CA Gen

Design Guide

Release 8.5

CA Technologies Product References

This document references the following CA Technologies products:

- CA Gen
- COOL: 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 7: Designing Windows 69

Chapter 8: Designing Screens 87

Appendix A: z/OS Block Mode Screen Design 125

Chapter 1: Introduction

This section contains the following topics:

Design Basics

Design is the process of defining an information system that meets the needs of the users. The human interface is considered as well as the characteristics of the specific environments in which the system runs.

In CA Gen, Design concentrates on selected areas of the entire system. These selected areas, called *business systems*, let you design a manageable portion of a system at a time. A business system is a group of related procedures. Each procedure can implement zero, one, or more elementary processes. An elementary process defines what must be done to support the business activities. A procedure defines how it is done. Elementary processes are intended to be independent of procedures and can be reused in different types of procedures, such as batch, terminal-based, GUI, and client/server.

To support the procedures, you may find it necessary to design other procedures, provide system or operational controls to convert data, and so on. The activities during system design, therefore, tend to be iterative. You design part of the system, such as the interface, see if it meets the needs of the users, and refine the design as needed. After your design meets the needs of the users, you can use that design in subsequent phases of development, such as Construction.

Prerequisites

Design can begin after you have a partial or complete Data Model and a business system defined. Other information, however, can be useful for design.

The following list offers one starting place of prerequisites to design. The list does not represent a sequence; it shows only a set of prerequisites that can be satisfied in different sequences.

- A Data Model, a partial or complete Entity Relationship Diagram
- Activity model, a partial or complete Process Hierarchy Diagram
- Interaction model, a partial or complete set of Process Action Diagrams

- Implementation plan, a partial or complete plan determined through matrix clustering and affinity analysis
- Business system definition, defined through Business System Definition

Several development life cycles use information-engineering techniques. You can prefer a different set of prerequisites or set of concurrent requisites. For a discussion of the alternative life cycles that use information engineering techniques, see the *Client* Server *Design Guide*.

Objectives

The major objective of system design is to develop the system as it appears to the users (the external aspects of the system). The external design of the system must be suitable for the proposed business users, locations, and technology. Detailed technical design issues, however, are normally deferred until the users agree with the functionality of the interface design. The interface design must first satisfy the needs of the users in its form and its suitability to meet the business objectives.

Other objectives of design or constraints on the form of the system may also exist. For example, an objective may be that the new system development exploit and protect other system development investment. This may involve designing procedures to make the best use of current (sometimes called legacy) systems and data. The design may also have to work with some other system that is currently planned or under parallel development.

After an overall design is agreed upon, the results of the design work can supply subsequent phases of development, such as Construction.

Tasks Performed During Design

Design requires you to perform several major tasks, each of which is described in detail in the *Client Server Design Guide*.

The tasks in the following table assume that you have already prepared for design by doing the following tasks:

- Assembling a design team
- Reviewing the results of previous phases of a life cycle
- Establishing design standards

The order of the tasks in the table does not imply the only sequence in which you can perform the tasks. In the actual design work, it is common to perform many tasks concurrently. The following table lists the major tasks performed during design.

Task	Purpose
Choose an application style	Map the tasks of the users to a type of application: Graphical User Interface (GUI) Client/server Terminal based (block mode) Batch (Batch is only available on MVS)
Design the dialog	Specify the procedures and the dialog flows used to navigate the system.
Design the interface	Build the screens, windows, and dialogs (the external interface).
Design the procedure logic	Specify the logic for each procedure.
Design the data structure	Transform the logical data model into a physical representation of the database.

After completing the major tasks, you can complete the design by verifying with the application users the consistency and completeness of the design.

Deliverables

After you complete design, you have the following set of deliverables to use for subsequent phases of development:

- Interface layouts that implement the data views for a particular procedure developed with the Screen Design tool and the Navigation Diagram tool
- A set of procedural logic programs and data views, called Procedure Action Diagrams
- A diagram of links and transfers among procedures based on commands and states that trigger these flows, called Dialog Flow Diagram
- A design for the physical structure of the database developed with the Data Structure List and, depending on the database management system, the Data Store List
- A set of specifications determined from the technical design to use during subsequent phases of development, such as Construction

In addition, you may need to update deliverables from previous phases of development. For example, you may discover during design that previous information is invalid or additional requirements need to be analyzed.

After You Complete Design

Typically, the tasks you need to complete after design involve generating and installing your application, or parts of the application. Installing the application lets you test the application and verify the design and functionality with users.

Note: For more information about generation and installation, see the *Workstation Construction User Guide.*

Chapter 2: Designing a Graphical User Interface

This section contains the following topics:

Graphical User Interface

A Graphical User Interface (GUI) is the type of interface used by GUI applications and by different styles of client/server applications (such as the distributed process style).

You create a GUI with the Window Design functionality within the Navigation Diagram tool.

More information:

Designing Windows (see page 69)
Client/Server Application Design Basics (see page 29)

Prerequisites

Before you begin designing a GUI, you should have either completed or have an understanding of certain tasks. However, the specific task list can vary depending on the information engineering techniques of the life cycle being used.

The following list offers a starting place of prerequisites. As you become more familiar with designing GUIs, or select a life cycle that works best for your needs, you may prefer a different set of prerequisites. For more information about the alternative life cycles that use information engineering techniques, see the *Client Server Design Guide*.

- An understanding of the tasks that the user performs with the application, which is determined from previous phases of a life cycle or by interviewing users as you design the interface
- A partial or complete data model defined with one of the Data Model tools
- One or more procedures defined with the Dialog Design tool

- Views for one or more procedures defined with the View Maintenance tool or the Action Diagram tool
- A business system defined with the Business System Definition tool

GUIs and User-Centered Design

As you design a GUI, you should consider some of the changes required for the interface user. These considerations apply to standalone GUI applications and client/server applications. A good place to start is by considering the interfaces for environments that are *not* graphical. In traditional online transaction processing, for example, a user interacts with the system using a terminal-based screen on dumb terminals. The system drives the dialog of the user and restricts options by what the menus provide. The work pattern for the user is one of procedural tasks controlled by the system.

As a designer, you are now faced with a change in the work patterns of users from procedural tasks to information gathering tasks. Users now expect greater control of application use and the sequence in which they perform application tasks. In short, users want applications, especially the interface, to empower them to better complete their work.

Applications must address this change in work patterns and offer a user-centered design for the interface. The interface should provide a flexible user dialog with a choice of activities. To the greatest extent possible, the user must be in control. The application becomes one of a set of tools by which the user accomplishes work.

The application user tends to view the entire desktop as a collection of tools. All of the applications work together, or should from the perspective of the user, to provide an integrated solution to information gathering tasks. For example, data could be shared among a CA Gen Client Server application, a word processor, and a spreadsheet. This type of integration represents what users want, and will require, in desktop environments.

How You Create a GUI

To create a GUI design, you have some preliminary decisions to make concerning the interface. CA Gen tools can often assist you as you make these decisions, but, overall, many of the preliminary decisions can be made independently of the tools.

For example, as you decide on the overall form and function of the interface, you would typically address tasks such as shown next:

- Determine the flow between windows and dialogs that corresponds to the order in which the user is most likely to use the application.
- Decide on any interface standards for the application, which could include decisions about:
  - When to use tool bars and status bars
  - The bitmap images to use as objects
  - The font and the foreground and background colors for windows, dialogs, and controls

After you have a preliminary design, you perform several major tasks with the Window Design functionality. Each of these major tasks has subtasks associated with it. The major tasks tend to group into the following broad categories:

- Tasks to create windows and menu items
- Tasks to create controls and specify control characteristics (controls are components that allow selection of choices, entry of information, or both)
- Tasks to fine tune the interface

The major tasks tend to be iterative. For example, since one procedure may have many windows associated with it, you would iterate the tasks of specifying the characteristics for each window, the characteristics for each control, and so forth.

You can follow the major tasks in sequence to create a GUI interface. This is not the only sequence in which you could build a GUI. The major tasks show one sequence that can guide you as you create your application. As you become more familiar with the Window Design functionality, you may prefer a different sequence.

The following table lists the tasks to create windows and menu items.

Task	Primary Subtasks
Add a window	Choose a procedure or procedure step with which you want to associate a window or dialog.

Task	Primary Subtasks
Specify window characteristics	Name the window. Specify if the window is the primary one with which the user interacts. Specify where on the desktop the window appears when first invoked. Specify if the window can remain open while the user performs other tasks. Specify if the window uses a bitmap for a background.
Add menu items to a window	Name the menu options that appear in the menu bar and menu bar pull downs. Determine the arrangement (hierarchy) of the menu options. Specify font and colors for menu items.
Specify menu item characteristics	Specify the text for the menu item. Specify the result that occurs when the user selects the menu item. Specify the command, if any, associated with the menu item.

The following table lists the tasks to create controls and specify control characteristics.

Task	Primary Subtasks
Add controls that implement views of attributes	Select attribute views. Specify the type of control. Specify prompts and edit patterns. Place the control on a window.
Add controls that do not implement views of attributes	Specify the result that occurs when the user selects the control. Specify the text for the control. Place the control on a window.
Specify control characteristics	Map import and export views. Specify the text for the control. Specify prompts and edit patterns. Specify behavioral results of the user interacting with the control. Behavioral results also includes event processing, which is listed as a separate major task because of the amount of information associated with events.

Task	Primary Subtasks
Add events	Add an event type. Create a name for the event action. Specify the action diagram logic for the event action. This subtask is performed with the Action Diagram tool.

The following table lists tasks to fine tune the interface.

Task	Primary Subtasks
Test the flow of user tasks from window to window	Decide which windows open or close which other windows. Test the flow from window to window.
Refine the overall appearance of the interface	Verify the implementation of any application-wide standards. Verify the use of tool bars and status bars. Rearrange the controls in a manner that satisfies the object/action flow in which a user performs tasks. Group related controls together. Disable controls where appropriate. Create or update any bitmap images to use as objects. Verify the choices for the font and the foreground and background colors for windows, dialogs, and controls.

Chapter 3: Designing a Terminal-Based Interface

This section contains the following topics:

Terminal-Based Interface Function

In an online environment, the primary visual layouts are screens and windows. A window is one of the graphical objects used by the type of interface called graphical user interface. GUIs provide great flexibility and user-controlled actions. A screen is a terminal-based interface, one in which the user interacts with the system using a dumb terminal or a PC with terminal emulation. The system drives the dialog of the user and restricts options to what the menus provide. The work pattern for the user is basically system-controlled procedural tasks.

GUI-based applications let the user *drive* the screen. Terminal-based interfaces offer the user predetermined routes and restrict input.

Each screen provides the user interface for one procedure step or single-step procedure. When you design a screen, you are creating a visual layout of the import and export views of a specific procedure step that was defined using the Dialog Design tool.

You create a screen using the Screen Design tool.

You design screens only for online procedure steps. You do not design screens for batch procedure steps. The primary visual layout in a batch environment is not a screen or window, but rather a report.

Note: Batch is only available on MVS.

More information:

Designing Screens (see page 87)
Designing Windows (see page 69)
Designing Dialogs (see page 49)

Design Objectives

When you design terminal-based interfaces using the Screen Design tool, your objectives are as shown next:

- Create screens that are easy to read and consistent across the business system
- Make the steps the user must follow clear and easy to implement

Prerequisites

Before you begin designing screens, you should have either completed or have an understanding of certain tasks. However, the specific task list and its contents can vary, depending on the information engineering techniques of the life cycle being used.

As you begin your screen design tasks, you can use the list of prerequisite tasks that follows to help you get started. As you gain experience in designing block mode screens for your application, you may choose to use this list as it is, delete some of these tasks, or add others.

This getting started list consists of the following tasks:

- An understanding of the tasks that the user intends to perform with the application, which is usually determined from previous phases of development, such as life cycle analysis using the Entity Life Cycle Analysis tool
- A data model defined with the Data Model tool, Data Model List tool, or the Data Model Browser tool
- One or more procedures defined with the Dialog Design tool
- Views for one or more procedures defined in procedure synthesis using the Dialog Design tool
- A business system defined with the Business System Definition tool

Types of Screen Objects

There are two types of objects you can define during screen design:

- Screens
- Screen templates

A screen is the view of the system for the user. Each online procedure step created in dialog design is associated with one screen.

A screen template is a definition of a part of a screen, typically a part that you may want repeated in the same position on many or all screens such as the name of your business. Because it can be used repeatedly, a template saves design time. It also gives screens uniformity.

Design Components in Screens and Templates

Screens contain the following design components:

- field
- literal
- prompt
- special field
- template

Templates contain the following design components:

- literal
- prompt
- special field

Fields are specific to a particular procedure step, since they are mapped to the attribute view of a procedure step. Since templates can be shared by many screens (and therefore, many procedure steps), templates do not support fields.

Screen Design Components

The following list describes the functions of the major components that you create and modify in the process of designing a screen:

- A field implements import and export data views. A single screen associated with each procedure step is responsible for providing data to its import view and displaying data from its export view. As a designer, you can use each field on the screen to implement both an import and an export view of an attribute.

  Note: Before you begin to add fields to screens, be sure you understand the relationship between screen fields and import and export data views.

- A literal is information that does not change such as the title appearing at the top of a screen. Presented as text, number, or symbol, a literal represents itself rather than a reference to something else.

- A prompt is a label for a field or a special field. A prompt is placed close to the field with which it is associated.

- A special field reflects a special attribute that is not derived from business requirements but is introduced during system design for a specific purpose. Examples of special fields are user IDs, timestamps, transaction codes, and system error messages.
- A template can also be part of a screen. It can consist of literals, prompts, and special fields.

How You Create a Terminal-Based Interface

To create screens or templates, you use the Screen Design option of the Design tool to perform the necessary tasks and subtasks. The tasks can be divided into the following broad categories:

- Create templates and screens step by step
- Create screens automatically
- Fine tune the interface

Note: If you plan to use templates, build them first. Then build the screens in which you plan to use them.

The following tables illustrate various tasks and subtasks.

The following table lists the tasks to create templates step by step.

Step	Task	Subtasks to Consider
1.	Open a new template	Define the template name. In the Description field, define how and where this template is used throughout the business system.
2.	Add a literal	Determine whether to use defaults for properties. Add literal properties. Determine positioning.

Step	Task	Subtasks to Consider
3.	Add a special field	Determine whether to use defaults for properties. Add a prompt. Add prompt properties. Determine field display length. Add field display properties. Add error display properties. Determine fill character. Define edit pattern. Add Help ID. Determine positioning.
4.	Copy a literal	Determine positioning.

The following table lists the tasks to create screens step by step.

Step	Task	Subtasks to Consider
1.	Use a template in a screen	Delete any unwanted literals or special fields contained in template.
2.	Add a field	Define import view. Add a prompt. Add prompt properties. Determine field display length. Add field display properties. Add error display properties. Determine fill character. Define edit pattern. Add Help ID. Determine positioning.
3.	Add a literal	Determine whether to use defaults for properties. Add literal properties. Determine positioning.

Step	Task	Subtasks to Consider
4.	Add a special field	Determine whether to use defaults for properties. Add a prompt. Add prompt properties. Determine field display length. Add field display properties. Add error display properties. Determine fill character. Define edit pattern. Add Help ID. Determine positioning.
5.	Copy a literal or repeating group	Determine positioning.
6.	Describe a screen	Include descriptions of source and destination procedure steps.
7.	Specify screen properties	Define screen name (changing the name changes the name of the associated procedure step and action block). Define Help ID. Determine whether to enable scrolling. Define whether unused occurrences are protected or unprotected. Define top on display-first return. Describe screen.
8.	View in different display modes	None.
9.	Run prototyping	None.

The following table lists the task to create a screen automatically.

Task	Subtasks to Consider
Create a screen automatically	None.

The following table lists the tasks to fine tune an interface.

Task	Screens	Templates	Subtasks to Consider
Assign commands to function keys	x	x	Define whether message type is Standard or Default. Check reserved command names before assigning a command name.
Access view maintenance	x		Define whether any attributes should be deleted.
Center a screen object	x	x	None.
Change the name of the screen or template	x	x	None.
Delete an object	x	x	None.
Delete an unused prompt	x	x	None.
Specify whether to display function keys and commands on generated screens	x		None.
Move an object	x	x	None.
Redefine an object	x	x	None.
Redefine a literal	x	x	Reposition literal after changing text.

Chapter 4: Designing Client/Server Applications

This section contains the following topics:

Client/Server Application Design Basics

When you create applications, the design decisions you make determine, in part, the tasks that you perform. For client/server applications, the design decisions are often numerous, which results in numerous tasks. For more information about the tasks for creating client/server applications with CA Gen, see the *Client Server Design* Guide.

Types of Logic

For purposes of distribution, it is possible to divide application software into three distinct types of logic:

- Presentation, which is associated with the user interface
- Business rule, which is associated with accomplishing a basic business process
- Data manipulation, which is associated with reading and writing persistent data

Presentation logic is associated with a procedure step definition. Business rule logic and data manipulation logic are generally found in elementary processes or process action blocks. In some applications, little or no business logic exists except for that directly governing data.

In simple client/server applications, these three types of logic can be divided into two components executing on different machines, one for presentation and one for data manipulation. Business rule logic is generally embedded in whichever component it appears to make sense, or it may be segmented into reusable action blocks. One reason for putting business rule logic in a reusable component is that you can easily generate and install the component for a different platform.

You do not have to decide what processing logic is required to support network communications. CA Gen for client/server products includes a communications runtime for you. For more information about the communications runtime components, see the *Distributed Processing - Overview Guide*.

The important task to remember is to design client/server applications to be modular and flexible. Maintain a clear distinction between presentation logic, business rule logic, and data manipulation logic so that you can change between client/server styles as needed. Maintain data manipulation logic as separate action diagrams or common action blocks to be used by server procedures. This approach also allows many client procedures to use the server procedures.

Presentation Logic

The client component of a client/server application requires a graphical user interface. For information about the special action statements for GUI applications, see the *Action Diagram User Guide*.

More information:

Designing Windows (see page 69)

Business Rule Logic

A major design consideration for the processing logic is *what* logic is required to fully support the tasks performed by the user. After you determine this, you can decide if the logic should be distributed or not. If the logic is distributed, you have considerations of which platform, client or server, is best suited for the particular functionality.

Data Manipulation Logic

A key design consideration concerning data is how important is access to *current* data. Depending on the application and business needs, data that is updated periodically, instead of continually, may be adequate. Data that is replicated on the client platform is often used only for read access (data look up). If other actions are required, such as create, update, and delete, the database management system (DBMS) must have the capability to handle the data integrity, node directories, and two-phase commits. Typically, the DBMS provides these capabilities and not the generated application. You can, however, build with CA Gen the time stamping and locking logic in your application, but you must also design for various failure conditions.

With read-only replicated data, you have several design decisions:

- How to distribute the data
- How to update the replicated data
- How often to update the replicated data and ensure that each client machine has the same data
- How to ensure consistency and synchronization of data after a component fails

You provide these capabilities through the processing logic in your procedure action diagrams (PrADs), the procedure steps on the Dialog Flow Diagram (DLG), batch transfer of data, and office procedures.

Design Alternatives

Dividing the logic also gives rise to three basic design alternatives for client/server:

- remote presentation
- distributed process
- remote data access

These design alternatives are discussed in detail in the *Client Server Design* Guide.

Distributed Process and CA Gen Toolsets

Most of your design work for a distributed process application occurs with the following tools in the Design Toolset:

- Navigation Diagram tool
- Dialog Design tool
- Action Diagram tool

You also have some considerations when you use the Construction Toolset.

Navigation Diagram Tool

The user interface for a distributed process application is a graphical user interface. Only the client component contains an interface. To design a GUI interface, see the chapter "Designing Windows."

Dialog Design Tool

Processing logic is often divided between hardware platforms. This lets you best use the processing power of each platform and to limit the quantity of data transferred over a network. This division requires separate procedures for the client and server components. The dialog flow between client and server procedures can pass commands and data as in any other dialog flow.

Identify Client and Server Procedures

After you identify the initial procedures on the DLG, you need to decompose these into client and server procedures (called client-to-server mapping). This mapping can take several forms, as the following bullets show. The last three bullets correspond to a style of client/server applications called the distributed process style.

- One Client Procedure: No Server Procedures

  This mapping assumes that all processing is performed on the client, with data stored remotely on a server or locally on a client, and accessed directly from the client procedure using a DBMS to obtain data from the server. The advantage of this mapping is that all the processing is in one place and maintenance is easier.

- One Client Procedure: One Server Procedure

  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.

  You can conveniently define this mapping as two procedure *steps* of the same procedure. The client procedure is the first step so that other client procedures can flow to it. Using two procedure steps of the same procedure, however, keeps other client procedures from accessing the server procedure step. The flow can go only to the first step of the multiple-step procedure.

- One Client Procedure: Many Server Procedures

  This mapping implies a multifunctional client procedure being serviced by many server procedures. This makes the server procedures relatively simple and individually easy to maintain. However, the client procedure may become more complex and the DLG is more complicated.

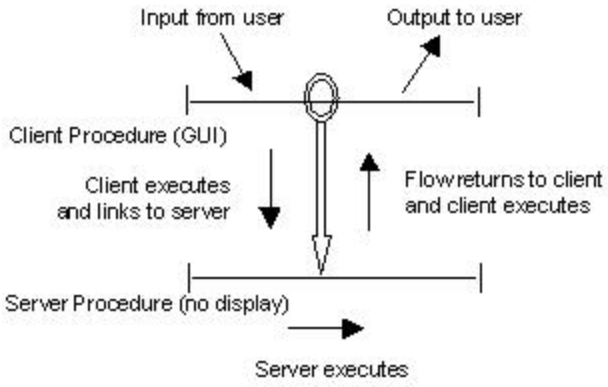
  You may find it more convenient to define this group of procedures as procedure steps of the same procedure. The client procedure is the first step and the server procedures are the following steps. With such a definition, other client procedures cannot access the server procedure steps.

- Many Client Procedures: Many Server Procedures

  In this mapping, the functionality of each server procedure is implemented to be reusable by many client procedures. Reusability in this case may result in complex server procedures as well as complex client procedures. On the DLG, each server must be a procedure (not a procedure step). Reusability might be better achieved by means of reusable action blocks.

Define Client/Server Flows

The following illustration shows what occurs as control flows from a client procedure to a server procedure and back.



After a distributed process has been split into client and server procedures, you need to add flows between the procedures. Flows to a server are defined to connect client procedures to the information processes, which are implemented as server procedures. The flows must always return and are defined as links. Maintenance of both client and server logic will be simpler if few dialog links between a client and server are defined. For instance, add, change, and delete requests should ideally be on one flow, not three.

One starting place is to first add dialog flows between the client procedures that implement user tasks. The server links are in addition to the client dialog flows. The users should not be able to tell when the client is accessing the server, so you can think of these additional flows as being invisible to the user.

Link flows must be used to join client and server procedures; transfer flows cannot be used. The link flow works in the following manner:

- When the client procedure completes execution an appropriate exit state is set and control is passed along the link flow to a server procedure. The properties of the link flow should be Execute First on arriving at the server (the server procedure has no display associated with it).
- When the server procedure completes execution an exit state that causes a return should be set in the procedure logic. Control is then always passed back to the calling client procedure.

Exit States

The Returns On flow supports any of the termination actions for exit states (normal, rollback, abort) and any of the exit state message types (none, informational, warning, error). Exit state messages associated with server procedures return to the client component and are displayed. One exception to this is if you specify a message type of none and specify a message text. The message is not returned for a message type of none that has message text.

Only data or a command may be passed along a flow. The value of the exit state cannot be tested by the client procedure logic.

Exit states that cause a flow for Returns On are optional. If not used, flow returns to the client procedure when the server procedure completes execution. This is no different from the behavior of any procedure.

The flow property should be set to Execute First on return. This ensures that the client procedure is always executed again to interpret the result of the server procedure execution.

Data in Flows

Data is normally passed in both directions along the flow. View matching on the flow specifies that the data needed by the receiving procedure be passed between a client and a server procedure. A well-designed client/server application should minimize network traffic. When you match views on a flow, move only essential information between the two procedures.

The data to be passed along a flow is defined using view matching. By introducing a work attribute view in the server procedure and a corresponding view in the client procedure, you can return the server procedure execution status to the client procedure for interpretation. The interpretation is performed by a CASE OF statement in the client procedure.

Summary of Requirements

The procedures and flows on the DLG must meet certain requirements for a distributed process application:

- Client procedures must be designated as online.
- Client procedures can flow to other client procedures or to server procedures.
- Flows from client procedures to server procedures must be *links*. The flow must return to the client.
- The link from a client procedure to a server procedure must be set to Execute First.
- Server procedures must be designated as online with no display.
- Server procedures can be subdivided into procedure steps, but the flow from a client procedure must go to the first step of the server procedure.
- The server procedure must set an exit state to return control to the client procedure at the end of a server procedure execution.
- When data is to be sent or returned across a link, you must map the information views on both the client and server procedures to one another with view matching.

Recommendations

In addition to the requirements, certain tips can help the design of your DLG:

- Generally, use as few dialog flows as possible between a client and a server. For example, add, change, and delete requests should be on one flow, not three. The client application can set the command to CURRENT on the flow to the server application. The logic in the server application tests the CASE OF COMMAND to determine which action to take. The reverse can happen on the return flow. In some situations, individual flows may be more efficient on a network because you pass only required data.
- Include the word *client* or *server* (or an abbreviation for them) as part of your procedure name, perhaps as a suffix. This helps you visually identify the two types on the DLG and during packaging.

Action Diagram Tool

The PrADs for distributed process applications must contain separate logic for the client and server components. The client logic has two parts. One part receives input from the user, begins processing, and passes control to the server. A second part receives control from the server and ends the processing for that procedure step.

Generally speaking, the separation of logic occurs along the following areas of processing:

- Client (Input)
  - Presentation of information
  - Dialog control
  - Input validation
  - Event processing
- Client (Output)
  - Error reporting
  - Reformatting of extracted data
- Server
  - Data actions against the database
  - Error recovery
  - Input validation requiring data access

Tracking between the client and server logic is easier if you match the command sent and received. You can accomplish this by creating a similar CASE OF COMMAND structure in the PrADs for the client and server components. Although the structure is similar, the actions for the commands differ in each PrAD.

For more information about the Action Diagram tool, see the *Action Diagram User Guide*.

Construction

The construction tasks for a client/server application differ little from those for other types of applications. You package procedure steps, define environment parameters, and generate/install the application. For a distributed process application, you perform these tasks for both the client and server components. For more information about generating and installing client/server applications, see the *Workstation Construction User Guide*.

How to Add Distributed Process to an Existing Model

With CA Gen Toolsets, you can modify an existing model and add the distributed process style. You can continue to use the existing application as you phase in the distributed process application. The action blocks in the existing model are reusable in the distributed process application.

This section explains how to add the distributed process style to a terminal-based (block mode) application, but the principles also apply to converting a GUI application.

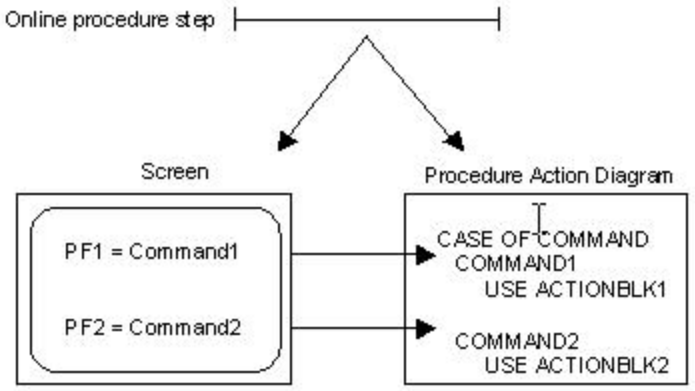
The following details the main tasks required for converting an existing application to a distributed process application:

- Create a separate business system
- Create the server procedure steps
- Copy the PrAD logic from the terminal-based PrADs
- Create the client procedure steps
- Modify the DLG
- Modify the client PrADs
- Modify the server PrADs
- Create the client user interface
- Package the application
- Generate/install the application

These tasks represent a fast way to convert an existing terminal-based application (preferably a small application). The intent is for you to create quickly a distributed process application that works. This lets you begin exploring client/server computing as soon as possible. Fast, however, does not necessarily mean best. Any redesign of an application should take into account all of the information discussed in the *Client* Server *Design Guide*. Even so, the knowledge gained from a quick conversion is beneficial to future design work.

Typically, a terminal-based application contains online procedure steps. The procedure steps are associated with screens, which use function keys to invoke commands. The commands are interpreted by the PrADs through a CASE OF COMMAND structure.

The following illustration shows an example of a terminal-based procedure.



Create a Business System

Create a separate business system for the distributed process application. Note that the same business system should contain both the client and server components of the application.

Including both in the same business system provides the following benefits:

- Avoids flows across business systems which can make subsetting cumbersome.
- Flows between procedure steps can be verified more easily.
- Load module packaging is not possible across business systems.

Create the Server Procedure Steps

Server procedure steps contain code used to execute the business logic that is currently contained in the corresponding block mode procedure steps.

Follow these steps:

1. Access the Dialog Design tool.
2. Create the server procedure steps for the DLG based on the ones for the terminal-based application.

You may find it easier to read from a list as you add the procedure steps for the distributed process application. Several reports, such as the reports for online packaging and procedure definitions, list the procedure step names.

Note: We recommend that you add the word server, or some designation that indicates server, to the procedure step name.

Copy the PrAD Logic

Now that you have a DLG for the distributed process application, you need to copy the PrADs from the terminal-based application.

Follow these steps:

1. Open the business system for the terminal-based application.
2. Select the Action Diagram tool.
3. Open a PrAD that you want to copy.
4. Open the business system for the distributed process application.
5. Select the Action Diagram tool. You now have two windows open for the Action Diagram tool.
6. Open the corresponding server PrAD.
7. Select all lines of the logic in the terminal-based PrAD. Do not include the views.
8. Select the blank line below the views in the distributed process PrAD.
9. Select XCOPY from the Edit menu to copy the logic from the terminal-based PrAD to the distributed process PrAD. The views get copied also.
10. Repeat steps 3 and 6 through 9 for each PrAD that you need to copy from the terminal-based application to the distributed process application. When finished, close the Action Diagram tool for the terminal-based application.

Create the Client Procedure Steps

Client procedure steps mimic the user interaction that the current screens facilitate. Logic in these procedure steps will be limited to interaction with the user and calls to the server procedure steps.

Follow these steps:

1. Access the Dialog Design tool.
2. Copy each server procedure step to create a client procedure step.

Note: We recommend that you add the word *client*, or some designation that indicates client, to the procedure step name.

Modify the DLG

In Dialog Design, the flows that currently exist between screened procedure steps are duplicated between the newly created client procedure steps. This allows the flow of the application from the user interaction point of view to be the same as it was before.

Follow these steps:

1. Select each server procedure step and set the step properties to online with no display.

   Note: The client procedure steps should already be set to online with display, so you do not have to change the step properties.

2. Join each client procedure step to its corresponding server procedure step with a link.

3. Set the Flows On property to an exit state of FLOW_TO_SERVER. Set the command to Current.

Modify the Client PrADs

For each of the client component PrADs, complete the following steps.

Follow these steps:

1. Select all lines of logic in the PrAD, except for initial statements that move imports to exports.

2. Delete the lines of logic.

3. Add logic that sets an exit state of FLOW_TO_SERVER and then escapes.

   You want the current command to get passed to the server procedure step. The server procedure step already has a CASE OF COMMAND structure to handle the command.

4. Increase the cardinality of the repeating group views.

   GUI client PrADs can handle more data than PrADs for terminal-based screens. A terminal-based screen is limited to 25 lines. Windows and dialogs can have numerous scrolling list boxes, drop down lists, and so on.

Modify the Server PrADs

For each of the server component PrADs, complete the following steps.

Follow these steps:

1. Delete all logic pertaining to the user interface, such as MAKE statements.
2. Retain the USE statements that use action blocks dealing with database manipulation.

Create the Client GUI

Select the Window Design functionality in the Navigation Diagram tool and create the windows and dialogs for the client procedure steps. You can convert a procedure step into a window or dialog based on its views. In the Window Design tool, the conversion process is called transformation. You access transformation from the Window Selection pop-up.

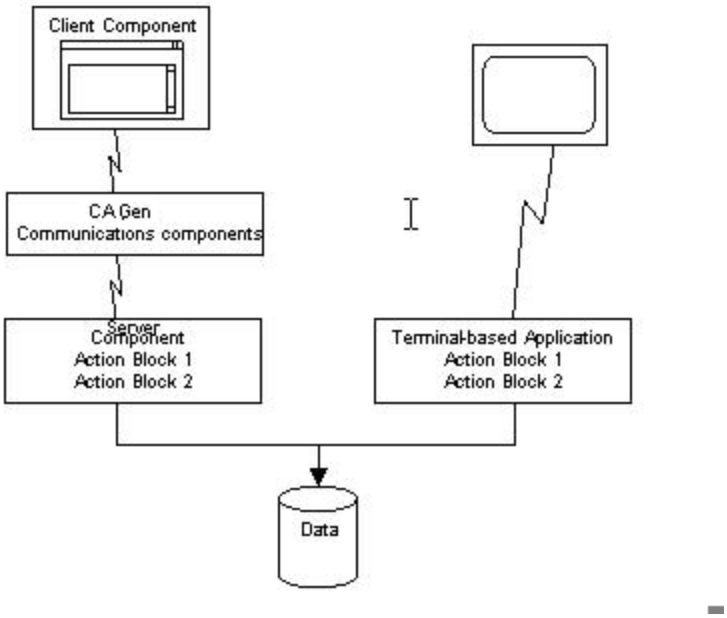
The conversion can create a window or dialog that serves as a good starting place depending on the type of functionality provided by the procedure step. For example, a procedure step that performs only relatively simple processing, such as an add capability, in the terminal-based application converts very well. A procedure step associated with a selection list on a screen and with dialog flows does not transform as well. You may want to convert the client procedure steps and check the results. You can delete the ones that do not convert to your satisfaction and build those as required for the design of your user interface.

Package and Generate the Application

Information about packaging, generating/installing, and testing a distributed process application is discussed in detail in the *Workstation Construction User Guide*. In short, package the distributed process business system for local testing and for production (the target platform). For local testing, package the application for window packaging. Package the client and the server procedure steps as GUI load modules.

Generate and install the distributed process application for local testing. Test the application. After you complete local testing, generate and install the application for the target platforms. The server component on the host platform can run concurrently with the terminal-based application. The same action block logic can access a common database.

The following illustration shows reusable action blocks for different types of applications.



Deploying Changes

After you install the distributed process application, revisions to it have to be redeployed. This can involve the client component, server component, or both. Deploying changes to a distributed process application has several considerations. For more information about deployment, see the *Client Server Design Guide*.

Chapter 5: Designing a Database

This section contains the following topics:

Database Design Team Tasks

Database design involves creating a relational database definition from the Entity Relationship Diagram (ERD). The ERD is the conceptual data model. The database design team usually involves the following members:

- Application development team
- Database specialists

The involvement in the database design can vary greatly for each audience. For example, one primary goal of the design team is to create an application with which the team can perform unit testing. The design team, at least initially, needs only a database design that can be generated and installed. The design team is less concerned with fine-tuning the database or with optimizing performance.

After completing unit testing and perhaps other areas of testing, the design team then needs to fine tune the database and optimize its performance. This is often when a database specialist assists the design team. The database specialist looks in detail at the logical and physical definitions for the database.

Since different audiences have different roles concerning database design, the audiences tend to use different tools in the Design Toolset.

Database Specialist Tasks

The database specialist is primarily concerned with the following tools:

- Data Structure List
- Technical Design Defaults
- Data Store List
- Referential Integrity Process

For more information about these tools and the tasks related to them, see the *Toolset Help*.

Design Team Tasks

The design team, in an effort to reach unit testing, is most concerned with the Transformation and Retransformation Tools. In addition, the design team may want to modify defaults with the Technical Design Defaults tool.

As development progresses, the design team may need to transform specific pieces of the data model as the data model changes. Incremental retransformation is used for this. In general, transformation is performed the first time and retransformation is performed thereafter as needed.

The physical database definition that is created when the data model is transformed does not contain a dynamic link to the data model. This means that the design team can continue to make changes to the data model without impacting the technical design. Changes may be made to the Data Structure List without affecting the data model. Changes to the Data Structure List are lost if a subsequent transformation is performed on the data model.

Initial Transformation Tasks

The design team has two primary tasks to perform the initial transformation:

1. Establish technical design defaults
2. Invoke the transformation

Technical Design Defaults

Technical design defaults provide control over the database definition that is created during the transformation process. In general, the design team does not need to adjust the technical design defaults. The defaults are sufficient to produce a database definition that can be generated and installed. The default values produce the data definition language (DDL) necessary for structured query language (SQL) generation and installation. Most of the defaults relate to specifics of the database management system being used and should be reviewed with a database specialist before being implemented in other environments.

Technical Design provides support for the following DBMS types:

- DB2 z/OS
- DB2 UDB
- JDBC
- MS/SQL
- ODBC/ADO.NET

- ORACLE
- SQL/MP

For more information about changing the defaults in the Technical Design Properties pop-up, see the *Toolset Help*.

Transformation

Transformation is the process of creating a data structure diagram (DSD) based on the ERD. You access the DSD through the combination of the Data Structure List and Data Store List Tools.

The DSD consists of data structure objects (tables, indexes) and data store objects (databases, tablespaces, indexspaces). Based on the data objects of the DSD, the DDL is created during Generation.

Before performing transformation, CA Gen runs a consistency check against the data model. If any portion of the data model is incomplete or inconsistent, the transformation fails. The design team can either correct the errors in the data model and then transform again, or manually transform parts of the data model that were consistent. Manual transformation involves using the Data Structure List tool and requires more work from you than when CA Gen performs transformation.

CA Gen transforms a data model in the following way:

- Each entity type in the data model is implemented as a table.
- Each attribute in the data model is implemented as a column.
- Each relationship with a cardinality of 1:1 or 1:M is implemented as a foreign key. A foreign key is a collection of one or more columns in one table that is used to identify another table.
- Identifiers are implemented as indexes.
- Foreign keys (implementing relationships) are placed in indexes.
- Data set sizes are calculated from entity type and relationship properties (maximum number of occurrences, and so forth).
- Names of tables and columns are determined in this way:
  - If the entity type (being implemented as a table) or attribute (being implemented as a column) has a Data Structure List name, it is used.
  - If there is no Data Structure List name, the entity type or attribute name is used.

- If the resulting name is a reserved word in the selected database management system, the name is modified slightly by CA Gen.
- All tables are placed in a single database.

Note: The characters in the table name are not verified for validity on the target DBMS OS system. The designer must ensure that any invalid characters generated in table or row names are replaced with valid character sequences. This can be done by defining a valid sequence of characters for the TD name of the entities and attributes or by replacing the generated TD names in the Data Structure List with valid characters sequences.

Retransformation

Retransformation automates the process of reconciling changes in the ERD to the DSD. Retransformation selectively updates the DSD after the ERD has been changed. The DSD is left intact and only those ERD objects that changed in such a way to leave the DSD inconsistent with the ERD are implemented again. This preserves any customizing you may have done on the DSD, such as changes to table sizes, and eliminates the need to delete and add a DSD table again to make changes.

It is intended for use when you are in a DSD maintenance mode. If there is no DSD or you do not want to preserve the existing one, a transformation serves you better.

Retransformation uses Consistency Check rules to detect differences between the ERD and DSD. A rule violation is required to trigger retransformation. Simply changing the name of an entity type in the ERD does not cause retransformation but making a structural change, such as adding an entity type, does. Deletes in the ERD do not result in any retransformation changes because they are immediately applied to the DSD.

Before performing retransformation, check the technical design defaults. Even though all databases are checked for consistency, the default database specified is the one to which the retransformation adds any new tables or tablespaces. Note that if you change the default owner ID between retransformations, the result will be some tables qualified by the new ID and some by the prior ID. This is because retransformation reconciles changes in the ERD to the DSD, not changes between one setting of defaults and another setting of defaults.

The retransformation pop-up lets you select which objects are retransformed. You can use the options to select the type of retransformation, and to limit the scope of the retransformation and the RI process to implement.

Type of Retransformation

You can use the following options to select the type of retransformation:

- Synchronize TD With Data Model causes changes in the data model to be implemented in the technical design.
- Specialize TD for Current DBMS causes each entity type in the TD to be specialized to the current DBMS.

Scope of Unimplemented Entity Type Implementation

The following options let you manage the scope of the retransformation:

- All causes every unimplemented entity type found in the ERD to be implemented in the default database. The process updates all tables, indexes, and constraints.
- Mandatory and Identifying Relationships refers to relationships. This option implements any unimplemented entity types that are related to an implemented entity type through identifying or mandatory relationships.
- Identifying Relationships Only implements unimplemented entity types that identify implemented entity types.
- None does not implement any unimplemented entity types but causes changes to existing DSD objects to be processed.

Scope of RI Process

The following options let you manage the scope of the RI process:

- All selects any unimplemented relationships in the data model to be implemented in the RI process.
- None selects any implemented entity with changed relationships in the data model to be implemented in the RI process. No unimplemented relationships are selected.

Retransformation Reports

The following reports might contain the results of a retransformation. The reports are created only if information is available to report.

- Implementation—This report is generated unless no violations of Consistency Check rules are found. The report lists what Consistency Check errors were found.
- Cleanup—Retransformation takes care of certain situations for you such as calculating missing space allocations. If a VCAT name is missing, the VCAT name in the DSD default panel is used. This report lists these types of changes performed by CA Gen tools.

- Protection Errors—This report lists if an action cannot be taken because an object is checked out in another subset.
- Unimplemented Objects—This report lists entity types and relationships that are not implemented. This report may appear if you do not select the All option. However, inconsistent entity types and relationships that cannot be implemented in the DSL also appear on this report.

Related Publications

For more information, see the following deliverables:

- Action Diagram User Guide
- Host Encyclopedia Administration Guide
- Online help for the Data Structure List, Data Store List, Technical Design Properties, and Referential Integrity Process Tools

Chapter 6: Designing Dialogs

This section contains the following topics:

Procedure Definition and Dialog Design

A procedure describes how a business will conduct its activities and consists of one or more procedure steps. The flow between procedure steps is known as a dialog and is represented in a Dialog Flow Diagram (DLG).

The DLG describes the interactions between a user and the generated business system. In an online environment, users interact with the system through screens, windows, or dialogs. In a batch environment, users interact with the system through reports and connections to data outside of CA Gen software.

When you use the Dialog Design tool, inputs to the DLG are as shown next:

- Elementary processes defined during a previous phase of development
- Procedures added to the DLG
- Flows added to connect procedure steps
- Commands set to invoke a procedure
- Function keys assigned to invoke commands

Dialog Flow Diagram

In the DLG, a procedure appears as a horizontal bar with vertical end posts. Horizontal bars without end posts represent procedure steps. Dialog flows appear as vertical magenta arrows indicating direction. Arrows without loops represent one-directional transfers, while arrows with loops represent
two-directional links.

The portion of the DLG where the procedure names appear is a fixed size (you can think of it as a window). The portion of the names you see depends on the Zoom level. The procedure bars are also in their own window. In a complex DLG, this window can be very wide. To see the entire diagram, scroll the diagram portion using the bottom scroll bar.

The CA Gen code generator uses the DLG to build a dialog control program. This program manages the data flow logic and the sequence of screens or windows in an application.

Note: For more information about dialog design tasks and concepts, see the *Toolset Help*.

Prerequisites to Using the Dialog Design Tool

Before using the Dialog Design tool, invoke CA Gen and open an existing model.

Build a Dialog Flow Diagram

To build a DLG, you create procedures, procedure steps, and dialog flows.

Create Procedures

To create a procedure, use the Edit and Add Procedure commands, or use the Copy command to create a new procedure from an existing one. Each of these activities is described in the following sections.

Ordinarily, you create procedures to implement processes that result from Analysis. You also can create procedures (called designer-added procedures) that improve the implementation of the business system, but are not a result of Analysis. The most common example of a designer-added procedure is a menu that calls other procedures, each of which implements one or more processes.

Note: When a designer-added procedure is required to perform CREATE, READ, UPDATE, or DELETE actions on an entity type, it indicates that an elementary process was not defined during Analysis. You can generate Create, Read, Update, and Delete logic automatically using Procedure Synthesis. Review the activity model before adding a designer-added procedure that performs Create, Read, Update, or Delete.

Add a Procedure

To add a procedure, select its location and assign it a unique name. The name should identify the work the procedure accomplishes. Most often, procedures are named with a verb, indicating the action, and a noun, identifying an entity type or an attribute. For example, if the procedure is designed for customer maintenance, enter a name such as MAINTAIN CUSTOMERS.

When naming procedures and procedure steps for client/server applications, indicate whether the procedure is processed in the client or server environment. For example: MAINTAIN CUSTOMERS CLIENT and MAINTAIN CUSTOMERS SERVER.

Adding *Client* and *Server* to the procedure name helps the person, responsible for load module packaging, distinguish procedures for the client environment from procedures for the server environment.

When a procedure is added it can be set to *online* or *batch* implementation. If you choose *batch,* the procedure name displays with *batch* in parentheses.

In a client/server application, client procedures must be set to online with display. Server procedures must be set to online with no display.

Perform Procedure Synthesis

To perform Procedure Synthesis, choose the action blocks (Analysis processes) and commands for each procedure to implement.

Note: For more information about procedure synthesis, see the *Action Diagram User Guide*.

Copy a Procedure or a Procedure Step

You can copy procedures and procedure steps in the following two ways.

- The first method, Copy, duplicates an existing procedure step after you assign a unique name to the copy. You select a new location for the duplicate procedure or procedure step. The only difference in the original and the duplicate is the name.
- The second method, Copy with Substitution, duplicates the existing procedure or procedure step and replaces the entity types with entity types you specify. A different business system also can be selected to replace the business system in the original procedure or procedure step.

Note: For more information about Copy with Substitution, see the *Toolset Help*.

Create Procedure Steps

Each procedure contains at least one procedure step. A procedure step is a subdivision of a procedure. Procedures with only one step are called single-step procedures. Create more than one step for a procedure when the contents of the data views of a single-step procedure are too large to fit on a single screen.

You can create procedure steps using the Edit and Add Procedure Step commands, or the Copy command. Both are described in the following sections.

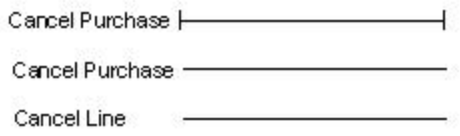
Add a Procedure Step

When you add a procedure to the DLG, CA Gen software adds it as a single-step procedure. When a procedure contains more than one step, the DLG distinguishes the procedure from the procedure steps.

The procedure is represented by a horizontal line with a vertical line at each end. Each procedure step is represented by a horizontal line *without* a vertical line at the end. The procedure steps appear immediately below the procedure. CA Gen software assigns the procedure name to the first procedure step. If desired, you can change the name of the first procedure step using the Detail or Step Properties actions. You assign a unique name to the second step and all other steps that you add to the procedure.

The following illustration shows a procedure and procedure step names. The Cancel Purchase procedure contains two procedure steps: Cancel Purchase and Cancel Line.

The following example shows a procedure with two procedure steps:



Procedure steps assume the type of implementation (online or batch) of the parent procedure.

Detail a Procedure Step

Using the Detail action, you can do the following tasks:

- Describe a procedure step.
- Specify the type of online procedure step (with display or without display).
- Define or redefine information views.
- Assign PF keys/commands and display them on screens. This task applies only to online procedures that are associated with screens.
- Specify dialog flows for the procedure step.
- Define how CA Gen accesses unformatted (clear screen) input.

Each of these topics is described under the corresponding heading.

Describe a Procedure Step

Using the Detail and Properties actions, type a brief description of the nature of the procedure step and its relationships to other steps.

Specify the Type of Online Procedure Step

Using the Detail and Step Properties actions, specify which of the following options is appropriate for an online procedure step:

- with display
- no display

When an application is designed for local processing, all online procedure steps can be set to with display or no display. When no display is selected, the online procedure step sometimes is referred to as a background online transaction.

When an application is designed for client/server processing, only online client procedure steps can be set to with display. Online client and online server procedure steps can be set to no display. These guidelines reflect the characteristics of a client/server application:

- User interface is handled only by client procedure steps
- Processing can be handled by client and/or server procedure steps

Specifying whether a procedure step is executed with display affects dialog flows, exit states, and clear screen input or unformatted data.

Online procedure steps with no display:

- Are required for all server procedure steps in a client/server application.
- Permit dialog flows to and from other online procedure steps when the application is designed for local processing.
- Are the destination of link dialog flows and require the Returns On exit state.
- Cannot receive clear screen input (unformatted data) from the previous procedure step.

Online procedure steps with display:

- Are permitted for all client procedure steps in a client/server application.
- Allow dialog flows to and from all procedure steps when the application is designed for local processing.
- Permit dialog flow links from client procedures to server procedures.

Define or Redefine Information Views

Using the View Maintenance option in the Detail pull-down list, you can define or redefine the information views of the procedure step.

Note: For more information about restrictions, see the *Action Diagram User Guide*.

Assign PF Keys/Commands and Display Them on Screens

Assigning commands to function keys expedites invoking commands. The PF Key Definitions pop-up window lists the function keys. By selecting a function key and Props, you display the list of commands defined in Business System Defaults. Commands that you select when setting system defaults apply to all procedure steps in the business system. When you add a command using the Dialog Design tool, its type is Local, which means that is in effect only for the particular procedure step. The Key Properties pop-up window lets you choose whether to display the key assignment on the PF Key line.

The function key commands set in System Defaults are defined as either Standard or Default. In a procedure step you can override default commands, but you cannot override standard commands.

Specify Dialog Flows for the Procedure Step

Specifying dialog flows is basically the same for procedures and procedure steps.

More information:

Create Dialog Flows (see page 57)

Define Clear Screen Input

When you invoke the first procedure step associated with a screen, Clear Screen Input lets you pass unformatted input to the procedure without displaying its empty formatted screen. However, you cannot pass unformatted input to procedure steps without screens.

To enter unformatted input on a clear screen, you must enter a transaction code followed by a list of parameters separated by the appropriate string and parameter delimiters. Business System Defaults contain the string and parameter delimiters defined for your system. By entering this data, you specify how the dialog manager of CA Gen will interpret the data that follows the transaction code on clear screen execution and will pass that data to the import views of the procedure step. The transaction code is required by teleprocessing monitors (for example, IMS and CICS) to identify the programs required to execute the procedure step.

The Clear Screen Input option can help you reduce execution costs and time by providing a short cut to invoking and using a procedure. In addition, experienced users can employ this option to allow a program generated outside CA Gen to access a program generated using CA Gen software.

The parameter list can be defined by keyword. Keywords are labels that identify the parameter and must be accompanied by the parameter data enclosed in delimiters.

Example:

<*trancode*> NAME'John Doe',NUMBER'123-45-6789'

<trancode>

Identifies the transaction code, single quotes represent the string separator

comma (,)

Identifies the parameter delimiters

Note: NAME and NUMBER are keywords. If spaces are not selected as the parameter delimiter, they are ignored in the clear screen input data stream.

The keyword can be the prompt that accompanies the field on the screen with which the procedure step is associated or some other label that can be associated with the field. All keywords must be unique. When you use keywords, the parameters entered at a clear screen need not be in the same order as the parameter list defined for the screen.

For example, the following two clear screen inputs are acceptable:

<trancode> NAME'John Doe',NUMBER'123-45-6789'
<trancode> NUMBER'123-45-6789',NAME'John Doe'

Keywords are case sensitive and cannot contain delimiters. However, you can append symbols. For example, if you append an equal sign (=) to the keyword NAME, the equal sign functions as a keyword delimiter. Using the example presented here, the result is NAME=John Doe.

If you do not define parameters with a keyword, you must enter the parameters at a clear screen in the same order as they are defined in the list. You can also combine keywords with parameters ordered by position in the list.

Combining Keyword and Positional Parameters

If you mix parameters with a keyword and positional input, you need to be aware of this rule. Positional input counts its position relative to other positional inputs. The following three examples illustrate the rule.

Example 1:

The clear screen input below is acceptable when the employee number (201289), division number (6), and cost center number (739) are the first, second, and third items, respectively, in the parameter list.

<trancode> 201289,6,739

Example 2:

If you add a keyword to the second item (DIV=6), and leave the other items unchanged, you create an error of conflicting unformatted input.

```
<trancode> 201289,DIV=6,739
```

Although the first positional parameter is still first, the cost center parameter (739) is now the second positional input field, not the third as it is in the parameter list.

Example 3:

To prevent the error shown in Example 2, code a second comma after the division parameter, as follows:

```
<trancode> 201289,DIV=6,,739
```

The second comma acts as the second positional parameter. It also makes the cost center the third positional parameter, as it is in the parameter list.

Define Input

When you select Clear Screen Input, you use the menus to define how the dialog manager of CA Gen interprets the unformatted data. Perform the following tasks to define the input:

- Have CA Gen initialize the input text-look in the import view of the procedure step and select all non-repeating group import views (unformatted input is not supported for repeating group views).
- Add import views individually by selecting Edit, Add Import.
- Select keywords for the import view.
- Map data to CA Gen special field Command.

After a list of parameters is selected, you can delete or rearrange items on the list.

More information:

Business System Defaults (see page 107)

Copy a Procedure Step

You can also create a procedure step by copying a single-step procedure or a procedure step to another procedure. This can be useful when you have defined a procedure step and want to use the same procedure step details or (in the case of a single-step procedure) the same elementary processes in another procedure.

Create Dialog Flows

A dialog flow represents the movement and transfer of system control between procedure steps. The dialog flow for a client/server application illustrates the passing of control from the client environment to the server environment and back to the client environment.

Dialog flows can occur within a business system (internal flows) or between business systems (external flows) within the same model. A dialog flow can usually originate from any procedure step, but must go into the first procedure step of the destination procedure. A dialog flow originating from a server procedure step can flow only to another step within that same server procedure. A flow cannot originate with a server procedure step and flow to client procedure step.

In the DLG, a flow is represented by a vertical arrow between procedures. The arrow indicates the direction and movement of control from one procedure step to another.

There are two types of dialog flows-a transfer and a link. Each type is represented by a different arrow on the DLG.

Transfer

A transfer is a type of flow in which control and, optionally, data pass from one procedure step to another.

In the DLG, the transfer of control is represented by an arrow pointing to the destination procedure or procedure step. The arrow has a flat end.

Link

A link is a type of flow in which control and, optionally, data pass from a source procedure step to a destination procedure step, and back to the source procedure step.

The destination procedure step returns control to the source when the destination procedure step is complete. Data also may be passed to the source procedure step when control returns. In addition, all data known to the source procedure step when it originally executes is saved and made available after the destination procedure completes and returns control.

In the DLG, the link and return to a procedure step is represented by an arrow with a circle at the end. The arrow points to the destination procedure or procedure step. The circle indicates control can return to the originating procedure.

In a client/server application, the dialog flows from client procedures to server procedures must be links. Links are required to ensure that system control returns to the client when server processing is complete.

Link and transfer arrows on the DLG point up and down to illustrate the transfer of control. The DLG groups together all arrows flowing from one procedure or procedure step. If the procedure step is near the top of the diagram, the arrows flowing from the step appear on the left side. If the procedure step is near the bottom of the diagram, the arrows flowing from the procedure appear on the right side.

Dialog Flows for Online and Batch Procedures

You can create dialog flows for online and batch procedures. The following restrictions exist with batch procedures:

- You cannot connect a batch procedure or any of its steps to another procedure
- Only transfers are allowed
- Transfers to a prior procedure step are not allowed (forward or self-referencing transfers only)

Both online and batch procedure steps can be joined to themselves (known as a self-referencing transfer or link). This feature lets you control the checkpointing frequency of your database tables. Online procedure steps can have self-referencing links and transfers. Batch procedure steps can have self-referencing transfers but not self-referencing links.

Dialog flows are permitted to and from online procedure steps without screens.

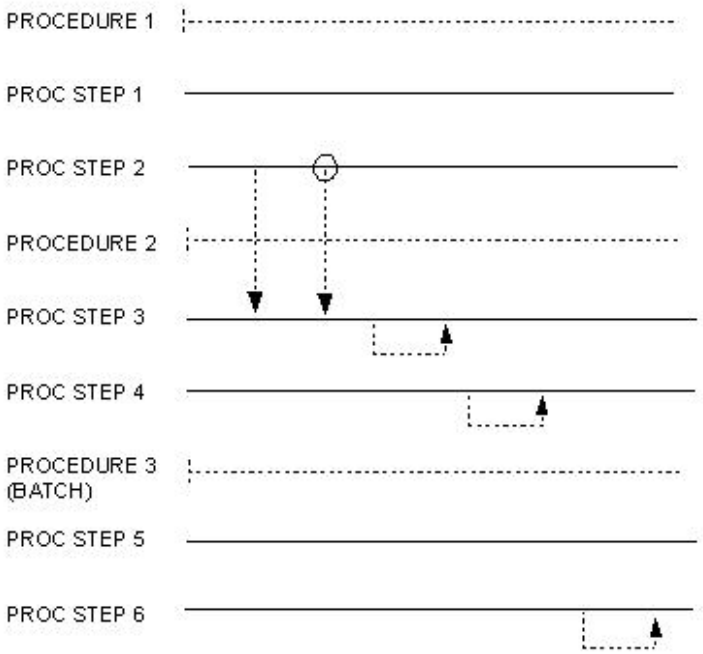
Note: Batch is only available on MVS.

Dialog Flow Example

The following illustration shows a DLG with the following types of internal flows from online and batch procedures:

- A transfer between two online procedure steps
  - (PROC STEPS 1 and 3)
- A link between two online procedure steps
  - (PROC STEPS 2 and 3)
- An online procedure with a self-referencing transfer
  - (PROC STEP 3)

- An online procedure with a self-referencing link
  - (PROC STEP 4)
- A batch procedure with a self-referencing transfer
  - (PROC STEP 6)



Dialog Flows and Client/Server Applications

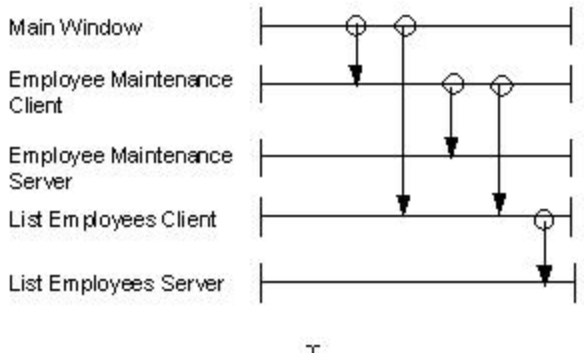
In an application designed for client/server processing, the following restrictions apply:

- Client procedures must be designated as online.
- Client procedures can flow to other client procedures or to server procedures.
- Flows from client procedures to server procedures must be *links*. The flow must return to the client.
- The link from a client procedure to a server procedure must be set to Execute First.
- Server procedures must be designated as online with no display.
- Server procedures can be subdivided into procedure steps, but the flow from a client procedure must go to the first step of the server procedure.
- If data is sent or returned on a link, you must match information views on the links from client procedures to server procedures.

- The server procedure must set an exit state to return control to the client procedure at the end of a server procedure execution.

When data is to be sent or returned across a link, the information views on both the client and server procedures are mapped to one another by view matching.

The following illustration shows a DLG for a client/server application.



More information:

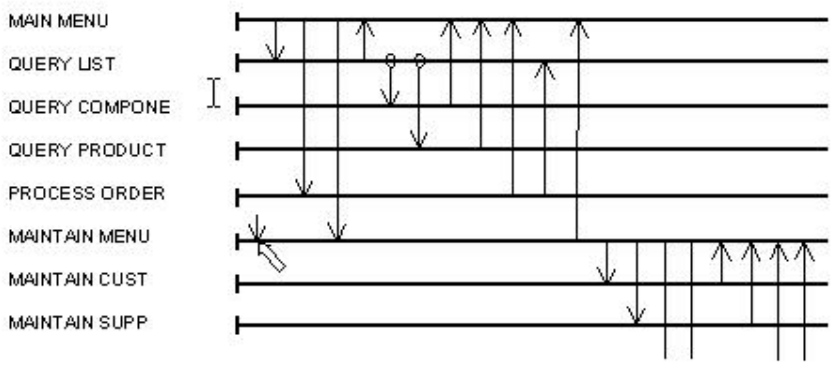
Designing Client/Server Applications (see page 29)

External Flows

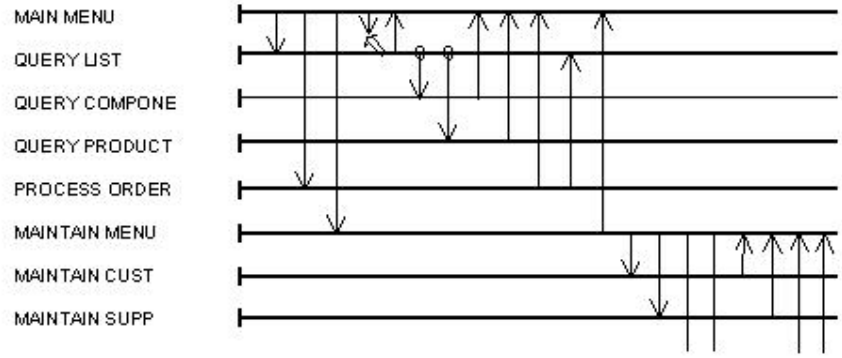
External flows specify a flow in or a flow out of the current business system. Use the Edit action and either the Add External Flow In action or the Add External Flow Out action to create a flow between business systems.

You can add an external flow into a destination business system and an external flow out of a source business system. An external flow into a destination system, known as an external flow in, can be to the first procedure step only. An external flow out of a business system, known as an external flow out, can originate from any procedure step.

The following illustration shows an external flow in.



The following illustration shows an external flow out.



Join Procedure Steps Within a Business System

Use the Edit and Join actions to create a flow between procedure steps within a business system. When you join a procedure step to a multi-step procedure, the destination must be to the first procedure step. Procedure steps other than the first are accessible only from within the procedure.

Detail a Dialog Flow

Use the Detail and Properties actions to detail a transfer or link dialog flow. This lets you do the following tasks:

- Specify the flow type
- Specify flow properties
- Specify commands to initiate dialog flow processing
- Specify conditions under which a flow occurs (exit states)

- Assign autoflow commands
- Match data views
- Describe the flow in words

Specify Flow Type

Specifying the flow type means specifying whether an existing flow is a transfer or a link. For a definition of transfer and link, see Creating Dialog Flows.

Default: Transfer

Specify Flow Properties

Specifying flow properties involves the following two activities related to the destination procedure step:

- specifying the first action by the procedure step
- specifying the command that the destination procedure step should execute

When a transfer or a link flows to an online procedure step that displays a screen, the destination procedure step can begin action in one of the following two ways:

- Display First—The generated system can display the screen or window for the destination step and wait for operator input to begin the logic of the procedure step.
- Execute First—The generated system can execute the logic of the destination procedure step. This results in either the display of the screen or window of the destination step, or the execution of a dialog flow. Links from client procedures to server procedures must be set to Execute First.

Select Display First when the procedure step requires input from the user of the application before procedure step logic can begin.

Select Execute First when the source procedure step provides all of the information required for the destination procedure step to begin processing.

Specify Commands to Initiate Dialog Flow Processing

A command value can be specified for each end of a dialog flow to direct the processing of procedure steps.

When two online procedure steps are joined with a link or a transfer, set a command that initiates the execution of the destination procedure step. When the dialog flow is a link, set a command that initiates the return to the source procedure step.

When a procedure step is activated from a screen, the application user enters a value in the Command field or presses a function key assigned to the command. This initiates the destination procedure step.

When a procedure step is activated from a dialog flow, a command value is passed by the dialog flow. The value is processed the same way a command entered by an application user is processed.

When you select a command to be passed along a dialog flow, you can have the command set in the source procedure step passed to the destination procedure step. To do this, you select the command option <CURRENT> when associating a command with a flow. In addition, upon return from a link, you can restore the previous command value. To do this you select the command option PREV when associating a command with a link flow. You can set the value of a command to <CURRENT> for flows within and between business systems. You can also set a command value to <CURRENT> for autoflows.

The <CURRENT> option can be used to add flexibility to menus: the user can select menu options that set an exit state and a command to be passed along the flow triggered by the exit state. The <CURRENT> option can be especially helpful if you have multiple options on the same screen menu, each of which uses a flow to the same destination procedure. For example, on a menu that includes options to add, delete, and fill sales orders, all three flows can occur to the same destination procedure. Instead of adding three exit states and three different commands, you can add a single dialog flow with one exit state and the command value set to <CURRENT>. By using fewer dialog flows, you develop smaller load modules during the Construction phase of CA Gen usage.

When you use the <CURRENT> value for command, be sure that the destination procedure step can respond to the command values passed to it. You must use the techniques in Action Diagramming to ensure that all commands passed to the procedure step can be acted upon.

When you set a command in a flow including <CURRENT>, you should set the properties of the dialog flow to Execute First.

Specify Conditions Under Which a Flow Occurs (Exit States)

For a dialog flow to take place, a condition called an exit state must occur. Defining an exit state includes assigning a value to the exit state and optionally a message that appears on the screen when the exit state occurs. For example, if there are procedure steps named MAIN MENU and CANCEL ORDER, the exit state that caused flow to occur from MAIN MENU to CANCEL ORDER could be assigned as CANCEL ORDER REQUESTED. The exit state message could be Order cancelled.

Every procedure step associated with a dialog flow must have at least one exit state set in its action diagram. Each transfer and link on a dialog flow diagram is associated with a Flows On exit state that causes the change of control from the source procedure step to the destination procedure step. On links, there is also a Returns On exit state associated with the return of control to the source procedure step. When a procedure step completes, it transfers control based on the value for NEXTTRAN, then the exit state. If the procedure step finds two exit states with the same value, the step transfers control based on the Returns On exit state before transferring control based on the Flows On exit state. For more information about NEXTTRAN, see the *Action Diagram User* Guide. For background (non-screened) transactions, it is important to set an exit state associated with a dialog flow to a screened transaction in the procedure step in order to respond to the terminal user.

In client/server applications, exit state messages associated with server procedures return to the client component and are displayed. Only data or a command may be passed along a flow. The value of the exit state cannot be tested by the client procedure logic. However, any exit state messages associated with server procedures do return to the client procedure and are displayed.

Assign Autoflow Commands

Assigning a command as an autoflow command lets you indicate that a particular command causes an immediate exit state to be set and the associated dialog flow to be executed without screen field validation or the invocation of a procedure step action block. For example, it allows the user to exit a screen to return to a menu without entering required data on the current screen. Autoflow is not required for prototyping, but it does facilitate it.

When you define autoflows, CA Gen software lets you add an exit state to a dialog flow and assign an autoflow command to the exit state. If the dialog flow is a link and inputs to the source procedure step need to be saved, autoflow updates the data views in the user profile without verifying user inputs. Autoflow also performs the indicated view matching and activates the destination procedure step. The input data is always verified immediately before execution of the destination procedure step.

Note: As you assign commands as autoflow commands, you can also assign the same commands to function keys and have the command display with the function key assignment on the screen. These steps improve operation during prototyping.

Note: For information about the steps to add an autoflow, see the *Toolset Help*.

Match Data Views

It is useful to send data between procedure steps in a dialog flow, especially when the same information is needed in both steps. Communicating data between steps can improve system flow and reduce the chance for error. During a transfer or a link, view matching is the way you specify the export data views of the source procedure step that should be placed in the import data views of the destination procedure step before the latter executes.

Flows to procedure steps in external business systems require view matching as well as flows between procedure steps within a business system. When matching views, you select the receiving views for the destination procedure step and the supplying views for the source procedure step. CA Gen guides you through the view matching by providing possible views to match and determining whether the match is possible.

In a transfer, control passes from the source step to the destination step. Match the export views of the source procedure to the import views of the destination procedure.

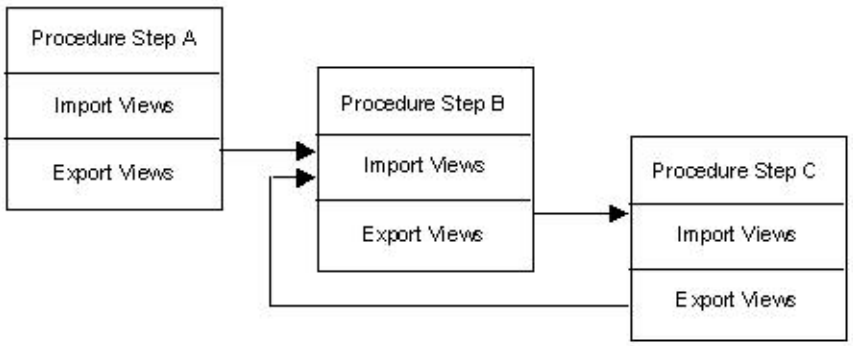
For example, in the next illustration, Procedure Step A transfers to Procedure Step B, and the export views of Procedure Step A are matched to the import views of Procedure Step B.

In a link, control passes from the source to the destination and, depending on the logic, may or may not return to the source. After matching the export views of the source to the import views of the destination, you match the export views of the destination to the import views of the source.

In the same illustration, Procedure Step B links to Procedure Step C and matches export to import in both directions.

Note: In a client/server application, minimize the volume of data transferred from the server to the client. When matching server export views to client import views, specify only the data that is required to complete a unit of work.

The following illustration shows an example of view matching export to import views.



Match and Copy Views

Use the match-and-copy function in the DLG when the source procedure step does not have a supplying or receiving view that corresponds to the views in the destination procedure step.

Unmatch Views

You can unmatch import and export views in the DLG. Unmatching a view *does not* delete either the supplying or receiving view. If you use the match-and-copy function to create a view and then you unmatch it, you do not delete the created view.

Describe the Flow in Words

You should briefly describe the nature of the flow and the procedure steps the flow connects on a description pop-up window.

Change Dialog Flow Diagram

After a DLG is created, one or more of the screen objects (procedures, procedure steps, or dialog flows) can be changed, deleted, or moved. Exit states, flows, and other characteristics of the dialog flow also can be changed.

The changes are organized into the following groups:

- Procedures
- Procedure Steps
- Dialog Flows

Change Procedures

You can change procedures in the following ways:

- Access view maintenance from a single-step procedure. For more information about view maintenance, see the *Toolset Help* and the *Action Diagram User Guide*.
- Change a procedure description in the same way that you enter a procedure description.

- Change the implementation type (batch or online) for a single-step online, multistep online, or multistep batch procedure when no dialog flows are connected to other procedures and the dialog flows within the procedure are downward transfers only. The implementation change also applies to the procedure steps.

  Note: Batch is only available on MVS.

  Delete an action block name from the list of action blocks implemented for a procedure. CA Gen removes the USE statement for the action block from the CASE OF COMMAND construct in the Action Diagram.

- Delete a procedure. When you do so, CA Gen deletes all of its associated procedure steps, screens, windows, and flows.
- Move a procedure. When you move a multistep procedure, its steps move with it. CA Gen does not allow you to move procedure steps out of their parent procedure. When you move a procedure, the dialog flows into and out of the procedure are redrawn to accommodate the new position of the procedure.

Change Procedure Steps

You can change procedure steps in the following ways:

- Access view maintenance from a procedure step. For more information about view maintenance, see the Toolset Help and the *Action Diagram User Guide*.
- Change a procedure-step description the same way that you enter the description.
- Change the name of a procedure step. When you add a procedure step to a single-step procedure, the first procedure step is initially named the same as the procedure. When you change the name of the first procedure step, you can also apply the name change to the procedure.
- Delete a procedure step. When you do so, CA Gen deletes its associated screens, windows, and all its flows.
- Move a procedure step. CA Gen redraws the dialog flows to accommodate the new position of the procedure step.

Change Dialog Flows

You can change flows in the following ways:

- Convert the flow type for online procedures. A flow to or from a procedure can be either a transfer or a link. After you join two procedures with a transfer, you can change the transfer to a link or change a link to a transfer. You cannot change the flow type for a batch procedure step.
- Change a dialog flow description in the same way that you enter a description.
- Delete a dialog flow, which severs the data connection between two procedures or steps.

- Remove an exit state from a flow. You may need to remove exit states to avoid a processing conflict after a model merge.
- Unmatch data views if you happen to match the wrong views.

Chapter 7: Designing Windows

This section contains the following topics:

Window Design Functionality

You use Window Design functionality to create a graphical user interface for an application.

With Window Design functionality, you can design the windows and dialogs required to support a GUI application and the appropriate controls for each.

For the sake of brevity, this chapter uses the term *window* to refer to both windows and dialogs. If a distinction is required, this chapter mentions the difference.

Prerequisites

Before you begin designing a GUI, you need to complete or have an understanding of a minimum list of tasks. However, the specific task list can vary depending on the information engineering techniques of the life cycle being used.

The following list offers a starting place of prerequisites to GUI design. The list does not represent a sequence. It shows only a set of prerequisites that can be satisfied in a number of different sequences. As you become more familiar with designing GUIs, or select a life cycle that works best for your needs, you may prefer a different set of prerequisites.

- An understanding of the tasks that the user performs with the application, which is usually determined from previous phases of a life cycle development
- A data model defined with one of the Data Model tools
- One or more procedures defined with the Dialog Design tool

- Views for one or more procedures defined with the View Maintenance tool or the Action Diagram tool
- A business system defined with the Business System Definition tool

Note: For more information, see the *Client* Server *Design Guide*.

Tasks Performed with Window Design Functionality

To create a GUI design, you have some preliminary decisions to make concerning the interface. CA Gen tools can often assist you as you make these decisions, but, overall, many of the preliminary decisions can be made independently of the tools.

For example, as you decide on the overall form and function of the interface, you would typically address tasks as listed here:

- Decide on a flow between windows and dialogs that corresponds to the order in which the user is most likely to use the application.
- Decide on any interface standards for the application, which could include decisions about the following criteria:
  - When to use tool bars and status bars.
  - The bitmap images to use as objects.
  - The font and the foreground and background colors for windows, dialogs, and controls.

After you have a preliminary design, you perform several major tasks with Window Design functionality. Each of these major tasks has subtasks associated with it. For more information about the tasks and subtasks, see the following table.

The major tasks tend to be iterative. For example, since one procedure may have many windows associated with it, you would iterate the tasks by specifying the characteristics for each window, the characteristics for each control, and so on.

You can follow the major tasks in sequence to create a GUI application. This is not the only sequence in which you could build a GUI application, nor is it a list of all the tasks that you perform with Window Design functionality. The major tasks show one sequence that can guide you as you create your application. As you become more familiar with Window Design functionality, you may prefer a different sequence.

The following table lists major tasks performed with window design functionality.

Major Task	Primary Subtasks
Add a window	Choose a procedure or procedure step with which you want to associate a window or dialog. Determine if you want to create an initial layout of the window or dialog based on the views for the procedure, based on a screen layout from the Screen Design tool, or based on your own design.
Specify window characteristics	Name the window. Specify if the window is the primary one with which the user interacts. Specify where on the desktop the window appears when first invoked. Specify if the window can remain open while the user performs other tasks. Specify if the window uses a bitmap for a background.
Add menu items to a window	Name the menu options that appear in the menu bar and menu bar pull downs. Determine the arrangement (hierarchy) of the menu options. Specify the font and colors for the menu items.
Specify menu item characteristics	Specify the text for the menu item. Specify the result that occurs when the user selects the menu item. Specify the command, if any, associated with the menu item.
Add controls that implement views of attributes	Select attribute views. Specify the type of control. Specify prompts and edit patterns. Place the control on a window.
Add controls that do not implement views of attributes	Specify the result that occurs when the user selects the control. Specify the text for the control. Place the control on a window.

Major Task	Primary Subtasks
Specify control characteristics	Map import and export views. Specify the text for the control. Specify prompts and edit patterns. Specify behavioral results of the user interacting with the control. Behavioral results also include event processing, which is listed as a separate major task because of the amount of information associated with events.
Add events	Add an event type. Create a name for the event action. Specify the action diagram logic for the event action. This subtask is performed with the Action Diagram tool.
Test the flow of user tasks from window to window	Decide which windows open or close which other windows. Test the flow from window to window.
Refine the overall appearance of the interface	Verify the implementation of any application-wide standards. Verify the use of tool bars and status bars. Rearrange the controls in a manner that satisfies the object/action flow in which a user performs tasks. Group related controls together. Disable controls where appropriate. Create or update any bitmap images to use as objects. Verify the choices for the fonts and the foreground and background colors for windows, dialogs, and controls.

Windows and Dialogs

Windows can be regarded as primary or supplemental. Primary windows provide the main dialog between the user and the application. Supplemental windows extend the dialog between the user and the application. Each application must have at least one primary window and may contain none or more supplemental windows. Primary windows are associated with procedures on the Dialog Flow Diagram.

Comparison of Windows and Screens

The other type of interface that you can create with CA Gen is a screen interface, which is used for terminal-based applications. Terminal-based applications are also known as block mode or 3270 applications. For a screen interface, each procedure may have only one screen. A one-to-one mapping exists between them. For a GUI, each procedure may have one or more windows. A one-to-many mapping exists between them. For both types of interfaces, a procedure may implement one or more elementary processes.

For screen interfaces, the application controls the dialog of the user and restricts options by what the menus provide. The interface suits a work pattern of procedural tasks controlled by the application. For GUIs, the application gives the user more control and lets the user continue with tasks, cancel tasks, and switch to other parts of the application. The interface suits a work pattern of information gathering tasks rather than just procedural tasks. The interface offers a more user-centered design and use of the desktop as a collection of tools.

More information:

Designing Screens (see page 87)

Add a Window

The objective of this task is to select a procedure and create an initial window or a dialog for it. You must have defined at least one procedure on the Dialog Flow Diagram before you can perform this task.

Add a Window using Business System Defaults

The following procedure explains how to add a window using the Business System Defaults menu.

Follow these steps:

1. Open a model.
2. Select Design, Business System Defaults.
3. Highlight a business system and right-click to select Diagram, Open.
4. Select Tool, Design, Navigation Diagram.

   The Navigation Diagram window appears.
5. Select Navigation Diagram, Edit, Add Window.

   The properties dialog appears.

6. Complete the properties and click OK.

   A window is added in the Navigation Diagram.

Adding a Window Using Dialog Design

The following procedure explains how to add a window using the Dialog Design menu.

Follow these steps:

1. Open a model.
2. Select Design, Dialog Design.
3. Select Edit, Add Procedure from the Dialog Design menu.

   The properties dialog appears.
4. Complete the properties and click OK.
5. Select the procedure that you have just added.
6. Select Diagram, Launch, Navigation Diagram.

   The Navigation Diagram window appears.
7. Select the procedure from the Network view.
8. Select Navigation Diagram, Edit, Add Window.
9. Complete the Properties box and click OK.

   A window is added in the Navigation Diagram.

Add a Dialog

The following procedure explains how to add a dialog.

Follow these steps:

1. Open a model.
2. Select Design, Business System Defaults.
3. Highlight a business system and select Diagram, Open.
4. Exit that menu and return to the Tool menu.
5. Select Tool, Design, Navigation Diagram.

   The Open Navigation Objects dialog appears.
6. Select a business system from the Business Systems drop-down.
7. Select a procedure step from the Procedure Steps drop-down.
8. Select one or more Windows from the Windows list and click Include.

9. Click Open.

   The Navigation Diagram window is displayed.

10. Select the procedure from the Network view.
11. Select Navigation Diagram, Edit, Add Dialog Box.
12. Complete the Properties box and click OK.

    The dialog box is added.

Menu Items

You specify the menu items on a menu design pop-up. The layout of the menu items is presented as an indented list. This makes the layout easier to design and view.

Add Menu Items to a Window

Only windows, and not dialogs, can have menu items. As you add menu items, you can also design the hierarchy of them in the menu bar and in the menu item pull downs.

Specify Menu Item Characteristics

Only windows, and not dialogs, can have menu items. The characteristics affect the appearance and the action that results when the user selects the menu item.

You specify the characteristics on pop-ups. The pop-ups vary depending on where in the menu structure hierarchy the menu item appears. Other characteristics of menu items, such as colors, disabled-by conditions, events, and help descriptions are also accessed from the pop-ups. You can access the pop-ups only from the Menu Design pop-up.

Controls

Controls are the interface elements that allow users to select choices, type information, or view information. Controls can be divided into the following categories:

- controls that implement views of attributes
- controls that do not implement views of attributes

Controls that Implement Views of Attributes

These types of controls primarily deal with data coming from a database and from a user and data displayed to the user. Therefore, these controls are associated with (implement) attribute views. These controls can also be used for programmatic control by triggering events.

The following table lists the types of controls that implement views of attributes and the pop-up from which you add the controls.

Control	Added from Field Design	Added from List Properties
Check box	x	
Drop List	x	
Entry field (multiple line)	x	
Entry field (single line)	x	
Hidden field	x	
Radio button	x	
List box		x
Enterable list box		x
Enterable drop-down list		x
Non-enterable drop-down list		x

All controls in the preceding table, except for list box, implement a single attribute view. The list box control can implement one or more attribute views. The attribute views for a list box, and for the other three types of List Properties controls, must belong to a repeating group view.

Attribute views implemented as Field Design controls may or may not belong to repeating group views. If you implement an attribute view that does belong to a repeating group view, the control is called a list occurrence. For example, if you implement an attribute view in a repeating group view as a check box, the control is referred to as a list occurrence check box. The only exception to this is the hidden field control type. There is no list occurrence hidden field because hidden fields are not placed (implemented) on a window.

For more information about the types of controls and specific information related to their use, see the *Toolset Help*.

More information:

Event Processing (see page 78)

Controls that Do Not Implement Views of Attributes

These types of controls are primarily used for programmatic control, appearance, information, or a combination of the three as listed:

- Group box
- Literal
- Menu Item
- Picture (bitmaps)
- Push button
- Special field

Menu items are discussed under their own topic.

Special fields provide read-only information from the operating system or from the CA Gen runtime. For more information about the View List on the Field Design pop-up for further information about special fields, see the *Toolset Help*.

The online help further defines the other types of controls and gives specific information related to their use.

More information:

Menu Items (see page 75)

Programmatic Control

Programmatic control is what you define to occur when a user interacts with a control. The action of interacting with a control can cause the following results:

- Trigger an event
- Execute the procedure step
- Execute a command, which, in turn, can cause an autoflow
- Execute a special action of OK, CANCEL, or HELP
- Invoke a dialog

All of the control types except literals, hidden fields, group boxes, and special fields can trigger events. Only menu items and push buttons can cause the other four types of results.

Control Characteristics

The control characteristics affect the appearance or the action that results when the user interacts with the control.

You specify the characteristics on pop-ups. The pop-ups vary depending on the type of control.

Prototyping

At any point after you add a menu item or a push button to a window, you can start testing the dialogs among windows and dialogs. This testing is called prototyping. You can use prototyping to verify the dialog flow with your users as you design the interface.

You perform prototyping by placing the Navigation Diagram tool into the prototype mode. The two modes are edit and prototype. During prototyping, the Navigation Diagram tool either presents you with a pop-up on which you specify the next procedure or displays a dialog. A dialog displays if the menu item or push button invokes a supplemental dialog.

Note: You cannot add or specify the characteristics for controls when the Navigation Diagram tool is in prototype mode. All the menu items in the Edit drop-down are disabled. To add controls or specify characteristics for them, select Mode, Edit.

Event Processing

Events are activities that occur during the execution of a GUI application, such as windows opening and closing, controls being clicked, and so on. These events can result from user interaction, from the GUI application itself, or both.

The CA Gen runtime keeps track of these events. Event processing is a notification mechanism. A CA Gen-generated GUI application can be notified after an event occurs. After the GUI application is notified, it can respond by executing procedure action diagram (PrAD) logic. You associate the PrAD logic with the event.

Notice that the notification occurs *after* an event has taken place. Event processing is not a capability of deciding what events do and do not happen. The application cannot change the event after it has occurred. The application can, however, be notified that an event has occurred and *then* process PrAD logic.

You can create the following types of events with the Navigation Diagram tool:

- Open
- Close
- Changed
- Click
- Double-click
- ScrollBottom
- ScrollTop
- User-defined

Different controls can be associated with different event types. For more information, see the *Toolset Help*.

Event processing does not change the overall process of creating applications, including client/server applications. It does make it easier, however, to design the user interface and create the PrAD logic as concurrent activities. You can chain back and forth between the PrAD logic and the Navigation Diagram tool conveniently.

As far as the design of client/server applications is concerned, events are only client based because they involve GUI applications. Event processing provides the capability to do more flows to a server because the flows could be from event actions. An application could flow to a server at almost any time based on the events that trigger the flow.

Why Use Event Processing

The following tasks can be accomplished with event processing:

- Control program execution at a more granular level by associating events to controls.
- More conveniently design the user interface and the associated PrAD logic.
- More conveniently provide functionality of an application.

Event Processing Characteristics

The following characteristics of event processing enable you to complete the tasks:

- Special PrAD statements defined for event processing can be included in the PrAD logic. The event processing statements execute just as any other statements would during the normal execution of the PrAD.

  Note: The CA Gen runtime components silently ignore any *failures* of the event processing statements and allow processing to continue. For example, CLOSING a window that is already closed does not produce an error; the CLOSE statement is ignored.

- Special blocks of PrAD logic, called event actions, can be created in a PrAD.
- You can create event actions in the Navigation Diagram tool or in the Action Diagram tool.

Event Actions

Event actions are special blocks of logic in a procedure action diagram (PrAD) that can execute separately from the main body of PrAD logic. An event action is an extension of a procedure step because it shares views with the procedure step. The main use of an event action is for the user interface functionality and is typically tied to the presentation. Action blocks, in comparison, are pieces of code that typically perform actions against the database and are not tied to the presentation. The executing application queues all events. Event actions are executed in sequence, never in parallel.

Event actions are reusable and can be associated with many different events within the same procedure step. In such a situation, the event action would typically have a CASE OF COMMAND structure. The commands in the CASE OF COMMAND structure would be associated with different events.

In the Navigation Diagram tool, you add event actions from the Events pop-up. The field Action Name on that pop-up specifies the name for the event action.

Event actions can be associated with windows, dialogs, and controls. The event action logic executes depending on the action taken with the window, dialog, or control. The action is a combination of what the user does and what the application does. For example, a user clicks a control. The click event sets an exit state that causes a flow which closes the window. The user did not directly close the window. But as a result of the click event by the user, the application logic closed the window.

Why Use Event Actions

Event actions enable you to control program execution at a more granular level by associating events to controls. In some cases, you can provide functionality of an application more conveniently with event actions than otherwise.

Event Action Characteristics

Event actions have the following characteristics:

- All event actions in a procedure step share the same views as the procedure step.
- We recommend you to perform entity actions in process or common action blocks and not directly in an event action. However, if you do choose to perform entity actions in an event action, you should limit them to READ and READ EACH only (no updates of any kind). Performing entity actions in action blocks also avoids the sharing of entity action views between event actions and between event actions and the procedure step. Sharing views may cause undesirable results with SQL optimization. For example, if you use the WHERE CURRENT OF construct with a READ statement, the READ action could fail. Additionally, if your logic performs update functionality (UPDATE, TRANSFER, ASSOCIATE, or DELETE statements), too many database locks may be generated. This does not affect data integrity, but response time may be unacceptable.
- By default, an event action moves import views to export views for all event types except the Changed event. With Changed events, the import view contains the current view of the data in the window and the export view contains the previous data. You can override the default for any event type by detailing an event action in the Action Diagram tool.
- Event actions never clear export views. When the main body of PrAD executes, the export views are cleared. Thus, an event action may begin execution with import values or export values that resulted from the execution of another event action.
- The logic in an event action executes without executing the logic in the entire PrAD. An event action cannot cause the procedure step in which it is included to execute. You can, however, set an exit state within an event action that causes a flow to another procedure step.
- CA Gen always places event actions after the main body of PrAD logic.
- The order of event actions in the PrAD is significant during Construction. Event actions produce entry points into the source file. Therefore, if you reorder event actions within a PrAD, you must regenerate the Window Manager and the procedure step.

- An implied commit point occurs when an event action completes execution, unless the event action sets an exit state that causes a rollback.
- An event action cannot be nested within another event action. A user-defined event, however, can trigger another event with the use of the special external action block TIREVENT. Refer to the online help for a complete discussion of user-defined events and TIREVENT.

Overall Appearance

The overall appearance of your interface helps the user work with the application. A layout that is consistent and aesthetically pleasing helps a user increase productivity. The following tasks can help you refine the appearance of the interface:

- Verify the implementation of any application-wide standards for the interface design.
- Verify the use of toolbars and status bars.
- Rearrange the controls in a manner that satisfies the object or action flow in which a user performs tasks.
- Group related controls together.
- Disable controls where appropriate.
- Create or update bitmap images to use as backgrounds and on controls.
- Verify the choices for fonts, foreground and background colors for windows, dialogs, and controls.

Toolbars and Status Bars

Only windows, and not dialogs, can contain toolbars or status bars. A toolbar is a visually separate area across the top of a window that can contain push buttons. A status bar is a visually separate area across the bottom of a window that can contain read-only controls. Neither the status bar nor its controls can receive input focus at runtime.

Size, Align, and Move Controls

After you create and lay out the controls for your window, you need to adjust the size and alignment of the controls. Sizing and aligning could be very tedious if you had to do them manually. The Navigation Diagram tool contains functionality that helps you adjust the sizing, aligning, and moving of controls.

The following table lists the functionality for sizing, aligning, or moving controls.

Functionality	Primary Use
Grid	Define a grid as a positioning and alignment aid. Cause an object to align (snap) to the nearest grid lines.
Move	Reposition a window or control.
Move Groups	Move a group box and all controls with which it intersects as a single object.
Position	Size other controls to the size of a reference control. Align controls vertically or horizontally. Separate controls vertically or horizontally. Move controls to the top, center, or bottom of a window.

Note: The size of fonts from platform to platform and the resolution of monitors can cause some overlapping or truncation of controls. You may notice this more if you develop an application on one platform and execute it on another.

The following suggestions can help you minimize or eliminate problems caused by different font sizes and monitor resolutions:

- Size the surrounding box around each control and prompt (the box that you see when you select a control) larger both horizontally and vertically.
- Change the font designation of the generated application on the target (execution) platform to match the font designation on the development platform.
- Position entry fields and their prompts with the Horizontal/Middle alignment option (select Edit, Position).
- Perform the design work on the lowest resolution monitor on which the application will execute. Controls sized for fonts on a low resolution monitor are usually sized adequately for a high resolution monitor, but not conversely.

Disable Controls

The design of your interface is more user-centered if you specify the conditions for which a user can and cannot interact with a control. Disabling a control prevents a user from interacting with it. You disable controls based on the conditions of other controls. For example, you can disable a push button until an entry field contains data.

Add Pictures

You can associate an event (click or double-click) with a bitmap object.

Bitmaps can also be the background for windows or appear on push buttons.

Note: CA Gen supports only 16 color (4-bit) and 256 color (8-bit) bitmaps. Most graphic software packages save bitmap images in one or both of these formats.

More information:

Event Processing (see page 78)

Fonts and Colors

The font and color specifications you make in the Navigation Diagram tool override any specifications you make in Business System Defaults. If you change the font for a window, all controls on that window inherit the font. If you change a color for a window, the following objects on the window inherit the color:

- Literals
- Group boxes
- Prompts
- Radio buttons
- Check boxes

Specify Fonts and Colors for a Window

The following procedure details how to specify fonts and colors for a window.

Follow these steps:

1. Open a model and select Design, Navigation Diagram.

   The Open Navigation Objects panel displays.
2. Select a business system from the Business Systems drop-down.
3. Select a procedure step from the Procedure Steps drop-down.
4. Select one or more Windows from the Windows List and click Include.

5. Click Open.
6. From the Navigation Diagram menu, double-click the procedure in the Network View.
7. Display the font and color properties pop-up:
   a. Select Detail, Video Properties.
   b. Select the display properties.
   c. Click OK.

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

Differences Between Design and Target Platforms

Because of differences in hardware and operating systems, applications may look or behave differently on the execution platform than on the design platform. The following points explain some of these differences.

- The size of fonts from platform to platform and the resolution of monitors can cause some overlapping or truncation of controls. These suggestions can help you minimize or eliminate problems caused by different font sizes and monitor resolutions:
  - Size the surrounding box around each control and prompt (the box that you see when you select a control or prompt) larger both horizontally and vertically.
  - Change the font designation of the generated application on the target (execution) platform to match the font designation on the development platform.
  - Position entry fields and their prompts with the Horizontal/Middle alignment option (select Edit, Position).
  - Perform the design work on the lowest resolution monitor on which the application will execute. Controls sized for fonts on a low resolution monitor are usually sized adequately for a high resolution monitor, but not conversely.
- Literals on the Windows platforms always align at the top of their surrounding box. Center Vertically is a property of literals.
- The Navigation Diagram tool supports only 16 color (4-bit) and 256 color (8-bit) bitmaps. Most graphic software packages save bitmap images in at least one of these formats.

- The Designed Position of a window or dialog can vary. Designed Position is a property of a window or a dialog. For applications executing on Windows, the position of the window or dialog at runtime is relative to its designed position on the design desktop. For example, suppose you design a window whose lower left corner is three inches horizontally and three inches vertically from the lower left corner of the design desktop. At runtime, that window will appear three inches horizontally and three inches vertically from the lower left corner of the runtime desktop.
- The behavior of the default push button when the Enter key is pressed varies by platform. On the Windows platform, the Enter key invokes the push button that has the focus regardless of which push button is the default. If the focus is on any other type of control, the Enter key invokes the default push button.

Construction

Construction of a GUI application produces a Window Manager file and associated files required by the target environment. These associated files can include a resource file, a help file, and a header file. The files produced have the same name as the load module to which they are associated. In addition, construction produces the standard files for a generated application, such as the source code files, the referential integrity library files, and the installation control monitor files.

Note: For more information about construction, see the *Workstation Construction User Guide*.

Chapter 8: Designing Screens

This section contains the following topics:

Screens

Each screen is associated with one online procedure step. A screen cannot be associated with a batch procedure step.

Note: Batch is only available on MVS.

More information:

Designing Dialogs (see page 49)

Block Mode Design

Screen design is often referred to as block mode. Block mode is a type of design that uses a static screen, one which cannot be resized the way a GUI screen can be. Unless a user is running an application in a window, block mode runs in what is called full-screen mode.

The following table summarizes the differences between the block mode design and GUI design.

Points to Consider	Block Mode	GUI
Single procedure step represented by	One screen	One or more screens

Points to Consider	Block Mode	GUI
Tool used	Keyboard	Mouse (primarily)
Type of design orientation	User-Data	Action-Object
Type of screen used	Static	Flexible

More information:

Designing Windows (see page 69)

Tasks and Objectives of Screen Design

Screen design basically requires that you repeat the following set of tasks until the screen you have created fits the needs of your target application:

- Create a template
- Create a screen
- Periodically preview a screen to be sure you are on the right track
- Make changes to the screen or template(s) as necessary

Use the Screen Design tool to accomplish the following objectives:

- Create screens that are easy to read and consistent across the business system
- Make the steps the user must follow clear and easy to implement

Prerequisites

Before designing screens, you need the following prerequisites:

- An understanding of the tasks that the user intends to perform with the application, which is usually determined from previous phases of development, such as life cycle analysis using the Entity Life Cycle Analysis tool
- A data model defined with the Data Model tool, Data Model List tool, or the Data Model Browser tool
- One or more procedures defined with the Dialog Design tool
- Views for one or more procedures defined in procedure synthesis using the Dialog Design tool
- A business system defined with the Business System Definition tool

Screen Design Display Objects

The Screen Design tool defines two types of display objects—templates and screens.

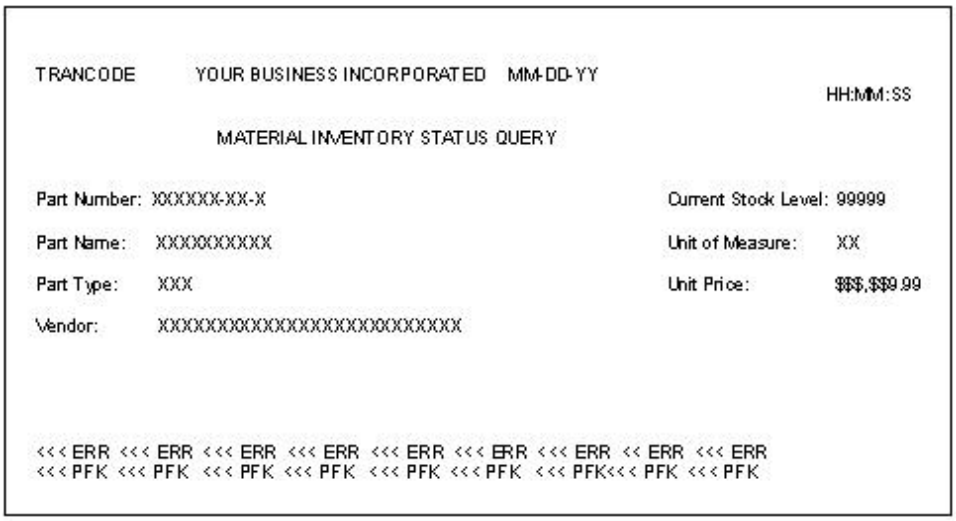
Templates are definitions of portions of a screen. A template can be used by many screens across a business system to define and position standard screen objects. In addition, one or more templates can be used in each screen.

Screens are the view of the system by the user and can contain one or multiple templates. A screen is associated with a single procedure step (or a single-step procedure) and is responsible for providing data to the import view of the procedure step and displaying data from the export view.

A screen in a generated system can appear in the following two states:

- Once as the system has formatted the export view as a response
- Once as the terminal operator has specified the data required in the import view

The following example shows a screen using a template.



Screens and templates may contain the following objects:

- Fields (screens only), which implement the attributes of import and export data views.
- Literals, which are constant information in text or symbols.
- Special fields, which reflect the set of special attributes (attributes not derived from business requirements but introduced during system design for a specific purpose). Special fields generally appear on all screens in the business system.
- Prompts, which are labels for fields and special fields.

Screen Design Guidelines

You can customize screens by specifying the location and characteristics (color, light, and intensity) of any field. The colors you define in the system defaults or override when you define screen objects locally will be the colors the user sees in the generated system. However, the display properties in the generated system will vary with the software and hardware used for development and display.

As a consequence, use the following guidelines as you design screens and templates.

- If you plan to use templates, build them first. Then build the screens in which you plan to use them.
- Since the Screen Design tool does not permit overlapping fields, add templates to the screen before adding fields. If you add a template and then determine you want to delete one or more literals or special fields the template provides, use the Delete command.
- Use field properties to highlight important information. Do not overuse highlighting as it might confuse the user.
- Arrange fields on the screen in the same order that the user is likely to use them. When both mandatory and optional fields appear, place the mandatory fields before the optional fields.
- Ensure that the order of fields on the screen reflects the reading order of the user; for example, in English, left to right and top to bottom.
- Be consistent across screens and screen types in format and sequence of fields.
- Use display properties to differentiate data fields from prompts and literals. Use discretion in color selection. Do not overuse color as tt makes screens difficult to read.
- Make literals consistent in text, color, and placement across screens in the business system.
- Use simple screen formats that convey information clearly and concisely. Do not crowd the screen; use additional screens when necessary.

At any stage in the design process, you can preview screens to see how they will appear to the end user.

More information:

Change Screens and Templates (see page 100)
Prototyping (see page 78)

Screen Design Tasks

To build screens and templates, you add or copy screen objects. The following table shows the suggested sequence of tasks and the related subtasks that you can perform to create templates using the Screen Design tool.

Each of these topics is discussed under the corresponding heading.

Step	Task	Subtasks to Consider
1.	Open a New Template	Define the template name. In the Description field, define how and where this template is used throughout the business system.
2.	Add a literal	Determine whether to use defaults for properties. Add literal properties. Determine positioning.
3.	Add a special field	Determine whether to use defaults for properties. Add a prompt. Add prompt properties. Determine field display length. Add field display properties. Add error display properties. Determine fill character. Define edit pattern. Add Help ID. Determine positioning.
4.	Copy a literal	Determine positioning.

The following table lists the tasks to create screens step by step.

Step	Task	Subtasks to Consider
1.	Use a template in a screen	Delete any unwanted literals or special fields contained in template.

Step	Task	Subtasks to Consider
2.	Add a field	Define import view. Add a prompt. Add prompt properties. Determine field display length. Add field display properties. Add error display properties. Determine fill character. Define edit pattern. Add Help ID. Determine positioning.
3.	Add a literal	Determine whether to use defaults for properties. Add literal properties. Determine positioning.
4.	Add a special field	Determine whether to use defaults for properties. Add a prompt. Add prompt properties. Determine field display length. Add field display properties. Add error display properties. Determine fill character. Define edit pattern. Add Help ID. Determine positioning.
5.	Copy a literal or repeating group	Determine positioning.
6.	Describe a screen	Include descriptions of source and destination procedure steps.

Step	Task	Subtasks to Consider
7.	Specify screen properties	Define screen name (changing the name changes the name of the associated procedure step and action block). Define Help ID. Determine whether to enable scrolling. Define whether unused occurrences are protected or unprotected. Define top on display-first return. Describe screen.
8.	View in different display modes	None.
9.	Run prototyping	None.

Prepare to Design Screens and Templates

To begin designing screens and templates, you access the Screen Design tool. Select the Screen Design tool only when a business system is defined or when Procedure steps exist for the business system.

Create Templates

Since most screens contain at least some common information, templates are a good way to minimize repetition and save design time. You can create a list of templates, and then include whichever ones you need in each screen.

Complete the following tasks to create a template:

- Add (and copying) literals
- Add special fields
- Add a description (optional but recommended)

When you create a template, the Screen Design tool displays each object you add in a different color:

- Literals appear in white
- Special fields appear in green
- Prompts appear in blue

Add a Literal

Literals are the constant information shown as text or symbols. An example of a literal is the title placed at the top of a screen.

Follow these steps:

1. Type a literal name.
2. Position the literal on the screen.

How to Create Screens

You can access automatic screen design from Design through the Action Diagramming, Dialog Design, and Screen Design Toolsets. In Action Diagramming and Dialog Design, CA Gen automatically generates a screen when you copy an action diagram or action block while substituting a new entity type.

Begin screen design after you create all the templates you think you may need. Each screen goes through the same basic steps, covering the iterative tasks in the following order:

- Use templates in a screen
- Add fields
- Add and copy literals
- Add special fields
- Add a description (optional but recommended)
- Change the screen

Use a Template in a Screen

Use the Uses option of the Detail command in the Screen Design tool to perform the following tasks:

- Select a template to be used in the current screen. You can use more than one template in the same screen, provided the literals or special fields on the templates do not occupy the same space.
- Identify the templates used in the current screen.

Add a Field

A field represents the implementation of an attribute. This means that you can add fields only to screens, not to templates. When you add a field, you must define and position it (except for hidden fields, which you only define).

Adding a field involves the following tasks:

- Add a field name
- Define a field
- Position a field

You can add individual fields, repeating groups of fields, and nested repeating groups. When you add repeating and nested repeating groups where the maximum cardinality exceeds the number of display lines, you should also add the special fields for scrolling (Scroll Location Msg, Scroll Amount Msg, and Scroll Indicator Msg) to either the template (if repeating groups are found on all screens) or the individual screen.

When you add nested repeating groups to a screen, you should plan the screen layout so that the presentation of fields reflects the subordination of one group to another. One way to do this is through indention of groups, similar to the indention of repeating group names on the Views panel during View Maintenance. CA Gen supports multiple levels of nesting, but only three levels can contain groups with a cardinality greater than one. Only the highest level of a repeating group can be scrolled.

Repeating group views can be mapped only to repeating group views. In addition, non-repeating group views can be mapped only to non-repeating group views.

A procedure step is associated with a single screen, which is responsible for providing data to import and export views of a procedure step. Since a field can represent both the import and export data views of an attribute, the field must be correctly mapped to both. During the definition of a field, you can specify the import and export views. During field definition, CA Gen automatically maps a matchable import view field to the export view chosen. In some cases, however, an export view cannot be mapped to a corresponding import view. In these cases, you should map the import view as not yet mapped. When you define a field with unmapped views, you should perform view mapping. If the field is contained only in the export view (only exported from the procedure step), the system maps the import view of the field to none.

Define a Field

You can only define a field after you have added the field, using Add Field. You then select the attribute, entity or group view to use as a field.

Use the Field Definition panel to define the following fields:

- Import view to be mapped to the export view.
- Prompt that accompanies the field. You can select an existing prompt, add a new prompt, or display the field without a prompt.
- Field characteristics-displayed or hidden.
- In an online system, you may need to save information between procedure step executions. To do this, you can hide the field that contains the information. This lets the data pass between the export view of the proceeding execution and the import view of the next execution without appearing on the screen. Any field identified as a hidden field must be associated with an import and an export data view; otherwise, the data cannot be passed between the executions of the procedure step.
- Length of the display and, for numeric fields, the number of decimal points.
- The length of the display is the number of character positions required to display the field on the screen. The display length may differ from the number of characters in the value because of the additional characters in the edit pattern. For example, the attribute Date Ordered for an entity named Customer contains a four-digit year, a two-digit month, and a two-digit day (attribute length of eight). For it to appear using the edit pattern YYYY-MM-DD, the display length is 10.
- The Screen Handler can accept only those numeric fields that have a length of 18 characters.
- Edit pattern to be used for field display. While it is preferable to keep edit patterns consistent across a business system, you may need to locally change an edit pattern on selected screens.
- The edit pattern is the way the information in a field is formatted. Edit patterns are used to make values in views more readable when they appear on a screen.
- Help identifier for the field. The identifier is associated with the text for the help system. The help identifier records information needed by the help system of the installation to locate help text for the screen.
- Field video properties.
- Error message video properties.

When setting field and error display properties, you can use system defaults or you can override the system defaults with local properties. You have the same options when setting properties locally as when setting system defaults.

For attributes within a group, you can define the fields individually or you can define all attributes in the group in one sequence. If you select a group view, CA Gen guides you through the definition of each field until all fields have been defined, including fields in lower level groups.

Describe a Screen

Describing a screen lets you explain its purpose. When you are describing a screen, you should include descriptions of the source procedure step and the destination procedure step, if any.

Position a Field

To position a field and its prompt, you use the incremental move box. The size of the box varies to reflect the number of characters in the prompt or field. First you select the attribute or group name from the views for the procedure step. You then position the prompt and field for each view selected. After you have positioned a field, you can review or redefine it only by using the Detail command from the Screen Design command menu.

The Screen Design tool prompts you for placement of the first occurrence of fields in both repeating and nested repeating groups. For nested repeating groups, you position the fields in higher level groups before the fields in lower level groups.

After you position the last field for the lowest level repeating group (one level for a non-nested repeating group). CA Gen prompts you to specify the options for the remaining occurrences in each group, starting with the lowest level group and working up to the highest level group. The number of occurrences depends on the cardinalities you assign to the group view during View Maintenance. You specify the remaining options for repeating group views on the Repeating Group Occurrences for Export panel.

The following table lists the options for repeating group views.

Repeating Field Positioning Options	Description
Number of additional occurrences	Defaults to the maximum cardinality minus the one current placement of the group.
Number of occurrences on each line	Lets you place columns of the group fields across the screen.
Number of blank columns between occurrences	Specifies the horizontal spacing of the group columns across the screen.
Display prompts with each occurrence	Specifies whether to include the prompt with each repetition of group fields. Default is not to redisplay it.
Spacing	Lets you specify vertical spacing (single, double, or triple lines) between each field.

Note: The path to the Repeating Group Occurrences for Export panel is Add Field\[set the product group or family](r)\OK\<prompt>(whatever you select)\OK. Then you place the first prompt and below it the corresponding field. When you have placed all prompts and fields in the first occurrence of the repeating group view, the Repeating Group Occurrences for Export panel appears.

You should carefully plan the spacing requirements for single-level and nested repeating groups. For example, in a repeating group where you place the prompt above the field, you may run out of vertical screen area if you choose to accompany each occurrence of the field with a prompt. For nested repeating groups, CA Gen positions only those occurrences in which all allowed levels of nested repeating groups can appear simultaneously.

Note: If the domain of an attribute is changed with the Data Modeling tool after the field for the attribute is positioned, the new domain can be reflected by deleting the field and positioning it again.

Define and Position a Field

You can add a field by defining and positioning it. For a repeating group, you continue defining and positioning fields until all fields in all groups have been positioned.

Add a Literal to a Screen

CA Gen lets you add literals to screens as well as templates.

Add a Special Field to a Screen

CA Gen lets you add special fields to screens as well as templates.

Copy a Literal or Repeating Group

To save time in preparing text for screens, you can copy literals and repeating fields to another part of the screen. To copy, you must have already added a literal or a repeating group. The Screen Design tool copies only literals and repeating groups, even if you select other objects to be copied.

Specify Screen Properties for Repeating Groups

If you add repeating groups, you use the Detail and Screen Properties commands to specify screen properties regarding scrolling. You have the following options:

- Allow scrolling. Scrolling lets the user scroll back and forth through a repeating group. For nested repeating groups (repeating groups within repeating groups), scrolling occurs on the highest level group. Multiple repeating groups placed on the same screen scroll synchronously.
- If the maximum cardinality of the entity represented by a repeating group exceeds the number of display lines, you should allow scrolling. If the cardinality is equal to or less than the number of display lines, scrolling is not necessary. The same is true for nested repeating groups.
- Allow or prevent the updating of unused occurrences of a repeating group. This prevents a user from scrolling forward when the last filled-in occurrence of the repeating group appears on the screen. If the user attempts to scroll forward, the NEXT command is sent to the procedure. If the unused occurrences are protected, the user cannot add occurrences to the end of the group. The default option is leaving unused occurrences unprotected to allow updates.
- Select which occurrence in the repeating group will be displayed on a return from a link. You can choose to redisplay the screen that was displayed before the link, or you can display the screen with the first occurrence in the repeating group. If you select the screen that was displayed before the link, you must have the dialog flow returns on properties set to Display First. The default option is the first occurrence in the repeating group.

You can select these options from the same panel.

More information:

Designing Dialogs (see page 49)

Screen Preview

After a screen is built, you can use Prototyping to preview what you have created.

After previewing a screen, you may want to fine tune the screen by making some changes to the contents of the screen or to one or more of the templates.

More information:

Prototyping (see page 78)

Change Screens and Templates

After creating a template or a screen, you can change it. The following table lists the tasks to fine tune an interface.

Task	Screens	Templates	Subtasks to Consider
Assign commands to function keys	x	x	Define whether message type is Standard or Default. Check reserved command names before assigning a command name.
Access view maintenance	x		Define whether any attributes should be deleted.
Center a screen object	x	x	None.
Change the name of the screen or template	x	x	None.
Delete an object	x	x	None.
Delete an unused prompt	x	x	None.
Specify whether to display function keys and commands on generated screens	x		None.
Move an object	x	x	None.
Redefine an object	x	x	To change a protected prompt, determine whether delete authority is needed.
Redefine a literal	x	x	Reposition literal after changing text.

Access View Maintenance

When using the Screen Design tool, you can access view maintenance only when you are working on a screen. For more information, see the *Action Diagram User Guide.* If a procedure step contains import and export views with cardinality greater than one and you have positioned attributes on the screen, do not change the structure of the fields. Do not move a view with positioned attributes into or out of a repeating group view. If such restructuring is required, delete the attributes from the screen first.

Assign Commands to Function Keys

Assigning commands to function keys provides the user of the generated system an easy means of invoking a command. The command, in turn, is tested for in the action diagram. You cannot assign function keys when designing a template.

The PF Key Definition panel lists the function keys and commands you have defined in Business System Defaults. When you add a command using the Screen Design tool, its type is Local - it applies only to the particular procedure step. The function key commands set in Business System Defaults are defined as either standard or default. You can override default commands in a screen, but not standard defaults.

When assigning command names to PF keys, keep in mind that CA Gen reserves certain commands, for example, RESET and RESTART. In addition, CA Gen recognizes the command names NEXT, PREV, TOP, and BOTTOM for scrolling. It also reserves the command names HELP and PROMPT for invocation of the help system of the installation.

Change Screen or Template Names

You can change the name of a screen within the screen. Similarly, you can change the name of a template from within the template.

The screen name is created when you name a procedure step in a Dialog Flow Diagram. When you use the Dialog Design tool to change the name of a procedure step or use the Screen Design tool to change the name of a screen, the change is reflected in both the DLG and the Screens and Templates panel (used for selecting screens). When you change the name of a template, the change affects only the template.

Delete Screens or Templates

When you delete a screen, the procedure step with which it is associated is *not* deleted. When you delete a template, all objects on the template are removed and do not appear on any screens that used the template.

Delete Screen Objects

The Delete option on the Screen Design command menu lets you delete one or more objects from a screen or template. If you delete an object that belongs to a template from a screen, all objects on the template are deleted *only* from the application of the template of that screen.

From screens, you can delete fields, prompts, literals, or special fields. From templates, you can delete literals and special fields.

When you delete a prompt, only the prompt is deleted. When you delete a field that was positioned with a prompt, the prompt is also deleted. When you select a field from a group, CA Gen also selects for deletion all the fields from the same entity that were positioned at the same time.

Unused Prompts

New prompts that are not used add to the size of the model and the number of objects the model must manage. If you have unused prompts, you should delete them.

Function Keys and Commands

Using the Screen Design tool, you can specify whether to display function keys and commands on a PF key line on generated screens.

Move Objects

You can move the following objects using the Screen Design tool:

- Objects positioned on a template only from within the template
- Objects positioned on a screen only from within the screen
- One or more objects on one or more consecutive lines

However, you cannot move objects to space that is already occupied.

Redefine Screen Objects

You can redefine objects positioned on a screen only from the screen. Similarly, you can redefine objects positioned on a template only from the template.

If you change the display length or edit pattern of a positioned occurrence of an attribute in a repeating group, all occurrences change. If you change the field or error properties, only the selected occurrence is changed.

When you change the width of a positioned field in a repeating group, avoid a length that extends the field past column 80. CA Gen wraps fields extending past column 80 to the next line.

You cannot change a protected prompt in a subset unless you have delete authority on the screens or procedure steps that use the prompt.

Redefine a Literal

When redefining literals, you can change the text and the display properties. After changing the text, you must reposition the literal.

The literal can contain up to 80 characters, including international characters.

If you size prompts or literals in one dialect, and then change the dialect and resize the prompt or literal, the larger size remains in effect for placement purposes (copy or move).

Suppose you have two dialects-Default and French. You create a literal under the default dialect with a size of 25 characters. Then, you select the French dialect and resize the literal to 10 characters. The literal you see in the Screen Design tool is 10 characters.

If you select the literal, however, and then select copy or move, the *box* that surrounds the literal is still 25 characters. The amount of space reserved for the literal remains 25 characters, which is the larger of the two sizes. This feature allows you to switch back and forth between dialects without causing spacing problems on the terminal screen.

Note that you can place the 10-character literal only as you could place a 25-character literal. Other fields, for example, cannot be closer to the 10-character literal than they could be to a 25-character literal.

Downstream effects

When a load module defined in Packaging contains multiple procedure steps, every screen in the load module must contain a unique identifying string as the first literal on the screen. Place this literal on the first line of the screen, following the transaction code.

The special field, panel ID, can be used instead of a unique literal. During screen generation, a literal equal to the member name of the screen is generated in the location of the panel identification.

Chapter 9: Refining Screen Design

This section contains the following topics:

Prototyping Tool

The Prototyping tool provides the ability to review the order and content of screens before detailing procedure logic and generating code. The PT tool can help you verify if your screens are complete and flow in the order that your business requires. By using the PT tool before refining the action diagrams for procedure steps, you can avoid costly changes later in the project.

Before using this tool, you must have at least one procedure step defined in the Dialog Flow Diagram and one screen defined in Screen Design for the application you are developing. On the Dialog Flow Diagram, you must detail the transfer or link between screens, using Detail and Flows On or Detail and Returns On options. You must have also assigned at least one command as an autoflow command.

More information:

Designing Dialogs (see page 49)
Designing Screens (see page 87)

How You Access the Prototyping Tool

After completing the minimum components of the Dialog Flow Diagram and Screen Design, you can access the Prototyping tool by clicking Design, Diagram, Open and selecting a beginning step (screen) from which to begin the prototyping session.

You can also chain to the PT tool from the following tools:

- Dialog Design
- Screen Design
- Action Diagram
- Action Block Usage

How You Use the Prototyping Tool

While you are reviewing the screens using the PT tool, you can interactively change screen definitions and the order in which screens are presented and review the results immediately. For example, if a user is reviewing the screens with you and suggests that one screen precede another, you can make that change by chaining to the source tool, making the modification, and then accessing Prototyping with the updated screen order. This ensures that you and the user agree on the change.

The objective of this tool is to further refine the screen definition. Therefore, when you build action diagrams for procedure steps, you will be working with stable information which has had the prior approval of the user and management.

You cannot enter data, interpret data, transfer data between screens, or simulate the look of data on the screen in the PT tool. These tasks are done in the CA Gen test facility. However, the edit pattern displays with each data field. The cursor position does not change to indicate which fields are protected and which will allow data entry. Also, error messages do not display when using this tool.

Select Another Beginning Screen

You can restart Prototyping to change to a different beginning screen or to start over with the session you are currently running.

Follow these steps:

1. Select the space bar to exit the Prototyping session.
2. Select the procedure step which is to be the new beginning screen.

When you have completed the review of the screens and verified any changes, you can chain to the Dialog Design tool, Screen Design tool, the Structure Chart tool, or the Action Diagram tool.

Chapter 10: Business System Defaults

As part of the project-planning task for a Design project, you can define standards known as business system defaults. Business system defaults, used with all design tools, provide a consistent user interface to the system you are constructing.

This section contains the following topics:

Consistency in Construction

You make all business system default selections from pop-up menus or panels that appear after you access Business System Defaults from Design. Each menu selection in Business System Defaults has a menu with its own set of tasks or panels. The panels contain fields to be completed or selected.

You can override business system defaults in most cases at the procedure step, screen, or template level.

Prerequisites

Before you begin defining system defaults, be sure you have either completed or have an understanding of the following prerequisites:

- A data model, which is defined with the Data Model tool, Data Model List tool or the Data Model Browser tool
- One or more procedures defined with the Dialog Design tool
- Views for one or more procedures defined with the View Maintenance tool
- A business system defined with the Business System Definition tool

Note: For more information about defining business systems, see *Analysis Tools Reference Guide*.

How to Set Business System Defaults

After accessing Business System Defaults, you can choose from the options shown in the following table. Each option is discussed under the corresponding heading.

Option	Allows You To
Properties	Rename or describe system.
Video Properties	Set the screen video properties such as the display properties of screen prompts, fields, error messages, and literals. Set the window video properties such as the color and font characteristics of windows and dialogs.
Commands	Set command names and the synonym associated with each name.
Exit States	Set exit states for action diagrams and dialog flows.
Function Keys	Assign commands to function keys.
Special Edit Patterns	Add editing patterns that control field display and input formats for special fields, such as system date, time.
Field Edit Patterns	Add editing patterns that control field display and input formats for other screen objects.
Delimiters	Specify the string separators and parameter delimiters that identify the separation of data items.

Set Properties

The Properties option lets you rename or describe the business system. To reach this option, you must first select a defined business system.

Set Video Properties

The Video Properties option lets you set screen video properties and window video properties.

Set Screen Video Properties

Click Screen in the Video Properties dialog tree view to set screen video properties. Screen video properties determine the display characteristics of the following controls:

- Prompts
- Fields

- Errors
- Literals

Set Prompt Properties

A prompt is a field label that indicates the contents of the field. You can set prompt display properties to distinguish them from the fields. See the following table for the options available. Setting the display properties defines how the generated system will display screens to the user.

Prompt or Literal Properties	Comment	Initial Default	Other Options
Color	Determines color or screen display characters. Used only on terminals with extended color and highlighting features.	Normal	White, Cyan, Yellow, Green, Pink, Red, or Blue
Intensity	Determines brightness of display.	Normal	High or Dark (no display; use for hidden prompts or literals)
Highlight	Determines how characters are highlighted. Used only on terminals with extended color and highlighting features.	Normal	Underscore, Blink, or Reverse Video
Justify	Specifies how data is to be aligned within the field when input.	Left	Right or Center

Set Field Properties

A field is an area of the screen in the generated system where an attribute value is displayed for, or entered by, the user. You define fields in terms of their display properties in order to distinguish them from other types of screen objects. See the following table, which identifies and explains the field properties you can set.

Field Properties	Comment	Initial Default	Other Options
Color	Determines color of display characters. Used only on terminals with extended color and highlighting features.	Normal (default for monitor)	White, Cyan, Yellow, Green, Pink, Red, or Blue
Intensity	Determines brightness of display.	Normal	High or Dark (no display)
Highlight	Determines how characters are highlighted. Used only on terminals with extended color and highlighting features.	Normal (default for monitor)	Underscore, Blink, or Reverse Video

Field Properties	Comment	Initial Default	Other Options
Justify	Specifies how data is to be aligned within the field when input.	Default (see Note 1)	Left or Right
Put Cursor Here	Indicates where user is to enter data when generated screen appears.	No	Yes
Protected	Prevents user from entering data in protected field.	Off	On
Blank when Zero or NULL	See Note 2.	No	Yes
Fill Character	Used for input of fields shorter than defined field length.	Blank	Type desired fill character

Note:

- Selecting Default lets you left-justify alphabetic text and right-justify numerals.
- If you select No and no data is present, blank numeric fields will be filled with asterisks (*) and blank text fields with underscores (_). If you select No and data is present, the import view will display 0 for numeric fields, spaces for text fields, 0 for date and time fields, and spaces for commands. If you select yes, spaces will appear where there are blanks and zeroes.

Set Error Properties

Error fields can be displayed with unique properties to distinguish them from other fields. The options you select should differ from those for the corresponding field properties for intensity, color, and highlight. The following table describes the error properties and their associated display options.

Error Properties	Comment	Initial Default	Other Options
Color	Determines color of characters used to display error. Used only on terminals with extended color and highlighting features.	Red	Normal (default), White, Cyan, Yellow, Green, Pink, and Blue
Intensity	Determines brightness of display.	High	Normal, Dark (no display)
Highlight	Specifies highlighting for characters in error. Used only on terminals with extended color and highlighting features.	Reverse Video	Normal (default), Underscore, Blink

Error Properties	Comment	Initial Default	Other Options
Justify	Specifies how data is to be aligned within the field when input.	Left	Right, Default
Put Cursor Here	Indicates where user is to enter data when the field is in error.	Yes	No
Protected	Prevents user from entering data in protected field.	Off	On

Set Literal Properties

A literal is constant information in text or symbols. Literals can be used on both screens and screen templates. Literals can be column headings, lines between columns, or other information that cannot be manipulated. Display properties must be set for literals so they can be distinguished from other screen objects. The property options available to define literals are the same as those used for prompts.

Set Window Video Properties

Click Window in the Video Properties dialog tree view to set window video properties. Window Video Properties specifies the color and font characteristics of an object on all of the windows or dialogs in a business system. If you change a font or color for a window, some or all of the controls of that window are affected.

The following table lists window controls affected by font or color changes.

Control	Affected by Font Change	Affected by Color Change
Group Box	Yes	Yes
Literal	Yes	Yes
Menu Bar	Yes	Yes
Prompt	Yes	Yes
Push Button	Yes	No
Status Bar	Yes	Yes
Toolbar	No	No
Entry Fields	Yes	Yes
Hypertext Links	Yes	Yes
Dialog Box	Yes	Yes
List Box	Yes	Yes
Window	Yes	Yes

In general, you can override the font or color selection for an individual control by selecting a different font or color.

More information:

Designing Windows (see page 69)

Set Commands

A command provides a way for you to direct the execution of a procedure, which, in turn, implements one or more processes. During dialog flow diagramming, you specify the command that tells the procedure step which action block to execute. You can use a command to call an action block that was created in Analysis or Design. When setting system defaults, you can assign the command name. You can also assign a synonym to the command name that the system will recognize and use to invoke the command (for example, A for Add). You can add, change, and delete command names and synonyms.

Note: RESET, RESTART, HELP, AND PROMPT are reserved words and cannot be used as command names. CA Gen reserves the command names HELP and PROMPT to invoke the help system of the installation. CA Gen recognizes the command names NEXT, PREV, TOP, and BOTTOM for scrolling.

Set Exit States

For data and activity control to flow between procedure steps, a condition called an exit state must be met. An exit state is the trigger that causes transfer of control. Exit State is the name of a CA Gen special attribute that links the Action Diagram and the Dialog Design Diagram. You can add, change, or delete exit states. For example, if a transaction code must be validated before another procedure can occur, you can add an exit state called INVALID_CODE if the code does not exist. You can set global exit states (exit states that apply across the model) or local exit states that apply only to the current business system.

You can set the termination action of an exit state-the action to be taken when the exit state occurs. The following table describes the types of termination actions.

Termination Action	What System Does When Exit State Occurs
Normal	Displays the message associated with the exit state only.
Rollback	Backs out the database update, displays a message associated with the exit state, and invokes the flow of control.

Termination Action	What System Does When Exit State Occurs
Abort	Aborts processing and executes a system dump.

In specifying the properties of an exit state, you can add, change, or delete a message that is associated with the exit state and displayed when flow does not occur. You can also assign the type of message.

Each type of message displays with unique properties. The following table shows the message types you can assign, what each message describes and how it appears on different types of monitors.

Message Type	Appearance on Color Monitors	Appearance on Monochrome Monitors	Description
None	Normal color (the color of fields on the screens)	Normal	Message type has not been defined. This option allows compatibility with messages created in earlier releases.
Informational	White	High Intensity	Message gives information only; no action is required.
Warning	Yellow	High Intensity	Message advises that user action may be required as a result of processing.
Error	Red	Highlighted	Message advises that an error occurred in an entry and action is required to correct.

You can define and assign these messages when defining system defaults, when diagramming dialog flows, or when defining action diagrams. For example, you can add the message CODE IS INVALID and associate it with the exit state INVALID_CODE. The message would display when the logic of the action diagram causes the exit state to occur.

Set Function Keys

You can use a function key (also known as a program function key or PF key) as a shorthand way to issue a command to a procedure in an online environment. For example, if you map PF key 2 to the Add command, you can invoke the Add command with one keystroke by pressing PF2.

When you associate a command with a function key, you can define the type as Default (can be overridden) or Standard (cannot be overridden). A Default command allows flexibility. For example, if a particular procedure does not have the function currently mapped to a function key, you can map another command to that function key for that procedure. A Standard command is assigned to its function key in every procedure step in the business system. For example, you could assign commands so that PF1 always invokes Help, regardless of the procedure you execute.

Set Edit Patterns

An edit pattern is a defined mask that controls the display and input format for fields. Edit patterns are classified as text, numeric, date, time, or timestamp. Each edit pattern class is associated with a set of supported edit pattern components. Some components display variable data, others contain formatting instructions, while others display the literal component itself. Any two components may be separated by a space or a supported literal character such as a slash (/), hyphen (-), period or decimal point (.), or colon (:).

The Business System Defaults menu lists two edit pattern options:

- Special Edit Patterns, which are used by the system to format System Date, System Time, Scroll Indicator, and Scroll Location.
- Field Edit Patterns, which are used to define edit patterns for data fields. Five edit patterns have already been created for your use. Their names are Date, Time, Timestamp, Quantity, and Currency. Quantity and Currency are both instances of the numeric class.

You can add new edit patterns for fields with data display requirements different from the displayed defaults. You can delete or modify any edit pattern that is supplied or that you add.

Edit patterns for a date typically include three components: year (YYYY), month (MM), and day (DD). The edit pattern, YYYYMMDD represents a style where the 4-digit year, the 2-digit month (with leading zeroes), and the 2-digit day are displayed contiguously, that is, without separators. The default edit pattern for Date is YYYY-MM-DD, where the hyphen is used as the separator; the default for Time is HH.MM.SS, where the period is the separator. Timestamps are stored internally as a combination of all the date and time values, plus microseconds. This value is stored in the following edit pattern: YYYYMMDDHHMISSNNNNNN.

The following table lists separators by class and description.

Class of Edit	Edit Pattern Separators	Separator Description
Date, Time, Timestamp, Text. Mixed Text, and DBCS Text	A hard space or one of the following symbols: / (slash) . (period) : (colon) - (hyphen)	A literal character or space used to separate the components of the edit pattern.
Numeric	. (decimal point)	The character used to separate the integer component from the mantissa component of an edit pattern for a decimal number.
	, (comma)	The character used to separate periods of a number over 999 and displayable after start of significance. Entry is optional.

The following table describes components that can be used when specifying an edit pattern for the following types of data: numeric, text, date, time, or timestamp. You may use these components to build new edit patterns or modify those that exist.

Class of Edit	Edit Pattern Components	Component Description	Valid Value and Default Value
Date and Timestamp	C	1-digit century code	Valid values: 0 through 9 inclusive, representing centuries as follows: 0 - 1900 1 - 2000 2 - 2100 3 - 2200 4 - 2300 5 - 2400 6 - 2500 7 - 1600 8 - 1700 9 -1800

Class of Edit	Edit Pattern Components	Component Description	Valid Value and Default Value
	YYYY	4-digit year	Valid values: 1-, 2-, 3-, or 4-digit number, where leading zeroes are significant. The value must be between 1600 and 2599, or 0 (zero). Default value for omitted leading digits: correct digit for current year Default value for unspecified component: current year unless specified date components are zeroes, then 0000.
Date and Timestamp	YY	2-digit year	Valid values: 2-digit number, where leading zeroes are significant. No default; if specified in the edit pattern, a value must be entered.
	MM	1- or 2-digit number for month of the year	Valid values: 1 through 12 representing the month, or 0 (a special case). Default value for unspecified component: current month unless specified date components are zeroes, then 00.
	DD	1- or 2-digit number for day of the month	Valid values: 1 through 31 representing the day, or 0 (a special case). Default value for unspecified component: current day unless specified date components are zeroes, then 00.
Time and Timestamp	HH	1- or 2-digit number for hour of the day, using 24-hour notation.	Valid values: 0 through 23; Default: 0
	MI	1- or 2-digit number for minute of the hour.	Valid values: 0 through 59; Default: 0
	SS	1- or 2-digit number for second of the minute.	Valid values: 0 through 59; Default: 0
Timestamp	N, NN, NNN, NNNN, NNNNN, NNNNNN	1- to 6-digit microsecond value, where the number of Ns determines precision. The value is aligned to an implied decimal point.	Valid values: 0 through 999999, where leading zeroes are significant and trailing zeroes are insignificant. Default: 000000

Class of Edit	Edit Pattern Components	Component Description	Valid Value and Default Value
Text	X	One or more 'Xs, where the number of 'Xs, literals and separators does not exceed the field length. Spaces are significant.	Valid values: All characters in the current codepage, including alphanumeric characters and symbols but excluding DBCS characters. Default: nothing
	A-L, N-Z, 0-9, No DBCS Characters	Literal text composed of all alphanumeric characters except 'X'.	Valid value: as is.
Mixed Text	M	One or more 'Ms, where the number of 'Ms, literals and separators does not exceed the field length. Spaces are significant. A single 'M' represents one byte. Two 'Ms, are required to hold a DBCS character.	Valid values: All characters in the current codepage, including alphanumeric characters, symbols and DBCS characters. Default: nothing
	A-L, N-Z, 0-9, Any DBCS Character	Literal text composed of all alphanumeric characters and DBCS characters with the exception of the single byte 'M'.	Valid value: as is.
DBCS Text	DBCS:X	One or more DBCS:'Xs, where the number of bytes of DBCS:'Xs, literals and separators does not exceed the field length. Spaces are significant.	Valid values: All DBCS characters in the current codepage. All single byte characters are excluded. Default: nothing
	Any DBCS Character	Literal text composed of any DBCS character.	Valid value: as is.
Numeric	*	Asterisk line leader fill character preceding data	Valid values: as needed to fill field.
	$	Floating dollar sign	Valid value: as is.

Class of Edit	Edit Pattern Components	Component Description	Valid Value and Default Value
	9	Numeric characters, including leading zeroes	Valid values: 0 through 9, inclusive
	Z	Numeric characters, excluding leading zeroes	Valid values: 0 through 9, inclusive; 0s are suppressed until significance has been established.
	0	A literal zero	Valid value: as is
	+	For signed data, displays both plus sign (+) for positive data and minus sign (-) for negative data.	Valid values: + (plus sign) and - (minus sign)
	-	For signed data, displays minus sign (-) for negative data.	Valid values: - (minus sign)

Rules for Using Established Edit Patterns

Follow the rules listed to use established edit patterns:

- When separators are not used between components in the date, time, or timestamp edit patterns, then the entry must include all digits of each component. Any white space included in the entry is ignored.

Edit Pattern	Entry	Result
YYYYMMDD	19930801	Accepted: 19930801
	1993 08 01	Accepted: 19930801
	20010104	Accepted: 20010104

- When date components are separated by valid separators in the edit pattern, it is not necessary to enter leading zeroes for components with values between 1 and 9, inclusive.

Edit Pattern	Entry	Result
YYYY/MM/D	1993/8/1	Accepted: 1993/08/01

Edit Pattern	Entry	Result
	2003/8/1	Accepted: 2003/08/01

- When YYYY is part of an edit pattern defined with separators, you may omit up to the first three characters of the year. The value for these characters will be taken from the date that is set in the system clock. That is, if your system clock is set to 1999, and you enter 83 for the year, the year defaults to 1983. If you enter only 3, the date defaults to 1993. If your system clock is set to 2000, the dates, in these examples, would default to 2083 and 2003 respectively.

Edit Pattern	Entry	Result
YYYY/MM/DD	89/11/26	Accepted: 1989/11/26 or 2089/11/26
	3/8/1	Accepted: 1993/08/01 or 2003/08/01

- When components are separated by valid separators in the edit pattern and you intend to accept defaults, you must enter at least one digit for each component.

Edit Pattern	Entry	Result
YYYY/MM/DD	3/08/01	Accepted: 1993/08/01 or 2003/08/01
	/08/01	Rejected: No value is specified for the year component.

- If multiple components of an edit pattern target the same value, then the rightmost component overrides. For example, the century code (C) and the 4-digit year code (YYYY) both supply the two most significant digits of the 4-digit year where:
  - C = 0 represents 1900,
  - C = 1 represents 2000,
  - C = 2 represents 2100.

Edit Pattern	Entry	Result
YYYY/MM/DD-C	1993/08/01-2	Interpreted: 2193/08/01

- A date value of all zeroes is a special case. If specified components are set to zeroes, then the unspecified components used for the internal timestamp are also set to zeroes. In the following example where YYYY is unspecified, the 4-digit year value is interpreted as 0000 for the associated timestamp.

Edit Pattern	Entry	Result
MM/DD	00/00	Timestamp: 00000000

Set Delimiters

A delimiter is a punctuation symbol that separates data items. In Design, delimiters refer to the parameter delimiters and string separators you can use during clear screen.

Parameter delimiters include spaces, commas (,), periods (.), semicolons (;), slashes (/), back slashes (\), carets (^), dashes (-), asterisks (*), and plus signs (+). String separators include double quotations (“?”), parentheses ((?)), brackets ([?]), braces ({?}), and single quotations ('?').

In string data that contains symbols that may be interpreted as parameter delimiters, enclose the parameter with string separators. For example, for the following string data:

```
John Smith, Jr.
```

Use a string separator such as double quotations, as shown in the following example:

```
“John Smith, Jr.”
```

This solution eliminates problems in parameter interpretation that may be caused by both the comma and the period.

More information:

Designing Dialogs (see page 49)

Reports

Four reports are available from the Diagram action on the Business System Defaults main window:

- Command List lists commands and synonyms defined for the business system.
- Command Uses (Command Where Used) lists one or more commands and the procedure steps by which they are used.
- Exit State List lists the exit states and messages defined for the business system.
- Exit State Uses (Exit State Where Used) lists one or more exit states and the processes and procedure steps by which they are used.

Chapter 11: Additional Design Tools

This section contains the following topics:

Event Browser

The Event Browser displays in a dialog format the event objects created during window design using the Event Processing tool. An event is a notification that something has occurred. It includes opening and closing windows, clicking on an item, and so on. An event action is a special block of logic in a PrAD that can execute separately from the main body of PrAD logic.

The Event Browser tool lets you do the following tasks:

- View logic for an event action and change it by chaining to Action Diagramming
- Determine which controls have had events defined for them and what types of events they are
- Create or change a control

Note: There is also a user-defined event type. For more information, see the *Toolset Help*.

More information:

Designing Windows (see page 69)

Work Set List

A work set provides information that a process, procedure step, or action block requires but cannot get from entity types in the model. Attributes (or characteristics) of work sets are like entity type attributes. A work view is a view of a work set that does not create permanent storage. A work set list displays a list of all work sets in a given model. The Work Set List tool implements a work view from a work set.

Work set lists let you use work sets to do the following tasks:

- Monitor execution time information, such as calculation of intermediate totals and counts
- View applications requiring information that is not available from entity views in the data model

Structure Chart

The Structure Chart tool shows the relationship between implemented action blocks and procedure action diagrams. This lets you do the following tasks:

- See the entire action diagram structure without chaining between action diagrams
- See procedure step action block decomposition in different ways
- Match import and export views of action blocks called by the USE command

Action Block Usage

The Action Block Usage tool creates a diagram that illustrates how an action block is connected to other action blocks, processes, and procedure steps through the USE command.

This tool eliminates the need to chain between action diagrams and is useful in performing the following tasks:

- Showing the different ways an action block is used in a model
- Providing a simple way to match import and export views of action blocks called by the USE command.

Dialect Definition

The Dialect Definition tool lets you design multilingual applications for screens in more than one language (each model has a default dialect). Using this tool, you can do the following tasks:

- Add dialects with the Dialect Definition tool and use the Dialect Properties pop-up to define the various properties, including dialect name, the direction the text will flow, separators and the scroll amount.
- Size prompts or literals in one dialect, then change the dialect and resize the prompt or literal in a smaller size without causing spacing problems on the terminal screen.

Note: Multiple dialects are supported in both screen design and window design functionality.

The following table lists other design tools.

Tool	Used In	Used by Code Generator	Also See
Event Browser	Navigation Diagram	Y	Event Processing
Work Set List	Navigation Diagram, client/server or Screen Design	N	
Structure Chart	Navigation Diagram, client/server or Screen Design	Y	Action Diagram
Action Block Usage	Navigation Diagram, client/server or Screen Design	Y	Action Diagram
Dialect Definition	Navigation Diagram, or Screen Design	Y	

Note: For more information about any design tool, see the *Toolset Help*.

Appendix A: z/OS Block Mode Screen Design

This section contains the following topics:

Operational Behavior

Much of the underlying operational behavior of the z/OS Block Mode Screen Manager is controlled in the way that the user defines screens and constructs edit patterns. Some of the behavior, however, can be controlled by modifying the generated source materials.

The Enhanced Screen Generator is the default for all z/OS block mode applications developed with CA Gen.

More information:

Numeric Edit Pattern Restrictions (see page 152)

3270 Information Display System

The screen presented to the user is formatted using the facilities and capabilities of the IBM 3270 Information Display System or a compatible terminal and controller.

Note: For more information about 3270 Information Display System, see the *IBM Programmer's Reference Documentation*.

Initial Write Compared to Subsequent Write

The initial write of a screen is the case where the screen must clear what may exist on the screen and write the first instance of its image on the screen. After it is written, the image is retained in the device and control unit buffers until it is changed or erased. The CA Gen implementation rewrites the content of fields that change. The software does not redefine the fields, nor does it rewrite the entire screen.

Additionally, on the initial write, CA Gen uses either the Erase/Write or the Erase/Write Alternate device commands. These commands first clear the buffer of all formatting information and set all display positions to nulls, which are displayed as blanks. Then, the data stream is processed to write new formatting and data into the buffer.

Data Stream Options

The generated screen manager program contains a Screen Table that controls the Data Stream Options and Alarm Options. The following five options can be changed by modifying the Screen Table in the generated code:

- Repeat to Address (RA) Order
- Set Attribute (SA) Order
- Program Tab (PT) Order
- Erase/Write Always
- Alarm Options

A value is assigned to each option in the screen table. To suppress any combination of these options, add the value assigned to each option. Enter the total in the value field for the Data Stream Options.

To control the alarm, set the value of the alarm options to the appropriate value.

```
03 DATA-STREAM-OPTIONS PIC S9 (4) VALUE 0 COMP.
88 EXTENDED ATTRIBUTES VALUE 1.
88 BINARY ADDRESSES VALUE 2.
88 SUPPRESS RA ORDER VALUE 4.
88 SUPPRESS SA ORDER VALUE 8.
88 SUPPRESS PT ORDER VALUE 16.
88 ERASE-WRITE-A;LWAYS VALUE 32.
03 ALARM OPTIONS PIC S9 (4) VALUE 0 COMP.
88 ALARM-ALWAYS VALUE 1.
88 ALARM-ON-ERROR VALUE 2.
```

Repeat to Address Order

The Repeat to Address (RA) order compacts the data stream by eliminating repeated characters, specifying the ending buffer address and the character to be repeated. The RA order can be used at any point in the data stream and can be used within a field at any point in the data.

The RA Order, address, and character are four bytes long. CA Gen uses the RA order when the repeated data exceeds four bytes.

The RA Order can be inserted any number of times into a stream of data, even when the data is for the same field.

For example, the string:

AAAAAAABBBBBBBCCCCCCC

Can be reduced to:

SA addrA SA addrB SA addrC

Where the address in each case indicates the buffer address of the last occurrence of the repeated character. This example saves nine bytes in the data stream. When combined with other fields, the savings can be significant.

When the RA order is suppressed by the user, the runtime does not reduce repeated strings. Instead, the runtime software sends the entire string unchanged to the device.

To suppress the RA order, modify the Screen Table in the generated screen manager program. Add 4 to the value currently shown for the Data Stream Options.

Suppressing this feature may make it possible to use non-IBM 3270 systems that do not support this order.

Set Attribute Order

The Set Attribute (SA) order can modify the highlighting, color, or character set used to display individual characters in a field. This supports enhanced error detection and user interface improvements.

When the SA order is suppressed, individual characters that are in error are not highlighted. Also, Shift Out (SO) and Shift In (SI) control codes are not suppressed in mixed DBCS/SBCS fields. This means unexpected spaces appear in mixed strings containing single byte and multiple byte characters. The spaces are displayed in place of the SO and SI control codes that mark the transition between single byte and multiple byte characters.

Note: When the TP monitor is IEFAE and extended attribute support is turned off, IEFAE supplies default colors for screens. The default colors cannot be changed using the video properties dialog.

Program Tab Order

The Program Tab (PT) order advances the current buffer address and optionally clears the buffer. This works well for clearing a field of its contents on an already formatted screen without transferring a large amount of data in the data stream. This is especially true for scroll areas where only a partial view is valid.

CA Gen runtime uses a PT order to clear fields when the result should be a blank field. If other data replaces the field, use a PT order at the end of the new data to clear the remainder of the field.

Consider the example of a field defined as 40 bytes in length. It currently contains a 32 byte string, which will be replaced by an eight byte string. The PT order can be appended to the eight-byte string in the data stream to clear the remainder of the existing field. In this mode, the PT order acts like a user placing the cursor in a field and pressing the Erase/EOF key. The current buffer address is advanced to the next field. The runtime requires a Set Buffer Address (SBA) order to move the current buffer address.

When the PT order is suppressed, the runtime software sends the entire string, padded with spaces or Nulls, to clear fields.

To suppress the PT order, modify the Screen Table in the generated screen manager program.

Suppressing this feature makes it possible to use non-IBM 3270 systems that do not support this order.

Erase/Write Always

The Erase/Write Always option forces the runtime software to perform an Erase/Write or an Erase/Write Alternate for all screen I/O, not just the first screen.

Device Alarm

If a device alarm is installed, it can be activated when data is entered in any field, or only when there is an error. Edit the screen table in the generated code to modify the Attribute State Flag for the Alarm Options.

To always sound an alarm, set the value to 1. To sound an alarm only when there is an error, set the value to 2.

External Data Representation

The external representation of data is the way data appears after the internal data is transformed to a character representation and displayed with the appropriate punctuation including commas, decimal points, and currency symbols.

Edit patterns

All fields have an edit pattern. If the user does not select an edit pattern, the generator synthesizes an edit pattern that matches the characteristics of the field and variable data types.

Character Sets

The 3270 display system defines a one-byte Logical Character Set ID (LCID) for each character set loaded or installed in the device. In addition, some of the LCIDs are reserved for permanently installed character sets and loadable character sets.

Loadable character sets are control memory that can be programmed by the user application by the Load Programmed Symbols structured field. Also, whether the character set is defined for a one-byte or two-byte codepoint is reserved as well. When using two-byte codepoints, select an LCID that defines a two-byte character set.

The CA Gen runtime software does not load loadable character set buffers (LCIDs) and does not use these LCIDs if present. DBCS character sets are always assumed to be in LCID x'F8', and the SBCS character set LCID is the device default, x'00'.

The assigned character set buffers (LCIDs) are:

- x'00' Default character set
- x'40'-x'EF' Loadable character sets
- x'F0'-x'F7' Non-loadable single byte character sets
- x'F8'-x'FF' Non-loadable double byte character sets

All 3270 character sets define 190 printable characters (glyphs). These characters are assigned codepoints x'41' - x'FE'. The first byte of a double byte character set also is restricted to the range x'41'-x'FE', and can be thought of as a selector of one of the 190 glyph sets. Therefore, DBCS characters can represent a maximum of 1+190X190 characters, or 36,101 characters (x'4040' is always a space, and is not part of the 190 groups of 190 characters).

The operation of double and single byte characters depends upon the character set for the field, the Input Control settings, the use of SO/SI characters on write to the device, and the use of SA order.

When a single byte or mixed field is created, the Start Field Extended (SFE) order sets the default character set LCID to x'00'. This selects the installed default for the target device. Although the default character set may differ from one device to another, x'00' always selects the installed default.

Typically, the default character set is a single byte character set, but it may not be English. This can be determined by querying the installed character sets of the terminal, by user definitions, or other mechanisms. The Character Set Descriptor self defining terms received on the inbound Query Reply (Character Sets) structured field contains the LCIDs of each installed character set, its characteristics (loadable, not loadable, number of panes, and so on), and its Coded Graphic Character Set Global ID (CGCSGID) and Coded Character Set ID (CCSID). The CGCSGID and CCSID identify the codepage number of the international standard codepage associated with that character set. Additionally, CICS queries each terminal and retains that information, allowing applications to query CICS to obtain it, rather than query the terminal. Under CICS, the standard API services should be used. Under TSO/CMS, the Query can be sent to the device to determine this information (as defined in the TIRDCX) exit.

Codepage Translations

Codepage translations also take place when querying the installed character sets on a terminal. When the application is generated, the generator stores the codepage ID of the application in a constant data area made accessible to the runtime. This codepage ID is the codepage in use when the screen was modeled. It is the codepage of all literals, edit patterns, and character data in the application referenced by the runtime.

The codepage of the terminal is determined directly from the device or from the TP monitor (such as CICS). The terminal codepage always is the target codepage on outbound operations and the source codepage on inbound operations. The application codepage is the source codepage on outbound operations, and the target on inbound operations.

The application codepage is the codepage used to define the model. The screen manager assumes that the data is in the same codepage as the application.

It is important to note that the internal representation of a character and the display image of the string can be a different size. This is especially true when field editing inserts literal characters and a codepage translates the data for display.

Single Byte Characters

The customer must verify that the default LCID is set correctly when the control unit and display stations are configured. If the default LCID cannot be configured, the correct codepage must be installed.

Not all 3270 display units report the installed codepage. In these systems, the Query Reply structured field indicates that a single LCID is installed, and that the CGCSGID is not present. This information is relayed in the Query Reply (Character Sets) structured field, in the character sets query reply base section. Specifically, the information is relayed in byte 4, bits 1 and 6. Also, when bit 5 is on double-byte (DBCS) fields are supported. When bit 5 is off, double-byte (DBCS) fields are not supported.

When the CGCSGID is not present, the runtime assumes that the codepage in use is 37 (US English EBCDIC). This is also true when CICS reports that the CGCSGID is not available.

During codepage translation from a single byte to a single byte codepage, the length of the strings does not change. However, if the translation is from a single byte to a multiple byte codepage, then the resulting multiple byte string may be the same length, or longer.

Multiple Byte Characters

Multiple byte characters can be used both on the display device and in the screen manager program data areas. When the screen manager data areas contain fill characters, punctuation characters, currency symbols or strings, the fields are defined large enough to contain multiple byte characters. No indication is made as to which is stored in one of these data areas. The values are implicitly defined. Since no byte within a multiple byte string can have a value less than or equal to a single byte space (with the exception of a double byte space, which is x'4040'), the runtime determines implicitly whether a character in a data area is a 1, 2, 3, or 4 byte value.

Multiple byte character strings are designated as such in the appropriate entry in the attribute table. All data within a multiple byte character string are assumed to be multiple byte characters. No Shift In or Shift Out control codes are used. When the data is presented to the user, the user is prevented from entering any single byte characters. To block the entry of single byte characters, the field must be created with the Input Control attribute set to disallow generation of SO/SI control codes.

When the field contains characters that occupy less space than is provided, the characters are left aligned. For example, if the fill character is defined as a PIC X(4) area in COBOL, and the fill character is a single byte asterisk (x'5C'), then the field is actually stored as x'5C404040'. Since the second byte is a x'40', it legally cannot be part of the character, and the x'5C' is assumed to be the fill character. If, on the other hand, the fill character is a double byte '?' (x'426F'), then the fill character field is stored as x'426F4040'. Again, the first occurrence of a space (x'40') signifies the end of the character sequence, which is x'426F'. This also works correctly for ASCII encodings, except that an ASCII space is x'20' and not x'40' as in EBCDIC.

The previous algorithm works when the field is defined to contain only *one* character. When a field contains a string, an indicator is needed to inform the runtime that the value is a multiple byte, mixed, or single byte string. This is provided for user-defined fields, as an indicator in the field table entry for the field. This algorithm is used for constant data encoded in the data areas of the screen manager.

When a multiple byte character is transmitted to a screen in a field, the appropriate mechanisms are used to represent it.

Fixed Length Character Strings

Fixed length strings have an invariant length, regardless of the length of the significant portion of the data.

When a fixed-length character string is output, it is edited to contain the appropriate formatting. All editing is performed using the application codepage.

When a fixed-length character string is input, it is translated into the application codepage, if needed. This makes it unnecessary to translate edit patterns, and to perform complex editing functions on mixed codepages because all literals, edit patterns, and permitted values are stored in the generated screen manager program in the application codepage.

Varying Length Character Strings

The defined length of a varying length character string is treated as significant.

The string is not trimmed. This means that the user logic can pad the string with any desired characters, including the defined fill character or white space.

When a varying length string is output, the current length determines the significant portion of the data. The string occupies a substring which is left aligned within a space large enough to contain the maximum defined string length. However, only the current length is valid, and any characters beyond the current length are ignored.

If the length is larger than the defined maximum length, then the length should be reset to the maximum length. This prevents any storage violations, overlays, or other problems downstream. However, such a condition is suspect, and if some diagnostic reporting is available (such as tracing), then the condition is reported.

The defined substring is edited using the edit pattern defined for the field. This results in a varying length intermediate string.

The intermediate string is then translated to the terminal codepage. This may result in an intermediate string being longer or shorter than the intermediate string that is output from editing. After translation, the output string is output to the device and justified.

When a varying length string is input, it is translated into the application codepage, if needed. The variable is not padded with spaces. Instead, the computed length is set into the current length of the variable. The variable is padded with spaces to its maximum defined length to support COBOL comparisons, which are always for the maximum defined length.

More information:

Input Justification (see page 140)

Mixed Strings

In a mixed string field, all strings are assumed to begin in single byte mode. The display system must distinguish where the mode changes from single byte to multiple byte.

When a mixed string is output, each character is transmitted to the device without alteration until an SO or SI control code is encountered. When an SO code is encountered, then an SA order specifying the appropriate character set (x'F8' for DBCS) is inserted into the data stream and the SO code is discarded (it is not removed from the variable however). Then, double byte characters are transferred to the data stream until an SI is encountered. When an SI is found, the SI is discarded and an SA order specifying the default character set for the field is inserted. The single byte characters are then copied into the data stream as before. This process can be continued for as long as possible, with any number of SO/SI pairs in the same string.

The controller synthesizes SO/SI control codes and places them in the data for the field returned in the inbound data stream. The control codes exist only in the data stream that the controller returns to the host program. The codes are synthesized from the character set selections made by the SA orders when the fields were written. Also, if the user is allowed to enter SO/SI codes, these are returned to the host program.

Numeric

Numeric fields can be signed or unsigned, and associated with zoned, packed, binary (either short or long), and floating point variables.

Significance

Significance is the point in which data becomes significant and, if truncated, would cause a loss of meaning. Significance can be forced by the edit pattern picture, or by the data itself.

For numeric, numeric/text, or picture data types, significant data begins with the left-most non-zero digit on the right of the decimal point.

For example, 00123.4500 is the same as 123.45. The leading and trailing zeroes add nothing to the meaning of the data.

After significance is turned on, it remains on until the second condition (right most non-zero digit after the decimal point) is encountered.

For example, the zeroes in the value 120.01 are significant, and cannot be removed without changing the value.

In picture strings, Z represents an optional digit, and 9 represents a mandatory digit which must be present even if the data is insignificant.

The Z specifiers can exist only on the left side of the picture string, before any 9 specifiers. The 9 specifier forces significance to be turned on, regardless of the data values, and to remain on for the rest of the pattern (proceeding from left to right). The Z specifier is illegal if it exists after a 9, since the significance has been forced on by the 9.

Algebraic Notation Rules

When the input of a numeric field cannot be matched to the supplied edit pattern pictures (positive, negative, or zero), then standard algebraic and quantity notation is attempted as a last course of action. If this fails, then the string is marked as error, and the user is prompted to specify the value again.

The following list shows the rules for interpretation of an input value that does not match any pattern:

- If the edit pattern indicates a signed quantity (a format 2 or 3 edit pattern), the input value is checked for the presence of a leading or a trailing sign, specified using the "+" or "-" characters. If either is found, then the sign is set accordingly and the character is removed from consideration in the input string. If no sign character is found, then the sign is assumed to be positive. If the variable is unsigned, then no check is made for a leading or trailing sign.
- Prefix and postfix literals are not allowed, and if present, fail the edit.
- Digits are extracted from the input string until an infix literal character is encountered or the end of the input is reached. If an infix literal is encountered, then it must match an infix literal in the supplied edit pattern, and the data is aligned over the edit pattern on that infix literal character. This supplies decimal point or separator alignment. After it is aligned, the input digit string must match the digit and infix literal specifiers of the edit pattern from that point forward.
- If no infix literals are encountered in the digit string, then the value is assumed to be an integer value. If assignment is to a variable that contains decimal digits, then the input value is aligned to the indicated decimal position.

The following table shows how the system interprets data that does not match a supplied pattern. The table shows various positive, negative, and zero edit pattern pictures. For each pattern, the table shows which pattern is matched, its variable width and decimal digits (w,d), and the resulting value extracted.

User Input	Positive Picture	Negative Picture	Zero Picture	Match	Variable (w,d)	Value
+123	+ZZ9	-ZZ9	N/A	Positive	(5,0)	00123
123	ZZ9	N/A	N/A	Positive	(5,0)	00123
1.23	ZZ9.99	-ZZ9.99	-0-	Positive	(5,2)	001.23
-0-	ZZ9.99	-ZZ9.99	-0-	Zero	(5,2)	000.00
-12	ZZ9.99	-ZZ9.99	-0-	None	(5,2)	-012.00
-12	ZZZ.ZZ9	-ZZZ,ZZ9	N/A	Negative	(5,2)	-012.00

Numeric/Text (Picture)

Picture fields can be used to model edit patterns for phone numbers, social security numbers, part numbers, and others that contain digits but are not formatted like numeric quantities.

Using a picture field, the user specifies an edit pattern string using the normal numeric specifiers Z and 9.

Edit patterns created in earlier versions of CA Gen are also accepted. They are converted to Z and 9 specifiers by the generator.

Dates

The external data representation of dates can consist of any combination of the component parts of date, or derivatives of these parts. The parts can be in any order, separated by any number of literals. Internally, dates have three components, the 4-digit year, 2-digit month of the year, and 2-digit day of the month. Other components can be derived from these values, including a 2-digit year, century code, week number, day of week number, month name (both abbreviated and long form), day name (both abbreviated and long form), and remainder day number. These components are specified using particular edit pattern picture specifiers, as defined later.

Not all component parts are required to be present in an edit pattern picture. For example, a date edit pattern picture may specify only the month and day. However, the internal representation must *always* result in a valid date. Therefore, default values must be supplied for all omitted component parts in the edit pattern picture.

The defaults for date edit patterns are defined in the following table.

Component	Edit Pattern Picture Specifier	Default Value
4-digit year	YYYY	Current value, unless ALL of the specified components are zero. If all are zero, the default is zero.
2-digit year	YY	Current value, unless ALL of the specified components are zero. If all are zero, the default is zero, including the century digits.
Century code	C	Current value, unless ALL of the specified components are zero. If all are zero, the default is zero, including the century digits.
Month Number	MM	Current value, unless ALL of the specified components are zero. If all are zero, the default is zero.

Times

The external data representation of times can consist of any combination of its component parts, or derivatives of these parts, in any order, separated by any number of literals. Times are internally composed of three component parts, the 2-digit hour, 2-digit minute of the hour, and 2-digit second of the minute. Other components can be derived from these values, including a time zone name, 12-hour clock AM or PM specifier, and 24-hour clock hours.

The following table lists the default values for time edit patterns.

Component	Edit Pattern Picture Specifier	Default Value
2-digit Hour	HH	Zero
2-digit minutes of the hour	MM or MI	Zero
2-digit seconds of a minute	SS	Zero

Timestamps

The external data representation of timestamps can consist of any combination of the component parts of the timestamp, or derivative of these parts, in any order, separated by any number of literals. Timestamps are internally composed of all of component parts of both dates and times, plus a 6-digit microsecond value. Derived components include all of those derived from dates, times, or both plus the microseconds value.

The following table lists default values for timestamp edit patterns.

Component	Edit Pattern Picture Specifier	Default Value
Fractional Seconds	n, nn, nnn, nnnn, nnnnn, nnnnnn	Current value unless ALL of the specified components are zero, in which case its default also is zero. If the execution platform is not capable of timer resolution to one microsecond, then the missing precision is supplied as zeros.

Field Fill Processing

Fill processing is performed on output whenever the data, after being formatted for presentation, is shorter than the presentation area. The defined fill character is added to the formatted data to make it the same length as the presentation area. Fill characters are added before, after, or before and after the formatted results.

Fill characters also are removed from the presentation area when the presentation space is read. This is done to obtain the significant data for editing. Input fill processing reverses the operations performed on the data during output fill processing.

Fill processing is related to justification. The operation on the fill characters and the presentation space depends on the justification modes in effect.

Blank When Zero or NULL

When the condition Blank When Zero or Null is true, fill characters are copied into the field. This applies to numeric fields, numeric/text (picture) fields, text fields, and timestamp fields that have a value of zero or no defined value.

No edit pattern processing is performed. Instead, the field is cleared and no formatted data is presented to the user.

When the condition Blank When Zero or Null is false, numeric, text, and date fields display differently.

When the condition is false for numeric and numeric/text (picture) fields, the edit pattern is formatted based on a value of zero. It is the same as if the value is supplied as zero. The standard formatting is applied.

When the condition is false for text fields, the edit pattern is copied into the field as if it is the intermediate result of editing. No formatting is performed. Fill characters are applied after justification.

When the condition is false for date, time, or timestamp fields, the numeric components are edited using a value of zero. Derived components that represent names are ignored as if they are not specified in the edit pattern. These include the month, day, calendar, AM or PM, and time zone names. Derived values that result in numeric values display as zeroes. These include the number of the day or the week. The fill character is applied after justification.

Transparency

Transparent fields are used for the direct entry of internal format data, such as file uploads and binary data. When a field is transparent, it can contain any value in any byte of its presentation space (x'00' - x'FF').

The field is assumed to be in the format of the internal variable. The data is not edited, nor checked for permitted value violations, nor converted from the terminal codepage to the application codepage.

The fields are not operated upon in any way, except to verify lengths. If the presentation space is shorter than the attribute, then the data is low-byte aligned and filled with nulls (x'00') in the attribute. If the presentation space is longer than the attribute, then the low byte of the presentation space is aligned with the low byte of the attribute, and the excess of the presentation space is discarded.

Shift Out/Shift In Generation

Shift Out/Shift In (SO/SI) generation is disabled for all fields except those defined as accepting mixed SBCS/DBCS characters. SO/SI generation is not allowed for exclusive SBCS or DBCS fields.

Dynamic Attributes

Dynamic attributes are supplied by the MAKE statement in the procedure step. These attributes are passed to the screen program in the export view. The view contains a 6-byte, character-encoded value preceding the attribute state variable for a particular attribute. The format of the attributes is defined by the current implementation and cannot be changed.

Output Justification

Output justification is the process of aligning the data within a presentation space based on the size of the data and the presentation space. CA Gen supports five justification modes. Not all are honored for all data types.

The five different supported justification modes are detailed in the following table.

Field Type	Left	Right	Centered	Word Wrap	ASIS
Numeric	Yes	Yes	Yes		
Text	Yes	Yes	Yes	Yes	Yes
Picture	Yes	Yes	Yes		
Date	Yes	Yes	Yes		
Time	Yes	Yes	Yes		
Timestamp	Yes	Yes	Yes		

When data is justified, the internal data is converted and edited to produce an intermediate text-like string. This intermediate string is justified within the presentation space. Every character of the intermediate data is considered significant.

The intermediate string is converted from the application codepage to the terminal codepage, if necessary. This intermediate result is then compared to the length of the presentation area. The significant portion of the resulting string may be longer, shorter, or the same size as the presentation space.

Fill and space characters are inserted into the string based on the selected justification.

Fill Characters Output Justification

Fill characters are added to the edited intermediate results prior to being sent to the presentation space. The type of justification determines whether fill characters are added to the beginning or end of the intermediate string.

Left and word-wrap justification adds fill characters to the end of the intermediate string.

Right justification adds fill characters to the beginning of the intermediate string.

Centered justification adds fill characters to the beginning and the end of the intermediate string.

ASIS justification adds only NULL (x'00') characters to the end of the string, if needed. Fill characters are never added to an ASIS justification field.

Truncation of strings, either on the left or the right, is done at a character boundary. Since the terminal codepage may be a multiple-byte character codepage, the truncation may leave an odd number of bytes unaccounted for. These bytes will be filled with the appropriate space character if possible, or nulls if not. These bytes in the intermediate string are called dead positions.

Input Justification

Input justification is the process of aligning the input data properly in the variable. This process differs from one data type to another.

Text Justification

If the justification mode is ASIS, the string is moved directly to the input variable unchanged. If the presentation space is larger than the variable, then no truncation is performed and the field is flagged as error (overflow). If the presentation space is smaller than the variable, then the variable is aligned left and filled with spaces.

For all other types of justification, the value in the text field is left-aligned. If the variable exceeds the size of the presentation space and significant data would be truncated, the field is flagged with an overflow error.

Numeric Justification

Numeric quantities are always right justified. Fill digits (zeroes) are added to both the left and right as necessary to align the decimal point (if one exists), to the defined implied decimal position.

If the variable is an integer variable (no decimal digits defined), then the value is right-aligned and filled on the left only.

Fill Characters

When the justification mode is ASIS, no fill characters are removed from the input. The entire input string is considered significant.

When the justification mode is LEFT, fill characters are removed from the right to left in the presentation space only. When the first non-fill character is encountered, removal of fill characters halts. The remaining presentation string is considered significant.

When the justification mode is RIGHT, fill characters are removed from left to right in the presentation space only. When the first non-fill character is encountered, removal of fill characters halts. The remaining presentation string is considered significant.

When the justification mode is CENTER or WORD-WRAP, fill characters are removed from both ends of the presentation space. Additionally, word-wrap justification results in the removal of multiple adjacent occurrences of fill characters or spaces WITHIN the presentation space. The multiple occurrences are reduced to a single occurrence of a space.

Decimal Point Alignment

When numeric attributes are defined in a data model or a work set, the size and decimal digits are defined using a syntax such as (w,d). W represents the width of the variable in total digits, and d is the number of digits assumed to be on the right of a decimal point. An integer, containing 5 digits, would be defined as a (5,0) variable. A real value, containing 12 digits total, with 5 decimal digits (and therefore 8 integer digits) would be defined as a (12,5) variable.

A floating decimal point is provided using a sliding pattern matching algorithm to match a literal character specified in the edit pattern at the specified decimal point position. If no literal character is present in the edit pattern picture string at the specified decimal position, then the right-most infix literal is assumed to be the decimal point and the input is aligned on that literal. If the data cannot be aligned to the implied decimal point, then it is assumed to be an integer, and a decimal point is forced to the right of the last significant digit.

The following table illustrates several alignment examples.

Case	Edit Pattern	Input	Aligned Input	Variable (w,d)	Value
1	ZZZ,ZZ9.99	123.1	000,123.10	(8,2)	00012310
2	ZZZ,ZZ9.99	1,234	001,234.00	(8,2)	00123400
3	ZZZ,ZZ9.99	1.0000	000,001.00	(8,2)	00000100
4	ZZ9.999999	1.20000	001.20	(5,2)	00120

Case	Edit Pattern	Input	Aligned Input	Variable (w,d)	Value
5	ZZ9,99	1,2	001,20	(5,2)	00120
6	ZZ9,99	12	012,00	(5,2)	01200
7	ZZ9$99	1$2	001$20	(5,2)	00120

When the edit pattern is defined, only digit specifiers are recognized. All other characters are literal. They are placed in the edit picture just as the user entered them. The literals are not processed by the runtime, except for formatting. A period in the edit picture is not assumed to be the decimal point.

The runtime locates the decimal point position within the edit pattern, and aligns it with the implied decimal point of the variable.

The value column does not have a decimal point because the actual internally-stored value has an implied decimal point based on the defined width and decimal digits. The input is aligned on the implied decimal point and the right-most infix literal. Then, zeroes are added or removed so the digit string conforms to the edit pattern.

Once aligned to the implied decimal point, the value is converted and stored in the internal variable. Alignment may discard insignificant digits, as in case 4 in the Input Decimal Point Alignment table. However, if any significant digits are to be discarded, then the field is flagged as an error and the user is allowed to correct the input (either underflow or overflow).

Case 1 in the Input Decimal Point Alignment table shows the input string "123.1" of the user is aligned over the edit pattern of "ZZZ,ZZ9.99", based on the right-most infix literal rule and the defined width and decimal place of the variable. The missing digits are then supplied as zeroes, resulting in an aligned input of "123.10". This is then converted to the internal representation and stored in the variable.

To obtain a fixed decimal point, associate the edit pattern picture with an integer variable and implement logic for the appropriate scaling.

Validation

After the input string is edited and found to conform to the edit pattern, the data may be incorrect. The edit pattern pictures convey formatting information only, not content. It is the job of validation to verify the data.

Validation takes place at one of three points in the processing of a transaction:

- Input of data from a formatted screen
- During processing and interpretation of unformatted input
- Dialog Flows

Input Data Validation

When an autoflow command is not used and validation errors are encountered for formatted input from a screen, the field is flagged in error and the user is presented with the same screen again to allow corrections.

When an autoflow command is used and validation errors are encountered, the dialog manager performs the flow and does not return control to the screen. The screen is not presented to the user for correction. The value in the export view variable associated with the erroneous fields remains unchanged from the most recent non-error value.

Unformatted Input Validation

When validation errors are encountered for unformatted input, the incorrect input is moved to the image buffer for the screen and displayed in error when the screen displays. When the unformatted input is associated with a variable that is unplaced (does not exist on the screen), then the incorrect input is ignored and discarded.

Dialog Flows Validation

Validation also takes place when control flows from one procedure step to another passing data on a view. The dialog manager calls the screen manager verification services to perform this validation. This is true even if the procedure step being flowed to is execute first, in which no screen is displayed until the procedure step executes. If validation fails, then the screen manager reports the error to the dialog manager, which passes control to the screen to cause the target screen to be displayed. In this case, the screen is displayed first, even if the procedure step was marked as execute first, to allow the user to correct the erroneous data on the flow. In other words, if the flow verification fails, the target of the flow is treated as display first, regardless of how it is modeled. If the screen manager does not fail the validation, then the execute first/display first characteristic of the flow is executed as normal.

Attribute State Variables Validation

Validation is also a place where the attribute state variable can be reset after the procedure step executes. The procedure step does not preserve the attribute state variables used by the screen manager. As such, when verification is called, it checks to see if the attribute state variables have been reset to an indeterminate state (spaces). If this is the case, then verification synthesizes new state information for each variable during verification. The synthesized value is based on domain checks, verification, and permitted value checking, which also is checked each time verification is called with determinate state variable values.

Domain

One aspect of the validity of a data item is the domain. If a variable is defined as character, then the value may be any byte value from x'00' - x'FF'. For signed zoned, the range of legal values is different. The domain checks are summarized for each data type. Also, note that the verification domain checking logic must perform this test such that it is not susceptible to abnormal termination.

The following table lists domain checking rules.

Data Type	Domain Checking Rules
Character	Any byte may be from x'00' through x'FF', inclusive.
Unsigned zoned decimal	Each byte must contain a valid numeric, in the range of 0 - 9. Each byte must have a zone value of x'F' (x'F0 - x'F9').
Signed zoned decimal	Each byte must contain a valid numeric BCD encoding for digits 0 - 9. All bytes except the right-most byte must have a zone value of: x'F'. The last byte must have a zone value of x'F', x'C', or x'D'.
Unsigned short	Value may range from 0 to 65535 (x'0000' through x'FFFF'.
Signed short	Value may range from -32768 to +32767 (x'8000' through x'7FFF' in 2's complement notation.
Unsigned long	Value may range from 0 to 4,294,967,295 (x'00000000' through x'FFFFFFFF'.
Signed long	Value may range from -2.147,483,648 to +2,147,483,647 (x'8000000' through x'7FFFFFF' in 2's complement notation.
Packed decimal	All nibbles of all bytes must have the value 0 - 9. The only exception is the low order nibble of the last byte. It must be: x'C', x'D', or x'F'. Based on the principles of operation for IBM mainframes, x'F', is legal. However, COBOL recognizes only x'C' and x'D'.

Data Type	Domain Checking Rules
Time	In addition to the validation rules for unsigned zoned decimal, the time must adhere to the format HHMMSS. where: HH represents the hours specified as 00 - 23, or 24 if all other components are zero. MM represents the minutes specified in the range from 00 - 59. SS represents the seconds specified in the range from 00 - 59.
Date	In addition to the validation rules for unsigned zoned decimal, the date must adhere to the format YYYYMMDD. where: YYYY represents the four-digit year. MM represents the 2-digit month of the year. DD represents the two-digit day of the month. The value must specify a valid month for the specified year. The day of the month must be a valid specification for a day within the indicated month and year. This check must account for leap years and centuries, as well as Gregorian calendar anomalies. A value of zero for the entire date value is legal for optional date fields.
Timestamp	In addition to the validation rules for unsigned zoned decimal, the timestamp must adhere to the rules for its date and time components. A value of zero for the entire timestamp is legal for optional timestamp fields. A value of zero for the date components, but a non-zero time component is an illegal combination in all cases. The microseconds component must be from 0 to 999999, inclusive.

Optionality

A variable can be defined as mandatory or optional. A variable is mandatory only when its view is marked mandatory and all containers of the variable are marked mandatory, as well as the variable itself. If any containing structure, sub-structure, or other recursive or nested container is optional, then the variable is optional.

An optional variable does not require a value. This is reflected in the attribute state variable.

The attribute state variable is sufficient to determine whether a value is assigned for all data types other than date and timestamp. If the attribute state variable indicates that the value is present, then optionality is always successful. If a value is not present and the variable is optional, then optionality is also successful. When a value is not present and the variable is mandatory, then the field is flagged in error.

In the case of date and timestamp, it is necessary to check components of the entry for zeroes. Zero dates and timestamps that are omitted or empty are legal only for optional variables. If specified for a mandatory variable, then the field containing the zero value is marked in error. Times are not included in this test because a value of zero is legal even for a mandatory variable.

Permitted Values

The user may associate one or more permitted values with a variable. The permitted values may be specified as individual values, lists of values, ranges of values, or any combination. There is no limit to the number of permitted values that can be specified. When permitted values are specified, the input of the user must match one of the values.

Permitted values are checked after domain and optionality are checked. The check for permitted values verifies optional variables where a value is entered and mandatory variables. Permitted value checking is not performed on optional variables that have no value, or any variable that has failed any of the previous tests (domain or optionality).

Error Processing

The screen manager detects and reports on many different types of errors and error conditions. The basic premise used throughout the error handling is to assist the user in correcting mistakes with the least amount of effort. Data entered incorrectly is not discarded. The erroneous portions are highlighted and brought to the attention of the user.

When extended attribute support is available, it is possible to highlight one or more individual characters within a field. When extended attribute support is not available, standard field-level processing is used.

The cursor is positioned on the first erroneous character of the first erroneous field when the screen is displayed again. Any error message displayed applies only to the first field in error.

Any field can have from zero to "n" characters (where "n" is the length of the field) flagged as errors. When extended attributes are supported, each erroneous character is displayed using the merged normal and error characteristics. Characters that are not in error are displayed using the normal characteristics for color and highlighting. When extended attributes are not supported, the entire field is displayed using the error characteristics.

Each field has normal and error display characteristics.

Extended Attributes

When the target device supports extended attributes and the SA order is not suppressed, errors in an input string are indicated by changing the characters to the defined error characteristics. When error characteristics and normal characteristics are incompatible, the two are merged and the error characteristics are dominant. Data in the field that is not in error is displayed using as much of the normal characteristics as possible.

In the following table, which shows error display processing for extended attribute support, the field contents are shown both for valid and invalid data.

Normal Characteristics	Error Characteristics	Valid Data	Invalid Data
Blue, Not Protected, Displayable, Not Selectable, No MDT, Alphanumeric Shift, Normal Highlighting	Red, Not Protected, Displayable, Not Selectable, MDT, Alphanumeric Shift, Reverse Video	2002/01/12 Blue, Not Protected, Displayable, Not Selectable, No MDT, Alphanumeric Shift, Normal Highlighting	2002/**13**/12 Field: Blue, Not Protected, Displayable, Not Selectable, MDT, Alphanumeric Shift, Normal Highlighting. Characters "13": Red, Reverse Video
Blue, Not Protected, Displayable, Not Selectable, No MDT, Alphanumeric Shift, Normal Highlighting	Red, Protected, Displayable, Not Selectable, MDT, Alphanumeric Shift, Reverse Video	2002/01/12 Blue, Not Protected, Displayable, Not Selectable, No MDT, Alphanumeric Shift, Normal Highlighting	2002/13/12 Field: Blue, Protected, Displayable, Not Selectable, MDT, Alphanumeric Shift, Normal Highlighting. Characters "13": Red, Reverse Video
Blue, Not Protected, Displayable, Not Selectable, No MDT, Alphanumeric Shift, Normal Highlighting	Red, Protected, Not Displayable, Not Selectable, MDT, Alphanumeric Shift, Reverse Video	2002/01/12 Blue, Not Protected, Displayable, Not Selectable, No MDT, Alphanumeric Shift, Normal Highlighting	XXXXXXXXXX Field: Blue, Protected, Not Displayable, Not Selectable, MDT, Alphanumeric Shift, Normal Highlighting.

Editing

Editing is the process of converting data from one form or representation to another, using a picture specification to control the process. This picture specification contains formatting information to convert the internal format to the presentation format, or to reverse the process and interpret the presentation format and convert it back to the internal format.

There are six types of edit patterns. These are:

- Numeric
- Text
- Picture (Numeric/Text)
- Date
- Time
- Timestamp

The numeric edit patterns also have three formats, known as format 1, 2, and 3. These different formats supply one, two, or three picture specifications related to positive, negative, or zero quantities.

Editing can take place either when the internal data is being displayed, known as output editing, or when the displayed data is being interpreted for storage back into the variable, known as input editing.

More information:

Numeric Edit Pattern Restrictions (see page 152)

Output Editing

Output editing converts the internal representation of a data item to the appropriate external representation, using a picture specification to control formatting. The result can be displayed on the screen or printed on a printer.

Numeric

Currently, numeric output is synthesized into a positive, negative, or zero edit pattern based on the numeric entry. Future enhancements will allow you to specify a positive, negative, or zero edit pattern picture string for output editing.

Positive and negative picture strings are interpreted to format the data for display. Specifier characters in the picture strings have specific meaning, and are used to format the data accordingly. All other characters are literal characters, and are copied from the picture to the output field as appropriate.

The *blank when zero or null* property of a field also controls the presentation of the data. When this property is enabled, a zero numeric value is blanked and the field is filled only with the fill characters.

The significant digits in a numeric string are the digits on the left or right of the decimal point that provide meaning and would alter the value if removed.

The numeric concept of significance is used. Both leading and trailing zeroes are considered insignificant. However, for formatting purposes, once significance has been forced on, either by the picture or the value, proceeding from left to right, it remains on for the entire field including any decimal digits. Specification of a specific precision for use in scientific and engineering disciplines is controlled by the specification of the edit pattern picture, which the data will be formatted to match on output.

The edit pattern picture specifies optional (suppressible) digits, or mandatory digits, or both. Optional digits are denoted by Z, and mandatory digits are specified by 9.

Significance can be forced on by the edit pattern or by the value of the data.

In an edit pattern, significance is forced on by the left-most 9 in the string. Significance is turned on at that position, and for all other positions to the right of that digit, regardless of the value of the data. When significance is turned on by the picture, then leading zeroes become significant.

The value being edited also can force significance on. When the value is edited, the left-most non-zero digit can be aligned with a Z specifier. When the digit is non-zero, significance is forced on. The left-most Z and all other digit specifiers to the right are treated as a 9.

The table below shows examples of numeric output editing behavior.

Case	Float Prefix Literal	Picture	Value	Presentation
1	No	ZZ9	000	__0
2	No	ZZ9	012	__0
3	No	999	000	000
4	No	999	012	012
5	No	ZZZ,ZZ9.99	00000000	_____0.00

Case	Float Prefix Literal	Picture	Value	Presentation
6	No	ZZZ,ZZ9.99	00123456	__1234.56
7	No	ZZZ,ZZ9$99	00123456	**1234.56
8	No	999,999.99	00000000	000,000.00
9	No	ZZZ,ZZZ.99	00000009	********09
10	No	999,999.99	00012345	000,123.45
11	Yes	+ZZZ,ZZ9.99	00012345	****+123.45
12	No	+ZZZ,ZZ9.99	00012345	+****123.45
13	No	$ZZZ,ZZ9.99	00012345	$****123.45
14	Yes	$ZZZ,ZZ9.99	00012345	****$123.45

The preceding table illustrates several cases where significance is controlled by the edit pattern. In all cases, an asterisk is the field fill character. This fill character explicitly demonstrates the behavior of the edit routines based on the data, pattern, and significance. Justification is not taken into account.

In case 1, significance is forced by the edit pattern picture at the last digit position (specified by the 9 specifier). The Z specifiers are optional digits, and correspond to insignificant digits in the value (leading zeroes). Significance is not already established. Therefore, the character positions occupied by the Z specifiers are removed from the result and are filled with the fill character.

Case 2 is a variation of case 1 in that the same pattern is used, but the value is now non-zero. In this case, the digit 1 in the value is the start of significance. This digit corresponds to a Z specifier, thus significance is forced on by the data, not the pattern. From that digit position onward to the right, significance is turned on, and all digit specifiers (Z and 9) are considered significant (as if they were 9s). The leading zero in the value is still associated with an optional digit, and significance has not yet been established, so it is suppressed and replaced with the fill character.

Cases 3 and 4 are additional variations, using a pattern picture that specifies only 9s. These specifiers force significance on, and cause all digits, whether they are zero or not, to be output. This can be seen with both a zero and non-zero value.

Cases 5 and 6 demonstrate the effect of infix literals and significance. Infix literals are always suppressed until significance is turned on. Then, they are copied into the output stream in the same relative position they hold in the edit pattern picture. The implied decimal point is placed based on the defined width and decimal digits as defined by the variable. This is further demonstrated in cases 7, 8, 9, and 10.

In case 5, significance is forced on by the 9 specifier in the picture, and thus all leading Z specifiers, and infix literals (,), are suppressed and replaced by the fill character.

In Case 6, the significance is forced on by the data, and corresponds to the digit position before the infix literal. Since significance is now on, all digits are significant as well as all infix literals in the edit pattern picture. It is also important to note that the decimal point character (.) in both cases is just another infix literal. There is nothing special about this character, or its position. The implied decimal point is placed based on the defined width and decimal digits as defined by the variable. This is further demonstrated in cases 7, 8, 9, and 10 of the table.

Cases 11, 12, 13, and 14 demonstrate the effect of floating or fixing the position of prefix literals. When the Float Prefix Literals property is enabled, the prefix literals (either a single character or string of characters) are floated along the field such that they immediately precede the start of significance. If the property is not enabled, then the prefix literals are placed in their exact position as specified by the picture, and suppression of insignificant digits and infix literals occurs normally (being replaced by the fill character).

The effects of output editing also depend on which picture string is used to format the data. Currently, numeric output is synthesized into a positive or a negative edit pattern based on the numeric entry. Future enhancements will allow you to specify a positive (format 1), negative (format 2), or zero (format 3) edit pattern picture string for output editing. See numeric edit pattern restrictions.

The picture string is determined by the sign and magnitude of the value, combined with the format of the edit pattern. When the edit pattern is a format 1 edit pattern, the sign is always disregarded. Negative values are displayed and formatted on output as positive, or absolute value, quantities.

When a format 2 or 3 edit pattern is defined, the sign of the value determines which edit pattern picture is used to edit the data. A zero quantity will select the positive edit pattern picture of format 2 edit patterns, or the zero picture of format 3 edit patterns. This behavior is further demonstrated in the following examples.

The different picture strings (positive, negative, and zero) of the format 2 and 3 patterns can be different lengths. As a result, the presentation space (field) length displayed will be the longer of the various patterns. When shorter pictures are formatted, they will be justified and filled based on the defined fill character and justification mode for the field.

The following table shows an example of numeric output editing using various edit pattern formats (1, 2, and 3).

Case	Format	Positive	Negative	Zero	Value	Presentation
1	1	ZZ9			123	123

Case	Format	Positive	Negative	Zero	Value	Presentation
2	2	+ZZ9	-ZZ9		123	+123
3	2	+ZZ9	-ZZ9		-001	**-1
4	2	ZZ9	(ZZ9)		-123	(123)
5	3	ZZ9CR	ZZ9DB	-0-	012	*12CR
6	3	ZZ9CR	ZZ9DB	N/A	000	N/A

More information:

Numeric Edit Pattern Restrictions (see page 152)

Numeric Edit Pattern Restrictions

The Enhanced Block Mode Screen Generator for the z/OS target platform synthesizes format 1 and format 2 edit patterns from the edit pattern syntax used before COOL:Gen 5.0. Format 3 edit patterns are not synthesized, nor can they be entered directly by the user. The discussion is included in this release for completeness, as the generated source file contains these different types.

The standard generator may be forced by defining an environment variable, named TIMAPGEN, and setting it to a value of 1. If omitted, or set to a value of 2, then the enhanced generator is used when targeting z/OS. If generating under z/OS, an option is provided on the generation dialogs.

Character

Character and character string edit patterns use X and Y as specifiers. Each specifier represents a single character (not necessarily a byte) in the data. Any character other than a specifier in the edit pattern is assumed to be a literal character.

The Y specifier represents a mandatory character position (like the 9 for numeric), while the X represents an optional character position (like the Z for numeric). When a Y is specified, then the character position must be filled with a non-space and non-fill characters. When an X is specified, then the position may or may not have a non-space or non-fill character.

Literal characters can be prefix, infix, or postfix. The operation and use of these literals is basically the same as for numeric and character strings. Significance is defined as being the left-most non-blank character for left-right orientation dialects, and the rightmost non-blank character for right-left orientation dialects. Significance may also be forced on by the picture, just as for numeric edit patterns (the first occurrence of a Y specifier, either left-right or right-left based on dialect).

Character edit patterns are always format 1 edit patterns. There is never a negative or zero picture. The positive picture supplies the edit pattern picture string for use in editing the field.

If a character string is edited, and there is less data in the string than required to satisfy the optional characters specified in the picture, then fill characters are inserted for these missing characters. This is true for fixed or varying length strings. In the case of varying length strings, the string may have fewer characters than those necessary to satisfy the optional characters in the picture (especially for NULL strings).

The following table illustrates character string output edit patterns.

Case	Picture	Value	Presentation
1	XXXXXXXXXY,YY	DALLASTX	____DALLAS,TX
2	(XXXX)XXXXXXX	DEWAYNE	______DEWAYNE
3	(XXXX)XXXXXXX	DLH DEWAYNE	(DLH_)DEWAYNE
4	(XXXX)XXXXXXX	XYZ ZEBRA	(XYZ_)__ZEBRA
5	(YYYY)YYYYYYY	XYZ ZEBRA	(XYZ) ZEBRA

Case 1 in the Character Output Editing table shows a picture that specifies a literal character (,) between two groups of significant characters. The result is edited, with the literal inserted, and the fill character pads the field to account for insignificant characters.

In case 2, the picture specifies entirely optional character specifiers. The data is the same length, but no significant data is present for the left portion of the picture. Prefix literals and infix literals are suppressed until significance is started.

Case 3 is similar to case 2, except that significant data exists in the variable to force significance on immediately. This causes the prefix literal to be displayed. The fill character is inserted for all optional character positions that do not have a character in the data or spaces in the data.

Cases 4 and 5 demonstrate the different behaviors when the data is shorter than the picture, and the picture specifies both optional and mandatory character specifiers.

Usage Note

Use of the edit pattern specifier is supported only by the Enhanced Block Mode Generator and only on the z/OS platforms. If used, the edit pattern specifier is not supported on other platforms or by the standard map generator as a literal string. Support for the enhanced generator on other platforms will be announced in the future.

Picture (Numeric/Character)

Picture, also known as Numeric/Text, is a logical combination of the behavior and specification of numeric and character edit patterns and capabilities.

The edit pattern specifiers in picture fields are detailed in the following table.

Specifier	Description
X	Optional alphanumeric character. It can be any value from x'00' to x'FF,' except x'11.' unless the field is transparent.
Y	Mandatory alphanumeric character. It can be the same value as an 'X' specifier.
Z	Optional digit, may be from 0 to 9.
9	Mandatory digit, may be from 0 to 9.
A	Optional alphabetic, may be any printable non-numeric character.
B	Mandatory alphabetic, may be any printable non-numeric character.

Picture fields are most often used for non-quantity numeric values, such as part numbers, phone numbers, social security, other identification numbers, and so on. Typically, they are associated with integer numeric variables, although they can also be associated with fixed or varying length character strings.

Significance is defined differently, based on the orientation of the dialect. For dialects with a left-to-right orientation, significance is defined by the left-most non-zero digit, or the left-most non-space character. For dialects with a right-to-left orientation, significance is defined by the right-most non-space character.

Significance can be forced on by the edit pattern picture string, using either the 9 or Y specifiers. Again as either the left-most 9 or Y for left-right orientation, or right-most Y for right-left orientation.

Prefix and infix literals up to the point of significance are suppressed. This differs from numeric editing, in that the prefix literals are either fixed or floated. In picture fields, prefix literals are suppressed UNLESS significance starts immediately following the prefix literal, in which case it is not suppressed. Infix literals are always suppressed until significance begins.

The following table illustrates significance in Picture fields.

Case	Picture	Data	Presentation
1	(ZZZ) 999-9999	0005756595	_____575-6595

Case	Picture	Data	Presentation
2	(ZZZ) 999-9999	2145756595	(214) 575-6595
3	XX99-9999Y	123456X	__12-3456X
4	XX99-9999Y	AB121234C	AB12-1234C
5	XX99-9999Y	AB1212349	AB12-12349
6	XX99-9999B	AB121234Z	AB12-1234Z

Case 1 of the Picture Output Editing Examples shows data that is not significant until the fourth digit (left to right). The edit pattern picture does not force significance until the fourth digit. This means that all prefix and infix literals, and optional specifiers, are suppressed until significance has been started.

In case 2, the data is significant in the first digit position, forcing significance on. This in turn causes the prefix literals to be output, because significance is on immediately following the prefix literal.

In cases 3 and 4, a picture edit pattern specifies both digits and characters. The digit specifier only accepts a digit (0-9), while the character specifier accepts any character. This allows the picture to specify, in this case, a part number using a mixed value of characters and digits. It is also significant to note that the data can have a character value that is the same as a specifier in the edit pattern picture. These are not the same, and should not be confused.

In case 6, the edit pattern string is changed to force the last position to be a mandatory alphabetic value. Case 5 indicates that the use of a Y specifier is satisfied by either a numeric or character value.

On output, data that does not conform to the edit pattern is flagged as an error, and the user is allowed to correct it. Thus, a field is flagged in error when a numeric character is entered where the edit pattern specifies an alphabetic character. Output editing not only flags the field in error, but also flags each character position that is incorrect. This is one of the few places where output editing changes the state of a variable to error and fails an edit; normally this is performed during input editing.

Date

Date editing accepts many different specifiers to indicate different portions of a formatted date value, such as the century, year, month, name of the month in the current calendar. The specifiers can be used in any combination, and in any order. Also, specifiers can be intermixed with any literal characters in order to provide the necessary punctuation.

The following table lists all of the date edit format specifiers.

Specifier	Description
YYYY	The century and year, formatted as a four-digit value, ranging from 0001 to 9999 for date variables and from 1600 to 2599 for numeric variables, or 0000 for both when all other components are also zero.
YY	The year of the century, formatted as a two-digit value, from 01 to 99 for either date or numeric variables, or 00 when all other components are also zero.
C	Century number, specified as a one-digit value, from 0 to 9.
MM	Current month number of the year, based on the current calendar. The number may be from 01 to 12 for Gregorian, or 00 when all other components are also zero.
MN	The full name of the month for the current calendar. This component results in a varying length value, corresponding to the name of the month. For example, MN DD YYYY displays July 4, 2000 for 7/4/00, and August 4, 2000 for 8/4/00.
MA	The abbreviation of the month name for the current calendar, if an abbreviation is allowed. If not allowed, the full month name is used. This component results in a varying length value.
DN	The full name of the day of the week for the current calendar. This component results in a varying length value, corresponding to the name of the day of the week.
DA	The abbreviation of the name of the day for the current calendar, if an abbreviation is allowed. If not allowed, the full month name is used. This component results in a varying length value.
JJJ	This is the sequence number of the day in the year. It is based on the current calendar for the date being edited. For Gregorian, the first day of each year is January 1, and the specifier is the sequence of days since the first of January. **Note:** January 1 is sequence number 1.
RRR	The sequence number indicating the number of days remaining in the current calendar.
CAL	The full name of the current calendar. This component results in a varying length result.

In some cases, more than one specifier can be used to specify the same portion of the internal date. For example, the specifier JJJ represents the day sequence number from the start of the year. This value actually implies the month and day of the month within the year. If JJJ is used in an edit pattern as well as either the month and/or day of the month, then an overlapping condition exists. In this case, output editing synchronizes the values. For input editing, however, these overlaps can be a cause for concern.

Time

Time editing accepts many different specifiers. These specifiers indicate different portions of a formatted time value, such as the hours, minutes, seconds, and time zone name. These specifiers can be used in any combination, in any order, and intermixed with any literal characters in order to provide the necessary punctuation.

The following table lists all of the time specifiers.

Specifier	Description
HH or 24	Two-digit hour of the day, in 24-hour clock format. The value ranges from 00 to 23, or 24 when all other components are zero.
12	Two digit hour of the day, in a 12-hour clock format. This value ranges from 00 to 11, or 12 if all other components are zero.
MI or MM	Two-digit minutes of the hour. This value may range from 00 to 59. When the hour of the day is 24 (24-hour clock) or 12 (12-hour clock), then the minute value must be zero.
AM or PM	AM/PM indicator. If the internal time hour component is between 0 and 12, inclusive, then the string AM is shown. For a value of 13 to 24, PM is shown. On input, if the value is AM, and the 12 component is between 0 and 12, then the internal hours and the value in the 12 component are the same. If the indicator is PM, then 12 is added to the value of the 12 component to compute the 24-hour clock value to be stored internally.

The operation and behavior of the various time edit pattern specifiers is shown in the following table.

Internal Time (HHMMSS)	Edit Pattern	Presentation
012345	HH:MM:SS	01:23:45
134455	24:MI:SS	13:44:55
012345	12:MM PM	01:23 AM
134567	12:MM PM	01:45 PM

Timestamp

Timestamp edit patterns use all of the specifiers in both the date and time edit patterns. In addition, timestamp edit patterns include a pattern for microseconds. These fractional portions of a second can be specified using from 1 to 6 digits of precision. Also, the MM component is ambiguous between dates (month of the year) and times (minutes after the hour). MM used in a timestamp edit pattern will be assumed to be the month of the year, MI will be the minutes after the hour.

The following table lists the descriptions for all the specifiers:

Specifier	Description
N	1-digit fractional second (1/10th Second)
NN	2-digit fractional second (1/100th Second)
NNN	3-digit fractional second (1/1000th Second) or milliseconds
NNNN	4-digit fractional second (1/10,000th Second)
NNNNN	5-digit fractional second (1/100,000th Second)
NNNNNN	6-digit fractional second (1/1,000,000th Second) or microseconds

The following table lists all of the timestamp format specifiers.

Internal Timestamp (YYYYMMDDHHMISSNNNNNN)	Edit Pattern	Presentation
19960528123456123456	HH:MI:SS.N	12:34:56.1
19960528123456123456	12:MI:SS.NN PM	12:34:56.12 AM
19960528234500123456	12:MI:SS.NNN PM	11:45:00.123 PM

Input Editing

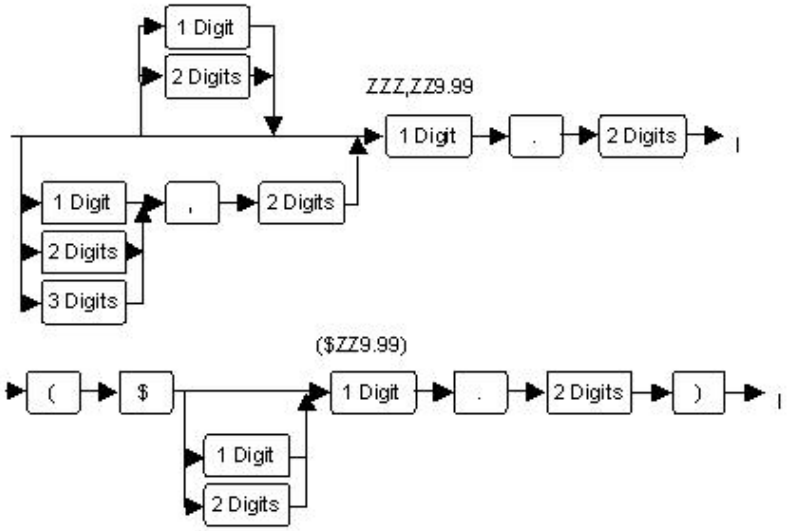
Input editing is the process of interpreting presentation data based on the edit pattern picture and converting it into the appropriate internal representation. The internal representation is then stored in the associated attribute within the view.

Input editing must support all of the same specifiers as output editing. The field may be formatted based on these specifiers, then displayed, modified (or maybe not, if the Modified Data Tag is set on), read back in, and processed. Therefore, input and output editing must be synchronized such that both support the same edit specifiers and behaviors. The difference is that in output editing, an internal value is formatted for presentation. In input editing, a presentation is being interpreted to derive an internal representation.

Numeric

Numeric editing is somewhat difficult because of the different edit pattern formats and the many different possibilities. Basically, input numeric editing attempts to match the edit pattern to the input presentation based on the order of digits and literals for each picture string in the edit pattern.

Pattern matching is based on the edit pattern picture string as the rules for interpretation. For example, an edit pattern of ZZZ,ZZ9.99 represents the following rule: 0 -3 optional digits, followed by an optional comma, followed by 0-2 optional digits, followed by a mandatory digit, period, and finally 2 digits. This is illustrated in the following syntax diagram for pattern matching of numeric edit patterns.



In the syntax diagram, the rules for interpretation of any edit pattern are somewhat vague. The diagram shows the translation of an edit pattern into a syntax diagram, but yields little information on how that transformation is performed.

The following list details the interpretation rules:

- Any prefix literals must be matched exactly (as in the case of "($" in the last example).
- Each Z specifier is interpreted as a parallel track to the main track through the syntax. This is because these specifiers are optional, and may not be present.
- Adjacent optional digits (Zs) are grouped together and may be satisfied using from zero (0) to the number of Z specifier digits of input.
- Infix literals following optional digits must be matched within the optional digits track.
- Mandatory digits must be matched exactly.

- Infix literals after a mandatory digit must be matched exactly.
- Infix literals after optional digits must be matched after the optional digits in that parallel track, not in the main track.
- Postfix literals must be matched exactly.

Pattern matching begins with the positive picture string. It continues with the negative picture string for edit pattern formats 2 and 3. When the positive and negative strings fail, the zero picture string is matched if the edit pattern is format 3. The sign or magnitude of the variable is known based on which pattern matches.

For example, if the positive string matches the input pattern, then the sign is set to positive. If the negative string matches, then the sign is set to negative. If the zero picture matches, then the magnitude of the value is set to zero and editing ends for this variable. Note that even if the positive or negative string matches the general pattern of digits and literals input, no value has yet been extracted.

When a positive or negative pattern is matched, then the value is extracted and converted to the domain of the variable. Editing then ceases for this field.

If no pattern is matched, then the input edit routines revert to standard algebraic notation rules for interpretation of the value. When the value is valid, both the sign and magnitude are extracted and set into the variable after domain conversion.

If the algebraic notation rules do not yield a result, the appropriate characters in the input are flagged in error and the field is set to an error state. Input editing of that field ceases.

More information:

Numeric Edit Pattern Restrictions (see page 152)

Character

Input editing of character strings is the reversal of the application of the edit pattern to remove any literal characters that may have been inserted into the output during output editing. It includes validation of optional and mandatory characters as well. The result of the editing is a single character or character string. In the case of a character string, the extracted characters are concatenated in the appropriate direction to compose the internal string value.

Character string edit patterns are interpreted much the same as for numeric strings, with the exception that there is only one format.

Picture (Numeric/Text, Numeric/Character)

A picture edit pattern may use an edit pattern that specifies alphanumeric, numeric, and alphabetic specifiers. The pattern can be associated with either a numeric or character variable. If alphabetic or alphanumeric specifiers are used, then the variable must be capable of storing alphabetic data.

The characters and/or digits are extracted from the input based on the edit pattern. The resulting value is then stored in the variable.

If the field is associated with a character or character string, then no domain conversion is performed. If the string is a varying length string, then it is set to the actual length of the extracted data. If the variable is a numeric variable, then the data is converted to the proper domain and stored in the variable.

Note that when the associated variable is numeric, the picture can contain only Z and 9 specifiers, and literal characters. This prevents a non-numeric value from being extracted which could not be converted to a numeric domain.

Date

Input editing of dates attempts to extract the information from the input field and compute an internal value comprised of either the year, month, and day in the form YYYYMMDD or the century, year, month, and day in the form CYYMMDD. Each component part of the date edit pattern specifies one or more portions of this value, either directly or by derivation.

For example, the YYYY edit pattern component directly supplies the year portion of the YYYYMMDD format, or both the century and year of the CYYMMDD format.

Overlap exists between many components used to specify dates. This overlap comes in the derivation of the internal value based on the component. For example, JJJ specifies the sequence number of days since the first day of the calendar, which implies the month and day of the month. If the JJJ specifier is used with the MM, or DD, or both specifiers, then duplicate or overlapping specifications exist.

When a date edit pattern contains overlapping specifications, then all of the overlapping specifications must be in agreement. Any overlapping specifications that conflict are flagged in error, in a left to right scan. This means that the first specification is assumed to be correct, and that any subsequent overlap that is in disagreement is flagged as an error. The actual error may be the first component, but editing has no way of knowing this.

Date components may be omitted from an edit pattern that is necessary to fully derive the internal date format (YYYYMMDD or CYYMMDD). When this happens, the internal values must be defaulted. Note that these defaults apply to computing the INTERNAL value from specifiers that are OMITTED from the EDIT PATTERN. If the specifiers are used in the edit pattern, and the user omits values for them in the input string, then the pattern flags the value as an error. The error occurs because the user did not enter all of the values specified in the edit pattern.

The following list details the default rules:

- The current 4-digit year is used as a mask on which the input of the user for the YY or YYYY specifier is right-aligned. The result is a merge of the current 4-digit year with the specified values, where the current value supplies all of the missing left-most digits and the user-entered value supplies all of the right-most digits.
- If the DD component cannot be computed from the JJJ or RRR components, then it defaults to 01, unless the DN or DA component specifies a day name, in which case it is defaulted to the day number corresponding to that day in the first week of the indicated month.
- If the MM component cannot be computed from the JJJ, RRR, MN, or MA components, then it defaults to 01.
- If the C specifier is used, then it overrides the default century (based on the current year).
- If all values for all specifiers in an edit pattern are zero, then the resulting omitted (internal) components are forced to zero to result in a zero date.

The following table illustrates date input editing behavior.

Presentation	Edit Pattern	Internal Date (YYYYMMDD)	Internal Date (CYYMMDD)
05/28/00	MM/DD/YYYY	20000528	0000528
05/28/0	MM/DD/YYYY	20000528	0000528
Jan 2, 2000	MA DD, YYYY	20000102	0000102
Jan 2, 00	MA DD, YYYY	20000102	0000102
January 2000	MN YYYY	20000101	0000101
Jan 00, Wed	MA YY, DA	20000103	0000103
00/00	YY/MM	00000000	0000000

Time

Time edit patterns extract the time components from a presentation space and construct an internal value in the form HHMMSS. Note that overlap may exist within times as well as for dates, especially in the use of the HH or 24-hour specifier combined with the AM or PM specifier. In these cases, the two groups of specifiers provide derivation information for the same part of the internal value. The action for overlapping specifications is the same for time as it is for date (disagreements are flagged as an error).

When a time edit pattern contains overlapping specifications, then all of the overlapping specifications must be in agreement. Any overlapping specifications that conflict are flagged in error in a left to right scan. The first specification is assumed to be correct, and any subsequent overlap that is in disagreement is flagged as an error. The actual error may be the first component, but editing has no way of knowing this.

The internal storage format for time is a 2400 clock format, regardless of how data is displayed. The use of the 12 specifier and AM or PM specifiers must derive the value based on a 2400 clock. When the 12 specifier is used, then the value of the AM or PM indicator determines whether 12 is added to the value specified by this component. If the AM or PM indicator is not used, but the 12 is used in the edit pattern, then the clock value can never exceed 125959.

Time components that are necessary to fully derive the internal time format (HHMMSS) may be omitted from an edit pattern. When this is the case, then the internal values must be defaulted. Note that these defaults apply to computing the INTERNAL value from specifiers that are OMITTED from the EDIT PATTERN. If the specifiers are used in the edit pattern, and the user omits values for them in the input string, then the pattern flags the value as an error because the user did not enter all of the values specified in the edit pattern.

The following list details the default rules:

- If the MI or MM component is omitted, then it defaults to zero (00).
- If the SS component is omitted, then it defaults to zero (00).
- If the HH, 12, or 24 component is omitted, then it defaults to zero (00).

The following table illustrates the time input editing behavior.

Presentation	Edit Pattern	Internal Time (HHMMSS)
12:34:56	HH:MM:SS	123456
08:30:00 AM	12:MM:SS PM	083000
08:30:00 PM	12:MM:SS PM	203000
20:30:00 AM	24:MM:SS PM	ERROR

Timestamp

Timestamps use the combination of both date and time editing, and add the microsecond specifier (as was the case for timestamp output editing). All of the same rules for dates and times apply to timestamps. In addition, there is a rule for a zero timestamp.

A zero timestamp can be valid only for an optional timestamp when all defined specifiers in the edit pattern have a zero value. Any non-zero time component with any zero date component is illegal.

Input Edit Exit

An input editing exit lets you tailor the behaviors for input editing. The exit is called after all parsing and interpretation is performed, but before the results are stored in the variable and before an error is reported. The exit provides the image of the input field, the characteristics of the field (its type, justification, fill character, state, and so on), the results of the edit (the intermediate result), the attribute address and its characteristics (type, length, and so on), the address of the error mask, and a work area.

The user exit can do the following tasks:

- Alter the image and cause editing to be repeated
- Change the intermediate result
- Set any value it wants in the variable
- Fail the edit
- Flag any character or characters that it wants to as error
- Clear the variable

The user exit reports the actions that it takes using a return code to the edit routines. The return code can be used to direct the edit routines to continue normal processing, re-edit the image, fail the edit (even though the normal action may not have been a failure), convert a new intermediate result and store it in the variable, bypass all processing, or erase the value of the variable.

Edit Pattern Conversion

It is necessary to modify some edit pattern strings stored in current models. The modifications generally involve conversion from numeric patterns to either format 1 or format 2 patterns, synthesis of edit patterns for unedited strings, translation of numeric/text patterns to picture patterns, and so on.

More information:

Numeric Edit Pattern Restrictions (see page 152)

Toolset Specification of Edit Patterns

From the perspective of the generator it makes no difference how an edit pattern is defined or where it is defined. However, it may make a tremendous difference to the user.

It is not necessary for a field to be the same length as the edit pattern. Fields can be the same length or LONGER than the edit pattern (they may not be shorter, however). This is especially true when specifiers such as DN (Day Name) result in a string that is much longer than the specifier. Specifiers need not be the same length as the data component that they specify.

Named Edit Patterns

The user can use business system defaults to define a field or special edit pattern for the entire model. These edit patterns are assigned a unique name and are available for any screen in the specified business system.

Special edit patterns can format system fields which are either derived from system information or used to display status information. These fields include scroll amounts and indicators, system timestamp, and other system values.

Field edit patterns format variables contained within the views, which are mapped onto the screen. The variables may be values from a supplied work set, but they are still variables within the view. Being provided as part of the IEF_Supplied group does not make them system variables. Also, some system variables (such as the command field) are copied into IEF_Supplied variables which can be used in the view, but when placed on the screen are placed as system fields.

Generally, named edit patterns are a means to define a common picture format for different types of data fields. If a consistent format is desired for a date or time, for example, then the edit pattern can be defined as a business system default and given a name to uniquely identify it. This named edit pattern is then added to the choices of edit patterns that are presented to the user when a field is placed on the screen in the screen designer.

Also, named edit patterns allow you to implement changes more easily. If it is necessary to alter the standard presentation format of some particular data item, and named edit patterns are used, only the definition in the business system defaults needs to be changed.

Unnamed Edit Patterns

Unnamed edit patterns are entered directly into the edit pattern control on the field properties dialog in the screen designer. The entered edit pattern is then associated with only that field instance. It is not available for use in any other field, even in the same screen. This is intended to provide an override capability, so that the user can enter a specific picture for a specific field on a specific screen, without changing the business system defaults.

User-Defined Edit Specifiers

Users can define their own edit specifiers and implement their behavior in the input edit exit (TIRIEX). These specifiers are treated as literals within the edit pattern specification, and are generated directly into the screen source code unchanged. These patterns are then passed to the runtime to interpret the meaning and edit the field contents on the screen.

User-defined edit specifiers may be individual characters (for example, 9 for mandatory digit) or short strings of characters (for example, DN for day name). Care should be taken that no user-defined specifiers are the same as any CA Gen specifiers. To assist in this effort, CA Gen does not define a specifier that starts with the letter U, thus allowing user-defined specifiers to start with this letter.

Conversion of Numeric Patterns to Format 1 or 2 Edit Patterns

Currently, the toolset allows only a single edit pattern for a numeric field. In this edit pattern, the user enters specifiers for optional and mandatory digits and leading or trailing signs. Punctuation is treated as literals, which may be changed based on the dialect. These specifiers are, for the most part, the same as defined by the generator, with a few exceptions.

In the toolset, a '+' represents a mandatory sign, and a '-' represents an optional sign. The sign may be leading (preceding all digits) or trailing (following all digits). A mandatory sign specifier means that the sign must always be displayed, even if the value is positive or zero. An optional sign is one where the sign is suppressed if the value is positive or zero, and displayed if the value is negative.

Also, the current implementation defines the '*' and '$' characters to mean special things. The '*' is considered a combination of fill character and optional digit (Z). If significance has not been established, then the '*' is used as the fill character. If significance has been established, then the '*' operates as a digit specifier 9. This behavior is extracted from the old edit pattern and is used to set the edit pattern correctly and to specify the fill character.

The '$' character is used to represent a floating currency symbol or optional digit. Much the same as the '*' character, if significance is not set, then the '$' is replaced by the fill character. This is true for all positions EXCEPT the one just left of the first significant digit, in which case the '$' character is inserted. All other '$' characters after significance start are treated as digit specifiers (9).

The generator translates numeric edit patterns specified using this syntax to either a format 1 or format 2 edit pattern using the new syntax. This conversion process must handle unsigned, mandatory signed, and optional signed patterns, as well as check-protection fill and floating currency symbols.

The rules for conversion of an old format edit pattern to a new pattern are:

- If there is no sign specified, either leading or trailing, then the old pattern is copied into the positive picture of a format 1 edit pattern.
- If there is a mandatory sign, then the old pattern is copied into both the positive and negative picture strings of a format 2 edit pattern. The “+” sign in the negative picture is then changed to a “-” character. The sign characters (+/-) are now considered only to be literals.
- If there is an optional sign, then the old pattern is copied into both the positive and negative picture strings of a format 2 edit pattern. The “-” sign in the positive picture is then removed from the string. The “-” character in the negative picture is now considered to be a literal only.
- If the '*' character is specified, then all occurrences of '*' are replaced with 'Z' characters and the fill character is set to '*'.
- If the '$' character is specified, then all but the first occurrence (or only occurrence) of the '$' is set to 'Z'. If there is more than one '$' specified (for example, $$$$9.99), then the float prefix literal property is enabled, otherwise (for example, $ZZZ,ZZ9.99) it is disabled causing the prefix literal to be fixed.
- If there is only a single '$' specifier, and it follows any '*' specifiers, then the '$' is replaced by a 'Z' and '$' is inserted as the prefix literal and the float prefix literals property is enabled.

The following table illustrates the translation process. The sign of the value determines which picture is used to format the value, resulting in the correct presentation. In the future, the toolset will be enhanced to allow the user direct entry of the positive, negative, and zero edit pattern pictures for numeric values.

Old Format Edit Pattern	Format	Positive Picture	Negative Picture	Fill Character	Float Prefix Literal
ZZZ,ZZ9	1	ZZZ,ZZ9	N/A	Default	No
+ZZ9	2	+ZZ9	-ZZ9	Default	No
ZZ9+	2	ZZ9+	ZZ9-	Default	No

Old Format Edit Pattern	Format	Positive Picture	Negative Picture	Fill Character	Float Prefix Literal
-ZZ9	2	ZZ9	-ZZ9	Default	No
ZZ9-	2	ZZ9	ZZ9-	Default	No
***,**9	1	ZZZ,ZZ9	N/A	*	No
***,*$9	1	$ZZZ,ZZ9	N/A	*	Yes
$**9.99	1	$ZZ9.99	N/A	*	No
$$$9.99	1	$ZZ9.99	N/A	Default	Yes
+$**9.99	2	+$ZZ9.99	-$ZZ9.99	*	No
$+**9.99	2	$+ZZ9.99	$-ZZ9.99	*	Yes
$$$9.99+	2	$ZZ9.99+	$ZZ9.99-	Default	Yes
$**9.99-	2	$ZZ9.99	$ZZ9.99-	*	No

Translation of User-Defined Specifiers

User-defined specifiers are not translated, assuming that they do not conflict with any defined specifiers. They are copied directly into the format 1, 2, or 3 picture strings unchanged.

Conversion of Picture (Numeric/Text) Patterns

The current implementation limits the use of picture edit patterns to a textual representation of a numeric quantity. This means that digits are specified using the 'X' specifier. This is defined as an optional alphanumeric specifier in the CA Gen generator. Therefore, old format numeric/text edit patterns would continue to operate, but would not provide the level of control as specified by the new generator. Domain validation would be the only mechanism to catch entry errors if the old format pattern is interpreted using the new rules.

Therefore, old numeric/text edit patterns that contain 'X' specifiers are converted to 'Z' specifiers, if the associated variable is a numeric variable. This is a simple transformation that guarantees that the edit pattern specifiers coincide with the type of data that can be entered in the variable associated with the field.

Note that this transformation is only performed if the pattern contains 'X' specifiers, AND the field is associated with a numeric variable. This limitation is imposed because in the future, when picture fields may be associated with text variables, the 'X' specifier may be valid.

Scrolling List

A scrolling list can display data stored in a repeating group view. Use scrolling when the maximum cardinality of a group view exceeds the number of display lines available on the screen.

The screen, or presentation space, displays fields, literals, and special fields.

The area of the screen where the repeating group view is displayed is called the presentation view. Data in the presentation view that is visible to the user is called the viewport.

The size of the repeating group view equals the cardinality of the view. This size is referred to as the data view extent. The maximum cardinality of the repeating group view defines the total number of entries that can be present in the repeating group. This sets the size of the data view extent in memory.

The size of the viewport is based on the number of rows it contains. This size also is known as the placed cardinality. The viewport can originate at any row in the data view extent. The row in the data view extent that is the first row in the viewport is called the viewport origin. This is an index number identifying the row in the data view extent that is to be mapped to the first row in the viewport. All subsequent rows are mapped from that point.

The data view encompasses all of the occurrences in a repeating group view. The data in a repeating group must be stored in memory in a table or in an array of data items. The entire collection of scrollable data can be up to 32 bytes in length, including the overhead of the view and other variables within the view.

The data population size is the number of rows actually populated in the repeating group view. Also called the current cardinality, this value can range from zero to the data view extent size. The data population size cannot be greater than the data view extent size, nor can it be a negative number.

Scroll Protection Property

In the toolset, a protection property can be set for each scrollable repeating group view. The view can be unprotected to allow updates, or protected to prevent updates.

When the view is protected, the data view boundaries are defined by the data view, and not necessarily by the maximum cardinality of the view. The population of the data view can range from zero (an empty view) to the size of the data view extent. The population cannot be negative, and cannot exceed the size of the data view extent.

When a protected data view is used in scrolling, unpopulated fields are outside the boundaries of the data view. These fields are protected and blanked to prevent the user from entering data. This is especially true when scrolling in two or more fields is synchronized.

When the view is unprotected, the user can add an occurrence to the end of the view. The viewport can be positioned anywhere within the data view extent.

If the user enters data in a previously unpopulated area of the data view, and there are unpopulated rows before the row that was modified, then the view is condensed. The new data is moved to the first unpopulated row and the unpopulated rows are removed. All unpopulated rows are collected at the end of the data view.

Note: The data view protection property controls the selection of areas that can be mapped to the viewport and determines whether users can add a row to the data view. Screen Design also allows a field to be protected, thus preventing users from entering data. The combination of data view protection and field protection determines whether the user can change existing data, add new data to unpopulated rows, or both.

The following table identifies the possible protection level combinations and the result of each common action.

Data View/Field Protection Levels	Change Data	Add Data
Data View Protected, Field Protected	No	No
Data View Protected, Field Unprotected	Yes	No
Data View Unprotected, Field Protected	No	No
Data View Unprotected, Field Unprotected	Yes	Yes

Scrolling Modes

The Scroll check box in the Screen Properties dialog determines if scrolling is handled by the user procedure step, or by both the CA Gen software and the user procedure step.

When the box is checked, the user procedure step and CA Gen cooperate to perform scrolling. This is called Cooperative Scrolling, or Mode 2 scrolling.

When the box is not checked, the user procedure step handles all scrolling functions. This is called User Scrolling, or Mode 3 scrolling.

Currently, Mode 1 scrolling is not available.

Scrolling is controlled by the data view extent, the protection property for the data view, and the positioning property for the map.

Cooperative Scrolling Mode 2

In Cooperative Scrolling, the user procedure step and CA Gen cooperate to perform scrolling. When the user places the viewport anywhere within the boundaries of the data view, then the CA Gen software performs the scrolling operation. Scroll action commands are *not* passed to the procedure step.

However, when the user moves the viewport entirely outside of the data view boundaries, then the procedure step is called. The command entered by the user is translated to a common scroll action command, which is then passed to the procedure step for processing.

The scrolling command can be dialect specific. It is associated with a standard scroll operation that is presented to the procedure steps. This way, the screen procedure step logic does not have to be duplicated, or sensitive to, dialect specific scroll commands. As a result, scrolling support can be constructed without regard to the dialect defined for a screen.

After the viewport origin is relocated, it is evaluated to make sure it is within the data view. If the viewport has been moved entirely outside of the data view boundaries either prior or beyond it, then the common command is passed to the procedure step.

User Scrolling - Mode 3

In User Scrolling, the screen manager performs no actions to support scrolling. All commands entered are passed to the procedure step. In addition, no command translation is performed, either on input or output, to common scroll actions. The user application procedure step is responsible for all actions and behaviors.

When the Scroll check box is not selected, the software suppresses the placement of scroll position, scroll amount, or the scroll indicator special fields.

In Mode 3, no viewport adjustment is performed. Instead, the command and the special fields are available to the user procedure step to make scrolling decisions. After this logic completes, the assigned values of the scroll viewport, scroll amount, scroll position, and scroll indicator are used to display the next screen.

Scrolling Commands

Scrolling actions can be associated with commands in any dialect. The scrolling actions are processed consistently regardless of the dialect of the command entered.

NEXT, PREV, TOP, and BOTTOM are English commands associated with scrolling actions. Scrolling commands in another dialect can be used in place of the English commands. The CA Gen software associates the non-English commands with the appropriate scroll actions. The entered command is checked against the defined scroll actions. If the command represents a defined scroll action, the action is performed and the command is replaced by a common command for the scroll action.

The procedure step of the user requires one Case of Command statement containing cases for each of the common scroll commands. It is not necessary to add individual case statements for each command in each dialect.

Data scrolls forward or backward based on the total number of lines in the viewport.

The following table defines the common scroll action commands.

Common Command	Scroll Action	Description
NEXT	Scroll to next unit	NEXT increments the viewport origin to the next entry that corresponds with the scroll amount.
PREV	Scroll to previous unit	PREV decrements the viewport origin to the previous entry that corresponds with the scroll amount.
TOP	Scroll to the first entry	TOP sets the viewport origin coincident with the first row of the data view.
BOTTOM	Scroll to the last viewport	BOTTOM sets the viewport origin to the row in the data view which, when mapped into the viewport, causes the last row of data to be coincident with the last row of the viewport.
LOCATE	Scroll to the indicated row	LOCATE sets the viewport origin to the row of the data view indicated by the locate amount.

NEXT is substituted when a scroll action indicates a sequentially forward movement by some scroll amount. PREV is substituted when a scroll action indicates a sequentially backward movement by some scroll amount.

TOP, BOTTOM, and LOCATE are substituted regardless of the scroll amount, and based solely on the scroll action entered.

Scroll Amounts

The scroll amount determines how many lines of data are scrolled forward or backward in the viewport. There are six scroll amounts.

The following table shows scroll amount and viewport displacement.

Scroll Amount	Description
PAGE	Data scrolls forward or backward based on the total number of lines in the viewport. The viewport displacement is the size of the viewport. When the command is NEXT, the displacement is added to the current viewport origin. When the command is PREV, the displacement is subtracted from the current viewport origin.
HALF	Data scrolls forward or backward based on half of the number of lines in the viewport. The viewport displacement is half the size of the viewport rounded down an integer. The displacement must be at least one. When the command is NEXT, the displacement is added to the current viewport origin. When the command is PREV, the displacement is subtracted from the current viewport origin.
CURSOR	Data is displayed in the viewport so that the row that contains the cursor is either the first row or last row in the viewport. The displacement is the same as it is for PAGE when the cursor is on the: First row of the viewport and NEXT is selected. Last row and PREV is selected.
MAX	The viewport origin is relocated to either the first row or the last row of the data view. When the command is PREV, the viewport origin is located to the first row. When the command is NEXT, the viewport origin is located to the last row. If the view is protected, the last row is the last populated row. If the view is not protected, the last row is the last row in the data view extent. This row may or may not be populated.
LOCATE	The viewport origin is relocated to the line number entered by the user. The line number cannot be zero or a negative number. If the number is larger than the number of rows in the data view extent, then the viewport is relocated to the last row of the data view, just as it would be for a MAX NEXT command.
NUMERIC	The viewport origin is relocated based on a number entered by the user. The number is the viewport displacement that is added to, or subtracted from, the viewport origin. The number cannot be zero or a negative number. This is similar to PAGE scrolling, except that the displacement equals a number provided by the user instead of the number of rows in a viewport.

The Page, Half, Cursor, and Numeric displacement scroll amounts are retained after they are set. The Locate and Max scroll amounts are processed and then discarded, returning the scroll amount to its previous setting. The setting remains in effect until the user changes the scroll amount to a different value.

Also, Page is the scroll amount assigned when an application is started without a profile, or reset if a profile exists. If a restart is performed and a valid profile exists, then the scroll amount is recovered from the profile and reset to the last value it had in the application.

Processing of all scroll actions is performed using a two-phase process which depends on the following criteria:

- Protect/unprotect property of the repeating group view
- Boundaries of the repeating group view
- Scroll viewport positioning property

In the first phase, the viewport origin is relocated based on the scroll request. In the second phase, the viewport origin is evaluated, a post processing action is performed, and the viewport origin is relocated if needed. The second phase processing depends on the mode of scrolling used, and the protection property of the repeating group view being scrolled.

Synchronized Scrolling

Synchronized scrolling applies to all screens having two or more scrollable fields. With this scrolling style, all scrollable fields are scrolled at the same time and in the same direction, based on the scroll command the user enters.

When the viewports of the scrollable fields are different sizes, the smaller viewport is used when determining how many rows to scroll forward or backward.

For example, if one viewport contains five rows and another viewport contains seven rows, then five is the viewport size CA Gen uses to scroll both fields.

When the data view boundaries are different for all fields, the largest data view extent is used to determine the boundaries for all fields. It does not matter whether the fields are protected or unprotected.

For example, when there are two repeating group views, one with an extent of 50 and the other with an extent of 100, then the extent size of 100 is used as the data view boundaries for both repeating groups for the purpose of scrolling. Also, if there are two data views, one protected and a population of 90, and another unprotected with an extent of 200, then the scroll boundary for both is 200. The protected data view is extended (logically) by 110 to make up the missing rows. In this case, the viewport for the protected data view is scrolled past the boundary and the viewport is spaced and protected to prevent entry even if the fields are not defined as protected.

When there are mismatches between the boundary sizes of the repeating group views, then the unpopulated areas of each view can be modified by the user if the screen property for unused occurrences is set to Unprotected to Allow Updates. The unpopulated areas of each view cannot be modified by the user if the screen property for unused occurrences is set to Protected to Prevent Update. In either case, the viewport is protected for undefined areas of any data views that are smaller than the data view extent.

Scroll Viewport Positioning Property

The scroll position property controls relocation of the viewport when the viewport is scrolled past the boundaries of the data view extent. This scroll position property forces relocation to the top of the data view, or causes it to remain in its previous location after adjustment to be inside the data view, if any. The scroll position property only applies to Cooperative Scrolling, also called Mode 2.

The position property controls the relocation of the viewport to ensure that the viewport always is placed within the data view boundaries. When the viewport is relocated to the top, or first row, of the data view, then the viewport origin is set to coincide with the first row of the data view. When the viewport is relocated to the end of the view, then the viewport origin is set such that the last row of the viewport is coincident with the last row of the data view boundaries. When the view is protected and the population does not equal or exceed the size of the viewport, then positioning to the end of the data view is ignored and the viewport is positioned to the beginning of the data view.

Relocation of the viewport depends upon the scroll positioning property specified by the user.

If scrolling is forward, then the NEXT common command is passed to the procedure step. If scrolling is backward, then the PREV common command is passed to the procedure step.

If a locate request is issued that positions the viewport outside of the data view boundaries, then it is treated the same as a NEXT common command that scrolled outside of the data view. The common command is set to NEXT.

If a Max scroll operation is requested, as well as a top or bottom request, then the viewport is positioned at the end of the data view boundaries appropriate for the request, and a NEXT or PREV common command is passed to the procedure step. The viewport is positioned at the beginning of the data view if the request was for TOP or Max backward, regardless of the positioning property. Likewise, if the command is BOTTOM or Max forward, then the positioning property determines the viewport relocation.

Index

A

B

C

D

E

F

G

I

J

L

M

N

O

P

R

S

T

U

V

W