

- DBMS—Use this option if you are going to let CA Gen name the tables and you do not want any reserved words from the DBMS used.
- None—Use this option 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—Use this option if you are going to let CA Gen name the tables for you, and you would like the tables in each targeted DBMS to be named exactly the same.

Setting Referential Integrity Enforcement

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

- CA Gen-enforced RI—Use this option if you want CA Gen to enforce RI. CA Gen generates pieces of code to handle all of the referential integrity.

- DBMS-enforced RI—Use this option 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.

Setting Permitted Value Default Enforcement

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

- Reads-Enabled—Use this option if you want the permitted value enforcement for read statements for all of the tables in the database(s) 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—Use this option 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—Use this option if you want the permitted value enforcement for row creates and updates for all tables in the database(s) 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 titled “Designing the Procedure Logic” to learn more about this technique.

- Creates/Updates-CA Gen—Use this option 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.

Setting 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.

Performing Transformation

Transformation uses the data model to derive a relational database design conforming to the technical design properties (see Setting 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 each DBMS, even if you are only going to target a single DBMS. This way, each of the implementations matches at transformation time.

Transformation creates these 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 and/or the Data Structure List to conform to local standards or optimize performance.

More information:

Completing the Data Structure Design (see page 80)

Using the 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.

The Data Structure List reflects any changes made in the Data Store List and vice versa.

Use the Data Structure List tool to:

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

More information:

Using the Data Store List (see page 78)

Data Structure Terminology

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

Data Structure Terms

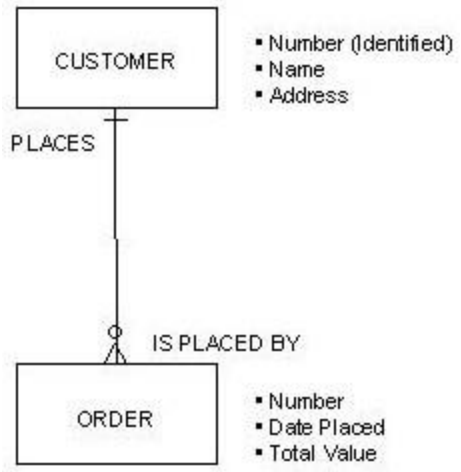
CA Gen Model	Relational Term	Production
Data Model	Data structure (physical design)	Database implementation by the DBMS
Entity Type and its Subtypes	One Table	Table
Attribute - Basic	Column	Column
Attribute - Designed	Column	Column
Attribute - Derived	Not implemented physically	Not implemented physically
Attribute - Auto Number	Column	Column
Relationship (1:1 or 1:M)	Foreign key	RI constraint
Relationship (M:N)	Table	Table

CA Gen Model	Relational Term	Production
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

A one-to-many relationship is illustrated in the following Illustration.

1:M Relationship



This illustration is a data model fragment showing:

- 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 figure "1:M Relationship" as a foreign key.

Implementation of the 1:M Relationship

Type	Macro Name	Format	Length Optionality
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)	Integer	4 Not Null
	NUMBER		
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 NUMBER	Integer	4 Null
RI Constraint	<No Name> CUSTOMER		
Index	FKEY		
Column	FK_CUSTOMER NUMBER	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.

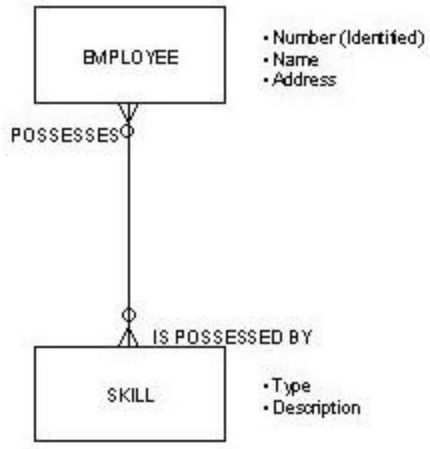
Many-to-Many Relationships

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 (see Associative Entity Type). 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.

A many-to-many relationship is illustrated in the following Illustration:



This figure is a data model fragment showing:

- 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 figure is a Data Structure List showing the implementation of the figure "M:M Relationship."

Type	Macro Name	Format	Length Optionality
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 Not Null
RI Constraint	<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)	FKEY (Primary)		
Column	FK_SKILL TYPE	Char	15 Not Null
Column	FK_EMPLOYEE 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 that 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 means that 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 terms used in the Data Structure List are:

- 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.

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

Properties of Tables

Entity Type

The name of the entity type in the data model that this table logically represents. This is a 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 that 18 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 30 characters long in the CA Gen model. Certain databases, however, only allow a maximum of 18 characters. Do not exceed 18 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

Select from the following options:

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-Select from the following options:

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 to:

- **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

Property	Description
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 that 18 characters long. There is no reason to change this name in the Data Structure List.

Property	Description
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 DBMS attempts to put rows in the clustering order by only actually organizing every row in a clustering order after 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 an 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 selection.

The RI constraint properties in the pop-up window that can be changed are listed in the following table.

Properties of RI Constraints

Property	Permitted Changes
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 that 18 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.

Using the 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.

The database administrator uses the Data Store List tool to:

- 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

More information:

Using the Data Structure List (see page 67)

Retransforming the Data Model

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 re-creation 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 each possible DBMS, 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.

Rules of retransformation for multiple DBMS targets:

- 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 each DBMS 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

Identifying 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:

- 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.

Completing 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 these tasks:

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

Changing 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.

Optimizing 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.

Some of the optimizations that can be performed in the Data Structure List are:

- **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—**A DBMS may 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.

Rearranging 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

Choose to Distribute or Replicate Data

The decision to either distribute or replicate data is a critical one because it involves significant effort.

Rules may already be in place that define how the data is to be distributed and or replicated.

Avoiding distribution and replication is the simplest and most expedient option.

The main reason for distributing and replicating data is to increase performance. Performance is an issue that becomes much more important in construction and testing. The most appropriate action to improve performance can be taken then.

Detailed discussions of the alternatives with DBMS and network specialists should be sought before proceeding.

Distributed Data Alternative

Distribution of data is the division of data between a number of servers.

The complexity of the division can range from simply placing different types of data (tables, or even individual columns) on separate servers to partitioning data of the same type between servers according to where it is created or needed.

The following list explains advantages of distributing data are:

- Brings complex or iterative maintenance of data closer to the user
- Provides for faster retrieval through dedicated database processors

The following list explains disadvantages of distributing data are:

- Can create extensive network traffic
- May not always be accessible as quickly as needed

Data can be distributed by using a database management system to automatically take control of the process.

Note: Controlling the distribution of data that is outside the scope of the DBMS can be achieved by specifying locations in procedure logic. Remember to consider the risks of failing to maintain data consistency, integrity, and security. This approach should be considered only if you are familiar with these issues, as well as the techniques for controlling distributed data.

Mechanisms distributing data depend on the currency requirement for the data:

- If the solution demands immediate, up-to-date data, a database management system should be used.
- If, however, the solution is a decision support system and requires data distribution on a monthly basis, it may be quite safe for an application or personal productivity software (such as a spreadsheet) to maintain the data locally.

Replicated Data Alternative

Replicating data (usually for use on a client) may improve performance considerably by freeing up processing time for other processes.

The following list explains considerations for replicating data:

- Do client procedures use replicated data locally rather than a centralized source?

  If a process requires read-only access to the data, it might be worth considering using replicated data on the client for that process to free up processing time for other processes.

- How current does read-only data need to be?

  If the process performs queries on the data, such as for statistical graphs, it might not matter if the data is replicated on a daily basis. The previous day's data might be sufficient.

Challenges of Distributed and Replicated Data

If you plan to use distributed or replicated data, you must carefully check for these:

- The target operational environment
- What the DBMS can achieve automatically

- What the CA Gen-generated application will provide
- What remains to be specified by the designer so that the full advantages of distributed and replicated data can be realized in a safe manner

If you plan to use distributed or replicated data, you must address these issues:

- Integrity
- Consistency
- Referential integrity
- System security
- Performance

Integrity of Distributed Data

If the data is not distributed and held on a single server, data integrity raises the same issues as in a traditional host/terminal environment. However, if the data is to be distributed across two or more processors, you will need to consider how the data is to be distributed.

Distributing data has a major consequence on the integrity of the data. Take special care in designing data distribution. The integrity of the data to be stored within the database must be guaranteed. It is vital that the database management system to be used fully supports the use of distributed data. In particular, the database management system needs to handle the checking and cancellation of unsuccessful changes to avoid integrity problems when updating the data.

The following list explains options for distributed data:

- Rows in tables may be arranged by value. For example, all rows for customers of each sales office could be placed contiguously.
- Updating the data is relatively straightforward. Each processor can update the data independently of the others.
- Columns in tables may be split by ownership. For example, the sales department has CUSTOMER NAME and ADDRESS, and the accounts department has CUSTOMER RATING.
- In this case, the data at one location should be maintained as the master. Data at any other location that uses or updates that data should be kept in synchronization with that master.
- A combination of splitting by rows and tables may be used.

Deletion of any data in a distributed environment can cause problems, as other processors may be dependent on the data for their own procedures. Deletions need to be synchronized, possibly by performing such operations at predetermined times, such as overnight.

Integrity of Replicated Data

With replicated data, the data is replicated as identical copies on the clients, or possibly on a number of distributed servers.

Integrity considerations for replicated data integrity:

- How will the data be updated?
- What mechanisms will be used to ensure that each client has access to the same data at all times?
- Where is the master copy of the data?
- Is the data needed immediately, or is a snapshot sufficient?
- Will the whole table be replicated or just a subset?

Note: Ideally, replication should be used only to give read-only processes better performance.

Updating replicated data should be avoided as it is a very high risk process. If replicated data is updated, the following minimum procedure is required:

1. Record the source of the data.
2. Update the source.
3. Apply the updates to the replicated data.

Consistency

If one change fails but changes to data on the other databases are successful, the changes to the other databases should be rolled back to ensure that no changes are made to any of the databases. This avoids data integrity problems.

This means using a mechanism to apply the changes to all the databases or not at all. The available DBMS may be able to achieve this using a multi-phase commit, whereby the update is made and logged to each of the databases. The DBMS then checks to ensure that the updates have been made successfully. If so, the updates are committed to the databases simultaneously.

Maintaining data consistency through the use of procedure logic is critical to data, and therefore to business process integrity. Such logic should be defined by experienced designers who have an understanding of the operation of the target DBMS and operating system.

Referential Integrity

Referential integrity requires that data that has been affected by a procedure must satisfy all integrity rules for that procedure to execute successfully. For example, when an ORDER is created, at least one ORDER ITEM must also be created.

Referential integrity considerations for database and application design:

- Location of boundaries—Every reference from data stored in one location to data stored in another must satisfy integrity rules before a change to data can be accepted as successful.

  Any physical boundary between related data can cause extra data accesses and checking to maintain referential integrity. The boundary should be positioned to minimize access by procedures to related data stored in more than one location.

- Relationships across a boundary—Mandatory relationships across a physical boundary must be maintained by procedure logic. This logic should identify and attempt to fix any discrepancies.

  For example, assume that CUSTOMERS may be stored in one location and EMPLOYEES in another location, and that each CUSTOMER must be the responsibility of an EMPLOYEE. The name of the RESPONSIBLE EMPLOYEE is defined as an attribute of CUSTOMER. When a CUSTOMER is created, the procedure checks the existence of a responsible EMPLOYEE. When an EMPLOYEE is deleted, a procedure checks that there is no CUSTOMER who is the responsibility of the EMPLOYEE.

- Use of foreign keys—Foreign keys can be useful to represent relationship memberships across a physical boundary between two data stores. Where the target DBMS is unable to maintain integrity across different physical databases, then the designer must specify procedure logic to maintain the data values in step.

- Deletions—Deletions, such as cascade deletes, pendant deletes, and restricted deletes, that affect related data held in more than one location, can be very difficult to complete successfully.

Ideally, the DBMS should make all checks. If this is not possible, data integrity should be maintained by procedures.

System Security

The more distributed the data is in the system, the greater the security risk.

Security considerations for distributed data:

- What authorization is required before a user can run certain procedures?

  Authorization with password may also be required before a user is able to use irreversible procedures, such as deleting data. This will probably mean holding the user identifiers and passwords centrally on the server for access by client procedures to disable options on the user interface.

- Should the system time-out after a prolonged period of inactivity?

  The enterprise's security procedures may dictate that users should not leave unattended machines logged on to systems. If so, a time-out facility may be required. However, this could lead to data integrity problems. For instance, data integrity may be impaired if the server times out but the client does not.

Performance

It is likely that the system design will need tuning for optimum performance. The precise nature of this tuning depends on the individual system environment.

Database performance considerations:

- Is database performance affected by the location of the data?
- If the data is distributed, is access to the data fast enough across the network?
- If the data is centralized, can the server handle the anticipated number of transactions?
- If necessary, consider using distributed data or replicated data.
- Is the performance of the network adequate?
- This can be assessed by looking at the number and size of data transfers. For instance, would a small number of large data transfers or a large number of small data transfers provide better performance?
- How many concurrent clients are there likely to be?
- If there are a large number of users accessing a single data source, will there be problems with users being locked out of the data?
- Is there a potential bottleneck at the communications bridge or server?
- Must the system be available at all times? Can the system cope with machine failures?
- If the workstations are considered inadequate for backup, a more controlled approach is needed.

Results of Data Structure Design

The results of data structure design are listed next:

- Data Structure List and a Data Store List that are accurate transformations of the data model, optimized if necessary for performance
- Data storage plan showing where data is stored and which replicated data is master data.

Chapter 5: Designing the Procedure Interaction

This chapter describes how to specify in detail the procedures and flows that will allow users to navigate through the system. Procedure interaction is what allows procedures to communicate with one another. This includes client-to-client and client-to-server procedure communication.

The GUI, procedure interaction, and procedure logic are best designed in parallel. Procedure interaction and layouts may then be refined, often by prototyping. When the initial design is stable, detailed procedure logic can be completed, and the flows and layouts can be finalized.

Together, the procedure interaction, layout design, and logic will yield a completed navigation diagram, fully detailed displays, and action diagrams that are the basis for generating a complete system.

Prerequisites

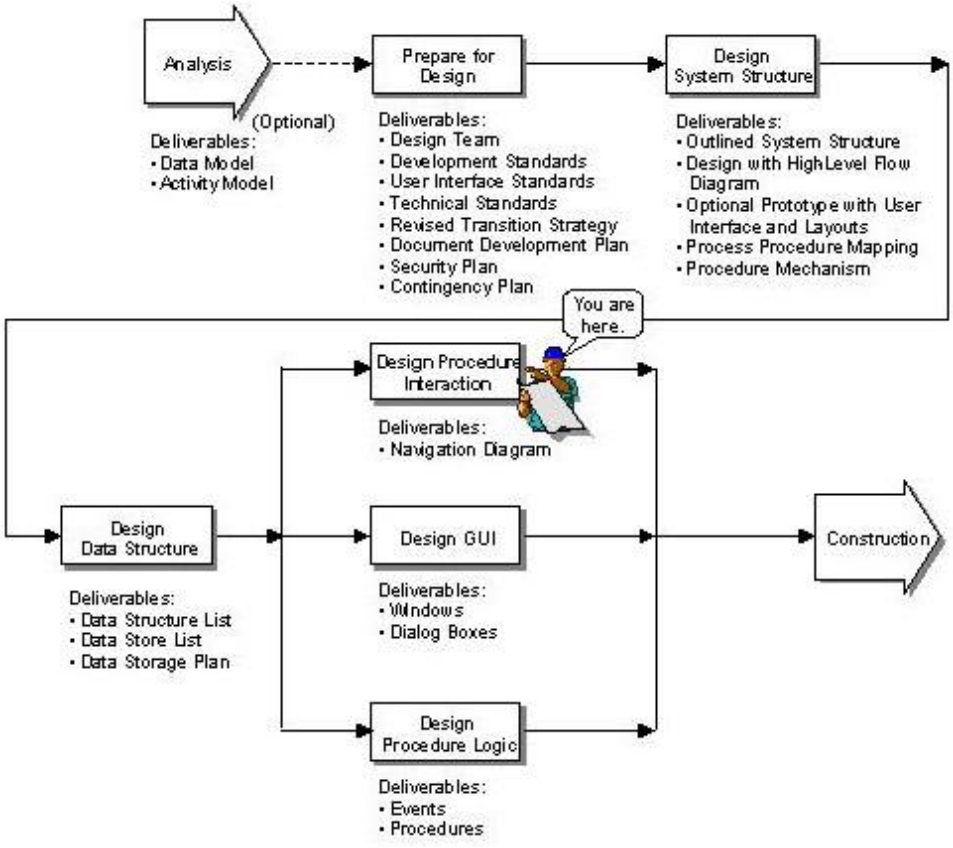
You need an outlined system structure design before beginning procedure interaction design.

More information:

Designing the System Structure (see page 33)

Design Process

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



Principles of Procedure Interaction

Procedure interaction can occur through a flow or Procedure Step USE statement:

- Flow (transfer or link)-A flow can be used to communicate between any two client procedures or client and server procedures.
- Procedure Step USE statement-This statement, sometimes generically called Remote Use, can be used to communicate between two client procedures if they reside on the same platform, and between any client and server procedure.

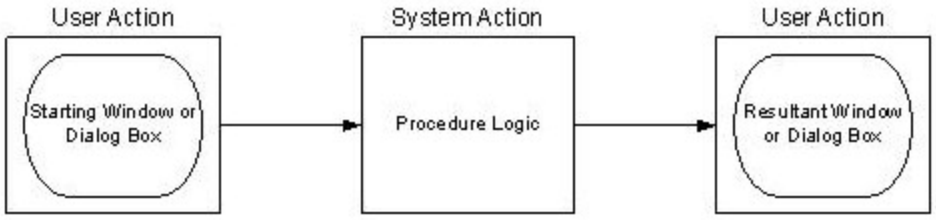
In a client/server environment, each client procedure may be associated with a procedure action diagram and a graphical user interface (GUI) display. You define the possible series of interactions between a user and the procedures in a business system through these associated GUIs. The GUI can be a window and associated dialog boxes as needed. You also determine when there is a need to communicate with a server procedure to gather the correct data for the user request.

An interaction 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 involves this procedure:

1. The user enters data on a window or dialog box.
2. The system acts and responds with another window or dialog box. The window or dialog box may be in the same format as the one on which data was entered, 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 interaction that begins with a user entering data on the starting window or dialog box.



When the user indicates that data entry is complete by selecting a menu item, push button, or a toolbar item, the system responds by processing the data according to the procedure definition. When processing is complete, the resulting window or dialog box is presented to the user. Remember that windows and dialog boxes are associated only with client procedures.

From the developer's perspective, an interaction involves an indeterminate number of procedures. Remember that the client procedure is what contains a window plus one or more associated dialog boxes. There may be a need to communicate with other client and/or server procedures before returning a window or dialog box to the user.

There are two possible procedure interactions:

- Client/client through a flow
  (only if on the same hardware platform)
- Client/server through a flow or a Procedure Step USE

The user does not see the procedure interactions. The window or dialog box associated with a client procedure presents only the data.

More information:

Choose Flow or Remote Use (see page 105)
Designing the Graphical User Interface (see page 117)

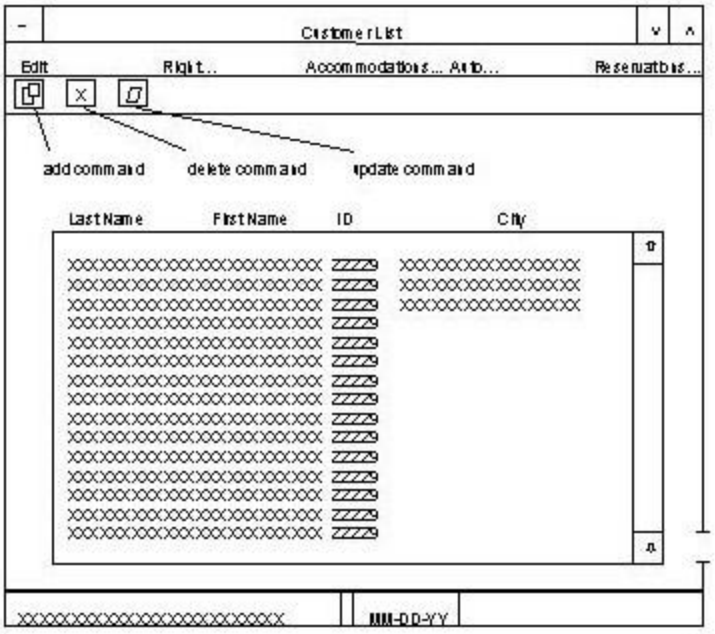
CA Gen Commands

A command is a special concept in system design using CA Gen. It allows you to specify how users can influence procedure execution. This special attribute is defined here because the concept of a command is important to the discussion of procedure definition.

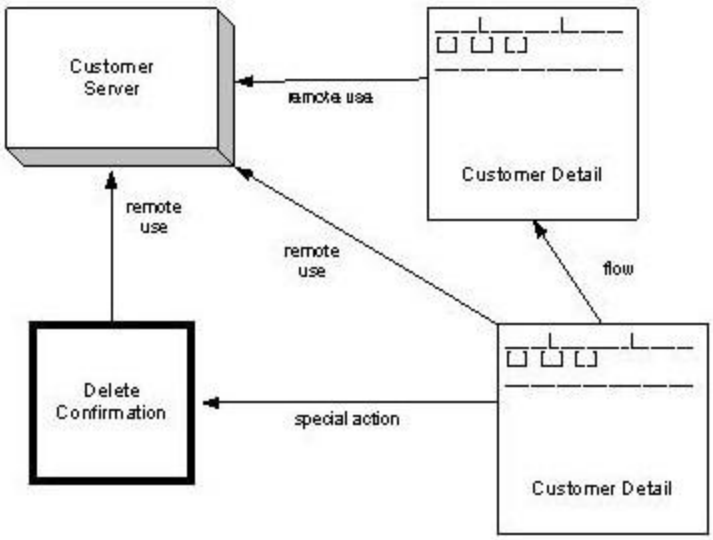
Commonly used commands should be standardized across the system.

You can associate a command with a graphical object, such as a push button or menu item, on a window. At execution time, a user can activate that graphical object by pressing the push button to cause a procedure to behave in a desired manner.

For example, the following figure illustrates a client procedure that provides the user a list of customers. The user can add a new customer or choose one to change or delete.



The user should be allowed to choose whether to add, change, or delete a customer at execution time by activating the appropriate graphical object. These commands are assigned and sent to the server as illustrated in the following figure.



Commands tell the server what action should be performed.

The following figure shows the statements for the procedure step Customer Server.

```
Procedure Step: CUSTOMER_SERVER

  1-    -----  CUSTOMER_SERVER
  2-   |               IMPORTS: Entity View import customer
  3-   |               EXPORTS: Entity View import customer
  4-   |       EXIT STATE IS prossing_ok
  5-   |    --- CASE OF COMMAND
  6-   |   |--- CASE delete
  7-   |   |    USE delete_customer
  8-   |   |                WHICH IMPORTS: Entity View import customer
  9-   |   |                WHICH EXPORTS: Entity View export customer
 10-   |   |--- CASE add
 11-   |   |    USE add_customer
 12-   |   |                WHICH IMPORTS: Entity View import customer
 13-   |   |                WHICH EXPORTS: Entity View export customer
 14-   |   |--- CASE update
 15-   |   |    USE update_customer
 16-   |   |                WHICH IMPORTS: Entity View import customer
 17-   |    ---             WHICH EXPORTS: Entity View import customer
 18-    ------
```

More information:

Designing the Procedure Logic (see page 163)

When to Use Commands

The special attribute command can be set in several places during design:

- Assigned to a graphical user interface object (such as a push button or menu item)—Assign a command to the GUI object only if you would like it assigned to any action that would take place with the object.

  You will not usually assign a command to the object since there may be several events associated with a single object. There may be an event associated with clicking the object, double-clicking the object, bringing the object into focus, and so on.

- Assigned to an event associated with a GUI object—This is the typical mechanism of assigning commands:

  - If the command is associated with an event, and a remote use of a server is executed in the event, the server procedure can evaluate the command and perform the correct logic.
  - If the command is associated with an event, and the exit state is set to cause a flow, and if you have defined the command to be "current" on the flow, the command will be carried to the server.

  The logic in the server will evaluate the command to understand what the user is trying to accomplish.

- Assigned along a flow—A command can be set along a flow, which means that every time that flow is executed, the command will always be the same.

  This will not be a typical use of setting the command for client/server or client/client procedure interactions.

- Set to current along a flow—For client/server and client/client procedure interactions, you will typically set the command to current along the flow.

  Since most of the flows in client/server design are links, you typically set to current in both directions of the flow. This does not assign a value to the command. It simply allows the value of the command to remain the same from the source procedure to the destination procedure as the flow is executed.

- Set to "previous" along a return link flow—Although you will typically set the command to current on the return link flow, you could set the command to whatever the command was when the send flow was executed. This means that if the command were changed in the destination procedure for some processing, it would be changed back to whatever is was when the send flow was executed. This allows you to know what command was sent to the destination procedure.

  This will become more useful as server-to-server flows are implemented.

- Set it in a logic statement—There is a procedure logic statement that sets the command whenever logic is being executed. For more information about how commands are set by a procedure logic statement, see the chapter "Designing the Procedure Logic."

  You will have many reasons to set the value of the command in the logic.

More information:

Designing the Procedure Logic (see page 163)

Reserved Commands

CA Gen's window and server managers, the execution-time components that govern procedure execution, reserve several commands for their own use. You should be aware of the commands listed in the following and use them appropriately.

- HELP—Causes the window manager to invoke its HELP exit.

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

- RESET—Causes the window manager to go back to the initial window or dialog box for the business system.

Exit States

Exit State is a special attribute in the CA Gen toolset. The value of this attribute controls what happens at the conclusion of a procedure.

Depending on the value of exit state, the procedure may do the following tasks:

- Display a message in the system message field
- Initiate a flow
- Roll back database updates resulting from the execution of the procedure
- Abort the procedure (roll back all database updates and abnormally terminate)

Components of an Exit State

Each exit state definition includes the components shown next:

- 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—A message appears in the system message field if its associated exit state value is set when the procedure concludes and the exit state value does not trigger a flow or an abnormal termination.
  Messages are displayed according to the message type (informational, warning or error). In a client procedure that contains a display, messages are displayed in an appropriate dialog box.

  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—A termination action indicates the type of processing that is performed at run time when the associated exit state value is detected.

  These are the possible termination actions:

  - Normal-the exit state message will be displayed or a flow will take place.
  - Rollback-all database updates (the results of CREATE, UPDATE, and DELETE actions) performed by the procedure since the previous checkpoint will be backed out.
  - Abort-the procedure will abnormally terminate and will roll back all database updates.

Exit State Definitions

Exit state definitions can be built using system defaults or from any tool that can reference exit states. In a CA Gen action diagram, exit states are set according to the action performed and its outcome.

The outcome of an action may be signaled by the following values:

- 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)
- Value of an Exit State returned to a client from a server procedure (for example return WITH ROLLBACK)

These are the options for ensuring 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.
- The default value is used if no other exit state value is set.
- You must ensure that every possible exception condition is identified, to avoid giving a false *positive* result.
- This is currently the most widely adopted option.
- 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 reset the exit state accordingly.
- You must ensure that every possible exception condition is identified.
- The default value is also used if no "positive" or known "negative" exit state value is set. 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 client, server, and common action block logic. Where a mixture of these options is already adopted, be especially aware of which option is employed by existing logic with which their procedures must interwork.

This example raises the following issues:

- Exit State is set to "Processing OK" before any other logic is executed (line 4). You assume that positive conditions are not addressed in the USED action blocks. Therefore, you initialize exit state to the positive condition here. This avoids handling exception conditions throughout the remainder of the procedure 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 unless you can guarantee correct handling of all exception conditions that may arise. For example, exception conditions could arise if the action blocks were subsequently modified.
- There is no handling of the exit state if the command sent to the procedure is not included in the case of command. This will be handled using one of the client/server error handling techniques.

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

These are the rules governing the assignment of exit state definitions to flows:

- An exit state value may appear on the flows on exit state list of only one flow from a given procedure.
- No exit state value that appears on the returns on exit state list of a link to a given procedure may appear on a flows on exit state list of flows from the same procedure.

More information:

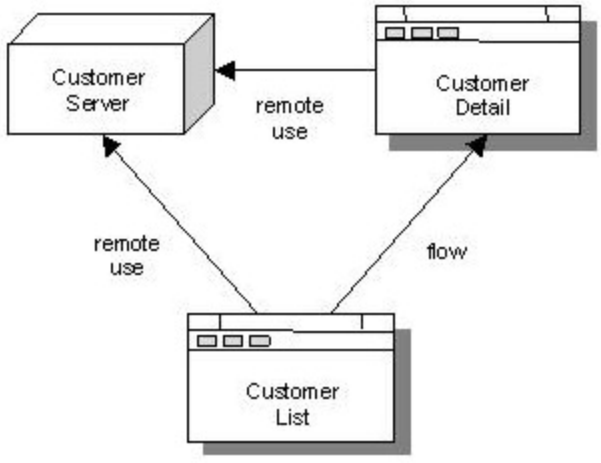
Error Handling (see page 219)

Passing Data

When data needs to be passed from one procedure to another, it is done using view matching for both flows and remote uses.

The navigation diagram in the following illustration represents a slightly more complex example.

The illustration provides an example of data passing for both a flow and a remote use would be if the user is on the customer list and decides to update information about a customer.



The illustration shows a navigation diagram collection of an airline reservation system:

- The user selects a customer.
- That customer identifier is view matched along the flow from the customer list client to the customer detail client.

- It is view matched along the remote use from the customer detail client to the customer server.
- The customer server reads all of the data that can be updated for the customer and returns that data to the customer detail client. The data is returned via a view match on the remote use.

Data can pass over both flows and Procedure Step USE to its destination. Any data in the source procedure's export data view can be passed to the destination procedure's import data view, as long as the views are compatible.

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

More information:

Designing the Procedure Logic (see page 163)

Flows

A flow can be used to communicate between any two client procedures or client and server procedures.

Types of Flows

There are two types of flows:

- Transfer
- Link

Transfer Flows

A transfer flow indicates that the source procedure passes control to the destination procedure when appropriate conditions are met. The source procedure may also pass data. For more information about how control is passed to the destination procedure, see How Flows Are Initiated.

The transfer flow is not used in the following scenarios:

- Not used for client-to-server flows because a return is necessary
- Not used much in GUI application since a graphical user interface is oriented around graphical objects and complete control transfer is not necessary

The transfer flow is used in block mode.

Note: For more information about how transfer flow is used in block mode, see the *Block Mode Design Guide*.

More information:

How Flows Are Initiated (see page 101)

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 passes control and, optionally data, to the destination procedure.

The link flow performs the following tasks:

- Allows the destination procedure to return control and data to the source procedure
- Saves all export data known to the source procedure when it completes its original execution and makes it available after the destination procedure returns
- Is always used in client-to-server flows because it is necessary to return from the server back to the client in client/server processing.
- If a flow is selected for the procedure interaction between a client and server procedure, the link flow is used.
- Is used in client-to-client flows since the processing is based on a graphical object, and although a flow may be used to open another primary window, the focus still comes back to the original object

For example, the illustration in the section Passing Data shows a Customer List. The user may add, update, or delete a customer from that list. This initiates a flow made to the Customer Detail client. After the action is completed, the focus comes back to the Customer List.

A link flow allows a user to acquire additional information from another procedure when it is needed by the initial procedure. After the information is acquired, the initial procedure executes again from the beginning with all the information from its initial execution intact. Any new data or commands returned from the destination procedure is also utilized.

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

More information:

How Flows Are Initiated (see page 101)

How Flows Are Initiated

Each flow is initiated by the source procedure setting an appropriate exit state.

Each transfer flow on a navigation 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 must have at least one Flows On Exit State.

Each link flow must also have at least one Returns On Exit State, with the exception of the client-to-server flows. There is an implied return from the server, and no exit state is necessary. The return flow will complete with the completion of the server procedure.

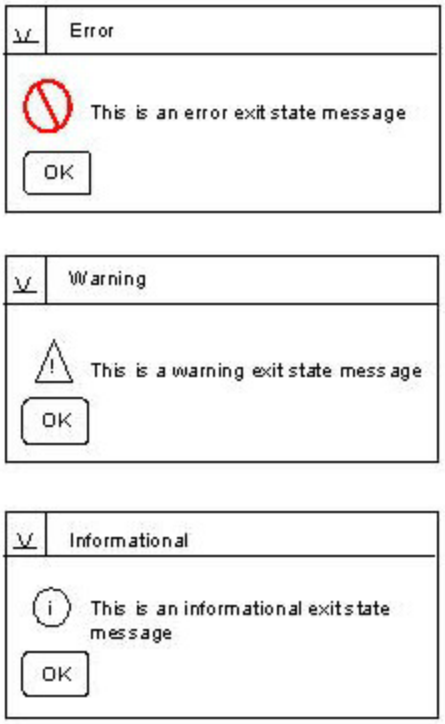
For a given procedure, only one flow can be associated with each exit state value.

After the destination procedure finishes executing, the Returns On Exit State causes control to be returned to the source procedure.

The use of exit states can be summarized as follows:

- When the logic of a procedure concludes, the value of Exit State is evaluated.
- If this value matches that associated with any flow from the procedure whether it be a transfer, link, or return from a link, the flow is executed.

- If this value does not match any exit state values associated with any flows, you must specify some corrective action.
- The corrective action is appropriate to the procedure implementation. In a client procedure, the exit state message would typically be displayed in a dialog box with the appropriate icon as shown in the following illustration.



The CA Gen window manager controls this dialog box and icon according to the message type specified. The message dialog box is modal; that is, it requires an input before proceeding.

Choose a Flow Action

Whenever a flow takes place between procedures in either direction, the procedure to be initiated begins in one of the following ways:

- Display First—This option shows the window or dialog box associated with the destination procedure and waits for user input to begin the logic of the procedure. Display first can be set only if the procedure is designated "online with a display" and contains a window or dialog box. A procedure that meets these criteria is a client procedure.
- Execute First—This option executes the logic of the procedure, which results in either the display of the procedure's window or dialog box or the execution of a flow. The distinction relies on the definition of a flow action associated with each procedure.

These are the guidelines for selecting a flow action:

- If the destination procedure requires information from the user before it can execute, the Display First option is used.
- If the procedure is a server, which contains no windows or dialog boxes, Display First is not an option.
- If the source procedure provides all the information required by the destination procedure, the Execute First option is used.

Using Autoflows

In many cases, the only processing associated with a value of command is the setting of an exit state with a particular value. An autoflow is appropriate if the only logic placed behind a GUI object on a window or dialog box sets an exit state to cause a flow.

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 is called an autoflow because it can be used to initiate a flow without any action diagram logic to support it.

At execution time, autoflows are processed as follows:

1. The CA Gen window manager evaluates the command special attribute. If the command special attribute value contains 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. If the value of exit state is associated with a flow, the flow happens as if Exit State had been set with the EXIT STATE IS action in an action diagram.

Executing a Command via a Flow

Most server procedures rely on the value in the command special attribute to direct their processing.

When a procedure is initiated from a window or dialog box, the user selects a push button, menu item, toolbar item, and so on, to influence procedure execution. The same kind of value is required when a procedure is initiated by a flow from another procedure.

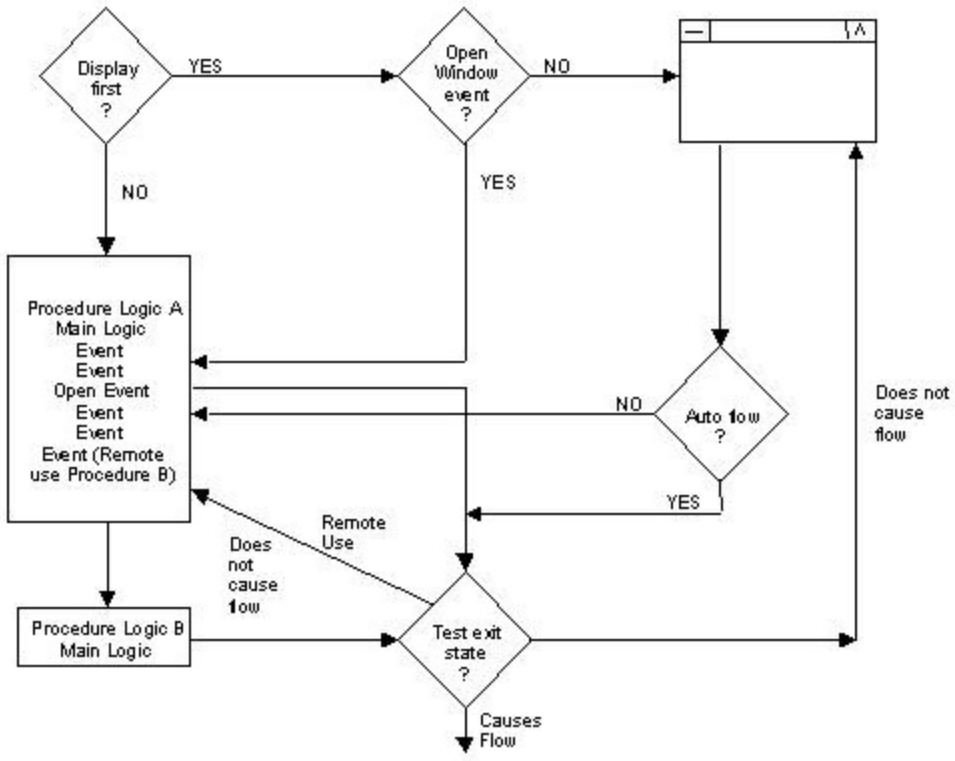
A command value can be specified for each direction of a flow. When the destination procedure of the flow executes, its underlying procedure action diagram treats the command value exactly as if it were entered on a window or dialog box.

You can choose to send one of the following:

- A specific command value
- For a transfer flow, specify the command value that triggers the destination procedure.
- For a link flow, specify both the command value that triggers the destination procedure, and the command value that triggers the return to the source procedure.
- The current value of command (its value at the conclusion of the source procedure's execution)
- For client/server, you will most often send the command of current. This lets the server know what the client is trying to accomplish.
- In the case of a return from a link, the previous value of command (the value in the command field when the procedure that initiated the link finished executing)

How Window Manager Controls a Flow

The following illustration shows how CA Gen's window manager controls a flow. (The diamond shaped boxes are the decisions being made by the window manager.)



The window manager controls a flow as follows:

1. As a procedure is being entered, the CA Gen window manager checks to see if the procedure is Execute First or Display First.
   a. If the procedure is Display First, the window manager checks to see if the procedure contains an open window event.
   b. If the procedure contains an open window event, the event is executed, and the primary window or dialog box is opened.
   c. If the procedure does not contain an open window event, the primary window or dialog box is opened. See How Window Manager Controls a Flow to learn more about open window events.
   d. If the procedure is Execute First, the main procedure logic is executed. See Choosing a Flow Action to learn more about the Execute First procedure.
2. At the end of the execution of the logic, the window manager checks to see if an exit state is set to cause a flow:
   a. If the exit state is set to cause a flow, go to step 1.
   b. If the exit state does not cause a flow, the primary window or dialog box associated with this procedure is opened.
3. For each selected GUI control that has an associated event, only the event handler logic executes. At the end of each event logic execution, the window manager will check to see if an exit state is set to cause a flow:
   a. If the exit state is set to cause a flow, go to step 1.
   b. If the exit state does not cause a flow, the window is presented, along with any dialog box that may have been opened in the event.
4. If any of the GUI controls are associated with a command that is designed as an autoflow on a flow the flow executes immediately. Go to step 1. See Using Autoflows to learn more about executing flows in this manner.
5. During any of the procedures main logic or event logic, if a remote use to another procedure is executed, the control is passed to the destination procedure. Upon return to this procedure, the window manager checks the value of the exit state to see if it causes a flow to another procedure.

Choose Flow or Remote Use

Procedure Step USE is an action diagram statement that allows interaction between procedures.

The procedure is still the commit point whether it is flowed to or used.

The following table shows the differences between the flow and Procedure Step USE procedure interactions.

Comparison of Flow and Remote Use

Design Parameter	Link Flow	Procedure Step USE
Command	Passed with <current>, or it can be changed along the flow	Passed to the procedure being used
Exit State	Causes a flow	Passed to the procedure being used
Views	View matched	View matched
Return from Procedure	The window manager: Interrogates the exit state to see if it causes another flow (if so, the exit state flows). If there is no flow, interrogates the exit state to see if it has a message associated with it (if so, the message is displayed). Executes the main logic.	

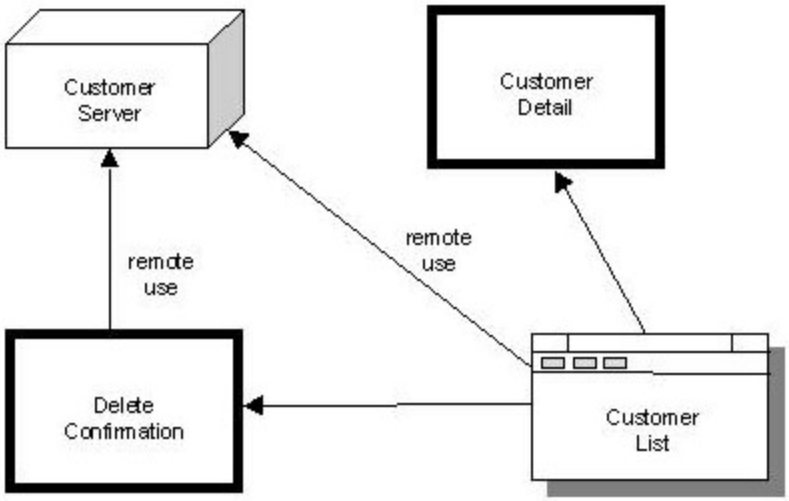
How to Design Procedure Interactions

A client procedure may have one primary window or dialog box and as many secondary dialog boxes as needed for the application. This allows two approaches for providing the same information in designing procedure interactions:

- Single procedure-using separate dialog boxes for each detail function
- Multi-procedure-using one detail dialog box with disable flags

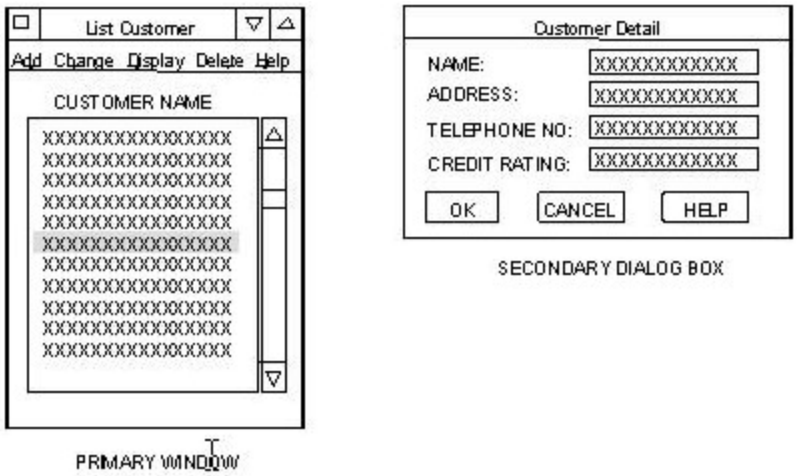
Single-Procedure Approach

One option in graphical dialog design is to include all listing display and maintenance functionality in a single procedure, as shown in the following illustration.



The procedure uses secondary dialog boxes as an auxiliary aid to a primary window.

The following illustration shows an example of the window interaction for a procedure called MAINTAIN CUSTOMER.



The following list explains the example in the figure:

- The List Customer window is a primary window used to select objects and actions. This primary window is the focus of the procedure.
- The customer detail secondary dialog box is used as part of the list customer procedure.

- Similar secondary dialog boxes are also specified for add and for update functions.
- A listbox is used on the update and detail dialog boxes, so that the customer occurrence selected in the list is the one displayed in the customer detail dialog box.
- The detail dialog box displays all the fields but does not allow changes.
- The update dialog box allows changes to be made to the fields.
- The Add dialog box requires a single occurrence input view so that the dialog box is displayed with empty fields into which the new customer details can be added.

If you add a new attribute to an entity, and you need to include that attribute on the dialog boxes, you must add the same field to all three dialog boxes. Even a small modification can introduce synchronization errors, and large changes can be extremely difficult to keep compatible in all three dialog boxes. For example, adding several fields and changing field order can introduce numerous synchronization errors.

Users will probably want the three dialog boxes to look the same except for field protection levels.

Another way to modify the design is to specify only one secondary dialog box, based on the Display Customer Detail dialog box. You then modify the appropriate field properties to support the Add, Update, and Delete commands.

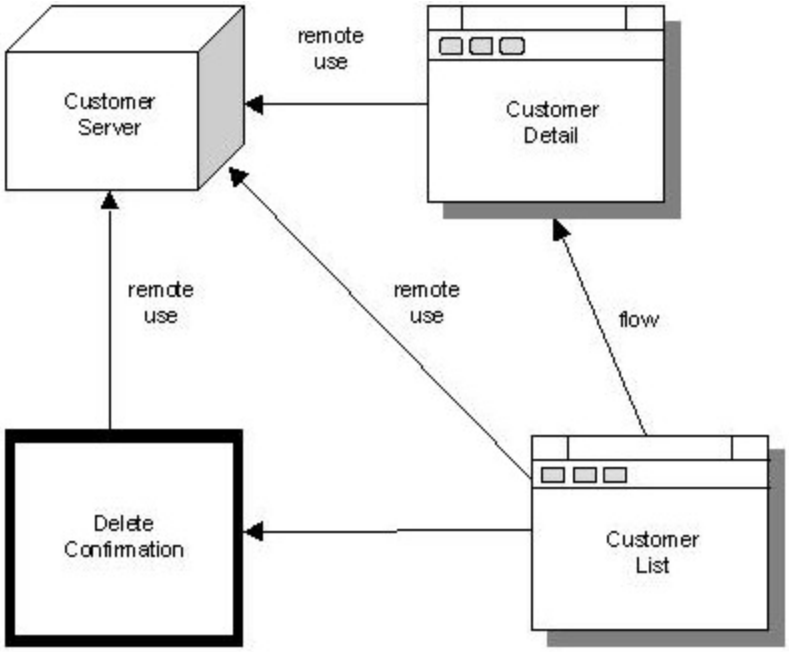
Performance may suffer from this approach. Using a list occurrence requires that all data to populate the dialog box be retrieved in the first execution. No procedure logic is executed when the dialog box is initiated. CA Gen uses the values in the selected row in the list and matches them to the fields in the dialog box. If a small amount of data is involved, the disadvantage is minimal. On the other hand, if views of several entity types are to be displayed, this could mean an extensive overhead in building the list.

More information:

Designing the Graphical User Interface (see page 117)

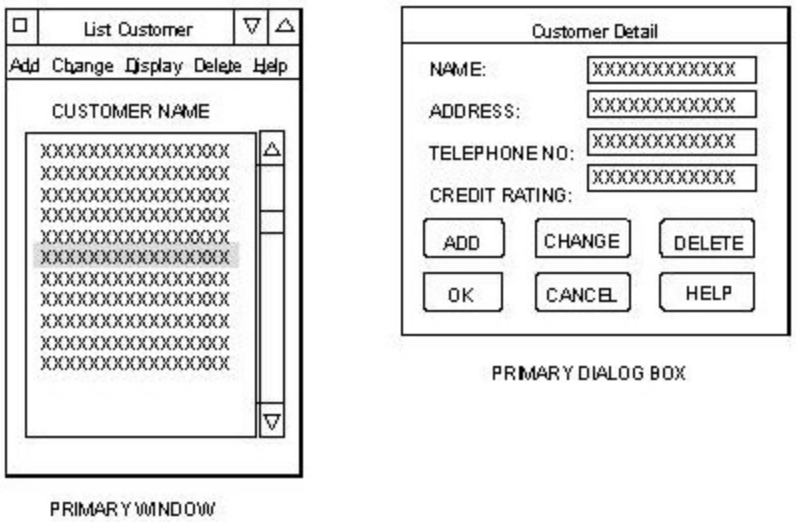
Multi-Procedure Approach

In this approach, the detail dialog box is implemented as a primary dialog box in a separate procedure that links back to the list customer procedure. An example of this approach is illustrated in the following illustration.



Using two procedures, with one detail dialog box employing disable flags in this way, results in windows that should be easier to maintain and execute because no flow is needed.

Consider the following illustration. Compare this illustration with the second illustration in the section Single-Procedure Approach.



The following list explains the previous illustration:

- List Customer is a primary window.
- Detail Customer is now a primary dialog box in a second procedure.
- Customer action buttons have been added to the dialog box.
- Embedding the action requires maintaining just one dialog box for add, update, and display actions.

In this primary window or primary dialog box interaction, the record selected and the action requested field both need to be passed or mapped to the detail customer procedure using view matching.

More information:

Designing the Procedure Logic (see page 163)

Comparison of the Single and Multi-Procedure Approaches

Using single procedures, with secondary dialog boxes, windows that can be modified by export view values, and modeless windows, offers greater flexibility in the design of the dialog.

The following list explains other advantages of the single-procedure approach:

- Allows the user to display multiple occurrences of a window, or to switch to another window without quitting the first window

- Furnishes additional action-related information without cluttering the primary window
- Simplifies the design by consolidating list and maintenance functions into one procedure

The following table points out the differences between the two approaches.

More information:

Designing the Graphical User Interface (see page 117)

Comparison of Approaches

Single-Procedure Approach	Multi-Procedure Approach
Multiple secondary dialog boxes	One detail version with disable flags
LIST and MAINTAIN in one procedure	LIST and MAINTAIN in at least two procedures
Menu items disabled according to function	Push buttons disabled according to function
Selected object used as a listbox on the appropriate dialog boxes	Select list occurrence moved to the output field and view-matched to the input field of the called procedure

These are situations where several procedures may be preferred to a single procedure:

- Processing associated with a dialog box needs be performed before it is displayed, for instance, to display a value that is derived or computed from primary window input as an output field.
- The procedure is too large and should be divided into more manageable units.

Handle Termination Conditions

The database commit point for a CA Gen application is upon successful execution of a procedure. This is true for both client and server procedures and whether the procedure interaction is a flow or remote use.

If an error occurs in the procedure, a rollback is to be performed.

Exit states are also used to determine the condition of the database access of the procedure.

You can assign the termination action to the exit state.

The server manager handles termination conditions for server procedures. CA Gen invokes the server manager at the end of the execution of a server procedure.

The window manager handles termination conditions for client procedures. CA Gen invokes the window manager at the end of the execution of a client procedure or at the end of the execution of an event handler.

Map Client and Server Procedure Interaction

After the initial system structure is complete, some of the identified procedures will need to be decomposed into client and server procedures. When you finish this decomposition, you will map these procedures. This task is called client-to-server mapping. Flows are not allowed to non-cooperative procedure steps.

Client Procedure Characteristics

These are the characteristics of a client procedure:

- Usually has a graphical user interface
- Should be designated online with display
- Can flow to other client procedures
- Can flow to server procedures
- Can remote use other client procedures if the client procedures are both implemented on the same platform
- Can remote use server procedures
- Contains a commit point at the end of execution

Server Procedure Characteristics

These are the characteristics of a server procedure:

- Does not contain a user interface
- Must be designated online without display
- Can return flow to client procedures
- Contains a commit point at the end of execution

Procedure Mapping Options

The options for mapping client and server procedures are:

- One client procedure/No server procedures
- One client procedure/One server procedure
- One client procedure/Many server procedures
- Many client procedures/Many server procedures

One Client Procedure and No Server Procedures Mapping

This mapping assumes that all processing is performed on the client. Data is stored remotely on a server and accessed directly from the client procedure using a DBMS to obtain data from the server. This is known as the remote data style of client/server.

Maintenance is easier with this approach because all processing is in one place. If there is no data maintenance, the procedure is there to facilitate the display of data.

However, a more powerful client machine is needed to run this kind of procedure Network traffic can be higher.

More information:

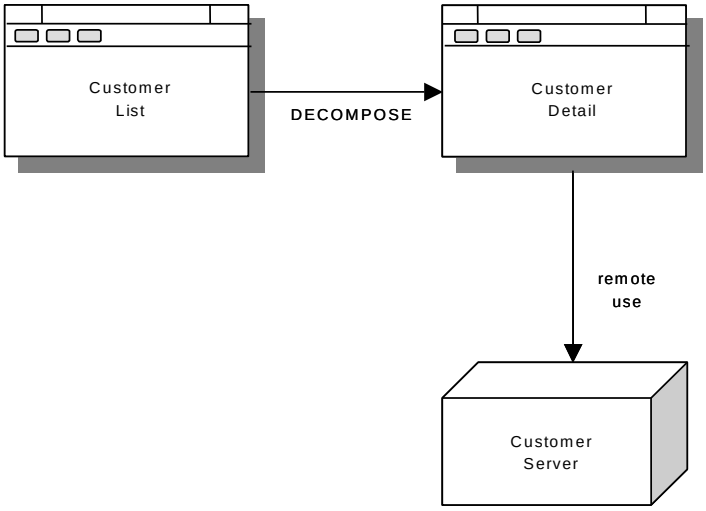
Designing the Procedure Logic (see page 163)

One Client Procedure and One Server Procedure Mapping

With this mapping, the client and server procedures are seen as closely coupled pairs:

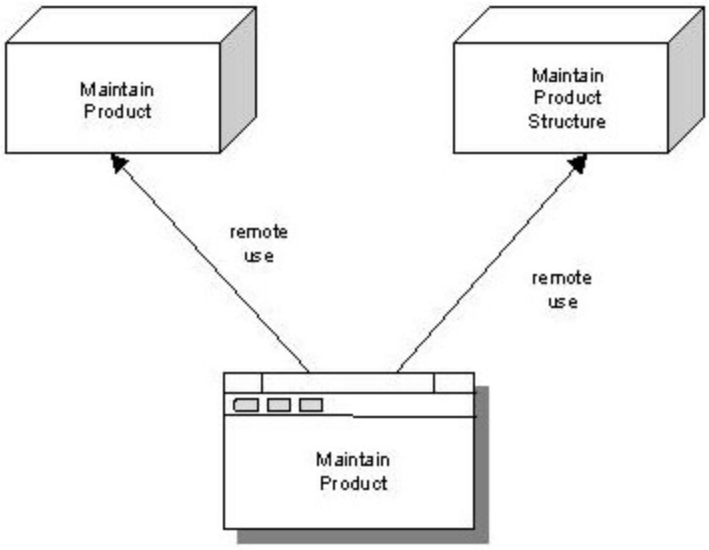
- The server procedure is developed specifically to serve the needs of the client.
- The client procedure is developed specifically to receive the data provided by the server procedure.

The following illustration shows this approach.



One Client Procedure and Many Server Procedures Mapping

This mapping implies a multi-functional client procedure being served by many server procedures, as shown in the following illustration.

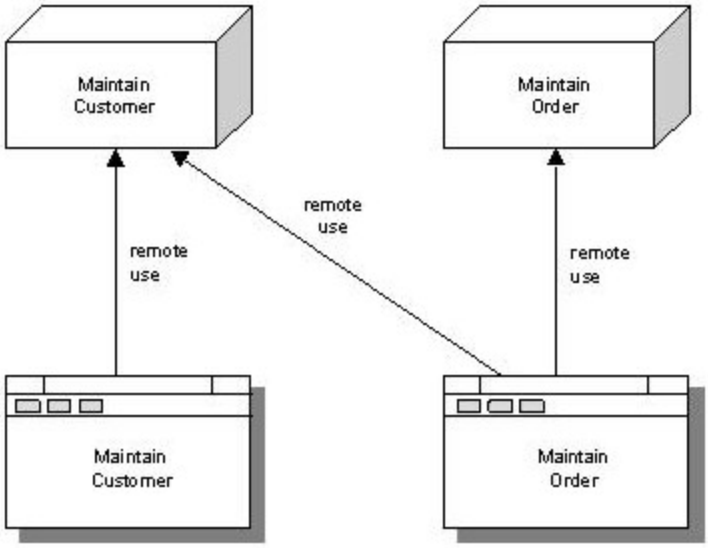


This mapping makes individual server procedures relatively simple and easy to maintain and reuse.

However, this approach makes the maintenance of the entire system slightly more difficult because several server procedures must be maintained at the same time. For instance, you will need to review the procedures that use a shared data definition each time that definition is changed.

Many Client Procedures and Many Server Procedures Mapping

This is an extension of the one-client procedure or many server procedures mapping. In this mapping, the functionality of each server procedure is implemented to make it reusable by many client procedures. This approach is illustrated in the following illustration.



Reusability is a laudable aim, but here it may mean a more complex server and client arrangement.

Results of Procedure Interaction Design

The results of procedure interaction design are mapped client and server procedures. Each mapping results in:

- A client procedure with associated windows and occasionally with local data access statements in its action diagram.
- A server procedure with no associated window(s) but one or more data access statements in its action diagram.

More information:

Designing the Procedure Logic (see page 163)
Designing the System Structure (see page 33)

Chapter 6: Designing the Graphical User Interface

This chapter describes how to design windows and dialog boxes. The Graphical User Interface (GUI) is the user interface built for client procedures. The GUI consists of windows and dialog boxes through which the user interacts with the application.

Client procedures are defined as online with a display. The display can be a GUI window, a dialog box, or some combination of windows and dialog boxes.

The GUI, procedure interaction, and procedure logic are best designed in parallel. GUI and procedure interactions may then be refined. Prototyping is an effective method for refining these interactions. When the initial design is stable, detailed procedure logic can be completed, and the GUI and procedure interactions finalized.

The GUI, procedure interaction, and procedure logic combine to yield a fully detailed user interface, a completed design navigation diagram, and action diagrams. These form the basis for generating a complete system.

More information:

Designing the System Structure (see page 33)
Designing the Procedure Interaction (see page 89)

Prerequisites

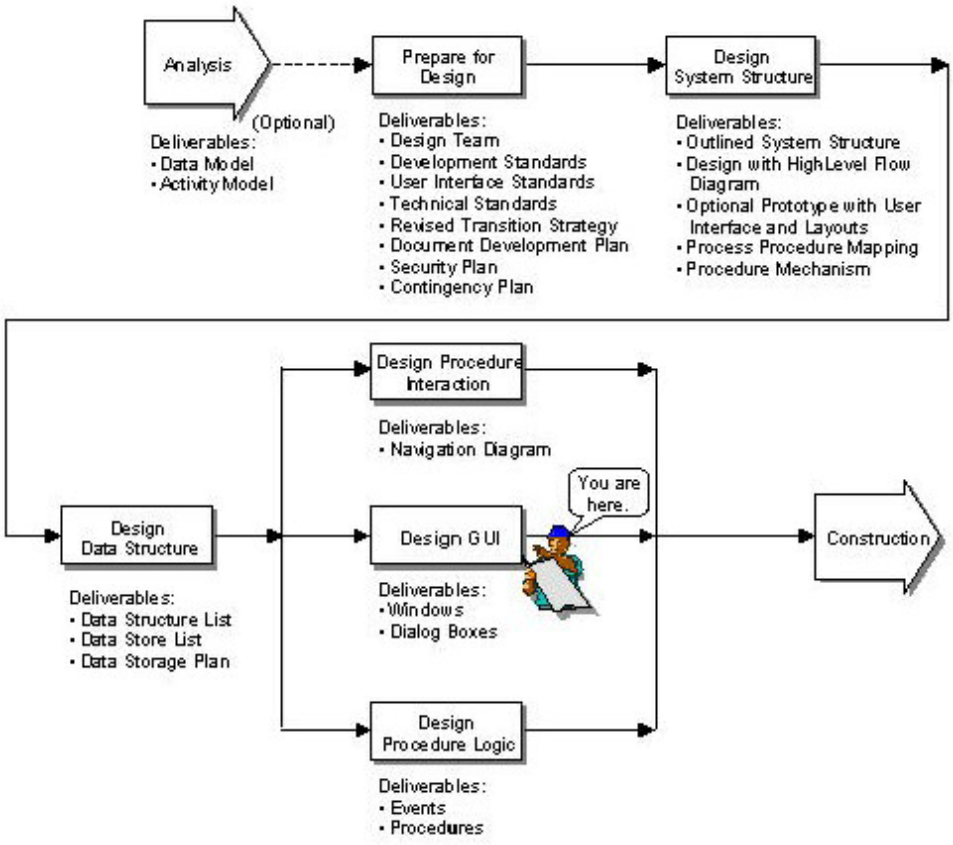
You need an outlined system structure design before beginning GUI design.

More information:

Designing the System Structure (see page 33)

Design Process

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



How to Learn About GUI

Note: You can skip this section if you regularly use sophisticated workstation tools that include graphical user interfaces.

If you are unfamiliar with the operation of a typical GUI, you should first become familiar with techniques for building an effective GUI. An effectively designed GUI provides visual cues so the user can quickly complete the correct tasks.

Do the following tasks to get started with GUIs:

- Acquire a workstation and some reasonable, commercially available workstation software. Spreadsheets and word processors running under Windows, Motif, or an alternative graphical environment will work well.
- Purchase one or more style guides. There are style manuals available that give guidance on creating user interfaces for all major environments, including the windows interface and Motif.
- Spend some time familiarizing yourself with the various elements of applications such as tool bars, menus, dialog boxes, and so on.
- Critique applications. Note the various areas that appear inconsistent. Identify the parts that are the easiest to use. It is even more effective to critique with a group.

This approach will help you to get a firm grip on both the grand scheme and the subtleties of how GUIs operate and what users expect to see.

Rapid Prototyping

CA Gen lets you rapidly build a prototype of the application for users to review.

CA Gen's Design Toolset provides window design functionality that uses definitions of views as a source of the data to be presented. If you retain these views, they can be promoted to data views after the GUI design is complete.

The data model does not have to be completely defined to design or prototype the user interface. If you wish to prototype flows as an aid to developing data and view definitions, you do not need to begin with prototypes produced using word processing or graphic tools. The CA Gen GUI designs can be arranged completely with or without the complete data model.

User Interface Design

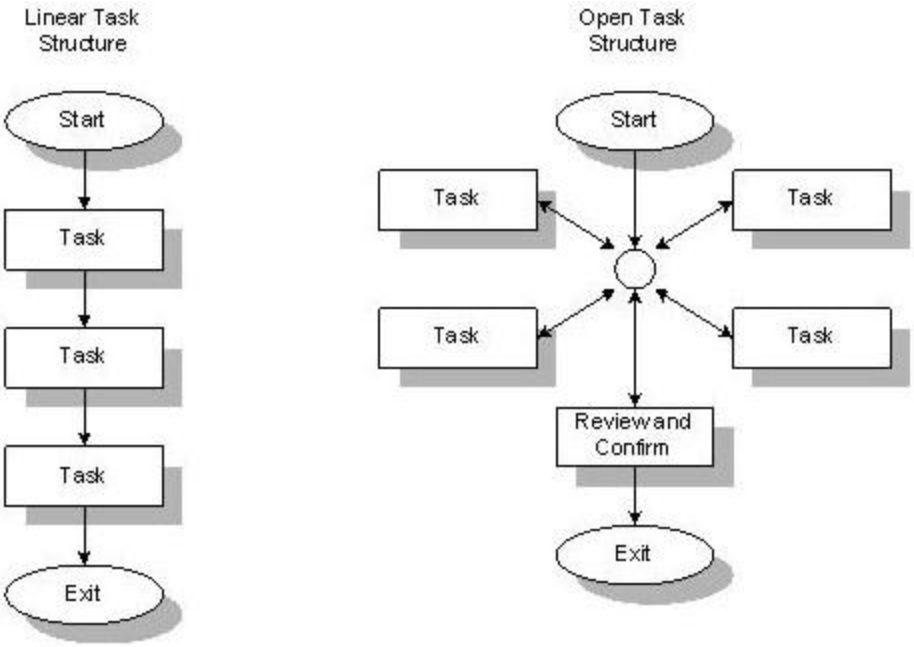
One of the fundamental principles of GUI is that the design must be closely mapped to the user's needs. To accomplish this, it is essential to have a thorough understanding of the users' needs. You must also understand who the users are and how they will use the interface. This understanding must form the basis of the windows you design.

Traditional block mode interfaces reflect an expectation that tasks are linear in nature. In a linear program, a well-defined series of steps should lead to a single definable goal. For example, consider the role of a data entry operator assigned to type data as quickly and accurately as possible. Such a task is repetitive and linear in nature, which makes it better served by a simple block mode, form-driven interface.

Many tasks are not repetitive or linear. Consider a more open-ended task such as word processing a document. This task may involve accessing several sources of information, such as other word-processed documents for text, a spreadsheet for a data, and a graphics package for a diagram. The performance of the task involves the ability to make decisions and, as a result, requires an open task structure.

An open task structure is also needed when the data entry operator's work may be interrupted by some external business event. The operator may need to suspend the task while switching to a higher priority assignment.

The linear versus open-task models are summarized in the following illustration.



The linear task structure is task-oriented. Block mode interfaces tend to be task-oriented.

The open-task structure is user-oriented. Properly designed GUIs tend to be user-oriented.

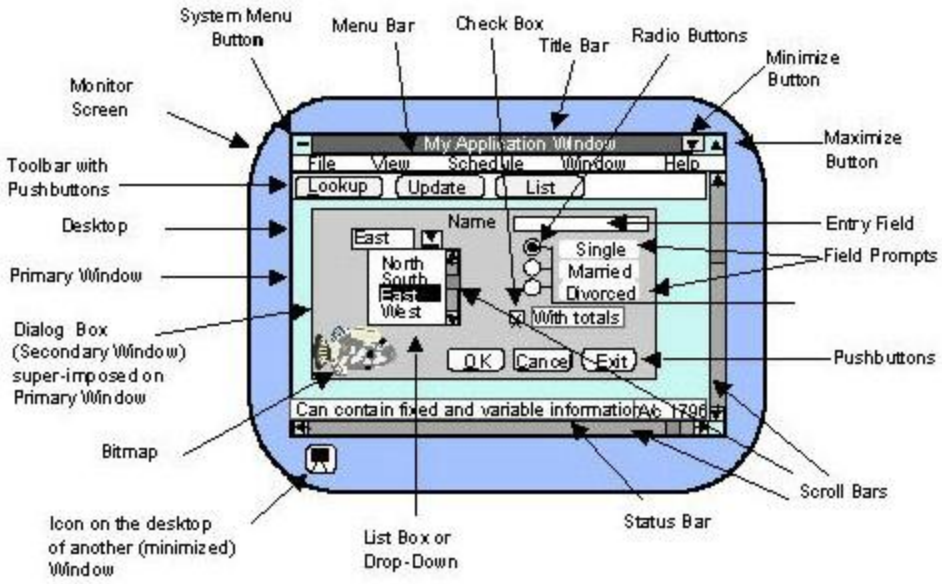
You should be aware of this major distinction as you move into the world of GUI design if you normally build block mode interfaces.

A GUI allows a closer mapping to the user task structure in the following ways:

- Interleaving—A user often starts a task, only to be interrupted with one of higher priority. The user suspends the first assignment while performing the high priority task. The user returns to the first task at the point of interruption after completing the high priority task.
- Single task, multiple application—The user employs several applications to complete one task. This task is usually of a managerial nature. For example, scheduling a meeting may require the use of a diary, electronic mail on the server. The user may also refer to the minutes of the previous meeting held as a word processed document on the client.
- Single application, multiple tasks—The user uses one application to complete several tasks. For example, employees use the electronic mail system for social interaction, cascading company notices to subordinate staff, and taking part in ad hoc discussions.
- Multiple representations—A variety of visual representations are used on the same underlying data. For example, numbers may be viewed one at a time, together in a table, or as a chart or graphic.
- Multiple sources—The user requires access to several sources of information or processes (typically on different machines) to complete the task.
- Reminding and feedback—The user may become disoriented in the computing environment, so information on what tasks are currently active (a *you are here* function) becomes essential.
- Context signaling and re-orienting—The users may require a reminder of what they were doing in a particular task. For example, on resuming a task the user is informed of the last command executed in the window before the task was suspended.

GUI Elements

The following illustration shows most of the elements of a graphical user interface.



The illustration does *not* show group boxes, literals, OLE areas, or OLE custom controls.

The following table lists all GUI elements for windows and dialog boxes.

GUI Element	Description	Window	Dialog Box	Provides Action	Provides Data	User Aid Only
Title Bar	Contains the name of the window or dialog box.	Yes	Yes	No	No	Yes
Menu Bar	Clicking each option in the menu bar reveals a drop-down list of menu items (not shown) designed by the application developer. Each menu item triggers its own piece of application logic.	Yes	No	Yes	No	No
Tool Bar	Optional element that may contain only push buttons for pieces of application logic.	Yes	No	Yes	No	No
Status Bar	Optional element that displays information, typically about the status of the process.	Yes	No	No	No	Yes

GUI Element	Description	Window	Dialog Box	Provides Action	Provides Data	User Aid Only
Scroll Bar	GUI scrolling device that allows the user to display information that is not visible on the screen.	Yes	Yes	Yes	No	No
System Menu	Operating system menu under the icon on the top left corner of the window.	Yes	No	Yes	No	No
Minimize Button	Shrinks the window to an icon on the desktop, where it can be maximized when you next need to view the window.	Yes	Yes	Yes	No	No
Maximize Button	Displays the window in the largest possible size, or in the size it was first opened, or last resized.	Yes	Yes	Yes	No	No
Check Box	Control that acts like a switch (on/off, yes/no, true/false, and so on).	Yes	Yes	Yes	No	No
Group Box	Margin box drawn around a collection of logically related fields.	Yes	Yes	No	No	Yes
List Box	Control that displays a series of values as a list.	Yes	Yes	No	Yes	No
Entry Field	Control into which the user types information.	Yes	Yes	No	Yes	No
Field Prompt	Text label that identifies a selection or entry field.	Yes	Yes	No	No	Yes
Literal	Text that is displayed but cannot be changed.	Yes	Yes	No	No	Yes
Radio Button	Control that offers one choice from among a group of choices.	Yes	Yes	Yes	No	No
Push button	Control that performs an action immediately when it is selected.	Yes	Yes	Yes	No	No
Icon	Small bitmap picture that represents a minimized window on the desktop.	Yes	Yes	Yes	No	No
Bitmaps	Graphics used to distinguish push buttons and as literals or icons.	Yes	Yes	No	No	Yes

GUI Element	Description	Window	Dialog Box	Provides Action	Provides Data	User Aid Only
OLE Area	In Win32 applications, an embedded area that allows communication with other applications.	Yes	Yes	Yes	Yes	No
OLE Custom Control (OCX)	In Win32 applications, an embedded control, such as a spin button.	Yes	Yes	Yes	Yes	No

Controls on Windows and Dialog Boxes

A control is a window component that allows the user to select choices, enter information, or both.

The following list shows some examples of controls:

- Entry fields
- Check boxes
- Radio buttons
- List boxes
- Push buttons

Windows and dialog boxes use controls to enable the user to interact with the application.

Because of the number of control types, GUI design is an approach that gives greater variety of choice and can increase functionality, but also can increase complexity and testing.

Primary Windows

A primary window is a re-sizable area of the desktop in which the user carries out the main dialog with the application.

The following list shows the elements of a primary window:

- Title bar
- Menu bar
- Tool bar
- Status bar
- Scroll bar
- System menu
- Minimize button
- Maximize button
- Check box
- Group box
- List box
- OLE area
- OLE custom control (OCX)
- Entry field
- Field prompt
- Literal
- Radio button
- Bitmaps

Dialog Boxes

A dialog box is a movable window that requests input from the user.

Dialog boxes are not resizable and do not have menu, tool, or status bars. Instead, they use push buttons to dictate procedure interaction.

- Primary Dialog Boxes—A primary dialog box is an alternative to using a primary window for a procedure.

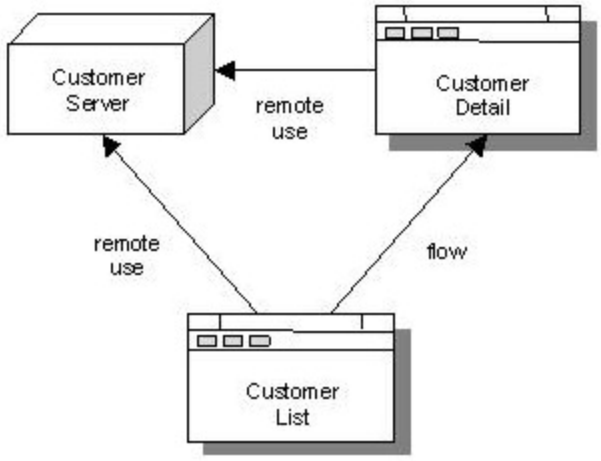
  A primary dialog box is appropriate when, for example, criteria need to be applied to retrieve data before it can be displayed, or when a password is required.

- Secondary Dialog Boxes—A secondary dialog box is a dialog box subordinate to a primary window or a primary dialog box.

  A secondary dialog box is often invoked either using a menu option from the menu bar in a window or a push button in another dialog box.

  A client procedure that is designated Online With Display may have one primary window or dialog box and as many secondary dialog boxes as needed for the application.

A secondary dialog box is dependent on its parent window or dialog box and may not be displayed on its own. The following illustration shows a primary window and secondary dialog box.



Note: Import views receive data. Export views populate display controls.

The following list shows the elements of a dialog box:

- Title bar
- Scroll bar
- Minimize button
- Maximize button
- Check box
- Group box
- List box
- OLE area
- OLE custom control (OCX)
- Entry field
- Field prompt
- Literal
- Radio button
- Push button
- Bitmaps

Message Boxes

A message box can display three types of information:

- Informational messages usually about status
- Action required by the user
- Warnings about problems

An informational message may or may not require the user to select a push button to end the message and close the box.

Action-required and warning message boxes require the user to select a push button to acknowledge the message.

CA Gen generates message boxes based on the value of the message associated with the exit state.

More information:

Designing the Procedure Interaction (see page 89)

Graphical Design Standards

Before identifying application standards, everyone involved should give careful thought to how an application should behave. You should discuss window appearance and how to implement common functions like adding, changing, or deleting database records.

Consistent application behavior will reduce training time and user errors. It will also decrease development and maintenance time and cost.

The standards committee should discuss these topics and understand how GUI issues affect the standards:

- Approaches for single windows
- Unit of work
- Modal versus modeless windows

Standardize the Approach for Single Windows

In setting standards, consider these approaches for single windows:

- Put all of the controls on one window
- Use progressive focus

All Controls on One Window Approach

In this approach, you design a window to handle all the tasks a user may want to perform. In this window, all the controls are available at the same time. For example, a list window may contain search criteria, sorts, filters, a detail area, and navigation buttons.

This approach lets the user enter search criteria, perform the search, select or add a row, make changes to it, save the changes, and flow to a related business object on one window.

All controls are available at the same time. This may confuse the user by presenting misleading results. For example:

- The user may perform a search, then change the search criteria, and then be distracted before performing the search. When the user returns from the distraction, there is no way to tell if the program performed the search. The user would perform the search again to be sure.
- If the user did not perform the search before the distraction, the data in the list would not correspond to the search criteria.

Progressive Focus Approach

Focus refers to the window or the part of a window that is active. We say that this window or part of the window is enabled.

For example, three windows are open on a desktop. The window whose title bar is highlighted has focus. This window is also referred to as the active window. A window typically becomes active when the mouse is clicked anywhere within the window's area.

Parts of a window can receive focus. For example, a radio button is clicked. Some of the window's controls are disabled and others are enabled. The enabled parts are said to have focus.

Only one GUI control can have focus at any given time. This is usually the control that contains the cursor on the active window.

Progressive focus refers to the windows, or the parts of a window, that will come into focus to complete a transaction.

The progressive focus approach minimizes the number of functions per window to make each window simple, user friendly, and limits focus to only the controls that pertain to the action being performed. Such a window might contain only a list and buttons.

Follow these steps:

1. Push a button or select a menu item to flow to a search window containing all the search criteria fields.

   The search window now has focus.

2. Enter information in the fields and click OK.

   The search is performed if the data passed the validations. The search criteria fields are displayed and protected at the top of the list window, so focus would be on the list, not the search criteria fields.

3. Press the appropriate button to add, change, or delete a record to flow to a window to perform the appropriate transaction.

   The resulting window has focus.

4. Click OK on the window.

   The window closes. The database is updated. The list containing the new data regains focus.

5. Click OK to perform the search.

   If the list window has focus, the search was performed.

In this approach, the user knows what steps are completed, based on which window (search criteria window, list window, and so forth) has focus.

Also, there would never be an instance where the data in the list does not correspond to the search criteria. Therefore, there are no misleading results.

The same holds true for updating a record. If the user were distracted while updating a record, the user would have no way of knowing if he committed the change when he returned. With progressive focus, the window performing the change transaction would still have focus, which lets the user that the change has not been committed.

Balance the Two Approaches

You should choose a happy medium between the two approaches. Apply the progressive approach to its fullest and then lessen it where the approach would hinder more than it helps the users.

This strategy is similar to developing a data model. Just as an analyst strives to build a third normal form data model and then downgrades to second normal form in some areas for performance reasons, you should design for progressive focus and downgrade in certain areas depending on the intended users' skill set and work flow needs.

In a client/server architecture, you must also consider the network. A design that gives the user focus on all controls at once might lead to unnecessary network traffic for actions that the user performed by mistake or are irrelevant.

Standardize the Unit of Work

This standards committee topic requires the analysis of a business scenario to determine the unit of work before windows are designed to support it.

A unit of work is the set of data necessary to complete a transaction.

A unit of work is usually a commit point, since it is the point where committing the data to the database completes the transaction. Since a commit occurs at the end of a unit of work, the unit of work must not violate any business rules.

For example, to add entity A, which has a mandatory relationship with entity B, the unit of work may include adding attributes of entity A and either associating it to an existing entity B or allowing the creation of entity B at the same time as entity A.

A unit of work may require more than one window to capture all necessary data.

Some transactions consist of a defined set of data while others consist of a minimum set of data. The unit of work for adding a customer, for example, might require that the customer's name, address and phone number be entered. This type of unit of work is typically applied to entity type maintenance.

As a minimum, taking an order may require the order header information, one customer, and one product. However, a customer may order several products at once. Therefore, the unit of work for taking an order requires the order header, one customer, and one or more products. This type of unit of work is typically applied to one-to-many relationship and many-to-many relationship maintenance.

Another example of the minimum set of data concept is when two entities exist and all a user needs to do is associate the two. This type of unit of work is typically applied to many-to-many relationship maintenance.

Window design standards should define the presentation and processing method for each type of unit of work-a defined set of data or minimum set of data.

The processing methods consider when the commit occurs. For entity type maintenance, whatever is on the window is applied to the database.

For the minimum set of data scenario, the user may visit many windows adding, updating, and removing entities or associating and disassociating entities. This brings up several questions such as listed next:

- Does the commit occur after each add, update, remove, associate, or disassociate?
- Is the data collected until the user is finished and then applied to the database?

If the answer is yes, these questions need answers:

- Is all the old data removed and the new data added?
- Are just the user's changes recorded and applied to the database?

You should also consider network traffic, view size, and processing complexity.

Standardize Modality on Windows

This standards committee topic requires deciding whether windows should be modal or modeless. For more information, see Set the Modality of Windows and Dialog Boxes.

Modal Windows

Modal processing is a way of enforcing the progressive focus approach. For a definition of focus, see Progressive Focus Approach.

A modal window is displayed in front of the window that invoked it. The modal window has focus. The user cannot obtain any previous window while the modal window has focus. However, the user can move the current modal window to view the contents of the previous window.

This approach ensures that the user completes the unit of work or cancels it before returning to the previous window.

Modeless Windows

Modeless processing allows access to more than one window without closing the active one first. This works well with unrelated objects.

These are sources of confusion with modeless dialogs:

- Two modeless windows displaying the same object can create havoc on the desktop and even cause data integrity problems or errors.
- If the user accidentally processes changes from two windows containing the same object, the later change would either wipe out the effects of the first or cause an error to occur if it is caught.
- There may be valid business reasons for needing to view different occurrences of the same object at the same time. For example, a user may need to compare multiple versions of a quote side by side.
- Another source for confusion is the implementation of implicit commit.
- If the user selects a row from a list and flows to a modeless detail window, makes changes, and then selects another row from the list, the changes are implicitly committed as long as no business rules are violated.
- If business rules are violated, the user is put into an awkward situation of answering validation checks before the change of focus can occur.

If there is a good case for modeless processing, guidelines should be set for processing considerations, such as the list window keeps track of which occurrences are open so the same occurrence cannot be opened twice.

Modeless processing is usually used for menu processing to flow between applications.

The primary windows attached to a menu are modeless. After a primary window is invoked, the user can click on another window selection on the menu to invoke another application's primary window. Then the two applications can run simultaneously, and the user can switch between them. However, within each window, the dialog traversed by the user would be modal.

GUI Design

You can use CA Gen to design windows and dialog boxes.

Note: The guidelines presented here are reasonable for GUI design using CA Gen. Do not construe them as hard rules. The user of the application is the final arbiter of what constitutes good practice.

Guidelines for Window and Dialog Box Design

Consider the following guidelines for designing windows and dialog boxes:

- When in doubt, make a window or dialog box modeless.
- Use the modal style only when there is a good reason (for example, when it is impossible to imagine the application continuing unless the interaction completes).
- Lay out controls with the specific user in mind.
- Place a logical unit of work on a window.
- Use pop-up dialog boxes for infrequently used data or for users to keep track of their work flow.
- Use a large window or dialog box instead of numerous pop-ups. This will have less fragmentation of work flow and is more efficient for the user.
- For specific pop-ups, provide an indicator on the primary window to inform the user if exits in the pop-up.

Set the Modality of Windows and Dialog Boxes

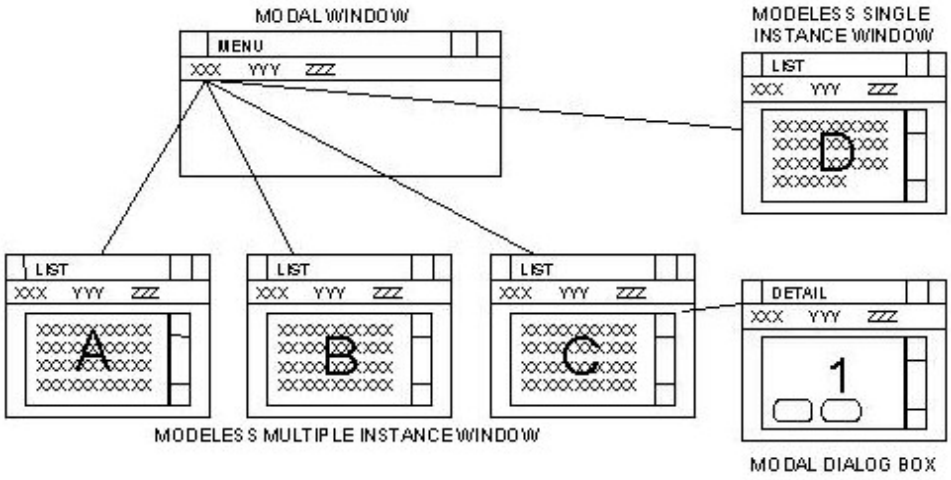
The modality of a system describes the freedom of movement that a user has among system components. This freedom involves the number of windows that a user can access and have open at any time.

Types of Modality

The following table lists the types of modality you can use in order of increasing freedom to the user.

Type	Description
Modal	In a fully modal system, the user may follow only one path through the dialog at any one time. When a user opens a modal window, other windows in the same application may be opened only by selecting commands directly from that window, or from windows opened as a result of those commands, and so forth.
Modeless Single-Instance	A modeless single instance window allows the user to switch to another window without closing the first window.
Modeless Multiple-Instance	A modeless multiple instance window allows the user to display multiple occurrences of that window, or to switch to another window without closing the first window.

The following illustration shows the layouts for part of a system where the modality has been varied.



In the illustration, note that only one modal Menu window may be displayed.

The List windows A, B, and C are all displays of the same primary modeless multiple window. They are all associated with the same procedure.

The dialog box 1 is defined as modal; when this dialog box is displayed it must be closed before the user can return to any other window.

Window D is modeless single instance. This lets the user switch to other windows (A, B, C or menu), but only one display of window D is ever shown. This avoids the confusion that can occur when using modeless-multiple instances (windows A, B, C) where the same data can be displayed in many windows. Thus if a user edits the data in one of the many windows displayed, confusion might result as to which window contains the updated data.

Guidelines for Setting Modality

You need to mirror how the user completes a series of user activities in a user task. The following table describes the options for modality on windows and dialog boxes.

If. . .	Use. . .
The task requires browsing or comparing many types of data before finalizing a task.	Modeless Window
The task requires browsing or comparing many instances of the same type of data before finalizing a task.	Modeless Multiple-Instance Window
Each discrete activity performed by a window's procedure must be completed before the user moves on to the next activity (guides the user through the work and helps to maintain data consistency).	Modal Window or Modeless Single-Instance Window
User actions need to be confirmed because there is a chance of an action being performed for the wrong object. This can be set as a development standard, for example, use a confirmation dialog box for all deletions.	Modal dialog box
Messages are presented to the user by exit states.	Modal Dialog Box

Set the Initial Window or Dialog Box Position

Initial position specifies the initial placement of any window or dialog box on the desktop. The following list describes the placement options for windows and dialog boxes.

- Mouse Alignment—The center of the generated window or dialog box falls wherever the mouse pointer happens to be.

  You may not want to use this placement because it will probably require user to move the window or dialog box to use it.

- Designed Position—The window or dialog box is placed where you indicate during design.

- Modeless Multiple-Instance—A modeless multiple instance window lets the user display multiple occurrences of that window, or to switch to another window without closing the first window.
- System-Placed—The operating system determines where to place the window or dialog box.

  Depending on the resolution of the users monitor, you may want to use the system placed position to let the monitor use the best fit algorithm.

The following list details guidelines for positioning windows and dialog boxes:

- Position windows and dialog boxes as you expect the user would position them.
- Size and position the window to be immediately useful when it is opened.
- Consider the window's or dialog box's relationship to other windows or dialog boxes opened at the same time.

Bitmaps

You can place bitmaps on push buttons, window backgrounds, and dialog box backgrounds. You can also use bitmaps as literals or icons. An icon is a small picture used to represent a minimized window on the desktop.

You can import bitmaps from any source that creates .bmp files.

Use bitmaps that can be easily recognized by the user. The image used should instantly convey its meaning and should not be more complex than the space allows. For example, avoid complex images on icons.

The state of a push button can be normal, disabled, activated, or selected. CA Gen enables the state of a push button bitmap to be displayed below the bitmap when the cursor is positioned over it for more than a few seconds. This is referred to as micro help.

Note: Bitmaps can be time-consuming to develop. When possible, either purchase bitmaps or have a graphic artist develop them.

Bitmaps can be dynamically changed during runtime through action diagramming statements.

Literals

A literal is a text or a bitmap that cannot be changed on windows and dialog boxes.

A text literal can contain multiple lines. Text literals wrap if they are not vertically centered and do not fit horizontally.

For literals, you can specify color and font:

- Color-Fields can appear in any of a defined palette of colors, or a mixed color, provided that the color is supported by the target workstation and its display
- Font-You can select the style and design of characters.

Visual Cues

A graphical user interface lets the application communicate flexibly and sometimes subliminally with the user.

Visual cues can be as listed next:

- Design visual cues
- Automatically created by the GUI environment
- Under the control of the application

Designed Visual Cues

The following list explains guidelines for designed visual cues:

- Have the current status of the application (for example: current customer name, or number of units of a product in stock, fact that the user is currently creating an order) continually updated at the bottom of the window.
- Distinguish different windows and parts of a single window by applying color and font differences or by including a picture.
- Disable buttons and menu items. They will be grayed out to show that they are unavailable at this moment.
- Enable the application to give the user visual feedback as actions are carried out.

  For example, icons are highlighted as they are selected, the mouse pointer may change shape as it moves over different areas of the screen, and so on.

More information:

Fonts and Colors on Windows and Dialog Boxes (see page 138)

GUI Environment Visual Cues

Many visual cues are under the control of the GUI environment. For example, radio buttons and push buttons appear to be physically pushed when the user clicks on them. They create this special effect by the appearance of a shadow around the edge.

Application Visual Cues

Many visual cues are under the control of the application, such as the disabling (or graying out) of fields, menu items, and push buttons when their use is inappropriate. For example, the OK button should be disabled until all the mandatory fields on a window have been completed.

Menu items should be disabled to prevent the user performing actions that are out of sequence, not relevant to the current processing or for which the user is not authorized.

Menu options can also be disabled to advise the user that a particular action is currently inappropriate.

Minimize and Maximize Buttons

A minimize button enables the user to shrink the window to an icon on the desktop, where it can be maximized when the user next needs to view the window.

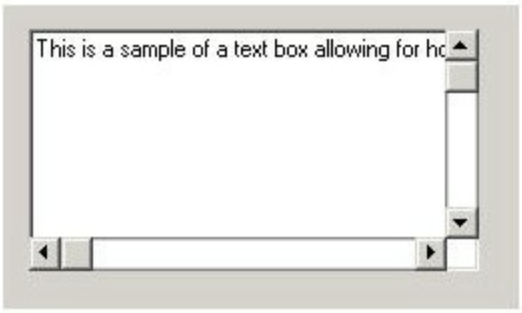
A maximize button enables the user to display the window either in the largest possible size, or in the size it was first opened, or last resized.

Use maximize and minimize buttons only on primary (top level) or modeless windows.

About Scroll Bars

A scroll bar is a scrolling device used in windows, dialog boxes, list boxes, and multi-line entry fields to let users see information that is too large to fit in the displayed view. Scroll bars conserve space.

A scroll bar consists of a scrolling area, a slider box, and boxes with arrows at each end. The following illustration shows horizontal and vertical scroll bars.



A CA Gen-generated application automatically handles vertical and horizontal scrolling on list boxes. Horizontal scrolling of prompts at the head of a list is synchronized with scrolling of the data to maintain the visual association of field names and field values.

Fonts and Colors on Windows and Dialog Boxes

There are a number of studies on best use of colors, use of colors in combinations (including undesirable combinations), and colors and fonts that are most effective.

The following list explains guidelines for using fonts and colors:

- Keep the concept LESS IS BETTER as your primary objective.
- Consider user community preferences.
- Avoid a riot of color and fonts, which may confuse rather than assist or please the user.
- Establish font and color standards to ensure consistency within and across applications.
- Avoid highlighting too many items in a window.

Note: Be aware that the fonts you select must be installed on every application user's PC.

You can dynamically change the colors and fonts at runtime using action diagramming statements.

Cursor Sequencing

The following list explains guidelines for sequencing the cursor:

- Make sure there is a well-defined default sequence of tabbing between fields.
- The cursor should normally tab through the fields from left to right and top to bottom.
- When the window is first displayed, the cursor should be placed in the first unprotected field in the sequence.

Title Bars Design

The title bar contains the name of the window or dialog box.

The following list explains guidelines for title bars:

- Wherever possible, use the name of the procedure supported by a window or dialog box as the title of that window or dialog box on the title bar.
- Titles on the title bar are generally objects (nouns) prefixed by actions (verbs), for example, Accept Policy, Create Mail, Confirm Order.

Menu Bars Design

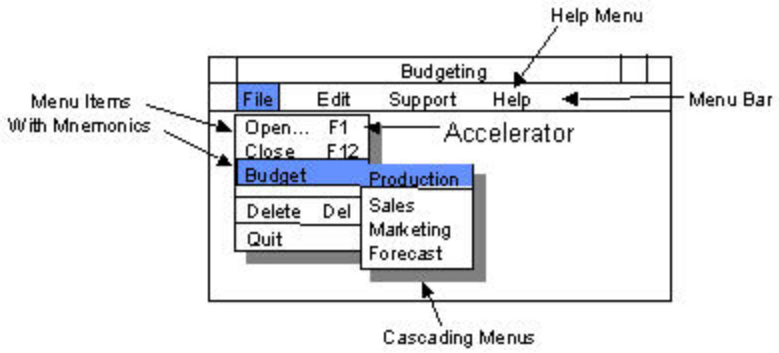
All windows should have a menu bar. Menu bars are not used on dialog boxes.

Selecting a menu item results in one of the following actions for the user of the CA Gen application:

- Initiates a flow due to a command set that is an autoflow.
- Opens a supplemental dialog box to this procedure
- Executes the procedure from the top.
- Executes an event.
- Executes a special action.

The following list explains guidelines for menu bars:

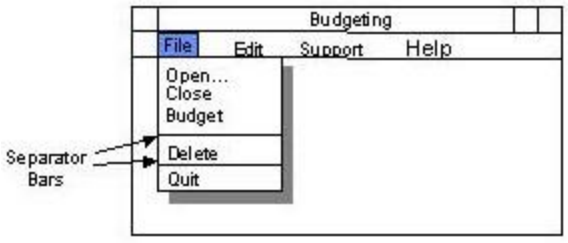
- Place functionality into broad categories on the menu bar.
- Limit the number of items on the menu bar. A rule of thumb is a maximum of 10 items.
- You may choose to design in the window and help menu items.
- Use menu names that would be of interest to the user, for example, Policy, Payment, and Customer.
- The drop-down list of cascading menu items underneath each menu should contain actions that can be performed on that object, for example, New, Change, and Display. The following illustration shows the presentation of the drop-down menus.



- Capitalize the first letter of each word, except for *noise words* (such as a, the, and of). All other letters should be lowercase.
- Assign a mnemonic to each menu and each menu item to let the user use the keyboard instead of the mouse.
- If the selection of an option results in the opening of a dialog box, add an ellipsis (...) to the menu option. For example, Rename as... opens a dialog for the user to enter a new name.

- Assign an accelerator (also known as a *hot key*) to a frequently used menu item.
- The accelerator should be listed to the right of the menu item on the drop-down.
- An accelerator is a key or combination of keys that perform some function defined by the application. The accelerator allows the menu item can be selected with the minimal number of key strokes, no matter at what level the menu item is located.
- Available key options are F1 to F12, 0 to 9, and A to Z. You can combine these options with Ctrl, Alt or Shift to provide more options.
- Place the most frequently used items first in the menu drop-down.
- Place sub-menu options as follows:
  - Place in order of frequency when top items are used most of the time
  - Place in order of importance to show critical items first
  - Place in alphabetic order if the user knows the exact wording of each item
  - Place logically when the order will show work flow or other criteria are not met
- Always gray out unavailable menu items.
- Use separator lines to visually separate choices on a drop-down menu.
- A separator line is a horizontal that focuses the user's eye on the fact that the action within the lines is not necessarily consistent with the rest of the menu items.
- You should use separator lines for those menu item actions that could cause a significant change to the database or window, such as:
  - Select All
  - Deselect All
  - Delete

  The following illustration shows the use of separators for the Delete and Quit menu items.



When to use a drop-down menu instead of the menu bar:

- Invoking an action from the menu bar itself should be discouraged. Users find them awkward and hard to notice because they are more the exception.

- If an action would be the only item in the drop-down menu, try to place it within another drop-down menu, using separator bars to logically differentiate it from the other items.
- If you must place the action in the menu bar, make it stand out from the other menu bar items by placing an "!" after the test to indicate that there are no options under the item and the action will be invoked when selected.

More information:

Designing the Procedure Interaction (see page 89)
Designing the Procedure Logic (see page 163)
Special Actions for Menu Items and Push Buttons (see page 154)
Mnemonics (see page 141)

Mnemonics

A mnemonic is an assigned character in a menu item, push button, or selection field prompt. The user can type the mnemonic, in uppercase or lowercase, for a choice to select it instead of using the mouse.

The mnemonic is identified by an underscore. This is a visual cue to the user that the mnemonic can be selected.

The following list explains guidelines for mnemonics:

- Use the first letter of the choice when possible; otherwise, assign the first available relevant, meaningful, and unique letter in the text.
- Use the same mnemonic for a choice throughout the application.
- Make each mnemonic unique.
- In a generated application, mnemonic selection is not valid if the cursor is positioned in an entry field or a list box. The mnemonic will work with the ALT key regardless of the accelerator setting.

Status Bars

The status bar is an optional window element that displays information about the current state of the application. Status bars are not used on dialog boxes.

The status bar can be used to provide the following information:

- Display static data, such as accounting period during journal entry posting or variable data related to the current transaction.

- Provide help by showing a brief explanation of the field to which the cursor currently points.

The status bar can contain single-line entry fields that are not modifiable by the user.

The fields within a status bar can be mapped to either an export view or a special field.

You place fields on the status bar just as any other fields are placed on windows or dialog boxes. They are left-justified on the bar. To achieve equidistant spacing, you can insert blank work attribute fields.

Tool Bars

A tool bar is an optional window element that provides quick and convenient access to frequently used choices and commands through push buttons. The tool bar is also used to invoke sub-applications within the application.

In CA Gen, tool bars are defined for windows only and not for dialog boxes.

The use of a tool bar is optional. Deciding whether to use a tool bar on a window should be based on how much space there is available and how the tool bar affects the overall appearance of a window's design.

The following list explains guidelines for tool bars:

- Use a toolbar for all frequently used operations to reduce the number of mouse moves and clicks the user must complete.
- Use micro help for all bitmaps on the toolbar.
- Always place tool bar near the top of the window.
- Use the tool bar in conjunction with a menu bar.
- The actions represented on the toolbar should also appear as menu items on the menu bar.
- Size the tool bar based on its contents.

More information:

Bitmaps (see page 135)

Entry Fields

An entry field is a control into which the user types information on windows and dialog boxes.

Note: A hidden field defines an import view of data to an export view of data without the data being placed on the window.

The following list explains guidelines for fields:

- Use margin boxes around all entry fields.
- Place the most important fields first (top left) and the least often accessed last (bottom right).
- Place logically related fields into a group box (a margin box is drawn around the collection of fields) and label the group box. For more information, see Group Boxes.

  For example, if a customer address is in four separate fields, these fields can be assembled into a group box labeled Customer Address.
- Follow the guidelines for field prompts.
- Display-only fields:
  - Change the color to the same as the background of the window and do not use a margin box around the field. This helps to avoid confusion between the prompt text and the field text.
  - Use a vertical separation between fields of 10 pixel elements. The maximum separation should be no more than 15 pixel elements.
  - Left-align text fields (have values start in the same position from the left).
  - Use decimal alignment for decimal fields.
- Identify mandatory entry fields using these techniques:
  - Disabling or enabling options
  - This can be confusing to the user if there are too many and they cannot easily determine how to enable an object.
  - Use bold text on prompts.
  - Be aware that this will take up more space, and mixing fonts can really slow down the user and make the window less appealing.
- Provide assistance within the application.
- Place separate group boxes around mandatory fields.
- If an attribute has an exhaustive, finite list of permitted values, use radio buttons, a check box, or a drop-down list box rather than an entry field. This gives guidance on the permitted values for the field and avoids the entering of incorrect values.

More information:

Field Prompts (see page 144)

Group Boxes

A group box is a margin box that is drawn around a collection of logically related fields.

The following list explains when to use a group box:

- Use a group box around similar information to focus the user's eye.

  For example, if a customer address is in four separate fields, and there is no need to have a prompt for each part of the information. You can assemble these fields into a group box labeled Customer Address.
- You do not have to name a group box, but it nearly always makes sense to do so.
- In complex windows, a combination of named and unnamed group boxes may be necessary if the space is available.
- An alternative to a group box is grouping related fields using the equivalent of blank lines and indentation.
- Place a group box around radio buttons and check boxes presenting options for the value of a single attribute.
- This focuses the user's eyes on the options for that single attribute.
- A group box is needed if you have more than one set of radio buttons or check boxes on a single window or dialog box.
- Use a group box around related items or a domain of choices.

Field Prompts

A field prompt is a text label that identifies a selection or entry field.

The following list explains guidelines for field prompts:

- Place the prompt near the field labels.
- Use the same prompt for an attribute wherever it appears on a layout.
- CA Gen keeps a list of each prompt used to implement each attribute across the system and allows you to choose among them.
- Capitalize the first letter of the prompt; make all other letters (except for proper nouns) lowercase.

- CA Gen creates default prompts in all capitals. We recommend that you delete these prompts and enter prompts with initial capitals instead.
- Right-justify all prompts.
- Keep field prompts short.
- Assign a mnemonic to each selection field to allow the user to use the keyboard instead of the mouse.

More information:

Mnemonics (see page 141)

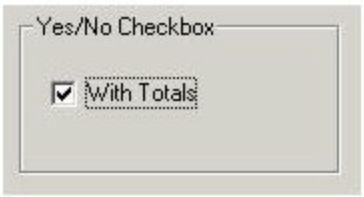
Check Boxes

A check box is a control that acts like a switch (on/off, yes/no, true/false, and so on) for selecting choices. It consists of a box that displays an X when the user selects it. (In some windows environments, the check box is marked with a _ instead of an X).

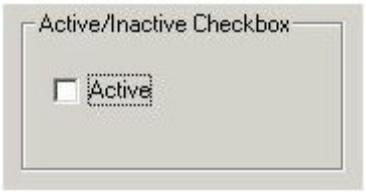
Each check box is independent of all other check boxes.

Using check boxes saves space on windows and dialog boxes.

For a yes/no example, you will have a single check box that is labeled *With Totals*. If the box is marked (selected by the user), the value is yes. If the box is blank, the value is no. The following illustration shows a check box that is marked for true.



Another example is a check box for an active or inactive customer. The check box is labeled *Active*. If the box is marked (selected by the user), the customer is active. If the box is blank, the customer is inactive, as is the case in the following illustration.



A check box is associated with an attribute. The true (yes) and false (no) value for the box can be any valid value for the attribute's domain and size selected from its permitted values.

You can group check boxes if you want multiple options to be set on or off. For example, the persons covered by a policy can be selected by a series of check boxes, as shown in the following illustration.

The following list explains guidelines for check boxes:

- Always use a single check box for independent true/false, on/off, yes/no information. You will recognize this information by attributes such as a flag and indicator.
- Group check boxes if you want multiple options to be toggled on or off. Limit the number of check boxes to 10-12 per related item.
- Align the check boxes vertically.
- Assign a mnemonic to each field prompt to allow the user to use the keyboard instead of the mouse (see the Assigning Mnemonics section in this chapter).
- Follow the guidelines for field prompts.

More information:

Field Prompts (see page 144)

List Boxes

A list box is a control that enables the display of a series of attribute values as a list. The user may select from the list and, optionally, enter new values in the list.

List boxes are used on windows and dialog boxes.

Note: A CA Gen-generated application automatically handles vertical and horizontal scrolling on list boxes.

You can specify the following for list boxes:

- Import and export views, which must be group views.
- Whether a user can select more than one entry at a time.
- Attributes and field prompts.
- Attribute that may be used as a selection indicator, which is flagged so that procedure logic can detect the entry in a list that has been highlighted by the user.

List Box Type	Possible Attribute Views	When to Use
Non-enterable list box (the default)	Several	The user will always need to see the list. There are multiple columns in the list. Multiple selections are desirable.
Non-enterable drop-down list	Only one	Space is limited. A list is rarely viewed (the default value displayed is usually accepted). A single choice is selected. While drop down lists can save space, too many of them on the same window can become hard for the user to navigate. **Note:** If there are fewer than 6 possible values, consider using radio buttons instead of a list box (see the Designing Radio Buttons section in this chapter).
Enterable list box	Only one	Data entry and list selection are allowed. Display as a drop-down or already dropped down if space is available.
Enterable drop-down	Only one	Data entry only is allowed.

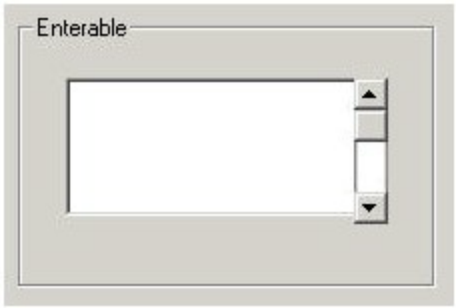
More information:

About Scroll Bars (see page 137)

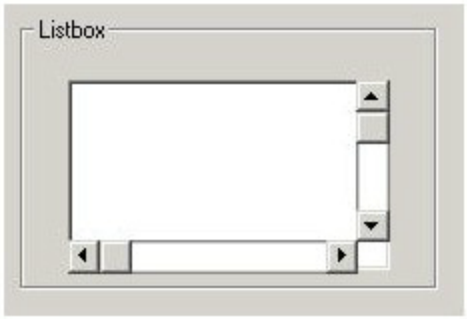
Types of List Boxes

The following illustrations show the types of list boxes.

- Enterable



- Listbox



- Enterable Drop-down

- Non-enterable Drop-down

The following list explains guidelines for list boxes:

- Make the list box as wide as necessary to limit horizontal scrolling
- Determine the default number of lines to display based on the precedence of the list box relative to other window controls, its specific use, the available space, and the user.
- The average position can see five to seven items without the user moving their eyes.
- If the list box is the main control in the dialog, up to 20 lines can be displayed initially, depending on the space available.
- Provide search or filtering capability if the user would need to constantly scroll through a long list (more than four or five page downs).
- If the list box contains many fields, improve scanning ease by placing a blank line after every 5th or 6th line.
- Follow the guidelines for field prompts.

More information:

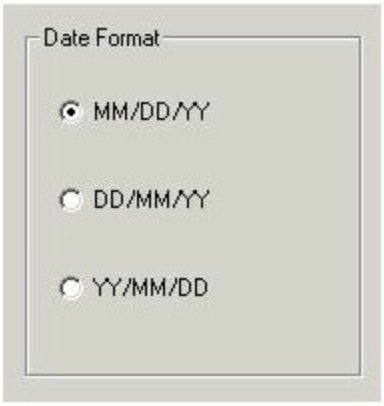
Designing the Procedure Logic (see page 163)
Field Prompts (see page 144)

Radio Buttons

A radio button is a control on a window or dialog box that offers the user one choice from a group of choices. It consists of a circle that is marked with a filled inner circle when the user selects it.

Radio buttons map to the permitted values of one attribute view. Only one button can be active at a time.

For example, the following illustration shows three choices for the date format. The user can select only one of the three. The format MM/DD/YY is selected.



The following list explains guidelines for radio buttons:

- Use radio buttons when there is a short list (up to about six) of possible values in a mandatory field.
- To avoid visual confusion, restrict the total number of radio buttons to six or less.
- You should have a minimum of two radio buttons.
- Align one to two radio buttons horizontally; align longer lists vertically.
- Follow the guidelines for field prompts.
- Use a group box around related radio buttons.
- Assign a mnemonic to the field prompt for each radio button to allow the use of the keyboard instead of the mouse.

More information:

Field Prompts (see page 144)
Mnemonics (see page 141)

Push Buttons

A push button is a control that performs an action immediately when it is selected. It is a rounded-corner rectangle with text or a bitmap inside.

Push buttons are used on dialog boxes, message boxes, and secondary windows that have no menu bars. Push buttons can also be used on a tool bar. For more information, see Tool Bars.

The following illustration shows the use of push buttons on a dialog box and a tool bar.

Selecting a push button results in one of the following actions for the user of the CA Gen application:

- Initiates a flow due to a command set that is an autoflow.
- Opens a supplemental dialog box to this procedure.
- Executes the procedure from the top.
- Executes an event.
- Executes a special action.

Examples of push buttons for dialog boxes are listed next:

- Apply-Causes the application to accept user changes in dialogs that set properties.
- Cancel-Closes the dialog box without performing uncommitted user changes.
- Exit-Closes the dialog box after the user has committed changes.
- Help-Displays online help.
- OK-Causes the application to accept any changed information.
- Reset-Cancels any uncommitted user changes and resets any changed fields to their initial values.

The following list discusses uses of push buttons:

- Exiting the dialog (OK, Select, Cancel, Exit)
- Invoking other functions or features (Options, Details, Help)
- Domain specific (does not apply to the whole dialog, for example, a prompt for a field; may invoke a list of values for that field)

The first two uses of push buttons are shortcuts for menu actions.

The third use of push button (domain specific) may not be provided as a menu item. This type of push button may be placed near the control to which it applies.

The following list discusses guidelines for push buttons:

- Provide a Cancel push button as required:
  - Provide a Cancel (or equivalent) push button if the OK (or equivalent) button causes processing to take place.
  - Do not provide a Cancel push button if the OK button is used to indicate that the reader has finished reading a purely informative message.
- Avoid yes/no and true/false text answers, as this will slow the user down by making them read the question.

By using a one-word description of the action to be performed, the user can make an immediate decision.

- If a push button invokes a secondary dialog, add ellipsis (...) to the text on the button to indicate that another dialog will appear.
- Position the push buttons with 10 pixel elements space between them and with the OK button in the bottom left corner of the dialog box.
- If a push button is not global to the dialog, place it near or within the group of related controls.
- Recommended placement for push buttons:
  - If push buttons are aligned horizontally across the bottom, place affirmative actions to the left and non-affirmative to the right.
  - If push buttons are aligned vertically down the right, place the affirmative actions above the non-affirmative actions.
  - If feature and exiting buttons are all aligned either horizontally or vertically, separate the two uses by white space (a blank equivalent to the size of a button).
- Identify a default action by making the border of its push button bold.

- The default push button should be that which may be expected of the user to choose most often, when it will not delay the user or affect the state of the data if chosen accidentally. Most often this would be the affirmative action (why the user came here in the first place). The exceptions are actions such as delete, purge, archive, and sometimes print if it can cause a bottleneck. In these cases, the non-affirmative action should be the default as this is a form of confirmation by forcing the user to deliberately select the non-default button (as opposed to selecting the enter key without thinking of the result and accidentally invoking the wrong action).
- Push buttons automatically provide visual feedback when selected. They can be distinguished if desired by applying a bitmap image.
- If a push button is covered by a bitmap, its associated text cannot normally be seen. CA Gen enables the text associated with a push button bitmap to be displayed below the bitmap when the cursor is positioned over it for more than a few seconds. This is referred to as the micro help technique.
- Assign a mnemonic to each push button to let the user use the keyboard instead of the mouse.

More information:

Designing the Procedure Interaction (see page 89)
Designing the Procedure Logic (see page 163)
Special Actions for Menu Items and Push Buttons (see page 154)
Bitmaps (see page 135)
Visual Cues (see page 136)
Mnemonics (see page 141)

Radio Buttons, Check Boxes, and List Boxes

Deciding whether to use a radio button, a check box, or a drop-down list box depends on a number of items, such as listed next:

- How much window space is available
- Whether the user will be primarily using the keyboard or some point-and-click device for data entry
- How frequently the item is accessed

Situation	Radio Button	Check Box	List Box
2-6 static choices	X	X	
7-12 static choices		X	
13 or more dynamic choices			X

Situation	Radio Button	Check Box	List Box
Yes/No, True/False, On/Off		X	
Many choices, limited space on window			X

Special Actions for Menu Items and Push Buttons

Special actions can be applied to either menu items or push buttons.

The names of the special actions do not reflect the name of the specific menu item or push button for which they are assigned.

The following list shows the categories of special actions:

- Execution control
- Help system
- Window control
- Edit
- OLE area control

Execution Control Actions

The special action options are:

- OK—Use this action to execute the users option on the window or dialog box and then close the window or dialog box.

  You will typically use this special action on a push button, but it can also be assigned to a menu item.

- Cancel—Use this action to simply cancel the user option on the window or dialog box and close the window.

  You will typically use this special action on a push button, but it can also be assigned to a menu item.

An example of the use of this special action is for a dialog box asking the user "Are you sure you want to delete this object?" (You will want to give the user the option to cancel the pending delete.)

Help System

You will typically place *help* on the menu bar. This gives you the option to format the help menu item to your own GUI standards.

The following special actions execute as part of the help system:

- Help
- Help for Help
- Extended Help
- Keys Help
- Help Index

Special Actions for Window Control

The following describes the options for controlling windows:

- Tile—Select a menu item that will tile the open windows in the CA Gen application.
- Cascade—Select a menu item that will cascade the open windows in the CA Gen application.

Edit

The following describes the special actions for user editing.

Cut

Copies the object in selected OLE area or selected data to the clipboard and remove it.

This option is disabled if there is no object in the OLE area or no data is selected.

Copy

Copy the object in the OLE area or selected data to the clipboard.

This option is disabled if there is no object in the OLE area or no data is selected.

Paste

Copy the object from the clipboard into the selected OLE area or copy the data from the clipboard into the selected area.

This option is disabled if the selected OLE area already contains an object, if there is no object or data in the clipboard, or the selected object in the clipboard is not compatible with the OLE area.

Clear

Remove the object or data from the selected area.

This option is disabled if there is no data or object in the selected area.

Undo

Reverse current actions.

OLE Area Control

Each special action that is to be available to the user during runtime must be associated with a menu item or push button. Since the menu items can be merged with the OLE object menu items (see Set Merge for OLE Menu Items), you will nearly always use these as menu items.

The special action options for the OLE Area are described in the following list.

Save

Saves the OLE object under the same name as it was inserted.
This option is only necessary if you have allowed the user to update the OLE object.

Save As

Saves the OLE object under a different name as it was inserted.
This option is only necessary if you have allowed the user to update the OLE object.

Load

Loads the OLE object form a file.

Insert

Inserts the OLE object while executing the application.
You must be sure that this special action is assigned to a menu item if you are not inserting the object at design time.

Paste Special

Selects one of multiple formats compatible with the OLE area that is to be pasted.

Object

Allows the user to see the verbs supplied by the OLE object.
The user will only have sub-menu items associated with the menu item that this special action is associated with when there is an object in the OLE field.

Return

Returns from the merged window back to only the CA Gen application window.

You will need to have this special action assigned to a menu item. To allow the user to return, that menu item must be merged into the menu bar when the OLE object is embedded (see the Setting Merge for OLE Menu Items section in this chapter).

More information:

Set Merge for OLE Menu Items (see page 158)

Embed OLE Areas in Windows and Dialog Boxes

Note: This information pertains to Microsoft Win32 only.

You can embed an OLE (object linking and embedding) area on a CA Gen window or dialog box to allow communication from the CA Gen application to another desktop application. The desktop application is embedded in the CA Gen application.

The CA Gen application acts as an OLE automation controller. Presentation of the desktop application data appears within the CA Gen window or dialog box.

The following list shows guidelines for OLE areas:

- A single window can contain multiple OLE areas.
- The number of areas is limited by space. More than four will start to become difficult to read without a rather large monitor.
- Only one OLE area can be active at any one time.
- When you design the OLE area you, keep in mind the size of the area needed by the desktop application. For example, if the OLE area is being designated to run a word processor, you will want to place the area with the maximum width and as much length as possible for reading the material.
- Which main window menu bar items you would like to have merged with the OLE application. For more information, see Set Merge for OLE Menu Items.
- The tool bar will be replaced by the OLE objects tool bar.
- When you bring the OLE object into place, there are no scroll bars for the area, unless the object brought the scroll bars with it.
- You can insert to OLE object for the user during design time, or allow the user to insert the object during application runtime (see Special Actions for Menu Items and Push Buttons). You can make the OLE object either read only or update.
- If you insert the OLE object during design, make the object read only. The reason for this is that each time the application is executed the object will start at its designed state. Allowing the user to update the object will give the false impression that work on that object is being accomplished.
- If you allow the user to insert the object (which will leave the OLE area unfilled during design), and the user needs the capability to update the object, allow the object to be updated.

More information:

Designing the Procedure Logic (see page 163)

Set Merge for OLE Menu Items

Note: This information pertains to Microsoft Win32 only.

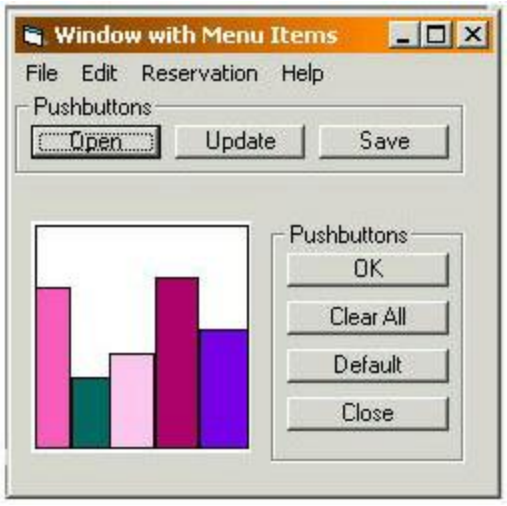
You can place a menu item in one of three menu item groups. The menu groups are named with some OLE naming conventions, but the main difference among the three groups is where the menu items will be placed in the merged menu bar.

You also have the option not to merge menu items.

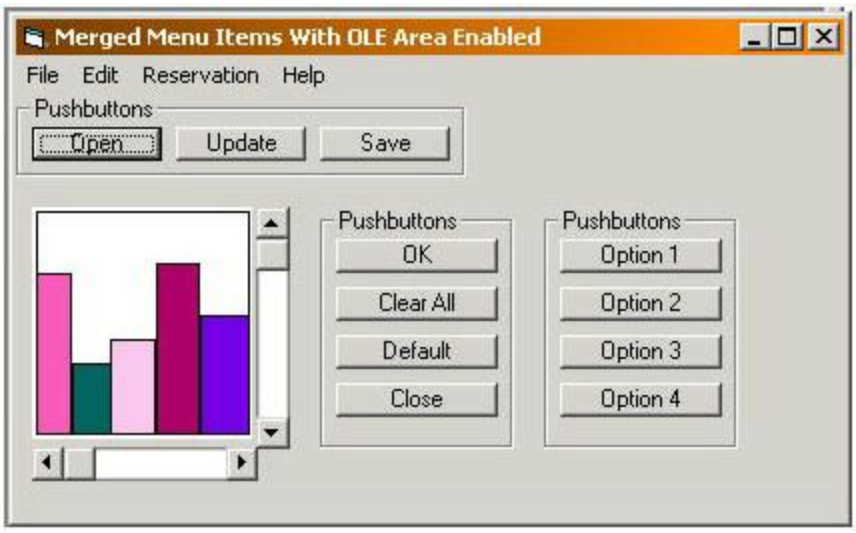
Option	When to Use
Do not merge	Not to be merged.
Merge as a member of the file group	There are file manipulation characteristics to be placed to the left of the merged menu bar.
Merge as a member of the container group	There are items to be placed toward the center of the merged menu bar.
Merge as a member of the window group	There are window manipulations to be placed toward the right of the merged menu bar.

Note: When you assign a merge to the top-level menu item, it carries with it all the sub-menu items.

The following illustration shows a CA Gen window at design time. The window has an OLE area with an embedded Excel spreadsheet that is inactive.



The following illustration shows a CA Gen window at design time. The window has an OLE area with an embedded Excel spreadsheet that has been enabled.



The illustration shows how menu items are merged when the embedded spreadsheet is active.

- The window tool bar has been replaced with the tool bar of the embedded spreadsheet.
- The menu items in the illustration were designed to be merged as follows:
  - File-Merge as a member of the file group
  - Edit-Merge as a member of the container group
  - Reservation-Do not merge
- After the menu items have been merged, there are two file and two edit menu items. The file and edit menu items from the CA Gen window were merged, as designed. The reservation menu item was not merged, as designed.

If you double-click on an embedded desktop application, the default verb for that application is executed. For example, the default verb for Microsoft Word is Edit. In a CA Gen window, when you double-click on an OLE area containing an embedded Word document, the document is presented in edit mode. The CA Gen window tool bar is replaced with Microsoft Word's tool bar, and the menu items are merged as you specified in the design.

More information:

Designing the Procedure Logic (see page 163)

Embed OLE Custom Controls

Note: This information pertains to Microsoft Win32 only.

You can embed an OLE custom control (an OCX) on a window or dialog box that is controlled by a CA Gen application.

Examples of OCXs are spin buttons, *gas* gauges, and other graphic gadgets. These controls let you give the user an indication of the time it will take to accomplish a specific task.

The following list detail considerations for OCX controls on windows or dialog boxes:

- The OLE object does its own resizing.
- Each OCX provides its own sets of events.
- In addition to the CA Gen properties, each OCX contains properties.
- The number of properties is not limited.
- Information about OCX properties should be delivered with the OCX itself.

More information:

Designing the Procedure Logic (see page 163)
Designing the Procedure Interaction (see page 89)

Results of GUI Design

The results of designing the GUI are:

- Designed windows
- Designed dialog boxes

Chapter 7: Designing the Procedure Logic

This chapter describes how to fully detail the procedure components of the outline system structure. Procedure logic design involves taking the outline system structure and fully detailing its procedure components.

The GUI, procedure interaction, and procedure logic are best designed in parallel. Procedure interaction and layouts may then be refined (often by prototyping). When the initial design is stable, detailed procedure logic can be completed, and the flows and layouts can be finalized.

Together, the procedure interaction, layout design, and logic yield a completed navigation diagram, fully detailed displays, and action diagrams that are the basis for generating a completed system.

Prerequisites

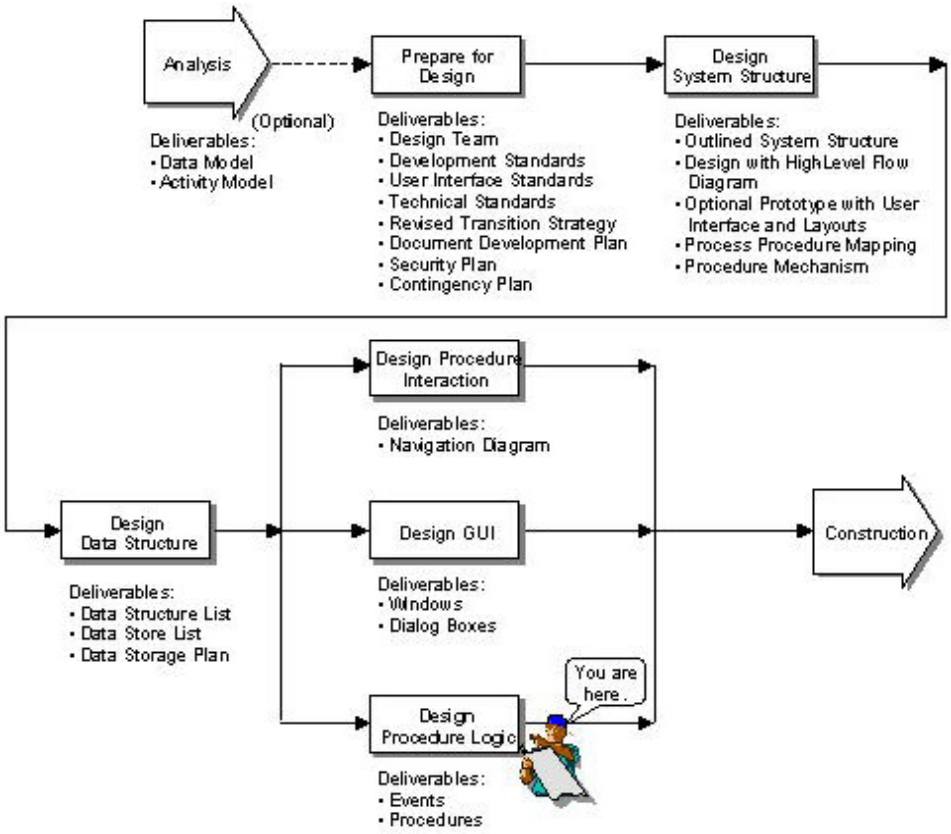
You need an outlined system structure design before beginning procedure logic design.

More information:

Designing the System Structure (see page 33)

Design Process

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



Client Procedure Structure

A client Procedure Action Diagram can be viewed in three parts:

- Information views—These views support all logic in the client procedure and included event handlers.
- Procedure main logic—This logic includes return status from flows to server procedures, execution first logic on flows from other procedures.
- Event handler logic—These are the handlers for each of the GUI events.

CA Gen automatically adds event handlers to the bottom of the procedure action diagram. The handlers can then be moved, but you should use a consistent approach to structure their action diagrams.

For client procedures, most logic can be placed in event handlers. Since they are focused groupings of logic, understanding a Procedure Action Diagram is simplified.

Keeping together window or dialog boxes that deal with related data minimizes the number of information views needed. This improves ease of maintenance by reducing the number of changes that have to be made to views. This approach also minimizes the number of flows, attendant view matching, and procedure logic. All of this leads to increased productivity and a reduction in the impact of change.

Client Procedure Standards

Completion and maintenance of action diagrams are easier if they follow a predictable pattern.

Guidelines for setting standards are categorized as listed next:

- Client procedure with flows (link) to server procedures
- Client procedure with remote use to server procedures

Standards for Client Procedure with Flows (Links)

In client procedures, event handlers are placed at the end of the action diagram but are executed independently of the main processing.

The following list shows standards for a procedure with flows to server procedures:

- View definitions:
  - Import views
  - Export views
  - Local views
- Initial processing:
  - Move import view contents to export views
  - Set initial values
  - Set a default exit state
  - Test for previously stored exit state and command to control subsequent processing

- Processing prior to any server:
  - Some application processing
  - Set and preserve values and commands for server procedures, and initializing links to servers (only one link is activated in any one execution of the procedure)
  - Move returned data to export views
  - Prepare for display of windows, dialog boxes, and fields
  - Initialize opening and closing dialog boxes
  - Set exit states
  - Exit
- Processing after server procedure “A”:
  - Test return code values for success or failure
  - Some application processing
  - Move returned data to export views
  - Prepare for display of windows, dialog boxes and fields
  - Initialize opening and closing dialog boxes
  - Set exit states
  - Exit
- Processing after server procedure “B”:
  - Test return code values for success or failure
  - Some application processing
  - Move returned data to export views
  - Prepare for display of windows, dialog boxes, and fields
  - Initialize opening and closing dialog boxes
  - Set exit states
  - Exit
- Event handlers only reached by a GUI event:
  - Respond to changes in windows, dialog boxes and fields and their selection
  - (Optionally) validate changed data
  - Prepare for display of windows, dialog boxes, and fields
  - (Optionally) Set exit states

Standards for Client Procedure with Remote Use

Standards for a procedure with a remote use to server procedures:

- View definitions:
  - Import views
  - Export views
  - Local views
- Initial processing:
  - Move import view contents to export views
  - Set initial values
  - Set a default Exit State
  - Test for previously stored exit state and command to control subsequent processing
- Processing prior to server procedure “A”:
  - Some application processing
  - Move returned data to export views
  - Prepare for display of windows, dialog boxes, and fields
  - Initialize opening and closing dialog boxes
  - Set exit states
  - Use server procedure “A”
- Processing after server procedure “A”:
  - Test return code values for success or failure
  - Some application processing
  - Prepare for display of windows, dialog boxes, and fields
  - Initialize opening and closing dialog boxes
  - Set exit states
  - Exit
- Event handlers only reached by a GUI event:
  - Respond to changes in windows, dialog boxes and fields and their selection.
  - (Optional) validate changed data.
  - Prepare for display of windows, dialog boxes and fields.
  - (Optional) use server procedure “A.”

- (Optional) processing after server procedure “A” (if used server procedure “A”).
- (Optional) set exit states.

Server Procedure Structure

A server Procedure Action Diagram contains actions needed to handle commands sent from the client procedure in order to direct the order of execution and pick up any application errors and send them back to the client.

Server procedures are designated On-line With no Display.

Server procedures do not contain event handlers since events are associated with the graphical user interface.

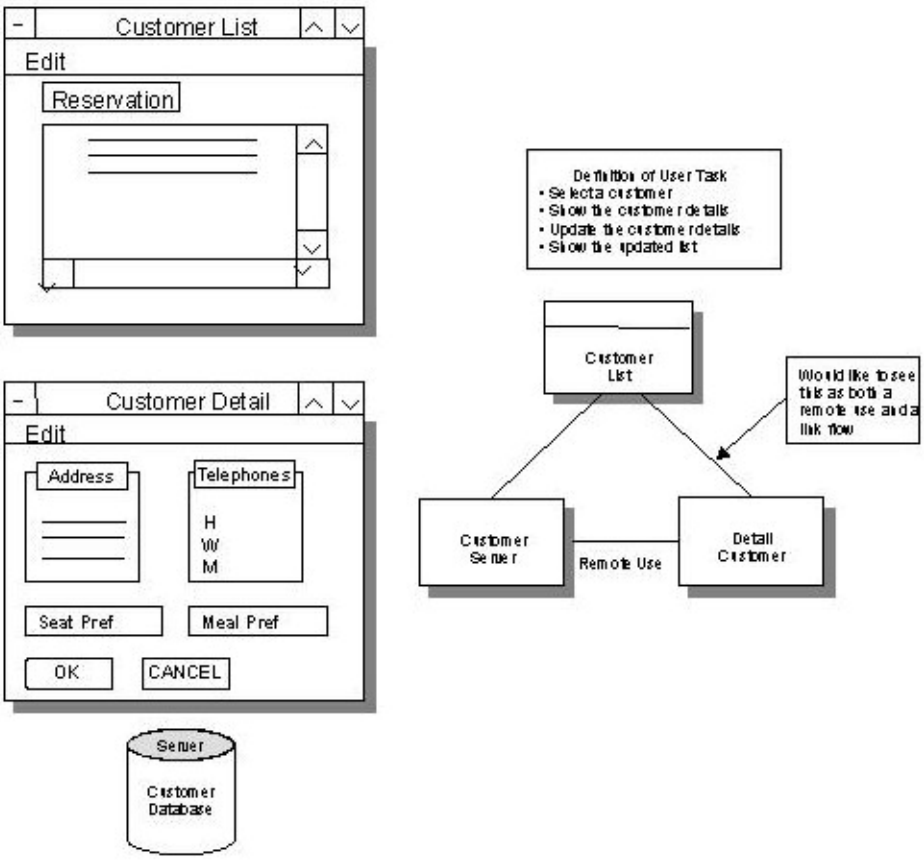
Completion and maintenance of action diagrams are easier if they follow a predictable pattern. The following outlines a possible standard action diagram structure for a server procedure that contains link flows or remote use statements, or both from client procedures. The structure is designed to handle links or remote use statements from more than one client procedure.

- View definitions:
  - Import views
  - Export views
  - Local views
- Initial processing:
  - Move import view contents to export views
  - Set initial values
  - Set a default return code
  - Set a return with command
  - Test for commands to control subsequent processing.
- Processing:
  - Application processing
  - Move data to export views
  - Set return codes
  - (Optionally) set exit states
  - Exit

Control Structure

Control is passed and results received through event handlers, commands, exit states, flows, remote uses, and so on.

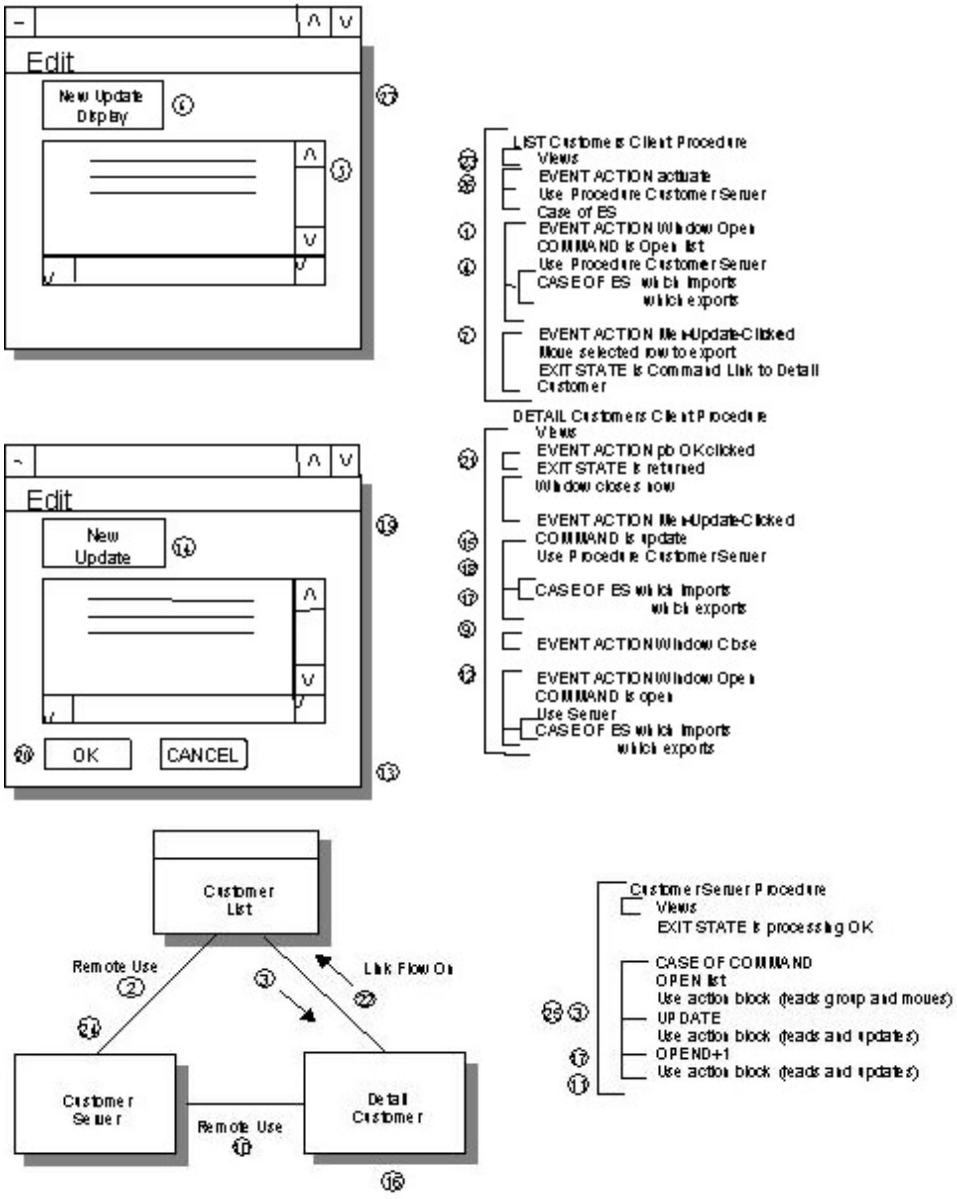
The following illustration provides an example of a task definition.



In this example, a design decision was made to use a separate client procedure and its primary window to present the customer detail. The alternative was to have an additional dialog box in the list customer client procedure. This decision was made to keep the amount of information about each customer returning from the customer server to the list customer window to a minimum since it is not necessary. Only when a new customer is added, an exiting customer is updated, or all data about a single customer need to be displayed will the detailed window be used. This choice significantly changes the logic.

It is very helpful if an organization decides on standards for common tasks such as this example. Standards ensure ease of maintenance during and after the development cycle by ensuring consistency of approach. There are user interface standards.

The following illustration and its key provide an example of the control structure. This example illustrates point by point, how control is passed and results are received through event handlers, commands, exit states, flows, and remote uses.



The steps in the next table lead you through the design shown in the illustration. Specific names of objects on the figure are shown in italics in the table.

Step	Action
1	The starting state for this example is that there has been a Display First flow to the List Customer window. Even though the procedure is Display First, the open window event will execute. The open window event sets the command to openlist and executes the remote use of the customer server procedure. **Note:** You can design this into the event handler such that each time the event is executed the command will be set. That is why you do not see any logic to set the command in the event handler.
2	Control is passed from the customer list client procedure to the customer server procedure via the window and server managers.
3	The customer server procedure processes the request of the customer list client procedure by executing the case of command. Since the open window event in the customer list client procedure sets the command to openlist, an action block is used to read and return the list of customers. **Note:** The detail of the logic of the action block is not included, nor is the view matching from the server procedure. If there were any problems while trying to read the list of customers, an exit state would have been set.
4	Since a remote use statement was used for the procedure interaction, and there is logic after the remote use statement, the control passes back to the OPEN event via the window manager. The window manager first checks to see if the exit state had changed to a value that will cause a flow. If the exit state has not been changed to a value that would cause a flow, it is checked for having a message and the message type is informational, warning, or error. If the message type is any of these three values, a dialog box with the message will be presented to the user. For this example, there were no errors. The case of exit state logic in the OPEN event is then executed. Since there were no errors, the control passes to the user via the window.
5	The user is presented with the window containing a list of customers. The user selects one row (customer) from the list box.
6	The user clicks on either the edit or update menu items or the update from the tool bar. This passes control to the menu update clicked event handler.

Step	Action
7	The menu update clicked event moves the selected row to the export view and sets the exit state to link to detail customer. This causes a flow to the detail customer client procedure.
8	Control is passed to the detail customer client procedure. This client procedure is Display First, flowed to on the link to detail customer exit state, and contains a view match of the selected customer. Since it contains an open window event, this event will be executed first.
9	The open window event sets the command to opendtl and executes the remote use of the customer server procedure. **Note:** The command opendtl was included at design time. There is no need for logic to set the command.
10	Control is passed from the customer detail client procedure to the customer server procedure via the window and server managers.
11	The customer server procedure processes the request of the customer detail client procedure by executing the case of command. Since the open window event in the customer detail client procedure set the command to opendtl, an action block is used to read and return all of the information of a the selected customer. **Note:** The logic detail of the action block is not included, nor is the view matching from the server procedure. If there were any problems while trying to read the list of customers, an exit state would have been set.
12	Since a remote use statement was used for the procedure interaction, and there is logic after the remote use statement, control passes back to the OPEN event via the window manager. The window manager first checks to see if the exit state had changed to a value that will cause a flow. If the exit state has not been changed to a value that would cause a flow, it is checked for having a message and the message type is informational, warning, or error. If the message type is any of these three values then a dialog box with the message will be presented to the user. For this example, there were no errors. The case of exit state logic in the OPEN event is then executed. Since there were no errors, the control is passed to the user via the window.
13	The user is presented with the window containing the detail of the customer previously selected from the list customers. The user modifies some information about the customer.

Step	Action
14	The user clicks on either the edit or update menu items or the update from the toolbar. This passes control to the menu update clicked event handler in the detail customer client procedure.
15	The menu update clicked event automatically moves the import view to the export view (you do not have to put this logic in the event handler) and sets the command to update. The menu update clicked event then executes the remote use statement to use the customer server procedure.
16	Control is passed to the customer server procedure.
17	The customer server procedure processes the request of the customer detail client procedure by executing the case of command. Since the menu update clicked event in the customer detail client procedure set the command to update, an action block is used to read, update and return all of the information of a the selected customer. **Note:** The detail of the logic of the action block is not included, nor is the view matching from the server procedure. If there were any problems while trying to read the list of customers, an exit state would have been set.
18	Since a remote use statement was used for the procedure interaction, and there is logic after the remote use statement, the control passes back to the menu update clicked event via the window manager. The window manager first checks to see if the exit state changed to a value that will cause a flow. If the exit state has not been changed to a value that would cause a flow, it is checked for having a message. The message type can be informational, warning, or error. If the message type is any of these three values, a dialog box with the message is presented to the user. **Note:** For this example, there were no errors. The case of exit state logic in the OPEN event is then executed. Since there were no errors, control is passed to the user via the window.
19	The user is presented with the window containing the detail of the customer previously updated.
20	The user clicks the OK push button. This passes control to the pb OK clicked event handler in the detail customer client procedure. This assumes that all of the information the user changed is presented properly.

Step	Action
21	The pb OK clicked event automatically moves the import view to the export view (you do not have to put this logic in the event handler). The pb OK clicked event then sets the exit state to return. Since this is the end of the pb OK clicked event, the window manager checks the status of the exit state. Since the value of return causes a flow to the list customer client procedure, control is passed to list customer client procedure.
22	Control is passed to the list customer client procedure. This client procedure is Display First, flowed to on the return exit state, and contains a view match of the selected customer. Since this procedure contains an open window event, this event is executed first.

More information:

Designing the Graphical User Interface (see page 117)

GUI Event Processing

GUI events (such as click on push button or alter field contents) are executed through individual blocks of logic called event handlers.

You can attach an event handler to a window or field (control).

User-defined events can supplement or replace standard events that are supported by the CA Gen window manager or by the window environment. This allows the user to have more control over the execution of GUI application procedures.

Note: There are other event types of interest to developers, especially events that are relevant to the business, such as *year end arrives*. However, in this discussion, the term event means GUI event.

Characteristics of Event Handlers

Event handlers have the following characteristics:

- Events are owned by the procedure and therefore have a name that is unique within the procedure.
- An event handler shares views with the procedure. All event handlers belonging to a procedure have access to its views. Therefore, there is no view matching between a procedure and an event handler.

- A single event handler may be invoked by multiple controls. They are shareable within a procedure.
- All types of action diagramming actions may be specified in an event handler.

The CA Gen generated window manager detects events. It calls the appropriate event handler. When the event handler completes execution, it returns to the window manager without having entered the procedure.

When the user or the system triggers an event, its code is immediately executed. An event handler's execution is similar but not exactly like a procedure's execution.

After an event handler executes, the following actions occur:

- Local and entity action views are cleared
- Commits take place
- Exit states are interrogated
- Export views are displayed

The default for the movement of the views is that the import views are automatically moved to export views, except for the *changed* event, when views remain as they were before the event logic was executed. You can change this default.

There is a reason for the exception made for the changed event. Based on certain conditions, the prior value in the field should be displayed instead of what was entered. For example, a default value may be displayed in an entry field. If the user makes an incorrect entry, the default value should remain displayed, not the entry that was made. This means that most action statements in event handlers should reference the export view. (However, that procedure logic must still move import views to export views.)

Further events (such as Open or Close) that may be triggered by an event are placed in a queue and executed in turn after the logic in the initial event handler has completed.

Standards for Naming GUI Events

You should establish standards for naming event handlers.

CA Gen will produce a default name for events such as:

```
window/dialog box name_[control type]-[object name]_event
```

The convention used in this example is as listed next:

- Control type is push button (pb), entry field (ef), and so on.
- Object name is the name of the push button, entry field, and so on.

Both of these may be absent depending on the event, for example, Open, Close.

Some event name examples are shown next:

```
COLLECTIONS_OPEN
COLLECTIONS_MN_LIST_CLICK
ACCOUNTS_EF_JOURNAL_NUMBER_CHANGED
```

Using this or a similar convention, you can group the event handlers by window or dialog box. The window or dialog box name becomes very important and should follow the object action format to keep the procedure action diagram well organized and understandable.

Windows and Window Controls That Signal Events

The following table defines the events signaled by windows and controls.

Window/Control	Signals These Events
Window or Dialog box	OPEN
	CLOSE
	ACTIVATE
	DEACTIVATE
Single-line entry field	CHANGED
	GAIN FOCUS
	LOSE FOCUS
Multiline entry field	CHANGED
	GAIN FOCUS
	LOSE FOCUS
Check box	CHANGED
	GAIN FOCUS
	LOSE FOCUS
Radio button group	CHANGED
	GAIN FOCUS
	LOSE FOCUS
Combination box group	CHANGED
	GAIN FOCUS

Window/Control	Signals These Events
	LOSE FOCUS
Listbox	CLICK
	DOUBLE-CLICK
	SCROLL BOTTOM
	SCROLL TOP
	GAIN FOCUS
	LOSE FOCUS
Listbox field	CLICK
	DOUBLE-CLICK
	GAIN FOCUS
	LOSE FOCUS
Bitmap or icon	CLICK
	DOUBLE-CLICK
Drop-down list, enterable	CHANGED
	GAIN FOCUS
	LOSE FOCUS
Drop-down list, non-enterable	CHANGED
	GAIN FOCUS
	LOSE FOCUS
Non-drop-down list, enterable	CHANGED
	GAIN FOCUS
	LOSE FOCUS
Push button	CLICK
Menu item (lowest level)	CLICK
OLE embedded area	GAIN FOCUS
	LOSE FOCUS
OLE control	GAIN FOCUS
	LOSE FOCUS

OPEN GUI Event

An OPEN event is triggered just before a window or dialog box is to be displayed. It can be used to perform any window or dialog box initialization before the window or dialog box is displayed.

OPEN can be triggered for the following actions:

- At initial module start up
- As the result of a flow
- Execution of an explicit OPEN statement as part of the procedure logic
- Selecting a push button or menu item with the result of initializing a dialog box

For example, to open a dialog box containing a list of departments, the procedure logic can include an OPEN statement.

An OPEN event handler contains statements to populate the list as shown in the next sample code. The list box for departments is then populated just before the Department dialog box is opened.

Initialization Accompanying an OPEN Event

```
114-    --- EVENT ACTION department_open
115-   |    SET SUBSCRIPT OF export_department TO 0
116-   |   --- READ EACH department
117-   |  |--- SORTED BY ASCENDING department_number
118-   |  |   --- IF SUBSCRIPT OF export_department IS LESS THAN MAX OF
119-   |  |  |                export_department
120-   |  |  |    SET SUBSCRIPT OF export_department TO SUBSCRIPT OF
121-   |  |  |                export_department = 1
122-   |  |  |    MOVE department TO export_line_department
123-   |  |  |--- ELSE
124-   |  |   --- ESCAPE
125-   |  |
126-   |   ---
127-    ---
```

CLOSE GUI Event

The CLOSE event is triggered when a window is closed. This even can be used to perform any window or dialog box clean-up required by the procedure.

CLOSE can be triggered for the following actions:

- As the result of a return or transfer flow
- Selecting a push button or menu item with a special action of OK
- Selecting a push button or menu item with a special action of Cancel

- Execution of an explicit CLOSE statement as part of the procedure logic
- Selecting Close from the System Menu

In the next sample code, the subscript of an export view is set to zero.

Clean Up When a Window Is Closed

```
211-  --- EVENT ACTION sort_close
212-  --- SET LAST OF export_sort_results TO 0
213-
```

ACTIVATE GUI Event

The ACTIVATE event is triggered whenever a window or dialog box changes from inactive or non-display to active. This occurs when a window is displayed initially, or when the user brings a window into the foreground.

If a window or dialog box is active and is clicked on again, the activated event is not triggered again.

When a window or dialog box is opened, the OPEN event is triggered before the ACTIVATE event.

DEACTIVATE GUI Event

The DEACTIVATE event is triggered whenever a window or dialog box is deactivated or closed. This may occur due to activating another application window, closing a window, or dismissing a dialog box.

When a window or dialog box is closed, the DEACTIVATE event is triggered before the CLOSE event.

CHANGED GUI Event

The CHANGED event is triggered when a user's actions cause data in the import view to change.

The CHANGED event is triggered for the following actions:

- Tabbing off an entry field after editing it
- Clicking on an unselected radio button or check box
- Selecting a different item from a drop-down list
- Clicking on an unselected row in a list box

The event handler runs after the import view has been updated with new data.

The CHANGED event is provided to allow actions such as validation of fields as they are entered and population of other fields as a result of changing a field. The user expects instant (or, nearly instant) response time when they tab. Processing in changed events should meet this expectation.

Before any logic is coded in-line in the event handler, you should ask, "Can this logic be reused?" If the answer is yes, then you should construct an action block to contain the logic.

The next sample code shows validating a department if the key is changed in a dialog box.

Read a Record if the Key is Changed

```
394-   --- EVENT ACTION department_number_changed
395-  |    MOVE input_department TO output_department
396-  |   --- READ department
397-  |  |      WHERE DESIRED department_number IS EQUAL TO input_department_
398-  |  |            number
399-  |  |--- WHEN successful
400-  |  |    EXIT STATE IS dept_already_set
401-  |  |--- WHEN not_found
402-  |  ---
403-   ---
```

GAIN FOCUS GUI Event

The GAIN FOCUS event is triggered whenever a control receives the input focus. This typically occurs when the user clicks or tabs onto a control, although it may be triggered for other reasons, such as window initialization and autotab.

If a control already has the input focus and is clicked on again, the GAIN FOCUS event is not triggered again.

LOSE FOCUS GUI Event

The LOSE FOCUS event is triggered whenever a control loses the input focus.

If the input focus is being transferred from one control to another, the LOSE FOCUS event for the control being left is queued before the GAIN FOCUS event of the control being entered.

The priorities of the GAIN or LOSE FOCUS events relate to existing events in the following way.

```
Changed > Lose Focus > Gain Focus > Click
```

CLICK and DOUBLE-CLICK GUI Events

These events are triggered when the user clicks the left mouse button on a standard list box, list box field (one column of a standard list box), bitmap, or icon.

The click continues to have the display effect it normally does. It causes a row of a list box to be selected or unselected.

The event handler runs after the click has been processed by the control. The event handler may use one of the following statements:

```
Get Row Clicked
Get Row Highlighted
```

Some applications use a single CLICK event to invoke processing on a row in a list and others use the DOUBLE-CLICK event.

For Example:

- Given a list of customers, the user clicks once on a row, and the selected customer's details are displayed in the detail area of the same window.
- To invoke a dialog box containing customer details, the user double-clicks on a row in the list.

In these cases, only one event is associated with clicking on a row in the list.

If both a click and double-click are associated with the same list, then both events are performed. This can be counter-intuitive and confusing.

You should come up with a consistent approach throughout the application on what a single click will do and what a double-click will do. The best approach would be to use only one event throughout the application. If this is not possible, then at least avoid using both types of events for the same list if it will produce counter-intuitive results.

SCROLL TOP and SCROLL BOTTOM GUI Events

A search is performed on a list window to limit the number of rows returned from the database, however sometimes the number of rows returned exceeds the cardinality of the group view.

The SCROLL TOP and SCROLL BOTTOM events allow processing to retrieve the next group of data in either direction.

When the SCROLL TOP EVENT is triggered, the following actions occur:

- Scroll indicator is moved to the top of the scroll bar.
- Up arrow key is pressed.
- First row in the group view is displayed.

When the SCROLL BOTTOM event is triggered, the following actions occur:

- Scroll indicator is moved bottom of the scroll bar.
- Down arrow key is pressed.
- Last row in the group view is displayed.

For example, the user requests all the customers with addresses in California be sorted by last name. There are 15,042 customers in California.

The group view containing the customers has a cardinality of 500.

After the user scrolls through the first 500 customers, a SCROLL BOTTOM event is triggered to retrieve the next 500 customers in last name order.

If the user wants to scroll back up to the first 500 customers, they move the scroll indicator to the top of the scroll bar. The SCROLL TOP event is triggered, and the previous 500 customers are retrieved in last name order.

Note: If new customers were added in the first 500 group, the first record is not displayed in the previous group of 500 customers.

The user is not able to look at the last window of data by moving the scroll indicator to the bottom of the scroll bar since this action triggers the SCROLL BOTTOM event (the same is true for the SCROLL TOP event). Even using the up and down arrow keys, the user may trigger a SCROLL TOP or SCROLL BOTTOM event unintentionally.

There are three scrolling alternatives:

- Retrieve only half of the number of records that are allowed in a list. This is the recommended alternative.

  For example, if the group view has a cardinality of 500, have the scroll bottom event retrieve the next 250 and append them to the previous 250, displaying the last row before the fetch at the top of the list window.

  This lets the user stay in the context of where they are working in the list. When they scroll to the bottom of the list, the next group is retrieved and appended to the first group. Then the scroll indicator bounces from the bottom to the middle of the scroll bar.

  If the user wants to scroll backwards from this point, the query does not have to be re-executed until the user scrolls to the top of the previous 250 records. If that happened, a Scroll Top event is triggered to retrieve the previous 250 records. The scroll indicator moves to the middle of the scroll bar, and the first row before the fetch is displayed at the top of the window.

- Provide *More* buttons in each direction, one for previous group and another for next group. The More buttons are enabled as long as data exists in either direction outside the current group view of the data.

  The SCROLL TOP and SCROLL BOTTOM events are not used, except perhaps to enable the More buttons if there is more data.

  If the user scrolls to the bottom of the list, and there is more data, the More button is enabled. When the user selects the button, the next group or half group of data is retrieved.

- Implement either of the previous alternatives without SCROLL TOP.

  SCROLL BOTTOM is used along with refresh to populate the first group view of the list.

  This avoids dealing with the complexity of SCROLL TOP processing.

User-Defined Events

User-defined events are useful when common logic is needed for several event handlers, and/or GUI statements are needed in the common logic.

A user-defined event shares views with the Procedure Action Diagram that contains the event.

The event USEs an external action block named TIREVENT, which must be added to the model. TIREVENT must have an import view of command, which will receive the name specified as the event type in CA Gen. This action block does not have to be generated.

Associate the event with the window or primary dialog box so that it can be shared and not duplicated within the procedure.

An example of how to specify the event type and action name for a window is shown in the next sample code.

Specify a User-Defined Event

```
Window Accounting
Define Event
Even Type Action Name Command
  makeacct ACCOUNTING MAKEACCT
READEACH ACCOUNT_READ_EACH
The next sample code shows a user-defined event that performs common logic and c
ontains a GUI statement.
 ---
|     --- EVENT ACTION accounts_pb_list_click
|    |    SET LAST OF export_account TO 0
|    |    SET local_ief_supplied_command to READEACH
|    |    USE tirevent
|     --- WHICH IMPORTS: Work_Viewlocal_ief_supplied
|
|     --- EVENT ACTION account_read_each
|    |   --- READ EACH account
|    |  |        SORTED BY ASCENDING account_number
|    |  |        WHERE DESIRED account_number IS GREATER OR EQUAL TO
|    |  |          input_account_number
|    |  | ADD EMPTY ROW TO export_account AFTER LAST OF export_account
|    |  | SET SUBSCRIPT OF export_account TO LAST OF export_account
|    |  | MOVE account TO export_line_account
|    |  |   --- IF LAST OF export_account IS EQUAL TO MAX OF export_account
|    |  |   --- ESCAPE
|    |   ---
|     --- SET output_account_type TO "E"
 ---
```

To trigger the user-defined event, the event type READEACH is passed to TIREVENT.

The event is executed when the current event handler or procedure logic is complete.

If a single event handler triggers multiple user-defined events, they are placed in a first-in first-out (FIFO) queue and executed in turn when the original logic completes execution.

GUI Presentation Actions

You can use these GUI statements in an action diagram control to open and close dialog boxes and manipulate fields:

- OPEN
- CLOSE
- DISABLE and ENABLE
- MARK and UNMARK
- REFRESH

These statements can be placed only in Procedure Action Diagrams or event handlers.

OPEN Statement

The OPEN statement causes the referenced dialog box within the current procedure to be opened after execution of the procedure or event handler.

Flows must be used to initiate opening windows or dialog boxes in other procedures.

OPEN is useful if additional processing is needed before a dialog box is displayed.

OPEN statements are not executed when they are encountered in the action language. They are placed in a FIFO queue. The dialog boxes are opened once the procedure or event handler completes execution.

The last dialog box opened gains the focus. *Focus* represents the part of the user interface that will receive the next piece of input if no navigation occurs before the input is generated.

If you are using the MAKE action to position the cursor to a field in another dialog box, you must consider when the dialog box is to open. If the MAKE action comes after OPEN, the MAKE will have no effect. This is because the opened dialog box obtains focus (control), which negating the MAKE action.

If multiple OPENs are executed for the same dialog box during an execution of a procedure or event handler, that dialog box will be opened only once. Subsequent executions of a procedure or event handler for a modal dialog box will not allow the box to be open multiple times.

A modal dialog box requires the user to respond to the dialog before continuing work in the application.

A modeless dialog box lets the user leave the dialog and continue working with other dialogs in the application.

```
 --- EVENT ACTION accounts_pb_sort_click
|    USE account_metadata
|       WHICH EXPORTS:   Group View export_sort_from
|                        Work View output_sort_metadata
 --- OPEN Dialog Box sort
```

In this example, data loads in a local view when the user clicks a push button. This is needed before the *sort* or *filter* dialog box is opened.

Conditional OPEN

```
 --- EVENT ACTION journal_entries_pb_balance_click
|    --- IF export_variance_ief_supplied_total_currency IS EQUAL TO 0
|   |    OPEN Dialog Box in_balance
|   |--- ELSE
|   |    OPEN Dialog Box balance
|    ---
 ---
```

In this example, a condition governs which of two dialog boxes is displayed.

CLOSE Statement

The CLOSE statement can act on both the primary window/dialog box or a secondary dialog box.

CLOSE pertains to only windows or dialog boxes in the current procedure.

A CLOSE statement cannot be executed at the same time as an OPEN for a dialog box. If the window is not open, no runtime error occurs.

A CLOSE for a window or dialog box can be contained in the OPEN event.

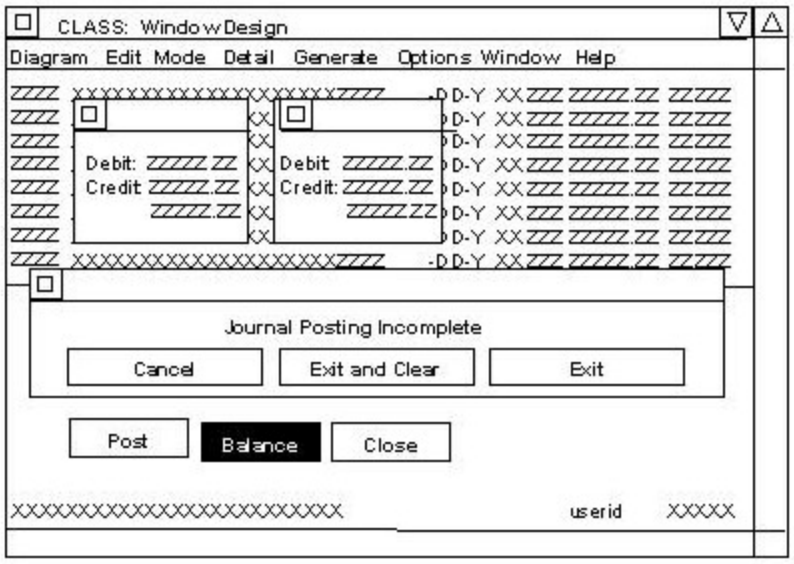
If the primary window or dialog box is closed, all secondary dialog boxes are closed.

Control returns to the last active window of the procedure that invoked the closed procedure. If the closed step was the first procedure, then control returns to the operating system.

CLOSE statements are not executed when they are encountered in the action language. They are placed in a FIFO queue. CLOSE statements are executed when the procedure or event handler completes execution.

In CA Gen, no relationship between secondary dialog boxes is maintained. It is the procedure's responsibility to close subordinate boxes if the procedure's structure has established a relationship between secondary dialog boxes.

The following illustration shows how the procedure establishes this relationship.



In the illustration, the Journal Entries dialog box is the primary dialog box. Each of the three other dialog boxes is dependent on it.

One of the two smaller dialog boxes is displayed when the *balance* push button is clicked on the Journal Entries dialog box. A green one is displayed if debits and credits are equal; a red one if they are not.

The third dependent dialog box is displayed when the Post push button is selected, and debits do not equal credits.

Whenever Journal Entries are discontinued, all four dialog boxes are closed.

The logic shown in the code in the next section closes the journal entries box. Therefore, the related boxes must be closed also.

CLOSE Statement for Logically Related Boxes

```
 --- EVENT ACTION journal_pb_exit_and_clear_click
|    SET LAST OF export_journal TO 0
|    CLOSE Dialog Box journal_posting
|    CLOSE Dialog Box in_balance
|    CLOSE Dialog Box balance
 --- CLOSE Dialog Box journal_entries
```

Note: Be careful of CLOSE events that use CLOSE actions triggering other CLOSE events. They may cause loops and unexpected results.

The code shown in next section demonstrates how a CLOSE can be used to close a window or dialog box if an error occurs.

CLOSE Statement for Errors

```
--- EVENT ACTION journal_entries_open
|   --- READ period
|  |        WHERE DESIRED period status IS EQUAL TO "A"
|  |--- WHEN successful
|  |    MOVE period TO output_period period
|  |    SET export_status_period ief_supplied sort_string TO
|  |        substr (textnum(period month), 14, 2)
|  |    SET export_status_period ief_supplied sort_string TO
|  |        concat (trim(export_status_period ief_supplied sort_string),"/")
|  |    SET export_status_period ief_supplied sort_string TO
|  |        concat(trim(export_status_period ief_supplied sort_string),
|  |        substr (textnum(period year), 12,4))
|  |--- WHEN not found
|  |    EXIT STATE IS no_active_period
|   --- CLOSE Dialog Box journal_entries
 ---
```

This example also uses the status bar to display information about the accounting period to which journal entries are being posted.

Other uses of the CLOSE statement are listed next:

- Closing a dialog box only if an operation is successful.
- Closing a dialog box during an OPEN event if a highlighted row was expected, and no row is highlighted.

DISABLE and ENABLE Statements

The DISABLE statement causes a menu item or push button to be disabled, which means it cannot be clicked on. The menu item or push button is disabled until its command is enabled using the Enable statement or the window/dialog box is closed.

CA Gen allows menu items and push buttons to be disabled. This feature should be used when possible to simplify procedure logic. However, when program control is desired, use the Disable command instead. The next section shows an example.

DISABLE Statement for a Push Button

```
 --- IF export_credit ief_supplied total_currency IS EQUAL TO
|        export_debit ief_supplied total_currency
|--- ENABLE COMMAND post
|    ELSE
 --- DISABLE COMMAND post
```

When using the CA Gen disabling feature built into menu items and push buttons, enabling is automatic. Menu items and push buttons are enabled when the disabling conditions are not met.

When DISABLE is used, the affected selection must be later enabled through action logic, if the selection is ever to be used.

Disable can also be used with modeless windows or dialog boxes to prevent users from inadvertently opening multiple dialog boxes at one time. For example, there are separate modeless boxes for add, change, and delete. When the user selects one of the push buttons, the other two are disabled.

Take care when disabling selections. With the emergence of the GUI, the traditional approach of letting the user type in all the data and then validating is replaced with disallowing certain controls and fields depending on which field the user entered data. Data validation went from reactive to preventive. It sounds like a good idea to disable the OK button until all the required data is filled in, and all the subordinate dialog boxes have been visited and their data has been filled in as well. But it turns out it is not such a good idea if there are several required and optional fields, several combinations of required fields, or required subordinate dialog boxes. A user can become frustrated trying to figure out what needs to be done before the OK button becomes enabled.

You should analyze each situation for the best approach to this issue. These are alternatives to consider:

- Change the color of the required fields and then disable the OK button until they are all filled.
- Users may be very familiar with the system and therefore knowing what is required is not an issue. In this case, it makes sense to disable the OK button.
- Have the system tell the user what is missing when they click the OK button.
- This approach is good for infrequent users.

- This approach does not have processing issues since the validation logic will probably reside in the client procedure.
- The disadvantage of this approach is the duplication of validation logic, which is also in the server procedure.

MARK and UNMARK Statements

A MARK statement immediately causes menu items that reference the command to display a check mark.

An UNMARK statement causes the check mark to be removed.

Only menu items placed on the windows/dialog boxes of the current procedure are affected by MARK and UNMARK statements. Push buttons that use the same commands are not affected.

MARK and UNMARK can be used to show what options have been specified for a window or dialog box. You can see this in the CA Gen Options drop-down menu where single or multiple adds for an object can be specified. You can also use these statements in any application that adds occurrences of entity types.

The next sample code shows examples of the use of these statements.

```
 --- EVENT ACTION collections_mi_single_add_click
|        SET local_add_option ief_supplied command TO COMMAND
|     --- IF COMMAND IS EQUAL TO singel
|    |    MARK COMMAND multiple
|    |--- ELSE
|    |    MAKE COMMAND multiple
|     --- UNMARK COMMAND single
 ---
```

REFRESH Statement

The REFRESH statement immediately causes all fields of all windows or dialog boxes of the current procedure to be updated with their export view values.

REFRESH can be used to provide status information to users during a long-running procedure. To serve this function, the REFRESH statement must be placed within the repeating logic that is causing the time delay.

The next sample code shows how this is done.

```
 --- EVENT ACTION click_update_progress_indicator
|    SET export work_fields temp_text_25 TO SPACES
|     --- FOR local number_control FROM 1 TO 24 BY 1
|     --- SET export work_fields temp_text 25 TO
|    |        concat(trim(export work_fields_temp_text_25),     )
|    |    REFRESH
|    |    FOR local_count work_fields rep_nbr FROM 1 to 5000 BY 1
|     ---
|    CLOSE Dialog Box modify_shipment
 --- OPEN?
```

As processing progresses, this logic extends a bar by appending a solid box to the bar every time a counter reaches 500,000.

Group View Manipulation

You can use these GUI statements in action diagram control for manipulating group views:

- SORT
- FILTER and UNFILTER
- ADD EMPTY ROW
- REMOVE ROW FROM
- HIGHLIGHT and UNHIGHLIGHT
- GET ROW HIGHLIGHTED
- GET ROW CLOCKED
- DISPLAY
- GET ROW VISIBLE

The same functionality is available for explicitly and implicitly indexed group views. The techniques employed are sometimes different, depending on the action. Explicitly indexed group views are used more in client procedures than implicitly indexed views. The examples for these statements use only explicitly indexed group views.

Restrictions on using the group view manipulation statements:

- These statements can only be placed in procedure action diagrams or event handlers.
- SORT, FILTER, UNFILTER, ADD EMPTY ROW, and REMOVE ROW from are only for export, local, or exportable import that includes a selection character.

SORT Statement

This statement sorts an updateable repeating group view based on a sort argument.

The sort argument consists of parameters made up of operators of the form n d. n is the sequence number of the attribute in the group view. The d is an A for ascending and a D for descending.

SORT Statement with a Character String

The next sample code illustrates the use of a character string to sort a group view on one field.

```
15-    --- EVENT ACTION accounts_pb_new_click
16-   |    --- CREATE account
17-   |   |    SET Description TO input account description
18-   |   |    SET type TO input account type
19-   |   |    SET number TO input account number
20-   |   |--- WHEN successful
21-   |   |    --- IF LAST OF export_account IS EQUAL TO MAX OF export_account
22-   |   |   |    SET SUBSCRIPT OF export_account TO LAST OF export_account
23-   |   |   |    --- IF input account number IS GREATER THAN export_line
24-   |   |   |   |        account_number
25-   |   |   |   |    EXIT STATE IS account_out_of_list_box
26-   |   |   |   |--- ELSE
27-   |   |   |   |    MOVE input account TO export_line account
28-   |   |   |    --- SORT export_account BY 4A
29-   |   |   |
30-   |   |   |--- ELSE
31-   |   |   |    SET SUBSCRIPT OF export_account TO LAST OF export_account +1
32-   |   |   |    MOVE input account TO export_line account
33-   |   |    --- SORT export_account BY 4A
34-   |   |
35-   |   |--- WHEN already exists
36-   |   |    MAKE output account number Unprotected Normal Intensity Normal
37-   |   |        Color containing Cursor
38-   |   |    EXIT STATE IS account_already_exists
39-   |   |    WHEN permitted value violation
40-   |    --- EXIT STATE IS account_invalid_permitted_values
41-   |
42-   |        GET ROW HIGHLIGHTED IN export_account STARTING AT 1 GIVING
43-   |            SUBSCRIPT OF export_account
44-   |    --- IF SUBSCRIPT OF export_account IS GREATER THAN 0
45-   |   |    MOVE export_line account TO output account
46-   |   |--- ELSE
47-   |   |    MOVE local_account_blank account TO output account
48-   |    --- SET output account type TO E
49-   |
50-    ---
```

In lines 27 and 32, the group view containing accounts is sorted ascending on the fourth field in the view, which is account number.

This example also shows how a new entity can be added to the bottom of a group view (if it is within the range contained in the group view). This logic is contained in the nested IF statements in lines 21 through 33. The group is then sorted to put the list back in sequence.

Another method of putting lists back in sequence is to use the ADD EMPTY ROW statement. For more information, see ADD EMPTY ROW Statement.

Always use a local character view for the sort argument character expression. Assign it a value in one place for ease of maintenance. Additionally, if plans call for the sort sequence to be user-defined, you must use a variable.

The next sample code shows one way to set up user definable sort sequences. The next sample code shows one way to set up user definable sort sequences.

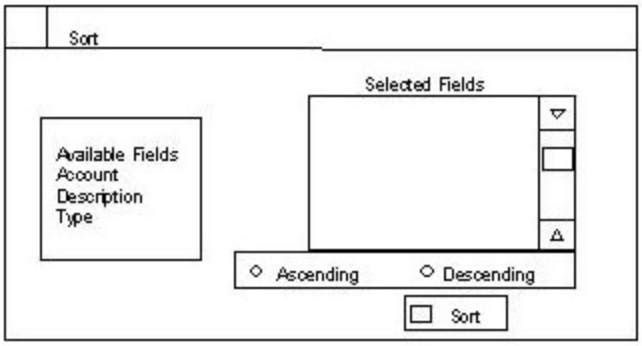
Load SORT Data

```
 --- ACCOUNT_METADATA
|      IMPORTS:
|      EXPORTS:
|      LOCALS:
|      ENTITY ACTIONS:
|
|    SET output_entity metadata entity_type TO ACCOUNT
|    SET SUBSCRIPT OF export_sort_from TO 0
|    SET SUBSCRIPT OF export_sort_from TO SUBSCRIPT OF
|        export_sort_from + 1
|    SET export_sort_from_line metadata name TO Account
|    SET export_sort_from_line metadata length TO 4
|    SET export_sort_from_line metadata number  TO 04
|    SET export_sort_from_line metadata type TO N
|    SET SUBSCRIPT OF export_sort_from TO SUBSCRIPT OF
|        export_sort_from +1
|    SET export_sort_from_line metadata name TO Description
|    SET export_sort_from_line metadata length TO 25
|    SET export_sort_from_line metadata number TO 02
|    SET export_sort_from_line metadata type TO T
|    SET SUBSCRIPT OF export_sort_from TO SUBSCRIPT OF
|        export_sort_from +1
|    SET export_sort_from_line metadata name TO Type
|    SET export_sort_from_line metadata length TO 1
|    SET export_sort_from_line metadata number TO 03
 --- SET export_sort_from_line metadata type TO T
```

The sample code shows the loading of data containing each attribute's name, length, sequence number, and domain in the view.

The attribute names are displayed on a dialog box. The user then selects the attributes in the preferred sort order. As the fields are selected, they are displayed in the selected fields list box.

Radio buttons on the dialog box are used to specify ascending or descending sort, as shown in the following illustration.



The next sample code illustrates the entire sort operation's logic.

User Defined SORT Sequence

```
 --- EVENT ACTION sort_open
|    SET LAST OF export_sort_to TO 0
 --- SET output_sort metadata ascend_descent TO “A”

 --- EVENT ACTION sort_1b_export_sort_doubleclick
|    GET ROW HIGHLIGHTED IN export_sort_from STARTING AT 1 GIVING SUB SCRIPT OF
|        export_sort_from
|    SET SUBSCRIPT OF export_sort_to TO LAST OF export_sort_to + 1
|    MOVE export_sort_from_line metadata TO export_sort_to_line metad ata
|    SET export_sort_to_ line metadata ascend_descend TO input_sort metadata
|        ascend_descend
|
|     --- IF input_sort metadata ascend_descend IS EQUAL TO “D”
|    |
|     --- SET export_sort_to_line metadata name TO
|             concat (trim(export_sort_to_line metadata name),   “(D)”)
 ---

 --- EVENT ACTION sort_pb_sort_click
|    SET local_sort_argument ief_supplied sort_string TO SPACES
|     --- FOR SUBSCRIPT OF export_sort_to FROM 1 TO LAST OF export_s ort_to BY:
|    |    SET local_sort_argument ief_supplied sort_string TO
|    |        concat (trim(local_ sort_argument ief_supplied sort_string),
|    |        concat (“ “, concat(export_sort_to_line metadata number,
|     ----    concat (“ “, export_sort_to_line metadata ascend_descent))) )
|
|     --- IF local sort_argument ief_supplied sort_string IS GREATER THAN SPACES
|    |     --- CASE OF output_sort metadata entity_type
|    |    |--- CASE “ACCOUNT”
|    |    |--- SORT export_account BY local_sort_argument ief_supplied sort_string
|    |     --- OTHERWISE
|     ---
 --- CLOSE Dialog Box sort
```

The sample code shows four event handlers:

- The event handler accounts_pb_sort_click calls the action block described in the sample code in Load SORT Data section to load the sort options into the export view.
- It then opens the Sort dialog box, which triggers the next event, sort_open.

- The event handler sort_open initializes the subscript of the fields the user will request as their sort preference and sets the default sort mode to ascending. The user specifies sort preferences by double-clicking on the available fields. Double-clicking triggers the event handler sort_lb_export_sort_doubleclick.

  The event handler sort_lb_export_sort_doubleclick moves the available field to selected fields list box. This shows the user the sort sequence that is being constructed.

  The last event handler is triggered when the user clicks on the sort push button in the sort dialog box.

  It builds the sort argument character expression. The expression would be constructed as "3 A 4 A" if the preferred sort sequence was account type (the third field in the view) and account number (the fourth field in the view).

- This event handler then sorts the view.

When building user-defined sort preferences, provide logic in the action diagram that allows a user to "undo" a selection.

SORT and FILTER each work on group views, not on a database. Thus, the occurrences that are being sorted or filtered may only be a subset of the data that should be sorted or filtered. It may not be possible for all the data to be returned to the group view.

The group view should be large enough to handle all occurrences that are to be sorted or filtered. This can be done for entity types whose number is small (usually low thousands at most) or by pre-filtering the occurrences that are to be sorted/filtered as they are read. Therefore, the SORT or FILTER statements will not always be used when sort and filter type logic is needed.

FILTER and UNFILTER Statements

The FILTER statement filters an updateable repeating group view to show only those occurrences of the view that match a filter character expression. The occurrences that are not shown remain in the group view and can be redisplayed by using the UNFILTER statement

The filter-character-expression contains an attribute, an operator (=, >, < and so forth), and a value.

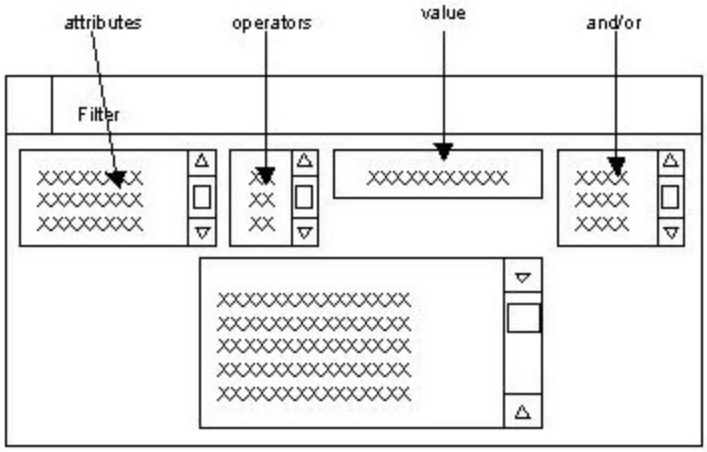
Attribute expressions can be *strung* together by using *and* and *or* operators.

For example, a filter to extract accounts in ZIP Code 75208 with a balance of over $10,000 can be expressed as:

```
zipcode = 75208 and balance > 10000"
```

As recommended for the SORT statement, the filter character expression should be a character view for ease of maintenance and flexibility.

A dialog box can be used to construct the filter expression, as shown in the following illustration.



The four boxes at the top of the dialog box (attributes, operators, value, and and/or) are the components of the filter expression. The user selects from these options.

The field on the bottom of the dialog box shows the FILTER expression as it is being constructed. The user undoes by double-clicking the filter expression line that is being constructed in the list box at the bottom of the FILTER dialog box.

The next sample code shows the use of the FILTER statement.

FILTER Statement for a Group View

```
 --- EVENT ACTION filter_ok
|    CLOSE Dialog Box filter 0
|    CLOSE Dialog Box filter
|    SET local_filter_big ief_supplied filter_big_field TO SPACES
|
|     --- FOR SUBSCRIPT OF export_selected_filter FROM 1 TO LAST OF
|    |        export_selected_filter BY 1
|    |     --- IF subscript of export_selected_filter IS EQUAL TO 1
|    |    |    SET local_filter_big ief_supplied filter_big_field TO
|    |    |        concat (trim(local_ filter_big ief_supplied
|    |    |        filter_big_field),
|    |    |        export_line_selected_filter ief_supplied
|    |    |        filter_input_statement)
|    |    |--- ELSE
|    |    |    SET local_filter_big ief_supplied filter_big_field TO
|    |    |        concat (trim(local_ filter_big ief_supplied filter_big_field),
|    |    |        concat (" ",export_line_selected_filter ief_supplied
|    |     ---         filter_input_statement)
|     ---
|
|     --- CASE OF output_meta_entity ief_supplied meta_entity_name
|    |--- CASE "JOURNAL"
|    |    FILTER export_je BY local_filter_big ief_supplied
|    |        filter_big_field),
|    |--- CASE "ACCOUNT"
|    |    FILTER export_account BY local_filter_big ief_supplied
|    |        filter_big_field),
|    |--- CASE "DEPT"
|    |    FILTER export_department BY local_filter_big ief_supplied
|    |        filter_big_field),
|     --- OTHERWISE
 ---
```

It uses data similar to that described for the SORT statement to populate the dialog box shown in the illustration in FILTER and UNFILTER Statements.

The logic in the FOR loop constructs the filter expression from the options specified in the FILTER dialog box.

The CASE logic filters the applicable group view.

After a group view is filtered, it must be unfiltered to display all group view occurrences. This is shown in the next sample code.

UNFILTER Statement for a Group View

```
 --- EVENT ACTION account_unfilter
|
 --- UNFILTER export_account
```

ADD EMPTY ROW Statement

This statement inserts a blank row into an updateable repeating group. The empty row can then be filled with data.

The empty row may be added BEFORE or AFTER the current index if the index expression is omitted.

An index expression may be specified for explicitly indexed group views. There is no need to refresh the list or to have complex expansion logic in the action diagram.

The next sample code shows adding an empty row and subsequently filling it.

```
 --- EVENT ACTION account_double_click
|    GET ROW CLICKED IN export_account STARTING AT 1 GIVING SUBSCRIPT
|         OF export_account
|
|     --- EXIT STATE IS aok
|    |--- FOR SUBSCRIPT OF export_budget FROM 1 TO LAST OF export_budget BY 1
|    |
|    |     --- IF export_line_budget account number IS EQUAL TO export_line
|    |    |        account number
|    |     --- EXIT STAT IS duplicate_accounts
|     ---
|
|     --- IF EXIT STATE IS EQUAL TO aok
|    |
|    |     --- FOR SUBSCRIPT OF export_budget FROM 1 TO LAST  OF
|    |    |        export_budget BY 1
|    |    |
|    |    |     --- IF export_line_budget account number IS GREATER THAN
|    |    |    |        export_line account number
|    |    |    |    ADD EMPTY ROW TO export_budget BEFORE SU BSCRIPT OF
|    |    |    |        export_budget
|    |    |    |    MOVE export_line account TO export_line_budget account
|    |    |     --- ESCAPE
|    |     ---
|    |
|    |     --- IF LAST OF export_budget IS EQUAL TO 1
|    |    |    ADD EMPTY ROW TO export_budget AFTER 0
|    |    |    SET SUBSCRIPT OF export_budget TO 1
|    |    |--- ELSE
|    |    |    ADD EMPTY ROW TO export_budget BEFORE LAST OF  export_budget
|    |     --- SET SUBSCRIPT OF export_budget TO LAST OF expo rt_budget  - 1
|    |
|     --- MOVE export_line account TO export_line_budget account
 ---
```

The logic operates on two group views that contain the account entity type. The first view contains a list of all accounts. The second view contains the list of accounts for which budgets exist.

The user double-clicks on an account from the first view to select it for inclusion in the second view.

Duplicates are not allowed and are checked in the first FOR loop.

The remainder of the logic places the account number selected into account number sequence in the second view.

REMOVE ROW FROM Statement

This statement removes a row from an updateable repeating group view.

For explicitly indexed group views, an AT INDEX expression appears when the statement is constructed.

If an entity is deleted from a list and removed from the database, there is no need to refresh the list or to have complex compression logic in the action diagram.

The next sample code illustrates the use of the REMOVE ROW FROM statement.

REMOVE with DELETE Action

```
 --- EVENT ACTION delete_department_pd_delete_click
|    GET ROW HIGHLIGHTED IN export_department STARTING AT 1
|         GIVING SUBSCRIPT OF export_department
|
|     --- IF SUBSCRIPT OF export_department IS GREATER THAN 0
|    |
|    |     --- READ department
|    |    |        WHERE DESIRED department number IS EQUAL TO
|    |    |            input_department department number
|    |    |--- WHEN successful
|    |    |    REMOVE ROW FROM export_department AT SUBSCRIPT OF
|    |    |        export_department
|    |    |    DELETE department
|    |    |--- WHEN not found
|    |    |    EXIT STATE IS department_not_found WITH ROLLBACK
|    |     ---
|     ---
 ---
```

The sample code shows the deletion of a department entity from the database and the updating of the group view through a refreshing READ.

Another example using REMOVE is shown in the next sample code.

REMOVE from Sort String

```
 --- EVENT ACTION so_lb_export_sort_to_doubleclick
|    GET ROW HIGHLIGHTED IN export_sort_to_STARTING AT 1 GIVING
|        SUBSCRIPT OF export_sort_to
 --- REMOVE ROW FROM export_sort_to  AT SUBSCRIPT OF export _sort_to
```

In the sample code, the REMOVE action removes a sort option if a user double-clicks on its row in a list box.

HIGHLIGHT and UNHIGHLIGHT Statements

The HIGHLIGHT and UNHIGHLIGHT statements highlight or unhighlight a given row of an export repeating group view.

For explicitly indexed fields, an AT index expression is used to specify the row in the group view to be highlighted.

Although a *clicked* field is highlighted by the system, it is sometimes useful to highlight a field under system control.

In the next sample code, highlight is used to show a default. In this case, it is the default sort order for a list of accounts.

HIGHLIGHT Statement to Show Default

```
 --- EVENT ACTION accounts_pb_sort_click
|    USE account_metadata
|        WHICH EXPORTS:  Group View export_sort_from
|                        Work View output_sort_metadata
|    HIGHLIGHT export_sort_from  AT 2
 --- OPEN Dialog Box Sort
```

The next sample code shows the data for sorting accounts was loaded.

HIGHLIGHT Statement for the Last Entity Added

```
 --- EVENT ACTION accounts_pb_new_click
|     --- CREATE account
|    |    SET description TO input account description
|    |    SET type TO input account type
|    |--- SET number TO input account number
|    |     --- WHEN successful
|    |    |    IF LAST OF export_account IS EQUAL TO MAX OF export_account
|    |    |     --- SET SUBSCRIPT OF export_account TO LAST OF export_account
|    |    |    |    IF input account number IS GREATER THAN export_line
|    |    |    |        account number
|    |    |    |    EXIT STATE IS account_out_of_list_box
|    |    |    |--- ELSE
|    |    |    |    MOVE input account TO export_line account
|    |    |    |    HIGHLIGHT export_account AT SUBSCRIPT of export_account
|    |    |    |    MOVE input account TO output account
|    |    |     --- SORT export_ account BY 4A
|    |    |--- ELSE
|    |    |    SET SUBSCRIPT OF export_account TO LAST OF export_account + 1
|    |    |    MOVE input account TO export_line account
|    |    |    HIGHLIGHT export_account AT SUBSCRIPT of export_account
|    |    |    MOVE input account TO output account
|    |     --- SORT export_ account BY 4A
|    |--- WHEN already exist
|    |    MAKE output account number Unprotected Normal Intensity
|    |        Normal Color Containing Cursor
|    |--- EXIT STATE IS account_already_exists
|    |    WHEN permitted value violation
|     --- EXIT STATE IS account_invalid_permitted_values
 ---
```

In the sample code, the account description was the second row in the available fields list. That row is highlighted as the default sort sequence. To select this sequence, the user presses Return when the sort dialog box pops up.

The next sample code illustrates highlighting the last account added to a list.

UNHIGHLIGHT Statement for a Changed Entity

```
 --- EVENT ACTION change_department_pb_ok_click
|    MOVE input_department department TO output_department  department
|    GET ROW HIGHLIGHTED IN  export_department STARTING AT  1 GIVING
|        SUBSCRIPT OF export_department
|     --- IF SUBSCRIPT OF export_department IS GREATER THAN  0
|    |     --- READ department
|    |    |    WHERE DESIRED department number IS EQUAL TO
|    |    |    input_department department number
|    |    |--- WHEN successful
|    |    |     --- UPDATE department
|    |    |    |    SET manager TO input_department department manager
|    |    |    |    SET description TO input_department department description
|    |    |    |--- WHEN successful
|    |    |    |    MOVE input_department department TO
|    |    |    |        export_department_line department
|    |    |    |    CLOSE Dialog Box change_department
|    |    |    |    UNHIGHLIGHT export_department AT SUBSCRIPT OF
|    |    |    |        export_department
|    |    |    |--- WHEN not unique
|    |    |    |    EXIT STATE IS system_error
|    |    |    |--- WHEN permitted value violation
|    |    |    |    EXIT STATE IS dept_invalid_permitted_values
|    |    |     ---
|    |     --- WHEN not found
|    |         EXIT STATE IS department_not_found WITH ROLLBA CK
|     ---
 ---
```

The sample code shows the removal of the highlight from a list box after the highlighted row has been changed. This is particularly important if the system allows multiple rows to be highlighted, each being processed in some defined sequence.

This was done to let the user easily change an entry just made. In addition to being highlighted, the account is added to the export view used to change accounts.

The user does not have to click on a field if the default highlighted is their choice.

GET ROW HIGHLIGHTED Statement

A GET ROW HIGHLIGHTED statement determines the index of the first highlighted row of an explicitly indexed repeating group view starting from a specified position.

A row can be highlighted by the user selecting it, or by using the HIGHLIGHT statement.

A conditional statement may be used to test whether an implicitly indexed repeating group view is highlighted or not.

The next sample code illustrates the use of this statement.

```
 --- EVENT ACTION accounts_1b_export_account_click
|    GET ROW HIGHLIGHTED IN  export_account STARTING AT 1 G IVING
|        SUBSCRIPT OF export_account
|    SET local ief_supplied subscript TO SUBSCRIPT OF export_account
|     --- IF SUBSCRIPT OF export_account IS GREATER THAN 0
|    |    MOVE export_line account TO output account
|    |    ELSE
|    |    MOVE local_account_blank account TO output account
|    |    SET output account type TO “E”
|     --- EXIT STATE IS nothing_selected
 ---
```

The logic in the sample code finds the first account highlighted and puts it into the single occurrence export view so that it can be changed. The row must be unhighlighted in the change logic, and the next highlighted row, if any, is put into the single occurrence export view.

If multiple rows are selected and need to be processed in the order in which they appear in the list, then GET ROW HIGHLIGHTED should be used (it works with both click and double-click).

If processing should occur as rows are clicked, then the next statement, GET ROW CLICKED should be used instead.

GET ROW CLICKED Statement

A GET ROW CLICKED statement can be used in the event handler associated with a click event to determine which element of an explicitly indexed repeating group was selected. This statement is used in a similar manner to GET ROW HIGHLIGHTED.

A conditional statement may be used to test whether an implicitly indexed repeating group view is clicked or not.

The next sample code illustrates the use of this statement.

GET ROW CLICKED to Find a Row

```
 --- EVENT ACTION account_double_click
|    GET ROW CLICKED IN  export_account STARTING AT 1 GIVIN G SUBSCRIPT
|        OF export_account
|     --- EXIT STATE IS aok
|    |    FOR SUBSCRIPT OF export_budget FROM 1 TO LAST OF e xport_budget BY 1
|    |     --- IF export_line_budget account number IS EQUAL TO
|    |    |        export_line account number
|    |     --- EXIT STATE IS duplicate_accounts
|     ---
|     --- IF EXIT STATE IS EQUAL TO aok
|    |     --- FOR SUBSCRIPT OF export_budget FROM 1 TO LAST  OF
|    |    |        export_budget BY 1
|    |    |     --- IF export_line_budget account number IS GREATER THAN
|    |    |    |        export_ account number
|    |    |    |    ADD EMPTY ROW TO export_budget BEFORE SU BSCRIPT OF
|    |    |    |        export_budget
|    |    |    |    MOVE export_line account TO export_line_budget account
|    |    |     --- ESCAPE
|    |     ---
|    |     --- IF LAST OF export_budget IS EQUAL TO 1
|    |    |    ADD EMPTY ROW TO export_budget AFTER 0
|    |    |    SET SUBSCRIPT OF export_budget TO 1
|    |    |--- ELSE
|    |    |    ADD EMPTY ROW TO export_budget BEFORE LAST OF  export_budget
|    |     --- SET SUBSCRIPT OF export_budget TO LAST OF expo rt_budget   - 1
|     --- MOVE export_line account TO export_line_budget account
 ---
```

In the sample code, budgets can be added for each account. The accounts are selected from an accounts list. This logic finds the account that was double-clicked and then checks for a duplicate budget. If no duplicate is found, the account is added to the budget list.

The selection indicator for the repeating group view that indicates what row is selected changes when associated with the clicked event handler:

- If the row is selected, it is set to +.
- If the row was clicked but not selected, it is set to -.

Note: The selection indicator remains an asterisk symbol (*) for non-event handler procedure logic.

GET ROW HIGHLIGHTED is preferred to GET ROW CLICKED because with the latter, the event handler logic must check the indicator to see if the row has been deselected and should not be processed. With GET ROW HIGHLIGHTED, the mouse controls the highlighting, and no checks are necessary to see if a row has been *unhighlighted*.

DISPLAY Statement

This statement places the referenced row of an export repeating group at the top of a list box:

- If the view is explicitly indexed, an index expression is used.
- If the view is implicitly indexed, the current row is displayed at the top.

The next sample code demonstrates the use of this statement.

Bring a Row to the Top with DISPLAY

```
 --- EVENT ACTION account_new_ok
|     --- CREATE account
|    |    SET type TO input account type
|    |    SET description TO input account description
|    |    SET number TO input account number
|    |--- WHEN successful
|    |
|    |    GET ROW HIGHLIGHTED IN export_account STARTING AT  1 GIVING
|    |        SUBSCRIPT OF export_account
|    |     --- IF SUBSCRIPT OF export_account IS EQUAL TO 0
|    |     --- SET SUBSCRIPT OF export_account TO LAST OF export_account
|    |    ADD EMPTY ROW TO export_account AFTER SUBSCRIPT OF
|    |        export_account
|    |    SET SUBSCRIPT OF export_account TO SUBSCRIPT OF ex port_account +1
|    |    MOVE input account TO export_line account
|    |    HIGHLIGHT export_account AT SUBSCRIPT OF export_account
|    |    DISPLAY export_account WITH SUBSCRIPT OF export_account ROW AT TOP
|    |    EXIT STATE IS enter_new_account
|    |--- WHEN already exists
|    |    EXIT STATE IS system_error_invalid_account WITH ROLLBACK
|    |--- WHEN permitted value violation
|    |    EXIT STATE IS system_error_invalid_values WITH ROLLBACK
|     ---
 ---
```

In the sample code, as new accounts are added, the last one added is brought to the top of the list and highlighted. The next account will be added to the list next the account at the top, unless the user has highlighted another account for the new one to follow.

GET ROW VISIBLE Statement

The GET ROW VISIBLE statement can be used with a filtered group view to select a visible row in an explicitly indexed repeating group view. The GET ROW VISIBLE statement is a companion to the FILTER statement.

This statement can be used to count the following rows:

- Number of rows visible or invisible so that the number of rows that were unfiltered or filtered can be displayed.
- Further process rows that were (not) filtered.

For example, the filter may have selected all accounts in ZIP Code 75208.

The next action is to add up all the balances for the selected ZIP Code. GET ROW VISIBLE is used to cycle through the filtered rows and add up the balances.

A conditional statement may be used to test whether an implicitly indexed repeating group view is visible or not.

Implicitly Indexed Group Views

With implicitly indexed group views, the subscript is controlled using repetitive logic, such as READEACH.

Most group view statements can be used, but GET statements for HIGHLIGHTED, CLICK, and VISIBLE are not available for implicitly indexed group views. In their place are tests that can be made, such as IS (NOT) HIGHLIGHTED.

ADD EMPTY ROW will add BEFORE, AFTER, or at the current subscript. REMOVE ROW will remove the row at the current subscript location. Highlight and UNHIGHLIGHT also affect the row at the current subscript location.

The DISPLAY statement uses the syntax WITH CURRENT ROW AT TOP.

Handle Repeating Groups

Each repeating group view is defined as either one of the following groups:

- Implicitly subscripted
- Explicitly subscripted

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 example, 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, the generate procedure automatically jumps to the next occurrence of the targeted repeating group view after each iteration (if the current occurrence is modified).

The following constructs support implicit subscripting and apply only to implicitly subscripted repeating group views:

- FOR EACH
- Targeting clause on the actions:
  - FOR EACH
  - READ EACH
  - WHILE
  - REPEAT
- Condition is:
  - FULL
  - NOT FULL
  - EMPTY
  - NOT EMPTY

In most cases, implicit subscripting is sufficient for the handling of repeating group views, particularly during analysis.

However, some cases exist where the processing requirement for repeating group views exceeds the capabilities of implicit subscripting. For example, 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 helpful.

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 test for the end of the group explicitly.

The example shown in the next sample code causes the size of each elephant in the repeating group view to be set to elephantine.

Explicit Subscripting Example

```
 --- 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
 ---
```

In contrast, had the repeating group view Elephants been defined as implicitly subscripted, the statement in the next figure would accomplish the same result.

Implicit Subscripting Example

```
 --- FOR EACH elephants
|        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 identifies the highest subscript value for which a repeating group view occurrence exists.
- MAX OF repeating-group-view special attribute identifies the maximum cardinality of the repeating group view.

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.

How CA Gen Supports OLE

This discussion is not an attempt to teach you about OLE (object linking and embedding), but to describe how CA Gen supports OLE concepts.

CA Gen supports the following OLE concepts:

- OLE automation
- Embedding
- OCXs (OLE custom controls)

  Note: This information pertains to Microsoft Win32 platforms only.

CA Gen Control through OLE Automation

OLE automation is controlling one application from within another application using objects. The objects used for this purpose are called OLE automation objects. An OLE automation object is an exposed component of an OLE automation server.

Automation objects provide properties and methods that allow the CA Gen application to control the server application, just as if you were issuing commands in the other application.

A method is a function that performs an action on the OLE automation object. For example if you are working with a column in a Microsoft Excel spreadsheet, you may apply a method to the column to sum the numbers contained in the column's cells.

Properties are the characteristics of an automation object that describe the object. For example, if you are working with a column in a Microsoft Excel spreadsheet, the column's width is a property that can be changed.

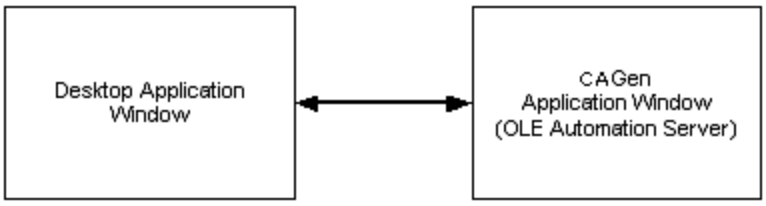
Not all OLE servers expose both methods and properties. If you are building a controller in CA Gen, you need to determine which methods and properties are available from the OLE servers you will be using.

CA Gen supports OLE automation by acting as an OLE automation controller or an OLE automation server.

CA Gen as an OLE Automation Controller

An OLE automation controller is an application that accesses and controls OLE automation objects.

CA Gen can be an automation controller. For example, OLE automation allows you to remotely operate OLE server applications such as Microsoft Word or Excel. The following illustration shows this role.



In the illustration, communication is from the CA Gen application to the desktop application (for example, an Excel spreadsheet). CA Gen acts as an OLE automation controller. CA Gen initiates the communication. The desktop application is presented in its own window in the CA Gen application.

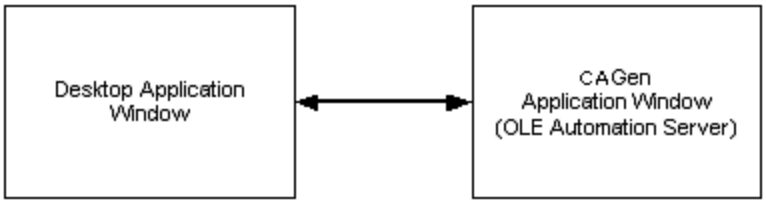
The other OLE application can have the following characteristics:

- Be completely external, including all of its presentation.
- Have its presentation embedded in the window of the CA Gen application. For more information, see CA Gen Control through Embedding.

CA Gen as an OLE Automation Server

An OLE automation server is an application that can be accessed through methods and properties and an automation controller.

CA Gen can be an automation server. The following illustration shows this role.



In the illustration, communication is from a desktop application such as an Excel spreadsheet. The desktop application initiates the communication. CA Gen becomes the OLE automation server.

The generation of the automation server is automatic. The following list shows the only tasks that you need to perform:

- Adding the names to the controls and windows in the CA Gen model. For more information, see the chapter "Designing the Graphical User Interface."
- A default name will be generated for each control placed on the window.
- Building scripts to access the interface
- These scripts are built outside the CA Gen application (for example, from Microsoft Excel or Visual Basic).

Use an OLE Automation Object

The following steps explain how to use an OLE automation object:

1. Dimension the object variable (create a work set in CA Gen).
2. Create an object (use a set statement to set a work view).
3. Set the object as the contents of the object variable (use a set statements to set a work view).
4. Do the work you wish to do with the object (use the invoke method statements to work with the object).
5. Clear the object when you are finished with it.

The next sample code illustrates an action diagram fragment that opens Microsoft Word and places the traveler's name into a field in a document.

```
1-    --- EVENT ACTION word open

2-   |    SET local guiobj word TO APPLICATION CreatObject(T:Word.Basic)
3-   |    INVOKE local guiobj word AppShow()
4-   |    INVOKE local guiobj word AppMaximize()
5-   |    INVOKE local guiobj word FileOpen (T:d:\reservation.doc)
6-   |    INVOKE local guiobj word EditGoTo (T:person)
7-    --- INVOKE local guiobj word InsertField (T:export_selected_customer)
```

The following table explains the sample code:

Line	Action
1	Builds an event to open a Word document and place information into it. This is the equivalent of the dimension statement (Step 1).

Line	Action
2	Sets a local view guiobj word to the CreateObject type library from Word. This creates the object (Step 2 object).
3	Invokes a method to show the Word application.
4	Invokes a method Word makes available to maximize the application (in this case the application is already running on the desktop).
5	Invokes a method to open the Word document reservation.doc.
6	Invokes a method to go the person place holder.
7	Invokes a method to insert the export view of the selected customer into the person place holder.

Name Work Views for Manipulating OLE Objects

Because you will use a significant number of work views for manipulating OLE objects, you need to set standards for naming them.

We recommend you to name them according to what objects you will be manipulating. For example, name the workset *Excel* and the attributes window, push button, and tool bar.

CA Gen Control through Embedding

CA Gen can control other OLE applications by embedding the presentation of the other OLE applications into the CA Gen window. This action uses OLE compound documents (in-place editing) technology.

The embedded application is run by the CA Gen application through action diagram statements.

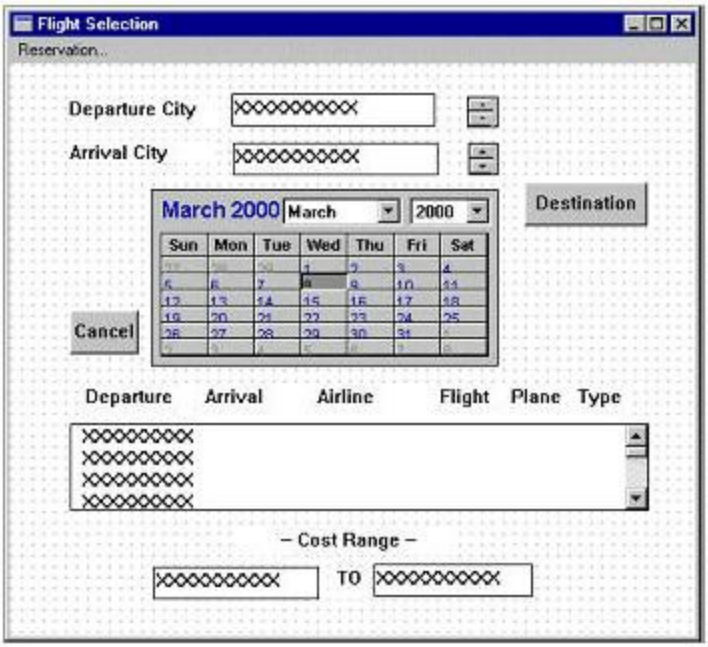
Since the desktop application is embedded, presentation commands need to be issued. The presentation commands are simply action diagram statements.

CA Gen Control through OCXs

CA Gen supports OCXs (OLE custom controls).

Each OCX comes with its own set of events. Some of the events may have the same names as the events CA Gen offers (for example, CHANGED or OPEN), but the events that come with the OCX are available only to the control itself.

The following illustration shows an example of three OCXs, two of which are spin buttons, while the other is a calendar.



The calendar contains the following events that it makes available to the CA Gen application.

- MOUSEDOWN
- MOUSEMOVE
- MOUSEUP
- ERROR
- CLICK
- DBLCLICK
- KEYDOWN
- KEYPRESS
- KEYUP
- CHANGE

These events can be filled by whatever action diagram statements are appropriate for each of the possible actions.

Attributes are provided by certain OCX events. For example, the Calendar OCX CLICK event makes available an attribute named *the date*. The next sample code provides an example of how the OCX CLICK event is used.

More information:

Designing the Graphical User Interface (see page 117)

OCX Click Event

```
 --- EVENT ACTION flight_select_olecontrol1_click
|   SET export flight departure_date to export change the date
|   USE flight_server (procedure step)
|       WHICH IMPORTS:  Entity View export flight to Entity View import
|          flight
|       WHICH EXPORTS:  Group View export_group FROM Group View
 ---       export_group_flight
```

The example in the sample code reads all of the flights from a certain departure city and a certain arrival city for a date selected through the calendar CLICK event.

The CLICK event sets an export departure date to the attribute supplied by the OCX click event *thedate*. It then remote uses a server procedure to read the flights from the departure city to the arrival city on the specific date and returns a group view of those flights.

External Action Blocks

You may need to use logic that was not defined using CA Gen. You can do this by using an external action block.

Use an external action block to do the following tasks:

- Bridge to an existing system
- Read from or write to an ASCII file
- Construct and print reports
- Perform database functions not supported by CA Gen
- Perform other C functions such as CA Gen assigns a small integer (8 bits) for number set to 1 position.
- If more granular typing (domain assignment) is required, an external action block can be used.

An external action block has only three components:

- Import views
- Export views

- External designation

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. This type of action block 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 next sample code illustrates a CA Gen-defined Procedure Action Diagram that references an external date manipulation routine called Date Services.

Procedure Action Diagram for Date Services

```
 --- CHECK_HOLIDAY
|      IMPORTS: Entity View incoming customer
|      EXPORTS:         Entity View work status
|      LOCALS:            Entity View work date
|
|    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
|    |
|     --- EXIT STATE IS shipment_requested_on_a_holiday
 ---
```

The next 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
 ---
```

Copy with Substitution

Copy With Substitution lets you copy an existing action block or event and substitute data during the copy. This capability saves time when you are completing similar actions on different data.

You can use Copy With Substitution when you want to select an event and copy that event either into the same procedure, another procedure in the same business system, or another procedure outside the same business system.

For many of the events (for example, the OPEN event), you will want to perform similar activities for all the list boxes. You can build a standard OPEN event, which other designers can copy and substitute their own data.

The action block Copy With Substitution works by performing the following tasks:

- Selecting an existing action block that references a single entity type
- Replacing the reference to that entity type and its attributes with another entity type of choice
- Creating a new action block

For example, consider an action block named Add Employee. Its logic may be quite similar to an existing action block called Add Customer. Using Copy With Substitution, you can request that an Add Employee action block be built from Add Customer, substituting references to Customer with references to Employee. It may be used on procedure or action blocks.

Copy With Substitution for an action block is presented as an alternative only when both of the following occur:

- The action block to be copied references entities of only one type.
- No relationships (except for ones 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, you should check that the action block produced by Copy With Substitution really does what it should.

Results of Procedure Logic Design

You can see the following results after procedure logic design:

- Designed client procedures
- Designed server procedures

Chapter 8: Error Handling

This chapter describes how CA Gen handles runtime errors and how to build logic to identify and communicate application errors to users. The ability to build CA Gen applications that can be distributed across multiple platforms opens up a key issue: how to capture and communicate errors in a distributed environment.

Errors can be classified into two types:

- Runtime errors—CA Gen automatically captures and communicates all runtime errors to users.
- Application errors—The designer builds logic to identify and communicate application errors to users.

Handle Runtime Errors

CA Gen automatically captures all runtime errors in the client or the server. Runtime errors include all non-expected database failures, communication, and teleprocessing monitor errors.

CA Gen automatically communicates runtime errors to users. Runtime error messages are displayed in a CA Gen dialog box. When a message is displayed the user can view the message, save it to a disk file, and then continue to work with the application. The message can be used for resolving the runtime error.

Design Error Handling in Applications

You are responsible for designing in the capture and reporting of application errors. Application errors include not found conditions, validation errors, and security violations.

Note: This discussion assumes you are using the distributed process model design approach for client/server. This model implies that business logic is split between a client procedure supporting primarily presentation and a server procedure primarily supporting data access. Your distributed process model could be implemented across multiple platforms or reside entirely on the workstation. (If the application executes entirely on the workstation, it will use remote data services to access data.)

There are two types of application errors that need to be captured when implementing client/server applications using the distributed process model:

- Client errors (those captured in the client procedure)

- Server errors (those captured in the server procedure)

Client Error Handling

Errors captured in the client procedure can be handled in the same way errors are handled for block mode and standalone GUI applications-the client procedure detects the error, sets the appropriate exit state, and the error is displayed by the window manager.

For client procedures, this technique is the most flexible and easiest to implement.

Use exit states to capture client errors no matter what technique you use for capturing server errors.

Server Error Handling

The results of a server action must be communicated to the client procedure so the user can be informed about the success or failure of a requested operation.

There are several design techniques to communicate errors from the server to the client. Each of the techniques provides different capabilities.

The following list shows the techniques for handling server errors:

- Flow or remote use and exit states
- Exit states and action block
- Server-supplied error information

Use Flow or Remote Use and Exit States

The flow or remote use and exit states technique provides a quick mechanism to communicate errors back to the client.

The interaction between client and server procedure is the link flow or a remote use statement.

On a flow, CA Gen captures the exit states that cause the client to flow to the server (the Flows On Exit State) and the exit state that causes the server to return to the client (the Returns on Exit State).

In the client/server environment, there is an implied return from a server to a client. Upon completion of the server, control returns to the client regardless of what exit state is set in the server. Defining the returns on exit state is not necessary.

When CA Gen returns on a link flow or a remote use from the server to the client, it can display the message associated with the exit state set in the server procedure when it completes.

To display a message, the exit state must have its message type defined as Informational, Warning, or Error. The message will be displayed automatically in a dialog box whether the returns on property is Execute First or Display First. After the user presses the OK push button on the message box, if the client/server procedure interaction is a link flow, one of the following two things happen:

- If the upon return property on the flow is defined Execute First, the client procedure is executed.
- If the upon return property on the flow is defined Display First, the primary window of the client procedure is displayed.

If the client/server procedure interaction is a remote use then the next statement in the procedure logic (main or event logic) executes.

Given this behavior, the server can set an exit state with the appropriate message to be displayed. At the client procedure, the message displays automatically.

A link can have up to eight exit states that trigger a return to the client defined on the flow. However because there is an implied return from the server to the client, defining the exit state is optional. As a result, the server has an unlimited number of returning conditions.

Each exit state set by the server will represent a different condition in which the server may return to the client. Although specifying the upon return exit states on a link flow is optional, an application may choose to identify them so that the communication between the client and the server procedure is completely documented. This documentation of a flow will help in maintaining the application.

The following list explains advantages of this technique:

- CA Gen handles the communication of the error messages from the server to the client. You do not have to design any communication architecture to support error handling.
- The message box displayed to the user will contain the appropriate icon to represent the type of error message being displayed, which improves the quality of end user communication.
- Changes to exit state messages are automatically applied by performing code generation on the affected components.
- Using exit states, you can design a flow to return the exit state from the server to the client, or a remote use will return the value of the exit state. As a result, if the client procedure can perform some processing (such as case of exit state) and determine what the server error was and determine how to handle it.

- You will need to turn off the error message (set it to "none") on the return exit state if you want the client procedure to interrogate the error before presenting the user with a message dialog box.
- Otherwise, if the exit state contains a message and it is error, warning, or informational, the window manager will capture that exit state and present the message to the user before it returns control to the procedure (regardless if it is Execute First or Display First).

The following list explains disadvantages of this technique:

- Distribution is a concern with this technique.
- Changing an exit state will require the client and server procedure that use that exit state be generated and redistributed. The reason is the exit state message is contained within the window manager as text. If the number of installed client locations is high, this distribution could be a challenge.

Use Exit States and an Action Block

The exit state and action block technique is similar to the flow or remote use and exit states technique.

The exit state and action block technique provides a robust error handling mechanism because the client can detect what has happened in the server. This mechanism can be very effective in all types of applications. It provides an application the ability to easily display messages by using the capabilities of CA Gen, and the technique lets a client procedure detect what has happened in the server.

With this technique, an error code is returned from the server to the client to indicate the results of the requested action. This is in addition to the exit state that causes the returns on flow or returning from the remote use.

The exit state is used to display the message. The returned error code is used by the client to execute action diagram logic based upon the server's results. This error code is view matched as part of the data returned on a flow or view matched with the remote use statement.

Setting exits states in the server procedure will communicate the message to the client workstation. In addition, the returns on property for the flow is defined as Execute First so that the returned error code can be evaluated.

An action block in the server can set the returned error code. The exit state set is evaluated and converted to an error code that is then returned to the client.

The conversion action block logic would consist of one large case of exit state statement with a case for every exit state. At the client procedure, the message is automatically displayed by the window manager (see the Using a Flow or Remote Use and Exit States section in this chapter). The error code returned is then evaluated in the client procedure to determine what processing may be required as a result of the server action.

The following list explains advantages of this technique:

- CA Gen handles the communication of the error messages, the display of the error messages, and the implementation of changes through code generation.
- The use of a return code enables the client procedure to determine what error has occurred in the server procedure.
- As a result, the client procedure can perform unique processing based on the code returned. This processing can include flowing to another window or marking a field in error.

The following list explains disadvantages of this technique:

- Distribution is a concern with this technique.
- If this technique is used every time an exit state is changed in the server, the clients that use that server must also be generated and redistributed.
- The maintenance of the exit state conversion action block could become difficult.
- Every time a new exit state is added the conversion action block would have to be updated and all servers would have to be reinstalled.

Use Server-Supplied Error Information

For complex applications, the server-supplied error information technique provides a robust technique to error handling. This technique eliminates the disadvantages of the other error handling techniques such as client distribution, maintenance, and lack of flexibility. The cost of these advantages is that the application must build and maintain the error message architecture. The type of application that may best use this approach is one with a large number of clients that may have messages change regularly.

With this technique, the server sends all the error information back to the client. This technique isolates the client procedures from changes in server errors. The error information may include the error code, the message text, and the message type (Error, Warning or Informational).

The server contains an action block that converts the exit state to a code and populates the other information. To populate the message and the type fields, the action block can use either set statements to assign literals, or read from a server-based table that contains that message information. The server table option provides the greatest flexibility because a message or message type can be changed without a logic change. For more information, see Create an Error Data Table.

After the message is assigned or read, it is returned using data returned on the flow or view matched with the remote use statement.

After the message information is assigned, the procedure sets one of two exit states to trigger the return flow to the client. One exit state is used when a rollback is necessary and the other when a rollback is not necessary. The exit state used for rollback is defined with the rollback property set.

The exit states should be defined with a message type of "none" to prevent any unwanted message displays.

If a flow is used for the procedure interaction, define the flow returns on property as Execute First.

At the client procedure, if a server message needs to be displayed (determined by evaluating the message type), the client will flow, using a link, to an error message display procedure. It will send the message using view matching on the flow.

All clients in the business system should reuse the error message procedure.

The following list explains advantages of this technique:

- The client is isolated from having to process server error messages.
- All the client does is check the error type code to determine if a message needs to be displayed and display the error message sent by the server. The server can add or change error messages without impacting the client. As a result the client procedure is easier to maintain.
- The distribution of data or load modules caused by message changes at the server are eliminated because all information is stored centrally.
- This makes management of the production environment simpler.

The following list explains disadvantages of this technique:

- Appearance of the message dialog box.
- CA Gen does not have the capability to display the error, warning, and informational bit map icons on the dialog box.
- You must develop all the logic to support server errors.

Create an Error Data Table

The server-supplied error information error handling technique uses database tables to find the error information to display.

The following table lists the attributes that will be represented on the ERD.

Attribute	Domain	Length
Error Code—Contains the error code for the exit state. This attribute is the identifier. It should contain the first 5 characters of the exit state name.	Text	5
Error Type—Contains the code to identify what is the type of the message. It will contain one of the following: ■ I (Informational) ■ W (Warning) ■ E (Error) This attribute should contain the same value as defined in the model for the exit state type property. It is used by the client to evaluate the severity of the error returned to determine if the message should be displayed. For example, a client procedure can suppress all informational messages that are received from the server.	Text	1
Error Action—Contains the code to identify the action for the server to perform when this error is encountered. The values for this attribute are either: ■ R (Rollback completion) ■ N (Normal completion) The server will use the code to determine which of the two returns on exit states to set. The value for this attribute should contain the same value as defined in the model for the exit state type property.	Text	1
Error Message—Contains the message that will be displayed to the end user. It should contain the same text that is defined on the exit state message property.	Text	80

Error Handling Technique Selection

The techniques described in this chapter can all provide effective error handling for client/server applications.

The technique you choose for an application will depend on the type of application being developed:

- For most applications you should use the flow or remote use and exit states technique. This technique provides the easiest and quickest way to handle client/server errors. You will want to select the remote use as your procedure interaction to allow more than the 8 exit states allowed on a return flow.
- If you need to use a flow for the client/server procedure interaction (perhaps you are putting error handling techniques in an existing system before the remote use was available), one of the other techniques can be effective.

Concerns that must be addressed are the cost of maintenance and production administration for the application system:

- The exit states and action block technique helps reduce the cost of development and maintenance because the code generator implements the exit states automatically, but it may provide some challenges for application maintenance and distribution.
- The Server Supplied Error Information technique provides some advantages in these areas because it does isolate the client from changes in server errors, but it requires the application design the error handling architecture.

Note: To position applications to take advantage of future CA Gen software releases, always use exit states and messages in the server procedure steps.

To choose an error handling technique, map the capabilities of each technique to the needs of the application's requirements and select the technique that best meets those needs.

You can modify the chosen technique to better meet the application's needs. For example, modifying a technique to return more than one error condition from the server to the client.

Standardize Error Codes

In several of the error handling techniques, an error code is used to communicate the server error to the client. This error code is directly related to exit states.

Maintaining this relationship is an important consideration to the assignment and management of the error codes. You should establish a naming convention for both exit states and error codes to enable this association.

An effective approach is to develop a code that will prefix all exit state names and represent the error code for that exit state.

The following table provides examples of a five-character code:

Exit State	Error Code
AA001_Customer_Not_Found	AA001
AA002_Customer_Add_Not_Successful	AA002
AA003_Customer_Type_Invalid	AA003

In this example, the error code is made up of two parts:

- The first two characters (AA) contain a unique code assigned to every business system.
- This code will ensure that exit state names and error codes are not duplicated.
- The last three characters (001, 002, 003) are a numeric sequence number.

  The business system is the boundary for this sequence number. Every time a new exit state is added to the business system the last number assigned is incremented by one and used to name the new exit state.

Chapter 9: Security

This chapter describes how to design security into client/server applications. The approach in this chapter is designed for an application that uses the distributed process model for client/server applications. Applications that use remote data access can apply some of the techniques with modifications.

Ensuring security of the client/server application is an important design consideration for all projects. Transaction environments are configured to use security when they require restricted access to their set of available transactions. Each transaction request received by a secured target server must contain security data such as a user ID and password. The target server uses the security data to grant execution access to users which it deems as authorized.

Client/server application security has two major components:

- Identify and verify the user at both the client and server

  The identification of the user includes capturing the security data of the person using the client software and providing the security data to the server. The security data is the user ID, password, and optionally a security token.

- Prevent unauthorized access to an application or its functions

  After the user is identified at the client and server, the access rights of that user must be determined and the application secured within the application itself.

  Securing a transaction and its functions is the application designer's responsibility. CA Gen verifies security in the server environment only.

User Identification

With CA Gen, security is a function of the distributed processing client, client manager, transaction environment, and distributed processing server software.

The identification of the user is a function of the client or client manager software.

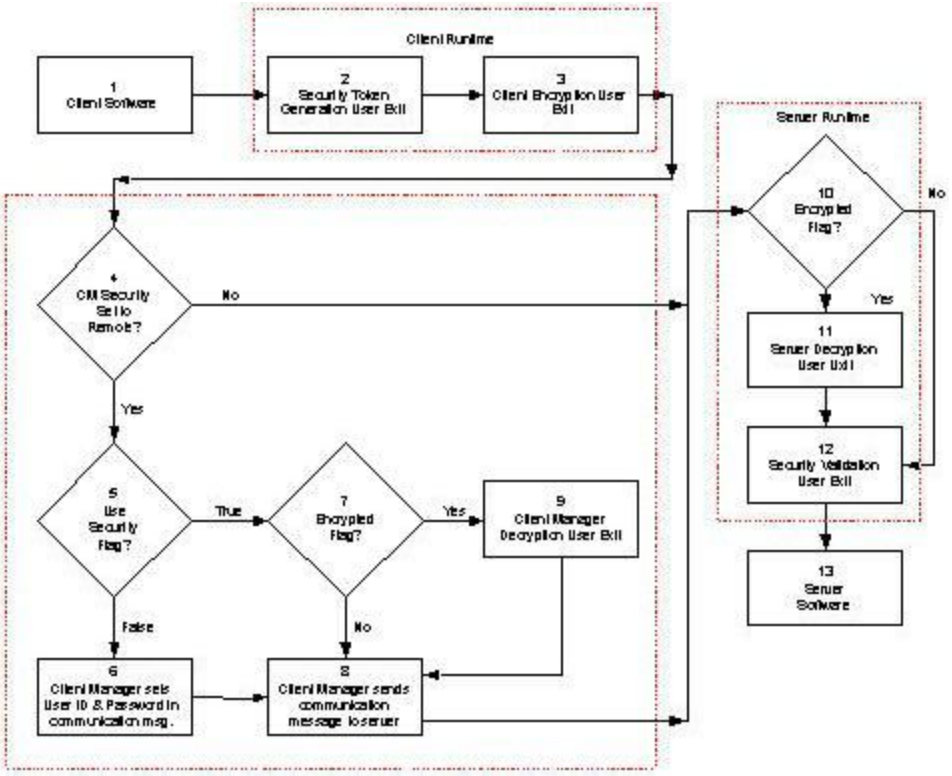
The user ID and password are both available in action diagramming statements. Therefore the user ID and password can be entered by the user in a graphical user interface and set in an action diagram statement or set in an action diagram statement by the system designer. The graphical user interface in this case is probably part of the user login process.

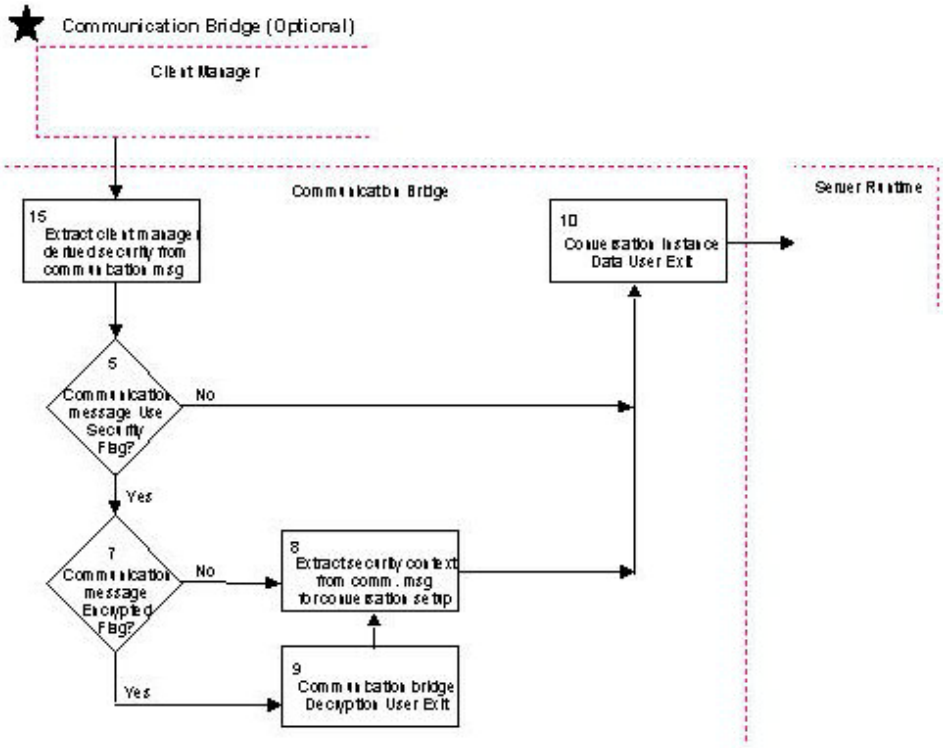
If the client software does not identify the user, it can be done at the client manager level through a login process or using the client manager configuration files.

The client manager has the responsibility to derive the security data to be used for each cooperative request. A cooperative request is the instance of the data flowing as a result of a remote use statement or a link flow.

Assuming the server transaction environment has enabled security, for example the asynchronous daemon is started with the -l option or a CICS connection is enabled for attach time security, then the target server uses the security data to grant access to users which it deems authorized. At the server, the security data is checked. How the check is performed varies by server environment.

The following illustration shows how security is implemented. The top portion of the illustration provides the implementation for a client-to-server procedure interaction and the middle portion of the figure shows the implementation for a server-to-client procedure interaction and the lower portion of the illustration shows the implementation for a communications bridge.





The following table provides descriptions of the parts of this illustration.

Key	Description
1	The client software is the code generated from the action diagramming statements in the client procedures. The user ID and password are both available in action diagramming statements. Therefore the user ID and password can be entered by the user in a graphical user interface and set in an action diagram statement or set in an action diagram statement by the system designer.
2	The Cooperative Runtime constructs the communication message and invokes the security token user exit. This security token user exit has the ability to influence whether the security data will be placed into the communication message. The security data can be supplemented by the generation of a security token using external security software such as adding Kerberos capability. The user exit allows you to enter the length of the token along with the value of the token itself.

Key	Description
3	Encryption is not required, but this user exit is invoked in the case you require encryption to be used. The client encryption user exit allows you to use your own or a third-party encryption capability. This routine encrypts the communication message from the client software to the server software. If the user exit return value indicates that encryption was used, the runtime will set a flag in the communication message indicating the communication message is encrypted. This allows the client manager and server runtime to know that the communication message is encrypted. **Note:** This implies that you will need to use a decryption routine that will decrypt according to the encryption routine used.
4	Each target server configured in the Client Manager has an associates security level which can be set to remote, defer, or none. The default security level in the Client Manager can be set to none or remote. For more information, see the *Distributed Processing - Overview Guide* and the *Distributed Processing - Client Manager Guide*. A security level associated with a target server can be set to remote in one of two ways: ■ The target server security level is set to remote. ■ The target server security level is set to defer and the Client Manager default security level is remote. In both of these cases, the target server security will be derived to be remote.
5	The client manager needs to determine where to obtain the security data to be used for conversation start up. If you set the flag in the security token user exit to: ■ BOOLEAN YES—The security data will be specified by the client software and extracted by the client manager ■ BOOLEAN NO—The security data will be provided by the client manager's configuration data
6	Derive the user ID and password for the target server that will be used in the conversation setup. At this point the security level of the target server has been derived as remote, the user ID and password will first be assigned from the target server information. If both the user ID and password are blank in the target server, then the conversation setup security information is obtained from the client manager defaults. If the user ID and/or password are defined in the target server, then the conversation setup security information is obtained from the target server. If both the user ID and password are blank in the target server and blank in the client manager, then blanks will be used for both the user ID and password in the conversation setup security information.

Key	Description
7	The client manager checks to see if the security information from the client software is encrypted. This flag was set by the cooperative runtime based on information returned from the client encryption user exit. If the flag is set to: ■ BOOLEAN YES—The communication message is encrypted ■ BOOLEAN NO—The communication message is not encrypted
8	Extract the security data information out of the communication message (as opposed to deriving the information from the client manager configuration - see #6).
9	This user exit allows for the use of your own or a third-party decryption capability to decrypt the communication message. **Note:** A key point here is that the decryption routine used for the client manager and the communication bridge (if used) must decrypt the information correctly from the encryption routine used in client encryption user exit.
10	The conversation instance data user exit provides the opportunity to interrogate, use, or modify the selected user ID and/or password that will be used by the conversational transport LU6.2. Example: Translate a lower case password to upper case. The user exit can be enabled or disabled. This user exit is called during the initialization of the client manager and communication bridge (if used). If the user exit returns disabled, it will not be invoked as long as the client manager (or communication bridge) is running. If the user exit returns enabled, it will be invoked for each conversation. **Note:** Any modifications made to the user ID and/or password in this user exit do not change the information in the communication message. The extracted, and possibly modified, data is for use only by the target server's associated transport layer.
11	The server runtime checks the communication message to see if the flag is set indicating whether the communication message is encrypted or not. If the communication message is encrypted, the server decryption user exit is invoked.
12	The server decryption user exit allows you to use your own or a third-party decryption capability. This routine decrypts the communication message from the client software. **Note:** A key point here is that the decryption routine used on the server must decrypt the information correctly from the encryption routine used in the cooperative runtime.
13	This user exit allows you to use your own or a third-party security package in the server environments.

Key	Description
14	The server software is the code generated from the action diagramming statements in the server procedures.
15	Extract the client manager derived security data from the communication message. See #6 for how the security data was derived in the client manager.

The following table provides security level, user ID, and password examples for target server, client manager, and derived security data.

	Target Server	Client Manager	Derived Security Data
Security level	remote	none or remote	
User ID	good-bye	hello	good-bye
Password	dolly	world	dolly
Security level	remote	none or remote	
User ID	good-bye	hello	good-bye
Password	<blank>	world	<blank>
Security level	remote	none or remote	
User ID	<blank>	hello	<blank>
Password	dolly	world	dolly
Security level	remote	none or remote	
User ID	<blank>	hello	hello
Password	<blank>	world	world
Security level	remote	none or remote	
User ID	<blank>	<blank>	<blank>
Password	<blank>	<blank>	<blank>
Security level	remote	none or remote	
User ID	<blank>	hello	hello
Password	<blank>	<blank>	<blank>
Security level	remote	none or remote	
User ID	<blank>	<blank>	<blank>
Password	<blank>	world	world
Security level	defer	remote	
User ID	good-bye	hello	good-bye
Password	dolly	world	dolly
Security level	defer	remote	
User ID	good-bye	hello	good-bye
Password	<blank>	world	<blank>

	Target Server	Client Manager	Derived Security Data
Security level	defer	remote	
User ID	<blank>	hello	<blank>
Password	dolly	world	dolly
Security level	defer	remote	
User ID	<blank>	hello	hello
Password	<blank>	world	world
Security level	defer	remote	
User ID	<blank>	<blank>	<blank>
Password	<blank>	<blank>	<blank>
Security level	defer	remote	
User ID	<blank>	hello	hello
Password	<blank>	<blank>	<blank>
Security level	defer	remote	
User ID	<blank>	<blank>	<blank>
Password	<blank>	world	world

How to Prevent Unauthorized Access in Applications

Application security includes securing the transaction and all functions within a transaction:

- Transaction security denies or permits user access to one or more transactions.

  For example, transaction security prevents a non-manager from accessing the maintain payroll transaction.

- Transaction security is typically a requirement of all application systems.
- Functional security determines a user's access based on the transaction function being attempted.

  For example, functional security allows a manager to access the maintain payroll transaction but will prevent a non-finance manager from approving the payroll.

- Functional level security is less common.

Design Transaction Security

You can design transaction security in one of the following ways:

- Use an external security package
- Build logic

Use an External Security Package

Wherever possible design transaction security without application programming because it is easier to implement and change.

This type of security can be done by using an external security package, such as RACF or ACF/2, that is automatically invoked by the teleprocessing monitor when a transaction executes.

When the user requests the execution of a transaction the external security package should automatically validate that the logged on user ID can execute that transaction. If this check fails, access to the transaction is denied, and no application logic is executed.

Using CICS as the server environment, this automatic checking is achieved easily and is completely external to the application. Transaction security for servers is set up and checked in the same manner as block mode applications.

Note: For more information about these security issues, see the *Host Encyclopedia Construction Guide*.

Code Security Logic

In environments without a teleprocessing monitor, such as UNIX and Linux, application logic must be coded for transaction security. To build logic for transaction security, the first thing a server must do is validate a user's authorization.

There are two ways an application can validate transaction security:

- Automatic transaction security checking can be simulated by using CA Gen TIRSECR user exit.
  - The server manager automatically calls this exit, and it can be customized to verify that the user who is logged on can execute the transaction. Both the user ID and trancode are supplied as input to the exit.
  - After the TIRSECR exit completes, the server manager checks the return code exported.
  - A space in this code indicates the user has authority to execute the transaction.
  - A non-blank value indicates a security violation.
  - The exit also exports a message field, a COBOL (z/OS only) or C variable that contains the error message to be displayed to the user. If a security violation is detected by the server manager (a non-blank value in the return code field), CA Gen abnormal termination routines are invoked, and the message exported by the TIRSECR exit is displayed to the user.
  - The TIRSECR routine is automatically linked into every load module during installation.

- The COBOL (z/OS only) or C code skeletons for the TIRSECR user exit routine are supplied with the implementation toolset. Any code can be placed in the exit routine to validate security. However, the code added to the skeletons must be written in COBOL (z/OS only) or C. Typically the code does the following:
- A read of a file or database is done to determine if the user has access to the transaction.
- If the user does not have authority the return code and message fields are set. These fields can be set to any value. A non-blank character in the return code causes the server manager to stop the execution of the transaction and display the exported message field to the user.
- If the user does have authority to execute the transaction, the return code and message are blanked.
- Calls to CA Gen-generated action blocks, such as the Security Template action block, can be placed within this exit. However, the calls to these action blocks must be written in COBOL (z/OS only) or C, not in CA Gen action diagram logic.
- Modifications to the TIRSECR user exit can be made once. All transactions using the Implementation Toolset automatically link in the modified exit.
- If you use the TIRSECR user exit is used, you do not have to build any action diagram logic for security. All transactions receive the security logic as part of the installation of the load module. As a result, transaction security is obtained in a way similar to using an external security package.

■ Transaction security can also be accomplished by invoking a common or external action block within the procedure.

- This action block call should be one of the first statements in the procedure.
- The called/used action block first needs to import the user ID and transaction code. Then the logic should read a file or a database table to determine the transaction authority for the imported information:
- If the user has the appropriate transaction authority, the action diagram returns an exit state or return code that allows further processing.
- If a user does not have the appropriate transaction authority, the execution of the transaction is terminated, and a message is returned to the client procedure.

This approach is similar to the way CA Gen Security Template implements security.

By placing the transaction authorization check within the procedure, an application has control over what information is available to check authorization and what information is returned from the check. Using the TIRSECR user exit, the information imported and exported is fixed and cannot be changed. An example of some additional information that may be required is a password. If the information imported and exported by the TIRSECR user exit is not sufficient to meet the application's security requirements, then using a common or external action block may be necessary. The disadvantage of this approach is that transaction security is not automatic, and application logic must be coded in every procedure. This disadvantage makes maintenance of security logic costlier.

Note: For information about the TIRSECR exit, see the *User Exit Reference Guide*.

Design Functional Security

Functional security restricts access to application functions within a procedure. The application must provide all logic to implement functional security.

The approach to implementing functional security in a client/server environment is similar to the common action block option for implementing application coded transaction security. For more information, see Coding Security Logic.

The functional security check should occur early in the procedure by using a common or external action block, preferably one of the first statements.

The called or used action block first needs to import the user ID, transaction code, and function requested. Then the logic should read a file or a database table to determine the transaction authority for the imported information:

- If a user does not have the appropriate functional authority, the execution of the transaction will be terminated and a message will be returned to the client procedure.
- If the user does have the appropriate functional authority, the action diagram returns an exit state or return code that allows further processing.

This approach is similar to how the CA Gen Security Template implements security.

Functional security can be enhanced in the client/server environment by enabling or disabling controls in the CA Gen GUI designer. Disabled controls on a GUI window can prevent a user from selecting an item to which they do not have access. By extending the functional security to the client window, the security integrates effectively with the GUI interface, and the application's usability increases.

For example, a customer information list procedure contains two functions, detail the customer information and view the customer credit history. Selecting a push button starts both functions.

Access to view customer credit history is limited to loan officers only. When a loan officer opens this window, both push buttons can be selected. When any other user opens the window the detail customer information push button is enabled, but the view credit history is disabled. The window manager prevents the non-loan officer user from selecting the push button.

The disabling can also proactively prevent unauthorized use. If the push button were not disabled, the non-loan officer user could select the view credit history button but be notified of a security violation after the selection.

The type of functional security described in this example can be implemented in all client/server applications. The security information used for disabling is retrieved from the server. The GUI controls are disabled based upon that information.

Follow these steps:

1. On all GUI windows that have controls (such as push buttons and menu items) and require security, create an import and export work attribute view for each control to be secured.

   The views must contain a 1-byte attribute for each control that will be disabled. The attribute will hold either a Y or a space:

   - The Y indicates that the user has access to the control and can perform the function represented by the control.
   - A space indicates the user does not have access to the control and cannot perform the function.

   The flag attribute will be populated by the server procedure via an export view. Using the customer information list example, the secured control is the view customer credit history push button.

   The import and export views to secure the push button are:

   - IMPORT SECURE
   - CREDIT_HISTORY_FLAG

     and

   - EXPORT SECURE

     CREDIT_HISTORY_FLAG

2. Place the flags on the window so that the Disabled By feature of the window painter can be used to disable the controls if necessary.

   Hide the flags from the user by one of these methods:

   - Extending the window or dialog box beyond its normal size
   - Adding a read-only attribute in an extreme corner of the window

- Resizing the attribute to the smallest size
- Returning the window to its normal size

In the customer information list example, the EXPORT SECURE CREDIT_HISTORY_FLAG will be placed on the window (see Step 1).

3. After the flag is placed on the window, use the Disabled By feature of the GUI painter to detail each control that requires disabling. Set each control to be disabled when the appropriate attribute flag does not have data.

   It is important that a space in the flag attribute indicate no access because when a window is initiated all functions will be disabled. If Y indicated no access then when the window is initiated, then all functions would be enabled. Continuing with the customer information list procedure example, the view customer credit history push button would be disabled in the following way:

   ```
   THE PUSH BUTTON 'View Credit' IS DISABLED WHEN

   ENTRY FIELD 'Credit_History_Flag' DOES NOT HAVE DATA
   ```

   Enable and disable the controls by defining each client procedure as Execute First. This property should be set for all procedure interactions and the clear screen input property of the procedure.

   The client procedure action diagram should have logic for the initial execution of the client that is sent to the server procedure to determine what functions the user can perform. This initial check can occur either by itself or bundled with another service request.

   The following sample code shows example action diagram logic for retrieving security information for the function Security Check.

   ```
    ---
   |    CASE OF COMMAND
   |     ---
   |    |    CASE firstime
   |    |
   |    |--- NOTE*****************************
   |    |    Initial execution
   |    |    Get function Security Information
   |    |    ********************************
   |    |    COMMAND IS security
   |    |    USE PSTEP server (or EXIT STATE IS link_to_server)
   |    |    CASE grafdel
   |     ---
    ---
   ```

4. At the server, a service must be provided that accepts as input the transaction code of the client procedure and the logged on user ID. It should return to the client procedure the access flags that hold the users authority. The following sample code is the action diagram logic that is contained within the server for the functional security check.

   The following sample code shows an example action block for checking security authorizations for the function Security Check.

```
 ---
|    CASE OF COMMAND
|     ---
|    |    CASE security
|    |
|    |    NOTE**********************************
|    |    Check Security Authorizations for user
|    |
|    |    USE check_security
|    |    WHICH IMPORTS: Work View import client_procedure_step
|    |    WHICH EXPORTS: Work View export secure
|    |
|    |    CASE initial
|     ---
 ---
```

5. Functional security should be re-validated at the server with each request for a secured function to assure integrity.

Security Approach Selection

The security approaches described in this chapter will provide sufficient transaction and functional security for most application systems.

These approaches can be enhanced to meet additional security requirements such as data or sub-functional security. For these additional requirements, you will have to develop application logic that functions similarly to the functional security technique.

Using the approaches outlined above, security for client/server applications becomes a server function.

The following list explains advantages of locating security within the server:

- All the authorization data and the security logic are located in one location. This enables security changes to be implemented more easily.
- Security functions are located in a more secure environment than on a user's workstations. This provides a higher degree of integrity.

By integrating the security approaches in this chapter with the capabilities of CA Gen, you can achieve effective transaction and functional security for an application. As a result, a client/server applications can be secured as effectively as any block mode application.

In addition, by integrating GUI design capabilities, you can significantly improve an application's usability.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W