CA SOLVE:Operations® Automation

Managed Object Development Services Guide

Release 11.9

CA Technologies Product References

This document references the following CA Technologies products:

- CA SOLVE:InfoMaster™

Contact CA Technologies

Contact CA Support

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

- 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

Provide Feedback

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

If you would like 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 4: MODS Administration 141

Chapter 5: CAS Programming Interface ($CACALL) 155

Chapter 1: Introduction

This section contains the following topics:

MODS Facilities

This section introduces facilities provided by Managed Object Development Services (MODS). MODS is a development environment that provides you with powerful tools for creating and customizing NCL applications. It provides comprehensive menu-and-panel-driven facilities for the definition and maintenance of data, presentation and behavior of applications.

MODS includes the Common Application Services (CAS) functions. These special purpose routines define application elements such as menus and lists that are common to all applications and provide a consistent user interface.

These same facilities let you easily customize delivered applications to your installation's requirements.

MODS provides the following benefits:

- Ability to quickly develop NCL based applications through the use of sophisticated, interactive development tools
- High reliability due to the use of common routines for standard application components
- A single interface for accessing data and invoking functions for all applications
- Ability to maintain test and production libraries and utilities to move components from one library to another
- Whole applications or individual components can be easily isolated, modified, or replaced due to the registration of application components

The MODS maintenance facilities have the following features:

- **Panel Driven—**Tailoring is performed through full-screen interactive panels. You do not need a knowledge of NCL to perform tailoring.
- **Data Input Assistance—**Selection lists of valid input are available to assist you in performing tailoring and maintenance functions.
- **Text Editor—**The editor allows easy text entry and alteration. The editor is available for maintaining help text, message text, panels, and presentation formats for lists and reports.
- **Help—**Comprehensive online help is available throughout the MODS maintenance functions. Help is invoked by entering the HELP command or pressing F1 (Help).

What is MODS?

The MODS environment comprises the following:

Application Register

The application register maintains application definitions. All applications that are built using MODS must first be defined in this register; applications are assigned an application identifier that is used to name all components that belong to that application.

Panel Maintenance

A facility for creating and tailoring full-screen panel definitions.

Common Applications Services (CAS)

The Common Application Services (CAS) functions are a collection of high quality special-purpose NCL routines designed to facilitate program development. The CAS application components manage the presentation aspects of applications—such as menus, lists, messages, online help, and panel navigation.

Report Writer

Report Writer (see page 257) is a facility for creating and tailoring report definitions, enabling you to tailor reports to your exact requirements.

Mapping Services

A facility that lets programmers define complex data structures for use by NCL applications.

Administration Functions

Facilities for maintaining MODS control libraries (for application component definitions) and panel libraries (for panel definitions).

Notational Conventions

The following conventions apply to statement and function descriptions in this guide:

UPPERCASE Characters

Commands and operands are presented as uppercase characters, but can be entered in upper or lower case.

***Italic* Characters**

Italic characters represent variables and show the type of information, rather than the exact information, that must be supplied. The actual entry replaces the italic description. The types of valid data are described in the *Operands* section of each command.

Underscored Values

An underscored value indicates the default optional value that is assumed for an operand if that operand is not specified.

Braces { }

Braces indicate the available options for a required operand. One of the alternatives listed must be selected. Do not include the braces when entering the desired option.

Square Brackets [ ]

Operands in square brackets, including any accompanying equal signs, are optional. Do not include the square brackets when entering the desired option.

Or-sign |

The or-sign is used to separate options. If a group of options is enclosed by square brackets, and the individual options are separated by or-signs, none of the options in the group must be chosen. If none of these operands is entered, the default value is used (default values are always underscored).

Commas, Quotes, and Equal Signs

Commas, quotes and equal signs must be entered as shown. When commas and equal signs appear within brackets, they are optional and are only used if the accompanying optional operand is used.

Related Documentation

Other documentation useful for MODS administrators includes:

- *Reference Guide*
- *Network Control Language Programming Guide*
- *Network Control Language Reference Guide*

Concepts and Facilities

This section describes the MODS development environment. MODS provides facilities for defining and maintaining common application components used to build an application.

MODS Facilities

MODS gives you a powerful set of development tools for building your own, or customizing delivered, applications.

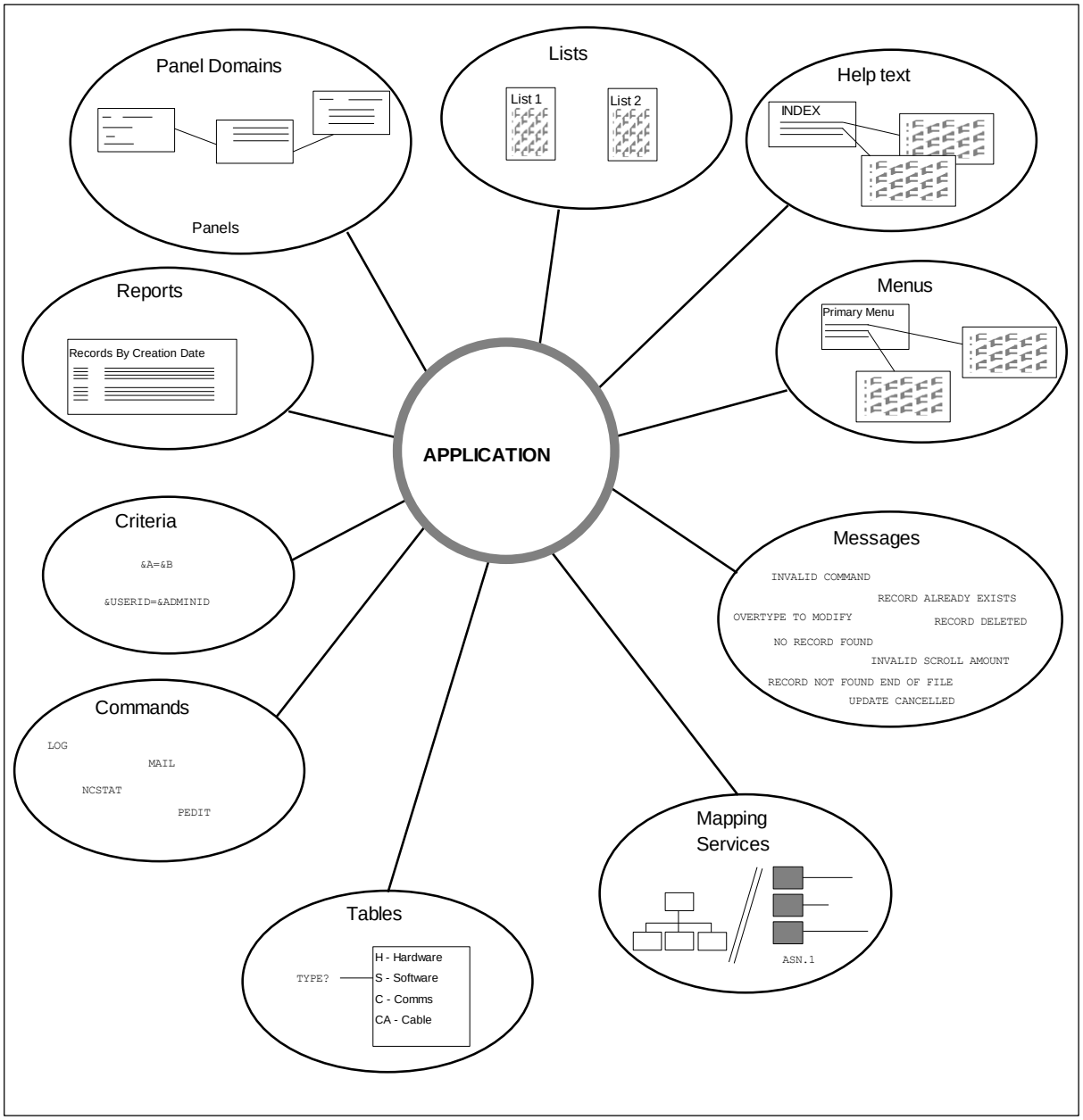
MODS provides you with facilities for defining, customizing, and printing the following application components: panels, menus, lists, reports, help text, messages, commands, criteria and tables.

Programming interfaces are available that let you include these components in your own NCL applications.

Note: Delivered applications should be customized only if this is expressly stated in the application-specific manuals.

The Mapping Services facility provided by MODS lets you provide a logical view of data without needing to understand its physical representation.

The following figure illustrates the components of an NCL application in the MODS environment.



Application Register

All applications must be defined in the Application Register.

When you define an application you must specify a unique 3-character identifier (the Application ID), which is used to tag all MODS components belonging to the application.

More information:

Naming Standards (see page 45)
Registering an Application (see page 48)

Panel Maintenance

The screens used in the applications are referred to as panels. Applications can make use of four general types of panels:

- Menu panels
- List panels
- Data-entry panels
- Text-entry panels

Menu and list panels are defined using the CAS Menu and List maintenance functions and their presentation is controlled by using these facilities.

Data entry panels are defined using MODS Panel Maintenance. A panel definition specifies the input and output fields that appear on a panel (as well as some aspects of their behavior), and controls the appearance of the text on the panel (for example, color and highlighting). These panels are invoked using the &PANEL NCL statement.

Note: For more information, see the *Network Control Language Programming Guide*.

Text-entry panels are used to display and maintain freeform text. These panels differ from other panels in that you do not need to provide a panel definition—a standard panel is presented by the CAS text editor (see page 31) facility.

More information:

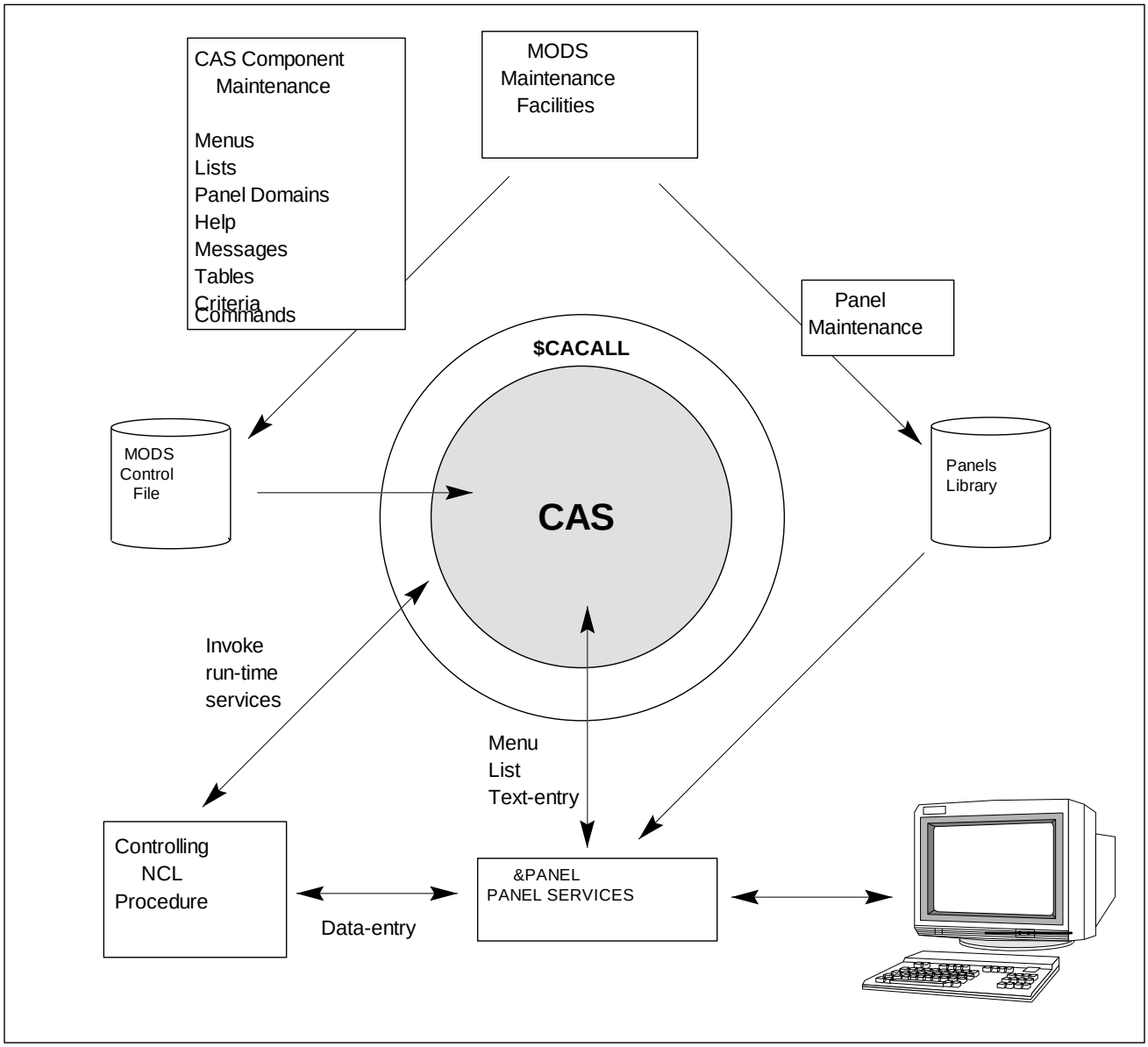
Maintaining Panels (see page 50)

Common Application Services (CAS)

Common Application Services (CAS) consists of development facilities that are used to define CAS components and run-time services that control interaction with these components during execution.

CAS application components are defined using the MODS maintenance facilities.

NCL based applications must use a controlling NCL procedure to invoke CAS run-time services through the CAS application programming interface (API), $CACALL. The relationship between panels, CAS application component definitions, and CAS run-time services is shown in the figure below.



CAS Programming Interface

The CAS application programming interface ($CACALL) is used to invoke CAS run-time services from a controlling NCL procedure.

The API requires you to specify the action that you want to perform and the type and name of the component to which the action applies.

For example, you can specify a class of MENU with the action DISPLAY and specify the menu name in order to display a menu definition that you have defined using the MODS menu definition facility.

Additional parameters may be required depending on the application component and action being performed.

More information:

CAS Interface Overview (see page 155)

Menus

A menu is a panel that presents the user with a number of options. Each option performs a specified action.

A menu is built from a menu definition that defines the format of the menu, the menu options and their associated actions, and any input fields required to support an option.

CAS supports panel-skipping between menus. For example, entering **D.C.O.L** goes directly to the last option specified, skipping the display of the three intervening menus.

You can control the behavior of a menu at various processing points, by specifying a menu exit procedure.

When a menu is invoked using the CAS API, CAS builds and displays the menu, and processes the user's selection. When a user selects an option from the menu, and at other defined processing points, CAS calls the menu exit procedure, if it is defined, to perform installation specific processing.

The action that is associated with a menu option can be a further call to the API. For instance, a menu option on a primary menu often leads to a submenu—you can display this menu by specifying a call to the CAS API as the action associated with the menu option. You can invoke other application components (a list, for example) from a menu definition in this way.

Lists

A list displays a series of items from which the user can make a selection of, or perform an action against, one or more items.

A list definition contains identifying information, the name of a service procedure that retrieves the list's entries, the identifier of an (optional) criteria definition that filters items for inclusion in the list, the name of an (optional) exit procedure that performs installation specific processing at various points, and the display format for the list.

The format of a list specifies the placement of list items and static text on the list panel. A list format can cover up to ten screens. That is, if the information you want to display for each item does not fit on a single screen, you can add a second screen, third screen, and so on. The user can scroll between these screens by entering the RIGHT and LEFT commands (or using the appropriate function keys).

The list service procedure retrieves list items and processes requests to perform actions against list items.

The list service procedure checks the list's data source (if it is defined) in order to determine how to retrieve list entries for different sources of data. For example, the data source could be an CA SOLVE:InfoMaster category or a file name. This lets multiple list definitions share the same service procedure. A data source need not be specified—in this case the service procedure retrieves entries from a source defined within the procedure.

A list can contain the identifier of an exit that is an NCL procedure used to perform installation specific processing. The exit procedure (see page 237) is called at various processing points; for example, during list initialization, or after an entry is retrieved.

The same list definition can be used to build four types of lists:

- **Action lists**—let the user apply actions (for example, Browse, Update, Delete, Copy) to one or more items on the list.
- **Single Select lists**—let the user select one item from the list. The selected item is passed back to the calling procedure.
- **Multiple Select lists**—let the user select one or more items from the list. All items selected by the user are passed back to the calling procedure.
- **Numbered lists (Pick lists)**—let the user select one item from the list by entering the appropriate number in the Select Entry field. The item corresponding to the number that the user selected is then returned to the calling procedure.

You specify the type of list that is to be built from a given list definition in the call to the CAS API.

The CAS list maintenance facilities (see page 98) enable you to define list definitions for use by your applications as well as modifying supplied list definitions.

CAS handles the selection of items for the list and the display of lists to the user, providing: full scrolling functions, the FIND and LOCATE commands, and confirmation of the user's selection.

Help

CAS provides a facility to define and display help text. Help text can be structured in panel format, in simple text format, or using a combination of both.

Facilities for constructing help menus, selection lists of help topics, help indexes and tutorials are available. Help text can be merged or copied, both during maintenance and while being displayed.

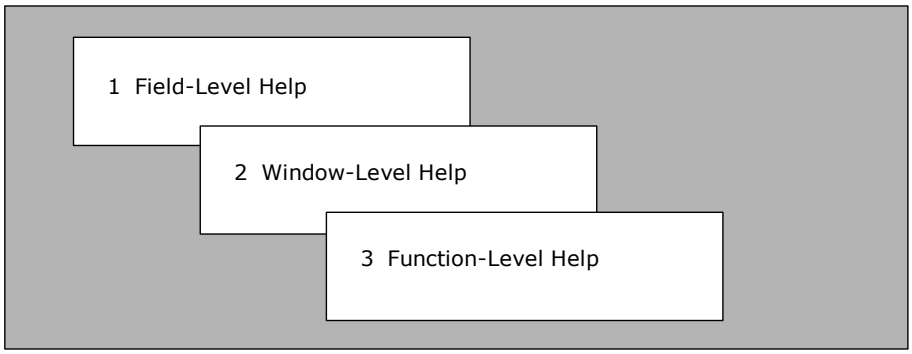
Help text is associated with a particular function, window (a logical area that facilitates context sensitivity), or field.

More information:

Maintaining Help (see page 106)

Help Hierarchy

There are three levels of help. Help text can be defined at any or all of these levels, as shown in the following figure.



The order in which help is presented to the user is levels 1 to 3, that is, beginning with the most specific help available, and becoming more generalized.

- **Field-Level Help** describes a single field, the data that can be entered by the user, and what happens as a result.
- **Window-Level Help** provides help for a group of logically related fields within a panel. A window is an area that covers part or all of a physical panel. There can be many windows defined for one panel; windows can overlap.
- **Function-Level Help** describes the function that the user is currently performing. Function-level help relates to one particular function within the application (for example, the Add Record function within a maintenance application).

For example, a panel might have two help windows defined as shown in the following diagram:

```
XXXXX---------------- Sample Application : Add Record -------------Page 1 of 1
Command ===>                                                      Function=Add



Record ID ...... ___          X            Userid ........________
Record Name .... ___________X              Date ..........________
Description .... ____________________

   Window A                                    Window B

 F1=Help      F2=Split     F3=File      F4=Save      F9=Swap      F12=Cancel
```

- If a user enters the HELP command from this panel, with the cursor positioned in the Record ID field, then CAS displays the field-level help describing the Record ID.
- If the user then re-enters the HELP command, CAS displays the window-level help for Window A (this file describes the Record ID, Record Name and Description fields).
- Entering the HELP command again displays the function-level help for the Add Record function

Messages

A message is a text string that is used to communicate information to the user—for example, an error message. CAS provides facilities for maintaining, building, and displaying messages.

Each message has a unique identifier and associated text containing an explanation of the message, the system action and the action that should be taken by the user when the message is displayed.

Variables can be included within messages. When a message is built the variables are substituted into the message to provide specific information relating to the message—for example, an error code.

CAS provides centralized control of messages. All applications call CAS to build messages for display.

More information:

Maintaining Messages (see page 112)

Validation

CAS provides facilities to validate data entered in input fields (or any other data) against either:

- A table of defined values
- Predefined rules (for example, a range test or a date format)

Tables

Tables contain a set of entries against which data can be validated.

A table comprises a table definition and a number of table entries that represent the valid values. CAS provides a facility for creating and maintaining table definitions and entries (see page 115).

Each table entry can have an abbreviated value, description, and up to ten associated data fields.

Valid values in a table can be specified explicitly as table entries or can be drawn from the following sources:

- A list of values in a field of a CA SOLVE:InfoMaster category
- A list of field names in a CA SOLVE:InfoMaster category
- A list of entries supplied dynamically through the CAS API

You can define an exit procedure for a table that performs installation specific processing during table entry maintenance—for example, to validate table entries or restrict the deletion of entries.

Prompted Fields

CAS supports field prompting: if the data to be validated contains a question mark (?), then CAS displays a list of all valid values for the field, with a description of each. The data can contain just part of the value, followed by a question mark; this displays all valid values that generically match the supplied value.

When you use the CAS API to validate data against a table this facility is provided by CAS with no additional code required on your part. This facility can be turned off through a parameter to the API (for instance, when you want a question mark to be valid input).

Other Types of Validation

CAS provides other types of validation including alphanumeric, hexadecimal, NCL keyword, and time. You pass the input you want validated to the CAS API and specify the type of validation. You do this by specifying one or more edit numbers in a parameter to the CAS API.

Criteria

A set of criteria is a set of rules that can be used to test a condition. For instance, a set of criteria can be used to select items to go in a list, or items to go on a report, or to validate users' input.

Criteria can simply compare static values or can be complex, combining numerous operators and values. Values can be variable (for example, the current date can be used as a value).

When you recall a set of criteria through the CAS API to perform a test, variable values are supplied by the calling procedure, interactively by the user (through a run-time panel), or by an exit procedure that you define. The criteria exit can process the entries made on a run time panel and also determine whether the panel was actioned or canceled.

You can specify a data source for the set of criteria, which the exit uses to determine how the criteria are processed. You can define exit parameters within the criteria definition for the exit. This lets you change the criteria behavior without having to rewrite the exit procedure and also lets you write a generalized exit that can then be used by other criteria definitions.

More information:

Maintaining Criteria (see page 118)
Criteria Exit Procedures (see page 245)

Commands

CAS provides facilities for defining and processing commands issued by users and by applications.

A command definition contains the unique identifier of the command and an action to be performed when the command is executed.

The CAS API can be invoked to execute defined commands.

More information:

Maintaining Commands (see page 123)

Text Editor

The CAS text editor is a full screen editor that can be included in any NCL application, providing the user with comprehensive editing facilities for up to 32767 lines of text (each of up to 256 characters in length).

- Standard text manipulation facilities are provided via line commands: insert, delete, move, copy, repeat, queue, text split, text flow, and text entry. These are supported singly and as block commands.
- Scrolling functions are supported (FORWARD, BACKWARD, LEFT, and RIGHT), including the ability to specify scroll amounts.
- The ability to find occurrences of a text string is supported, as well as the ability to change one or all occurrences of a nominated text string.
- The ability to position the text on a given line number is also supported.

You can provide text editing facilities from your application by invoking the text editor through the CAS API and passing the text to be edited. The edited text is returned to the calling procedure.

More information:

Using the Text Editor Commands (see page 347)

Text Browse Facility

Text can be displayed in Browse mode, letting users view but not update text. This facility can be included in any application, providing comprehensive text browsing facilities for up to 32767 lines of text.

The text browse facility is invoked through the CAS API.

Function Key Areas

The function key area (FKA) refers to the bottom two lines of a panel, where function keys and their labels are displayed.

You can modify the labels and actions associated with function keys through the CAS API.

CAS provides predefined function key sets for specific purposes, such as Browse and Update.

The FKA lines are formatted and returned to the calling procedure either when the CAS API is invoked to set key settings or when the KEYS command is executed by the user.

Report Maintenance

Report Writer is an application that is used to define user reports.

Report Writer has the following features:

- A full screen report design and maintenance facility
- A report generation facility
- A scheduler for automating the production of reports
- A maintenance function for defining applications to Report Writer

A report definition is created using a text editor that lets you define the layout of a report on the screen. Report definitions are stored on a database and can be recalled at any time to produce the report or modify its format and contents.

Report Writer is designed to operate independently of the database in which the report data is contained and can create reports from data stored in any database that is defined to it.

A report definition consists of the following components:

Description

Defines control information about the report such as its name, description, and the application that it belongs to.

Sort fields

Define the order in which records are sorted on the report.

Format items

Are lines of text that are printed on the report—each format can consist of any number of lines, made up of both constant and variable data.

The following report items are defined for each report:

Report Header

Defines material printed at the top of the report.

Page Header

Defines material printed at the top of every page.

Data Formats

Define material printed for each record that is passed to Report Writer—there can be multiple data formats.

An NCL exit procedure can be used to determine which data format or group of data formats to use for each individual record. If there is no exit procedure, all data formats are printed.

Control Break Headers

Prints headings above groups of data. Control break headers can be printed each time a field on which the data is sorted changes value.

Control break Trailers

Prints trailers below groups of data. Control break trailers can be printed each time a field on which the data is sorted changes value and are commonly used for printing subtotals and totals.

Page Trailer

Defines material printed at the bottom of every page.

Report Trailer

Defines material printed at the end of the report.

Data is secured against illegal access by using the User Access Maintenance Subsystem (UAMS) facility. Report Writer interfaces to Print Services Manager (PSM) for the management of report output.

Mapping Services

Mapping Services is a facility that gives NCL access to complex data structures.

Mapping Services is designed to separate the application's data processing requirements from a need to understand the actual organization of the data.

It means that NCL procedures deal with the logical relationships and usage of data (the data protocol), while the system manages and maintains the physical representation of the data (the data format).

Conventional NCL processing deals with simple data items accessed as NCL variables (or tokens). If a number of variables are logically related, the programmer must understand, through naming conventions or other disciplines, how data items are related and how they must be managed.

Difficulties arise in a number of circumstances, such as where data to be processed by NCL is sourced externally, or where NCL must define an interface to some other processing system. In such cases data is exchanged across a program interface, according to a strict protocol, and can be conveniently represented as some sort of protocol data unit. Such a protocol data unit is usually composed of one or more logical components, but must be presented across the interface as a series of bytes, and hence is structured according to some encoding technique.

Nearly all management data conforms to this simple model; however, there are many different encoding techniques employed. These range from very simple rules involving fixed length fields, one following another, to more complicated rules involving variable length structures, and even more complex rules involving self-defining lengths, tags, or similar structures.

The use of variable length data items, and tagged data structures, is popular because it encourages programming precision, and provides a continuous upward migration path. By extending the length of existing structures, or inventing new ones within a data unit, it can retain its original character while evolving to keep pace with new requirements.

Maps and Mapped Data Objects

Mapping Services provides NCL with a logical view of data, while removing from NCL the requirement to understand the physical representation of data. It does this by interposing a map that is used to interpret the data. A map contains the definition of the logical components of a complex data structure, as well as the data's physical representation.

A complex data structure is processed by NCL as a Mapped Data Object (MDO). The NCL procedure can connect an MDO to a map. Only through the map can the NCL procedure access the logical components of the MDO. Once the map connection is made, the NCL procedure can reference the logical entities contained within the MDO by their symbolic names defined within the map.

Each time NCL references a symbolic name, Mapping Services locates the definition within the map, determines how the component is organized, and with a knowledge of the physical representation, navigates the data structure to access the logical data items.

Using this technique, only the system need be aware, through definitions contained within the map, of the actual representation of the data. The NCL programmer, and the procedure itself, need only understand and reference the logical data components.

By modifying the map alone, it is possible to alter the underlying physical data definitions as managed by the Mapping Services facility, without having to change any NCL code.

Map Library

Maps comprise a number of definition records, and are compiled to a loadable form. Map source can be kept in any convenient source library in a similar fashion to NCL procedures.

All compiled maps are lodged in a file that serves as the Map Library. From this library, the loadable form of a map can be accessed on demand. Normally a map is loaded only when specifically requested for use by an NCL procedure. When no longer required, it can be deleted from storage by the system.

Note: For more information about the use of maps in NCL procedures, see the *Network Control* Language *Programming Guide* and the *Network Control Language Reference Guide*.

More information:

Mapping Services (see page 297)

MODS Administration Facilities

This section provides an overview of MODS administration facilities. Use these facilities to control the storage and maintenance of:

- Panel definitions in panel libraries and the panel paths to be used to retrieve panel definitions
- MODS component definitions in the MODS Control File and the concatenation path to be used when retrieving definitions

Panel Library Maintenance

Panel definitions are stored in VSAM data sets for fast retrieval and update. A VSAM data set containing panel definitions is called a panel library. Multiple panel libraries are supported.

Individual panel definitions are referred to as library members.

A library can be used as the sole source of panel definitions, or it can be concatenated with other libraries defined to the system. A concatenation of libraries is called a panel path. Each user can be defined to use a different path.

The administration functions let you:

- Define and maintain panel libraries
- Copy panels between libraries on different paths
- Temporarily define and maintain panel libraries

More information:

Maintaining Panels (see page 50)

Concatenation Path

The concatenation path can contain multiple files. The concatenation path is defined during initialization in the PANELLIBS parameter group.

MODS Control File Maintenance

A MODS control file contains the following records:

- Application dependent components:
  - $AR—Application Definitions
  - $CR—Criteria
  - $HM—Help
  - $LD—Language Services Definitions
  - $LH—Lists
  - $MH—Menus
  - $MS—Messages
  - $RW—Reports
  - $VM—Tables
- Common (non-application dependent) components:
  - $CM—Commands
  - $PS—Print Services Manager (PSM) Definitions
  - $LD—Language Definitions

Control File Maintenance Facilities

The control file maintenance facilities let you:

- Copy records from one control file to another (for example, from test to production)
- Delete records from a control file
- Browse the records stored on a control file
- Search records stored on a control file

Control File Concatenation

Control file concatenation lets you access multiple control files concurrently via the concatenation path, and provides a distinct separation between distributed, production, and test definitions to simplify maintenance and provide control over the development environment.

The concatenation path can contain multiple files. The concatenation path is defined during initialization in the MODSFILES parameter group.

Merged View

All MODS component lists display a merged view of all files in the path. This means that the distinctions between individual files are ignored and the contents of all files in the concatenation path are eligible for selection.

In the case of duplicate components present at different levels in the path, the topmost instance has precedence over those at lower levels. The identifier of the file in which the component is located is displayed on the right side of component lists.

Automatic Promotion

Only the topmost file in the path is modifiable by means of any of the maintenance facilities. A request to update any definition not resident in the top file retrieves the definition from the lower file and stores it into the topmost file (this is known as automatic promotion). New records are always added to the topmost file in the path.

Manual Demotion (Lodgement)

To move any definition downward in the path, the MODS : Definition Utilities must be used and the source and destination files specifically named.

These utilities also provide a means of viewing the contents of files in isolation. That is, they do not present a merged view, but list only those components that are actually contained in the named file.

Two major benefits flow from control file concatenation:

Operational Flexibility

The ability to give multiple groups of users of the same application different views of that application. A *view* in this context constitutes both presentation and behavioral differences. Views can be used to implement significant usage differences, such as TEST and PRODUCTION.

Maintenance Advantages

Library concatenation greatly eases maintenance difficulties and facilitates better change control with inbuilt backout capability. It also lets you manage changes to distributed applications.

Sharing Control Files

MODS control files can be shared between regions where multiple product regions exist (for example, a production and test region). You must ensure, however, that only one of these regions is capable of updating records stored on the shared control files. Data corruption can occur if this is not done.

Update capability is specified when defining the concatenation path in the MODSFILES parameter group for each product region.

WebCenter

The WebCenter files are stored in MODS. These files have no relation to the other MODS components.

Chapter 2: Accessing the MODS and CAS Facilities

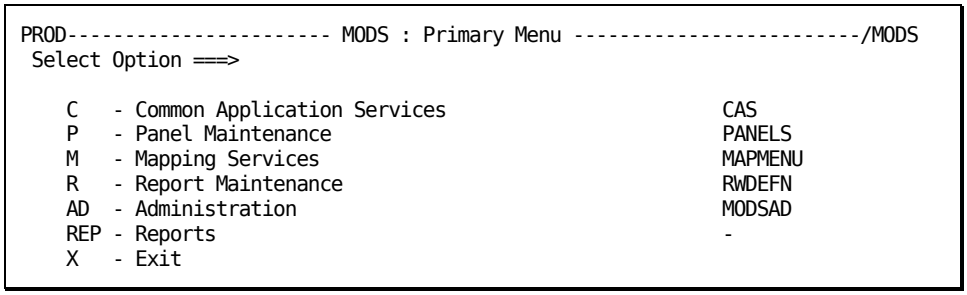
This section contains the following topics:

Accessing MODS Facilities

All facilities described in this guide are accessed through the MODS : Primary Menu. You can display this menu by entering **/MODS** at the command prompt.

MODS Primary Menu

The MODS menu structure is displayed in the following figure.



The following options appear on the MODS : Primary Menu:

C – Common Applications Services Maintenance

This option displays the CAS : Maintenance Menu used to maintain CAS application components (see page 42).

P – Panel Maintenance

This option lets you maintain individual panels (see page 50).

M – Mapping Services

This option lets you maintain ASN1 maps (see page 127).

R – Report Maintenance

This option lets you add, update, delete, copy, list and view report definitions (see page 257).

AD – Administration

This option displays the MODS : Administration Menu, giving you access to:

- Panel library maintenance facilities (see page 141)
- MODS control file maintenance facilities (see page 145)

REP – Reports

This option displays a list of predefined MODS reports that you can generate (see page 136).

X – Exit

This option returns you to wherever you were prior to accessing the MODS : Primary Menu.

MODS Access Authority

To access the MODS : Primary Menu, you need to have the Managed Object Dev. Services field on the UAMS : Access Authorities panel set to Y.

Note: For more information about the MODS : Primary Menu, see the *Security Guide*.

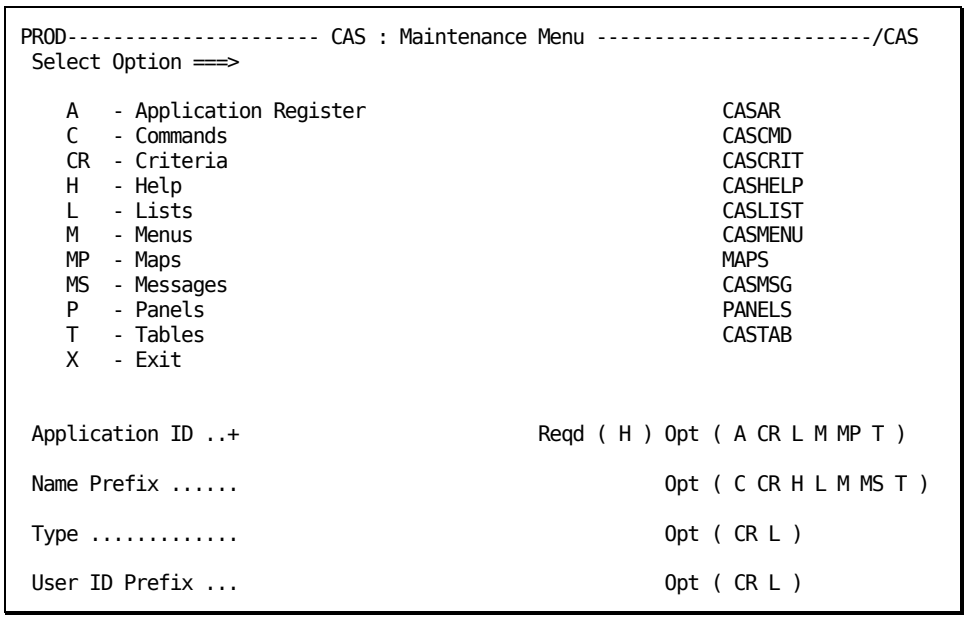
Accessing CAS Facilities

All Help facilities described in this guide are accessed through the CAS : Maintenance Menu. You can display this menu by entering **/CAS** at the command prompt or entering **C** at the MODS Primary Menu.

CAS Maintenance Menu

This menu enables you to maintain CAS definitions, including menus, messages, and help definitions that you may use when developing applications.

The CAS menu structure is displayed in the following figure.



The following options appear on the CAS : Maintenance Menu:

A – Application Register

This option lets you maintain application definitions. The first step when creating an application is to define the application in the Application Register.

C – Commands

Displays a list of command definitions.

CR – Criteria

Displays a list of criteria definitions.

H – Help

Displays a list of function-level help definitions.

L – Lists

Displays a list of list definitions.

M – Menus

Displays a list of menu definitions.

MP – Maps

Displays a list of map definitions.

MS – Messages

Displays a list of message definitions.

P - Panels

Displays the MODS Panel Maintenance Menu.

T - Tables

Lists table definitions.

CAS Application Components

You can maintain the following CAS application component types:

- Application Register (see page 48)
- Commands (see page 123)
- Criteria (see page 118)
- Help (see page 106)
- Lists (see page 98)
- Menus (see page 94)
- Messages (see page 112)
- Tables (see page 115)

Select any of these options to display a list of application component definitions where common actions are provided to maintain individual definitions as follows:

- To add a definition, press F4 (Add).
- To browse, delete, copy, or update a definition, enter the appropriate mnemonic (B, D, C, or U) beside the entry for the definition.

The panels that appear vary depending on the application component type. Some of the application component types support additional actions.

Chapter 3: Maintaining Application Components

This section contains the following topics:

Naming Standards

This section describes naming standards that must be used for applications that you develop.

Note: These standards do not generally apply when you are customizing a supplied product. Where customization is permitted for a given product, the specific standards that apply are described in the documentation for that product.

More information:

Application Register (see page 21)

Application Definitions

The primary method of applying naming standards is through an application identifier.

An application is a group of logically related functions and/or data. Every application is defined in the Application Register and named with a unique identifier. This application identifier provides a unique name space for application components.

Before you can define application components, you must define an application definition within the Application Register.

The following sections detail how individual application components are named.

Note: For applications that you define, ensure that the first character of the application identifier is Y. This provides name space protection from distributed product definitions and your applications.

Messages

The message prefix, specified within an application definition, must be the same as the application identifier.

Panels

The first three characters of the Panel identifier must be the same as the application identifier.

NCL Procedures

The first three characters of the procedure name must be the same as the application identifier.

CAS Components

The application identifier must be explicitly specified for Menus, Lists, Panel Domains, Help, Tables, and Criteria Definitions.

Note: The help function names OVERVIEW and INDEX are reserved for application overview help and the help index, respectively.

The first three characters of Message and Command identifiers must be the application identifier.

Reports

The first three characters of a report application identifier must be the same as the application identifier.

The Report Application Identifier must be explicitly specified within Report Definitions.

Map Definitions

The first three characters of a Map Name must be the application identifier.

Externally Visible Entities

In addition to application components there are other entities that are visible outside an application and therefore require name space protection by prefixing their names with the application identifier. These include the following:

- Global variables
- Global vartables
- Locks
- Variables/MDOs that are shared between applications
- Data Domain names
- File identifiers
- NDB identifiers
- EDS events
- NDB Global Format names

Maintaining Application Component Definitions

You can do the following with application component definitions:

Browsing an Application Component Definition

Select option B to browse an application component definition. The panels that appear will be specific to the type of definition that is selected. The function is Browse and the fields cannot be modified.

Updating an Application Component Definition

Select option U to update an application component definition. The panels that appear will be specific to the type of definition that is selected. When you have finished updating the definition, press F3 (File). To cancel the update, press F12 (Cancel).

Deleting an Application Component Definition

Select option D to display a message requesting you to confirm the deletion. Press Enter to delete the definition, or press F12 (Cancel) to cancel the deletion.

Copying an Application Component Definition

Select option C to copy an existing application component definition. The panels that appear will be specific to the type of definition that is selected.

The panels are primed with data from the copied definition. Modify the new definition as required.

When you have finished updating the definition, press F3 (File). To cancel the update, press F12 (Cancel).

Adding Application Component Definitions

You can create an application component definition by adding a new definition, either by using F4 (ADD) on a list, or by copying an existing definition and making changes to the copy.

When you have finished adding the definition, press F3 (File). To cancel the update, press F12 (Cancel).

Note: For information about the fields and actions on any of the CAS panels, press F1 (Help).

Registering an Application

An application is a group of logically related functions and/or data. Before an application's components can be defined, an application definition must be created in the application register.

This section describes the facilities available for adding and maintaining application definitions.

More information:

Application Register (see page 21)

Application Definitions

An application definition consists of a three-character identifier, a descriptive name, a one- to eight-character message prefix, and, optionally, some comments.

The application identifier (or application ID) provides a unique naming space (see page 45) for an application's components—it prefixes the identifiers of all these components. This makes it easy to find all components that are used by a particular application and simplifies maintenance of component definitions.

Defining an Application

You must specify the following information when defining an application:

- Application ID
- A brief description
- A message prefix

This is shown in the following sample.

```
PROD------------------- CAS : Application Definition ---------------Page 1 of 1
Command ===>                                                    Function=Browse

Application ID ... $AN
Description ...... SNA Management Services (APPN)
Message Prefix ... AN
Comments .........









 F1=Help      F2=Split     F3=Exit
                           F9=Swap
```

For information about the fields displayed on the panel, press F1 (Help).

After specifying this information, press F3 (File) to add the application definition. To cancel the application specification, press F12 (Cancel).

Maintaining Application Groups

Any application that uses lists and reports must have its own copy of the table $ADGROUP in which the names of associated groups are defined. The $ADGROUP table is currently available only to CAS List Services and Report Writer. Regardless of which application contains the table, $ADGROUP's attributes must always be defined (except for the Appl ID and Field description fields, which need to be changed for the relevant application).

```
PROD--------------------- CAS : Table Description -----------------Page 1 of 2
Command ===>                                                    Function=Browse

Appl ID .................. $AD   Field name ............... $ADGROUP
Field description ........ Sample Group Defn.
Edit type ................ TABLE (TABLE, OSATT, OSFLD, IMFLD or IMREC)
For Edit type = TABLE:
   Validation exit ................. $CAVM040
   Sequence numbers ................ NO  (YES or NO)
   Load table? ..................... YES (YES or NO)
   Max abbreviation length .........     (3 - 8 or blank if none)
   Max full value length ........... 12  (3 - 20)
   Max description length .......... 38  (3 - 38 or blank if none)
For Edit type = IMFLD or IMREC:
   InfoMaster category .............
For Edit type = IMREC:
   InfoMaster field ................
   InfoMaster description field ....
For Edit type = OSATT or OSFLD:
   Object Services Class ID ........
For Edit type = NDBFL:

 F1=Help      F2=Split     F3=Exit                              F6=Entries
              F8=Forward   F9=Swap
```

More information:

Maintaining Tables (see page 115)

Maintaining Panels

This section describes the MODS (see page 52) : (see page 52) Panel Maintenance facilities (see page 52) for creating, maintaining, and customizing full-screen panel definitions (see page 58).

More information:

Panel Maintenance (see page 21)
Panel Library Maintenance (see page 36)

Panel Maintenance

Panel definitions can be used by NCL processes executing in any NCL environment associated with a display window. The panels enable NCL processes to display output data, and can be designed with input fields for communicating back to these NCL processes.

For information about how NCL processes use panel definitions, see the *Network Control Language Programming Guide*.

Panels are created and changed using an online editor. You must be authorized to use this facility, and installations can limit the number of concurrent edit users by restricting the amount of storage available to the editor.

The split-screen facilities are very useful when designing panels: the panel editor can be used on one window, while the panel view facility of MODS : Panel Maintenance displays the current version of the panel being developed on the other.

Concepts and Terminology

Panels are stored in VSAM data sets for fast retrieval and update. A VSAM data set containing panel definitions is called a panel library, with individual panel definitions being termed members of the library. Each member maintains details on the date of creation, the date of last modification, the user ID, the number of statements in the member, and the modification level.

The product region supports multiple panel libraries. A library can be used as the sole source of panel definitions, or it can be concatenated with other libraries defined to the system. A concatenation of libraries is called a panel path. When a user is defined, the panel path they use is defined. Each user can be defined to use a different path. The default path is called PANELS. For more information about panel paths, see the *Administration Guide* for your product.

When a library is defined, the person defining it can determine if the library can be edited on the system. When a path is defined, the definer can choose to allow or disallow edit of selected libraries in the path.

Note: To update panel definitions in a library using the MODS panel maintenance facilities, the library must have edit allowed by both the library and path definitions.

More information:

Maintaining Panel Libraries (see page 141)

Retrieving Panels From Panel Libraries

When required, panel specifications are retrieved from a library in the user's current path.

To eliminate overheads associated with retrieving the panel from the library, an in-storage queue of active panels is maintained. When a panel is first referenced it is retrieved from a panel library and stored on the active panel queue.

Thereafter, the panel is retrieved from the active panel queue without reference to the panel's library. If one of these panels is modified (using the online editor), any old copy is removed from the active panel queue so that the next reference retrieves the updated panel.

Note: If a panel library is being shared by more than one product region, a modified panel is only removed from the active panel queue of the product region on which the panel change has been made. The other product regions continue to use the old panel until it is rolled off the active panel queue by other panels being used in the system. The LIBRARY REFRESH command can be used to drop all panels loaded from a library from the active panel queue.

Defining and Maintaining Panels

You can create a new panel definition by selecting the Add Panel option from the MODS : Panel Maintenance Menu, or by copying an existing panel definition using the Copy panel option on a list of panels. You can browse, update, delete, list, view, print, rename, and display existing panels using options on the list of panels.

MODS : Panel Maintenance Menu

The MODS : Panel Maintenance Menu can be accessed by selecting option P from the MODS : Primary Menu (or use the /PANELS shortcut).

```
PROD------------------ MODS : Panel Maintenance Menu -------------------/PANELS
Select Option ===>

   A   - Add Panel
   I   - Display Panel Information
   L   - List Panels
   M   - Move/Copy Panels Between Libraries
   S   - Search Panels
   X   - Exit

Path ........... PANELS

Panel Name ..... ____________          ( Required A I Optional L M )

Library Name ..+ PANLUSR               ( Required A L S Optional M )









 F1=Help      F2=Split     F3=Exit      F4=Return
                           F9=Swap
```

All panel maintenance facilities operate within the path defined for your user ID. The editing options shown on the MODS : Panel Maintenance Menu let you add or list panels, or copy the source data for online panels within libraries or between libraries in your path. You can also display or test existing panels.

Your library path is the name of the panel library path defined for you in your user ID definition. The default path name is PANELS.

For information about the fields displayed on the MODS : Panel Maintenance Menu press F1 (Help).

Library Selection List

If you enter a question mark (?) in the Library Name field on the MODS : Panel Maintenance Menu or the MODS : Panel Move/Copy Menu when you select an option, the MODS : Panel Library List is displayed.

All libraries in the current path are displayed. If you select a library by entering its number in the Select Option field, processing continues as if you had typed the library name yourself on the original menu and pressed Enter.

The top right corner of the panel shows the path name. The fifth line of the panel displays the path name and its description.

Note: For information about the fields and options on the panel, press F1 (Help).

Selecting a Panel Maintenance Function

Select one of the functions displayed on the MODS : Panel Maintenance Menu by entering the letter next to the desired option in the Select Option field, and pressing Enter.

Some options require that a library name is specified in the Library Name field. This defaults to the first editable library in your path. You can change this to any other library in your path. If you are unsure what libraries are in your path, enter a question mark (?) next to this field. The MODS : Panel Library List is displayed, which is a selection list of all libraries in the current path.

Note: You can only use panel maintenance facilities on libraries in your path.

Adding a Panel Definition

Select option A to create a new panel definition. Enter the name of the library where you want the panel to be stored in the Library Name field and the name of the panel in the Panel Name field on the MODS : Panel Maintenance Menu. The library name entered must be editable on your library path.

Listing Panel Definitions

Select option L from the MODS : Panel Maintenance Menu to display a list of panels defined in a particular library.

Enter the name of the library in the Library Name field on the MODS : Panel Maintenance Menu. The MODS : Panel List panel is displayed, showing information about each panel defined in the library.

If you make an entry in the Panel Name field on the MODS : Panel Maintenance Menu, MODS : Panel List displays the list of panel definitions starting with the panel name that matches the value you enter. (If this is a partial panel name match, the first panel name that partially matches the value that you entered appears as the second item in the displayed list.)

Note: For information about the fields and actions on the panel, press F1 (Help).

Special List Commands

In addition to the standard commands available with all lists, the following special commands are available on the MODS : Panel List:

S *panelname*

This command lets you select a new or existing panel for update, without having to scroll to the correct place in the selection list. Similarly, any of the other line selections (except Delete) can be entered as primary commands if followed by the panel name. For example, you can view a panel named MYPANEL by entering **V MYPANEL** on the command line.

SORT

This primary command can be used to change the sort order of the list. The list is normally sorted by name, but the sort command can be used to sort it by any of the other columns displayed on the list. For example, **SORT CRE** can be used to sort the list by creation date.

Secondary sort fields can also be specified. For example, **SORT ID SIZE** sorts the list by the name of the user who last updated the panel, and by size for each user ID. The valid sort fields are Name, Created (or Cre), Modified (or Mod), Mlev, and ID. The sort process can take some time if you are listing a large panels data set.

Viewing a Panel Definition in Display Format

On the Panel List panel, enter **V** beside a panel definition to see what it looks like when displayed by an NCL procedure.

The view function displays the panel in the nominated library. Note that the displayed panel is not necessarily the panel that would be displayed by any NCL user with the same panel library path. A panel with the same name can be in another library in the path.

Printing a Panel Definition

On the Panel List panel, enter **P** beside a panel definition to print it. The panel definition is printed using Print Services Manager (PSM).

For more information, see the User Guide for your product.

More information:

MODS Component Reports (see page 137)

Renaming a Panel Definition

On the Panel List panel, enter **R** beside a panel definition to rename it within the selected library.

Display Information About a Panel Definition

From the Panel Maintenance Menu, select option I to display a list of all the libraries, in your path, that contain a specified panel. You must specify the name of the panel in the Panel Name field on the menu.

All libraries in your path are displayed in concatenation order.

The top left of the display shows the library name that was specified on the MODS : Panel Maintenance Menu, called the current library. The top right of the panel displays the path name.

If the panel is not defined in a library, ***Not Present is displayed next to the library definition. For information about the displayed fields, press F1 (Help).

Statistics for the panel in the current library are displayed in high intensity. This information tells you where the highest instance of a member is in the library concatenation.

Enter an **S** or **B** next to a library to browse the panel definition stored in that library.

Moving and Copying Panel Definitions Between Libraries

From the Panel Maintenance Menu, select option M from the MODS : Panel Maintenance Menu to move or copy panels from one library to another within your path.

If you enter a name in the Library Name field or the Panel Name field on the MODS : Panel Maintenance Menu, this information is used on the Move/Copy Menu.

Note: You can also copy panels between libraries on different paths (see page 141).

The MODS : Move/Copy Menu is displayed, letting you select the libraries and panels that you want to move or copy.

```
PROD-------------------- MODS : Panel Move/Copy Menu --------------------------
Select Option ===>

   C   - Copy Panel Definitions
   M   - Move Panel Definitions
   X   - Exit

Path ..................... PANELS         ( Your path name )

'From' Library ..........+

'To' Library ............+ WORK

Panel Name ...............                ( Blank, Full or Generic name,
                                           e.g. '*'  for all panels, or
                                                'D*' for all starting with D )

Replace Like-Named Panels? NO              ( YES or NO )

Copy All Matching Panels?  NO              ( YES or NO )


 F1=Help      F2=Split     F3=Exit      F4=Return
                           F9=Swap
```

When the Copy option is chosen, the definition is loaded from the FROM Library and stored into the TO Library. When the Move option is chosen, the definition is deleted from the FROM Library and added to the TO Library.

To Move or Copy a panel, enter the names of the FROM Library and TO Library. Both libraries must be in your path (your path name is displayed on the screen, and cannot be changed). The TO Library must be editable. For Move processing, the FROM Library must be editable as well. To select from a list of library names in your path, enter a question mark (?) next to either Library field.

For information about the fields on the panel, press F1 (Help).

Panel Move/Copy List

The MODS : Panel Move/Copy List is displayed when you select Move or Copy on the MODS : Panel Move/Copy Menu and leave the Panel Name blank or specify a generic panel name.

The top left of the display shows the name of the library being listed (the FROM Library); the top right of the display shows the FROM Library and the TO Library. The list shows panels in the FROM Library, based on the panel name you specified.

Search Panels for a Character String

You can search all or specific panels defined in a particular library for a string of characters.

To search panels for a character string

1. On the MODS : Panel Maintenance Menu, specify the name of the library that contains the panels to be searched in the Library Name field and select Option S.

   The MODS : Search Panel Definitions panel appears, letting you enter the character string and optionally a panel name prefix.

```
PROD---------------- MODS : Search Panel Definitions -----------------------
Command ===>

Search String ..... _____________________________________________________
Panel Name Prefix ..... __________
```

   Note: For information about the fields, press F1 (Help).

2. After you fill in the fields, press F6 (Action).

   The PSM : Confirm Printer panel appears.

3. If necessary, change the values in the fields, then press F6 (Confirm) to start the search.

   If the number of panels to be searched is 100 or more, a panel appears on your screen to advise you of the progress of the search. When the search is complete, a message appears on your screen to advise you of the success or failure of the search. The search results in a PSM report that lists the panels containing the specified string of characters. If the report is on hold, you can use the PQ[UEUE] command to access the PSM output queue and view the report.

Designing Panels

This section describes what to put in a panel definition: how to define constant data, and input and output fields. The panel control statements can be used to embed comments in panel definitions, to define fields, and to interact with NCL processes.

Note: For information about how NCL processes use panel definitions, see the *Network Control Language Programming Guide*.

Panel Control Statements

Optional control statements can precede a panel to specify the particular requirements for that panel. These are as follows:

#ALIAS

Defines an alternative name for an input variable.

#OPT

Defines optional operational requirements.

#FLD

Defines or modifies a field character's attributes.

#ERR

Defines the action to be taken for an error condition.

#NOTE

Provides installation documentation (this is ignored during processing).

#TRAILER

Provides a means of placing specified panel lines at the end of the panel (regardless of screen size).

Control statements included within panel definitions must precede the displayable portion of the panel (as determined from the first non-control statement encountered). Control statements must start in column 1 of the lines on which they appear.

Before a panel is displayed, its associated control statements are parsed and any variable substitution performed. This lets you tailor control statements dynamically.

Data in Panels

Panels contain a combination of fixed data and variable output data:

- Fixed data is the screen captions, field identification text, and other static screen information defined when the panel is created. This does not change when the panel is displayed.
- Variable output data is data generated by the system while the panel is being displayed. It replaces variables positioned within the panel created by the editor. Data is extracted from NCL variables available at the time the panel is invoked.

Variable output data can be displayed in:

- Protected output-only fields (where the data comprises either system or user variables), or
- Unprotected input fields, for any user variables. Once displayed, you can enter data into the unprotected input fields. Panel Services then inserts this data into the user variable for each field, so it is available for further processing by NCL procedures.

You can use syntax to define variables within a panel (see page 73).

Panel Design

A panel design contains a series of lines, each of which can contain one or more fields.

Each field is preceded by a field character that identifies the attributes for that field. These attributes specify:

- The field type (input, output, selector pen detectable (SPD), or null)
- The intensity (brightness) of the display
- Optional formatting rules
- Optional editing rules
- The color and extended highlighting used when the field is displayed (for appropriate terminals)

Field Characters

Each panel line has one or more fields, starting with a field character that specifies the field attributes.

Within the #FLD control statement (see page 73) at the top of the panel definition, you must specify which characters are required for different types of field.

You can use the following methods to define field characters:

- **Character mode**—To specify character mode, use any special character other than an alpha or numeric character, and excluding ampersand (&), blank, or null.
- **Hexadecimal mode**—To specify hexadecimal mode, enter the hexidecimal value for the character (for example, as X‘FA’). Use any hexidecimal value in the range X‘00’ to X‘FF’, excluding the values X‘00’ (null), X‘40’ (blank), X‘50’ (ampersand—&), X‘0E’, and X‘0F’. Hexadecimal mode is used when you need a very large number of field types within one panel and there are insufficient special keyboard characters available to accommodate all of the field characters you require.

Field Types

Each field is allocated a field type that specifies the method for processing the field. The following field types are supported:

OUTPUT

Display only—no data can be entered from the screen.

INPUT

You can both display and enter data.

SPD

Selector pen detectable—data cannot be typed in.

NULL

Display only—although unprotected, any data entered is ignored.

Any mixture of the above field types can be defined to suit the requirements for a panel you are designing.

The field character that precedes each field determines:

- The field type.
- The display characteristics of the field (such as intensity, color, highlighting, justification, and capitalization).
- For input fields, the internal validation rules that must be obeyed for data entered in that field. Such rules can specify, for example, that a field is mandatory, must be numeric, cannot contain imbedded blanks, or must be a valid date.

Each field character that you define occupies the equivalent screen position when the panel is displayed, but appears as a blank character (the attribute byte).

The field proper starts from the next position after the field character, and continues to the next attribute byte on the same line, or to the end of that line where there is no intervening field. Fields do not wrap round from one line to the next.

Field characters can be specified either in character, in which case they are always special characters (non-alpha, and non-numeric; for example, *, %), or in hexadecimal.

The standard default field characters are as follows:

%

High-intensity, protected (no input)

+

Low-intensity, protected (no input)

–

High-intensity, unprotected (input, no validation)

These standard default field characters do not require definition by a #FLD Statement statement.

Define any additional field characters you need using the #FLD Statement statement. The attributes for the above default field characters can be modified. You can use the #OPT statement (see page 87) to nominate alternative standard field characters, so that %, +, and _ can be used within the panel and not processed as field characters, if required.

Column 1 of each line of a panel must be a valid field character; if one is not defined, then the attributes for the second standard field character (normally +, for low-intensity, protected) are used to replace any data incorrectly placed in that column.

In the figure below, all fields preceded by a percent sign (%) display in high-intensity and are protected from data entry. All fields preceded by a plus sign (+) display in low-intensity and are also protected. The only field available for input is on line 16 of the text data, preceded by an underline (_). The word newpanel identifies the NCL variable that receives the data that the user enters in this field once the Enter key is pressed.

By default, the cursor is placed at the beginning of the *cmd* field, as this is the first field requiring input—no other cursor position has been specified.

Note: The ampersand (&) that precedes a variable is omitted when specifying an input field.

```
LIB: PANLUSR----------------- MODS : Edit Panel ---------------NAME: PANEL001
 Command ===>                                                 Scroll ===> CSR


 LINE <---+----10---+----20---+----30---+----40---+----50---+----60---+----70-->
 **** ****************************** TOP OF DATA ********************************
                                   %MODS : Rename Panel
      +Command ===>_cmd



      %   Current Panel Definition

      +   Path ...................%&path      +

      +   Library ................%&lib  +

      +   Panel Name ..............%&oldpanel +

      %   Enter New Panel Name

      +   New Panel Name ..........._newpanel +
```

When a panel is displayed, field characters are removed and the required terminal attribute characters substituted. The following figure shows how the panel is displayed.

Note: In all figures, the underline symbol (_) designates the cursor location.

```
                              MODS : Rename Panel
Command ===> _


   Current Panel Definition

   Path ...................______________

   Library ................_______

   Panel Name ..............__________

   Enter New Panel Name

   New Panel Name ..........________
```

Allowing Long Field Names in Short Fields

An input field is defined on a panel by inserting an appropriate attribute character followed by the name of the NCL variable that contains the input data. Unfortunately this means that input fields cannot be any shorter than the variable name that contains the input data.

The #ALIAS control statement lets you define an alias name for a variable. The alias can be used where the variable would have been used. A range or list of variables can be defined and referred to by the same alias name in the panel definition.

Output Padding and Justification

Careful use of padding and justification greatly enhances the look and effectiveness of panels for end-users.

Panel Services includes extensive facilities to manipulate displayed data. Padding and justification qualities are specified by the #FLD Statement statement. There are two justification categories—field-level justification, and variable-level justification. Both can be used concurrently.

Field-Level Justification

This is performed on an entire field as delimited by defined field characters. Field justification analyzes the entire field, strips trailing blanks, and pads and justifies the remaining data. The #FLD Statement operands controlling field-level justification are JUST and PAD.

The various ways data can be manipulated are best described by a series of examples. These examples show a mix of fields each defined with a different field character and each showing a different display format. Study the #FLD Statement statements and observe the results achieved.

```
#NOTE   This sample panel definition gives examples of the
#NOTE   use of field level justification and padding.
#FLD   #
#FLD   $   JUST=RIGHT
#FLD   @   JUST=LEFT PAD=<
#FLD   ?   JUST=RIGHT PAD=>
#FLD   /   JUST=CENTER PAD=.
#&VAR01                                 +
@&VAR03                                 +
?&VAR04                                 +
/&VAR05                                 +
```

Assume the following variable assignment statements are executed by the NCL procedure before displaying the sample panel:

```
&VAR01 = &STR Left justified null padding
&VAR02 = &STR Right justified null padding
&VAR03 = &STR Left justified with padding
&VAR04 = &STR Right justified with padding
&VAR05 = &STR Center justified with padding
```

The default values are JUST=LEFT and PAD=NULL, as shown by the first line in the example below, where field character # is used with no attributes other than the defaults. The sample panel is displayed as follows:

```
Left justified null padding
         Right justified null padding
Left justified with padding<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>Right justified with padding
.............Center justified with padding..............
```

Variable-Level Justification

This operates independently of field-level justification, and applies to the data substituted for field variables defined as requiring variable-level justification. Variable-level justification is designed to help tabulated output, where data of differing lengths is substituted for a series of variables and where the normal substitution process disrupts display formats. The #FLD Statement operands that control variable-level justification are VALIGN and PAD.

The substitution process substitutes data in place of the &variable without creating additional characters. Thus, if a variable (for instance, &VARIABLE) is replaced by data (for example, Data), any characters following this are moved left to occupy any spaces remaining after substitution (this occurs if spaces are freed going from a long variable name to a shorter data length).

```
#NOTE   This sample panel definition gives examples of the
#NOTE   use of variable justification, padding, and field
#NOTE   justification.
#FLD # VALIGN=LEFT
#FLD $ VALIGN=RIGHT
#FLD @ VALIGN=CENTER
#FLD ? VALIGN=LEFT PAD=.
#FLD / VALIGN=RIGHT PAD=.
#FLD } VALIGN=CENTER PAD=.
#FLD ! VALIGN=LEFT JUST=RIGHT PAD=.
#&VARIABLE other data               +
$&VARIABLE other data               +
@&VARIABLE other data               +
?&VARIABLE other data               +
/&VARIABLE other data               +
}&VARIABLE other data               +
!&VARIABLE other data               +
```

Variable-level justification, controlled by the VALIGN operand of the #FLD statement, lets you influence the way substitution is performed.

Note: Variable-level justification is only performed if the length of the data being substituted is less than the length of the variable name being replaced, including the ampersand (&).

Assume the following variable assignment statement has been executed by the NCL procedure before displaying the sample panel:

```
&VARIABLE = Data
```

&VARIABLE is the only variable within a field that contains the words ‘other data’. Where both field justification and variable alignment are used, the padding character applies to both, as shown by the last line of the example for the field character !. The sample panel is displayed as follows:

```
Data    other data
Data other data
Data  other data
Data..... other data
.....Data other data
..Data... other data
.............................Data..... other data
```

Input Padding and Justification

Fields to which the PAD and JUST operands of the #FLD statement are applied can be defined as input fields. If an input field is primed with data during the display process, the alignment of data within that field when displayed is similar to output padding and justification, except that JUST=CENTER is treated as JUST=LEFT.

When Panel Services processes input from the screen, input fields defined using the PAD and JUST operands are specially processed using the following rules:

- Trailing blanks and pad characters are stripped off, unless the pad character is numeric.
- If JUST=RIGHT is specified for the field, leading blanks and pads are stripped off (including numeric pads).
- If JUST=ASIS is specified for the field, trailing blanks and pads are stripped off, but leading blanks and pads remain intact.

```
#NOTE   This sample panel definition gives examples of the
#NOTE   use of input padding and justification.
#FLD # TYPE=INPUT
#FLD $ TYPE=INPUT JUST=RIGHT
#FLD @ TYPE=INPUT PAD=< JUST=LEFT
#FLD ? TYPE=INPUT PAD=> JUST=RIGHT
#FLD / TYPE=INPUT PAD=0 JUST=LEFT
#FLD } TYPE=INPUT PAD=1 JUST=RIGHT
#VAR01                                  +
$VAR02                                  +
@VAR03                                  +
?VAR04                                  +
/VAR05                                  +
}VAR06                                  +
```

Assume the following variable assignment statements are executed by the NCL procedure before displaying the sample panel:

```
&VAR01 = Walt
&VAR02 = Tom
&VAR03 = Dick
&VAR04 = Harry
&VAR05 = John
&VAR06 = Vicky
```

The sample panel is displayed as:

```
WALT
                                        TOM
DICK<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>HARRY
JOHN0000000000000000000000000000000000000
1111111111111111111111111111111111VICKY
```

If control is passed back to the NCL procedure without any data entered into the input fields, the variables are set to the following values:

```
&VAR01 = WALT
&VAR02 = TOM
&VAR03 = DICK
&VAR04 = HARRY
&VAR05 = JOHN
&VAR06 = VICKY
```

Note: If the line JOHN000... is modified, it is padded to the right with zeros. The variable values are translated to uppercase because the default for input fields is CAPS=YES.

Displaying Function Key Prompts

The SAA Common User Access (CUA) standards require that a list of function keys and their functions be displayed at the bottom of a panel. These function key prompts are displayed on the last lines of the physical screen.

The #TRAILER control statement can be used by NCL procedures to nominate lines that appear at the bottom of the panel. The function key prompts are then always displayed at the bottom of the panel regardless of the screen size.

Panel Statements

This section describes the panel statements.

#ALIAS Statement

This statement defines an alternative name for input variables and allows the panel definition to contain alternative names for variable names in TYPE=INPUT and TYPE=OUTVAR fields.

This facility is useful if you require short fields with long variable names. Each reference to *name* in the panel definition is regarded as a reference to the next name from the list of VARS specified.

This statement has the following format:

```
#ALIAS name
     { VARS=prefix*[ RANGE=(start,end) ] |
       VARS={ vname | (vname,vname,…,vname) } }
```

name

Specifies the alias name that appears in the panel definition. Whenever this name occurs in a field declared as TYPE=INPUT or TYPE=OUTVAR on the #FLD statement, panel services logically replaces it with the next available name from the VARS list.

The name can be from one to eight characters in length. The first character must be an alphabetic or national character. The remaining characters, if any, must be alphanumeric or national characters.

The same name can be used on multiple #ALIAS statements. The variable names are added to the end of the list of names to which the alias name refers.

VARS=*prefix [RANGE=(*start,end*)] |**
VARS=(*vname,vname, ..., vname*) }

Specifies the list of names that replaces the alias name in the panel definition. Each time the alias name is encountered in the panel definition it is replaced by the next available name from this list. The format of the operands associated with VARS= is:

VARS=*prefix** denotes that the variable names used are *prefix*1, *prefix*2, and so on. The RANGE= operand can be specified to indicate a starting and ending suffix number. The default is RANGE=(1,64). The format *prefix** cannot be used in conjunction with other variable names on the same #ALIAS statement.

VARS=*vname* is the name of a variable excluding the ampersand (&).

Examples:

```
#ALIAS Z VARS=LONGNAME
#ALIAS Z123 VARS=(SATURDAY,SUNDAY)
#ALIAS AVAR VARS=LINE* RANGE=(10,20)
```

Notes:

- Multiple #ALIAS statements can be used for the same alias name if insufficient space is available on a single statement**.**
- If an alias name appears in the panel definition after all the variable names in the alias list have been used up, the alias name itself appears in the panel.
- Symbolic variables can be included in the #ALIAS statement. Variable substitution is performed prior to processing the statement, using variables available to the NCL procedure at the time the &PANEL statement is issued.

#ERR Statement

This statement defines action to be taken during error processing.

The #ERR statement is a panel control statement that determines the processing required when a panel is being redisplayed following an error condition.

An error condition can be detected either by Panel Services internal validation or by the processing NCL procedure. If detected by internal validation (and &CONTROL PANELRC is not in effect), error processing is automatically invoked by Panel Services. If detected by the processing NCL procedure, error processing is invoked in one of two ways:

- Using the &ASSIGN OPT=SETERR verb
- By nominating the name of the variable that identifies the invalid input field on the ERRFLD operand of the #OPT statement. This is dynamically invoked by using a symbolic variable with the ERRFLD operand and setting this variable to the name of the variable (minus the ampersand) that identifies the field in error.

Note: For information about using this technique, see the *Network Control Language Programming* Guide.

When #ERR processing is initiated, the cursor is positioned to the first field in error and the panel is redisplayed, applying the attributes defined on the #ERR statement to the fields in error. This technique provides the panel user with a simple means of drawing the terminal operator's attention to the field in error. This is particularly effective on color terminals where the color of any field in error can be altered for the duration of the error, and reverts to normal when the error condition is rectified.

One or more #ERR statements can be defined in any order. However, as with #OPT, #FLD, and #NOTE statements, any #ERR statement must precede the start of the panel, which is determined by the first line that is not a control statement.

This statement has the following format:

```
#ERR [ INTENS={ HIGH | LOW } ]
   [ { COLOR | COLOUR }={ BLUE | RED | PINK | GREEN |
                          TURQUOISE | YELLOW | WHITE | DEFAULT } ]
   [ { HLIGHT | HLITE }={ USCORE | REVERSE | BLINK | NONE } ]
     [  ALARM={ YES | NO } ]
```

INTENS={ HIGH | LOW }

Determines the intensity of the error field when displayed. The INTENS operand is ignored for terminals with extended color and highlighting when either the COLOR or HLIGHT operands are specified.

HIGH specifies that the field is displayed in double intensity.

LOW specifies that the field is displayed in low or standard intensity.

{ COLOR | COLOUR }={ BLUE | RED | PINK | GREEN | TURQUOISE | YELLOW | WHITE| DEFAULT }

Determines the color of the field. It applies only to IBM terminals with seven-color support and Fujitsu terminals with three- or seven-color support.

The COLOR operand is ignored if the terminal does not support extended color. This enables COLOR to be specified on panels that are displayed on both color and non-color terminals. COLOR can be used in conjunction with the HLIGHT operand.

For Fujitsu terminals that support extended color data streams, but support only three colors, the following color relationships are used:

GREEN

When GREEN is specified, it produces a result of GREEN on a Fujitsu three-color terminal.

RED

When RED is specified, it produces a result of RED on a Fujitsu three-color terminal.

PINK

When PINK is specified, it produces a result of RED on a Fujitsu three-color terminal.

BLUE

When BLUE is specified, it produces a result of GREEN on a Fujitsu three-color terminal.

TURQUOISE

When TURQUOISE is specified, it produces a result of GREEN on a Fujitsu three-color terminal.

YELLOW

When YELLOW is specified, it produces a result of WHITE on a Fujitsu three-color terminal.

WHITE

When WHITE is specified, it produces a result of WHITE on a Fujitsu three-color terminal.

DEFAULT

When DEFAULT is specified, it produces a result of GREEN on a Fujitsu three-color terminal.

Fujitsu seven-color terminals are treated the same as IBM seven-color terminals.

The DEFAULT keyword indicates that the color of the field is determined from the INTENS operand. This is particularly useful if you want to set the color from an NCL procedure (that is, COLOR=&COLOR is specified and the NCL procedure can set the &COLOR variable to DEFAULT).

{ HLIGHT | HLITE } = { USCORE | REVERSE | BLINK | NONE }

Applies only to terminals with extended highlighting support, and determines the highlighting to be used for the field.

Because the HLIGHT operand is ignored if the terminal does not support extended highlighting, HLIGHT can be specified on panels that are displayed on terminals that do not support extended highlighting. HLIGHT can be used with the COLOR operand.

When NONE is specified, the HLIGHT operand is ignored and no extended highlighting is performed for this field.

ALARM={ YES | NO }

Turns the terminal alarm on or off if the panel is displayed with an error condition. This works independently of the ALARM operand on the #OPT control statement.

Examples:

```
#ERR COLOR=RED HLIGHT=REVERSE ALARM=YES
#ERR COLOR=YELLOW HLIGHT=BLINK INTENS=HIGH
```

Notes:

- Only those attributes defined on the #ERR statement are modified for the field in error. All other attributes associated with the original field, such as internal validation, remain intact
- Symbolic variables can be included in a #ERR statement. Variable substitution is performed before processing the statement, by using variables available to the NCL procedure at the time the &PANEL statement is issued.
- When &CONTROL PANELRC is in effect, internal validation does not automatically reshow the panel with the error message, and so on. In this case, the procedure regains control following the &PANEL statement with the &RETCODE system variable set to 8 to indicate that an error has occurred. The &SYSMSG variable contains the text of the error message that describes the error and the &SYSFLD variable contains the name of the field in error. This name is the name of the variable in an input field that receives the data entered into that field.
- The &ASSIGN statement SETERR option provides a mechanism for assigning #ERR field attributes to multiple (input field) variables before displaying a panel. This lets you accept input from a number of different fields on a panel, validate all the fields and then redisplay the panel with all incorrect fields displayed with the #ERR attributes. This shows the user all the errors at one time, rather than field by field.

#FLD Statement

This statement defines or modifies a panel definition field character.

The #FLD statement is a panel control statement used to tailor the operational characteristics of a panel.

When a panel is defined, it is constructed of a number of lines that, in turn, are made up of a number of fields. Each field commences with a field character that appears as a blank on the panel when displayed. Each field character determines the attributes that are to be associated with the field following the field character itself. A field is delimited by the next field character or the end of the panel line. Fields cannot wrap from one line to the next.

The first field on a line always starts in column 1. If no field character is defined in the first position of the line, the attributes of the second of the three standard field characters (usually a plus sign (+), TYPE=OUTPUT, INTENS=LOW) are forced and replace any non-field character incorrectly placed in this position.

Before parsing, the #FLD statement is scanned and variable substitution is performed. This makes it possible to dynamically tailor any of the options or operands on the statement.

As many #FLD statements as required can be specified. They can be defined in any order. However, as with #OPT, #ERR and #NOTE statements, all #FLD statements must precede the start of the panel, which is determined by the first line that is not a control statement.

This statement has the following format:

```
#FLD { c | X'xx' }
     [ BLANKS={ TRAIL | NONE | ANY } ]
     [ CAPS={ YES | NO } ]
   [ { COLOR | COLOUR }={ BLUE | RED | PINK | GREEN |
                  TURQUOISE | YELLOW | WHITE | DEFAULT } ]
     [ CSET={ ALT | DEFAULT } ]
     [ EDIT={ ALPHA | ALPHANUM | DATEn | DSN | HEX |
              NAME | NAME* | NUM | REAL | SIGNNUM | TIMEn } ]
   [ { HLIGHT | HLITE }={ USCORE | REVERSE | BLINK | NONE } ]
     [ INTENS={ HIGH | LOW | NON } ]
     [ JUST={ LEFT | RIGHT | ASIS | CENTER | CENTRE } ]
     [ MODE={ SBCS | MIXED } ]
     [ NCLKEYWD={ YES | NO } ]
     [ OUTLINE={  {L R T B} | BOX } ]
     [ PAD={ NULL | BLANK | char } ]
     [ PSKIP={ NO | PMENU } ]
     [ RANGE=(min,max) ]
     [ REQUIRED={ YES | NO } ]
     [ SKIP={ YES | NO } ]
     [ SUB={ YES | NO } ]
     [ TYPE={ OUTPUT | INPUT | OUTVAR | SPD | NULL } ]
     [ VALIGN={ NO | LEFT | RIGHT | CENTER | CENTRE } ]
```

***c* | X'*xx*'**

The field character:

c is the character that is used in the panel definition to identify the start of the field. This is known as a field character. This must be a single non-alpha and non-numeric character. Any special character (for example, an exclamation mark) can be used, with the exception of an ampersand (&), which is reserved for use with variables.

X'*xx*' is the hexadecimal value of the field character. Use this notation to specify any value in the range X'01' to X'3F'. Do not use values that correspond to alphanumeric characters, or X'0E' (shift in) or X'0F'(shift out).

Although the panel editor does not let you enter non-displayable hexadecimal attributes (X'01' to X'3F') in the body of the panel, you can use the preparse option to prime the field character value in the panel before issuing the &PANEL statement.

The first #FLD statement to reference a particular field character defines a new character. Subsequent statements referencing that same field character modify or extend the attributes of the field character. Three standard field characters (%, +, _ unless altered by the #OPT statement) are provided. If a default field character (usually % + or _) is referenced, it is the same as extending or modifying the attributes of an existing field character.

If no additional operands are defined following a new field character, the following defaults apply:

```
TYPE=OUTPUT INTENS=LOW
```

No special attributes or internal validation apply.

```
BLANKS={  TRAIL | NONE | ANY  }
```

This determines the format for entering data in input fields. By default, a field can contain imbedded blanks (BLANKS=ANY). Specification of this operand ensures that the entered data does not contain imbedded blanks and contains only trailing blanks (TRAIL) or no blanks at all (NONE). This operand works independently of the REQUIRED operand. For optional fields this operand can still be specified to ensure that when data is entered, it is in the correct format. If &CONTROL PANELRC is not in effect, BLANKS=TRAIL is specified, and the data is in error, Panel Services redisplays the panel with the &SYSMSG variable set to:

```
INVALID IMBEDDED BLANKS
```

If BLANKS=NONE is specified and the data is in error, Panel Services redisplays the panel with the &SYSMSG variable set to:

```
INCOMPLETE FIELD
```

If &CONTROL PANELRC is in effect, control is returned to the NCL procedure for error handling instead of being handled totally by Panel Services. In this case, &SYSFLD contains the name of the field in error and &SYSMSG the error message text.

CAPS={ YES | NO }

Applies to input fields only and determines if entered data is converted to upper case before passing it to the NCL procedure in the nominated variable. Conversion to upper case is also performed for the data associated with an input variable before displaying the panel. This does not impact the current contents of the variable unless the data is modified and entered by the operator. Output fields are displayed exactly as defined and are not subject to upper case conversion.

Note: The effect of CAPS=NO can be negated if the variable that receives the data is used in an assignment statement (for example, &A = &DATA) within the processing NCL procedure, as data can be converted to upper case before performing the assignment; see the &CONTROL UCASE option. The CAPS operand is ignored when operating in a system executing with SYSPARMS DBCS=YES.

{ COLOR | COLOUR } = { BLUE | RED | PINK | GREEN | TURQUOISE | YELLOW | WHITE | DEFAULT }

Applies only to IBM terminals with seven-color support and Fujitsu terminals with three- or seven-color support, and determines the color of the field.

Because the COLOR operand is ignored if the terminal does not support extended color, COLOR can be specified on panels that are displayed on both color and non-color terminals. COLOR can be used in conjunction with the HLIGHT operand.

For Fujitsu terminals that support extended color data streams where only three colors are available, the following color relationships are used:

GREEN

When GREEN is specified, it produces a result of GREEN on a Fujitsu three-color terminal.

RED

When RED is specified, it produces a result of RED on a Fujitsu three-color terminal.

PINK

When PINK is specified, it produces a result of RED on a Fujitsu three-color terminal.

BLUE

When BLUE is specified, it produces a result of GREEN on a Fujitsu three-color terminal.

TURQUOISE

When TURQUOISE is specified, it produces a result of GREEN on a Fujitsu three-color terminal.

YELLOW

When YELLOW is specified, it produces a result of WHITE on a Fujitsu three-color terminal.

WHITE

When WHITE is specified, it produces a result of WHITE on a Fujitsu three-color terminal.

DEFAULT

When DEFAULT is specified, it produces a result of GREEN on a Fujitsu three-color terminal.

Fujitsu seven-color terminals are treated the same as IBM seven-color terminals.

The DEFAULT keyword indicates that the color of the field is determined from the INTENS operand. This is particularly useful if you want to set the color from an NCL procedure (that is, COLOR=&COLOR is specified and the NCL procedure can set the &COLOR variable to DEFAULT).

CSET={ ALT | DEFAULT }

Applies to output fields only. The operand determines which terminal character set to use to display the field. If you specify CSET=ALT (or ALTERNATE), you can draw box shapes using the following characters:

e	is displayed as	│
s	is displayed as	─
D	is displayed as	└
E	is displayed as	┌
M	is displayed as	┘
N	is displayed as	┐
F	is displayed as	├
G	is displayed as	┴
O	is displayed as	┤
P	is displayed as	┬
L	is displayed as	┼

Note: CSET=ALT supersedes CSET=ASM in Version 3.1.

EDIT={ ALPHA | ALPHANUM | DATE*n* | DSN | HEX | NAME | NAME* | NUM | REAL | SIGNNUM | TIME*n* }

For input fields this determines additional internal editing to be performed by Panel Services. By default no editing is performed. Specification of this operand ensures that the entered data conforms to the nominated type. If a field is mandatory, then REQ=YES should also be specified.

ALPHA

Only accept A to Z.

ALPHANUM

Only accept A to Z, 0 to 9, #, @ and $.

DATE*n*

Field must be a valid date. The DATE*n* keyword must correspond to one of the &DATEn system variables, and indicates that the input field must contain date in the format associated with that system variable. For example, EDIT=DATE5 indicates that the input field must always contain a date in the format corresponding to the &DATE5 system variable (MM/DD/YY).

DSN

Field must be a valid OS/VS format data set name. If required, a partitioned data set (PDS) member name or Generation Data Group (GDG) number can be specified in brackets as part of the name.

HEX

Only accept 0 to 9 and A to F.

NAME

Field must commence with alpha (A to Z, #, @ or $) and be followed by alphanumerics (A to Z, 0 to 9, #, @ or $).

NAME*

Field must commence with alpha (A to Z, #, @ or $) and be followed by alphanumerics (A to Z, 0 to 9, #, @ or $) but can be terminated with a single asterisk (*). This allows a value to be entered that can be interpreted as a generic request by the receiving procedure.

NUM

Only accept 0 to 9. Specifying EDIT=NUM, in addition to internal Edit validity checking, sets a hardware flag to inhibit alpha input. This flag is display system dependent for its implementation; either in hardware or emulation software.

REAL

Input in this field must conform to the syntax for integers, signed numbers or real numbers, including scientific notation. For information about real number support, see the *Network Control Language Programming Guide*.

SIGNNUM

Field must be numeric but can have a leading sign (+ or -).

TIME*n*

Field must be a valid time. The TIME*n* keyword must correspond to one of the &ZTIME*n* system variables and indicates that the input field must be in the format associated with that system variable.

When invalid data is detected and &CONTROL PANELRC is not in effect, standard error processing is invoked by Panel Services and control is not returned to the NCL procedure until the error is corrected.

For EDIT=NUM the panel is redisplayed with the &SYSMSG variable set to:

```
FIELD NOT NUMERIC
```

For EDIT=REAL the panel is redisplayed with the message:

```
FIELD NOT REAL NUMBER
```

For EDIT=ALPHA, ALPHANUM, HEX or NAME the panel is redisplayed with the &SYSMSG variable set to:

```
INVALID VALUE
```

For EDIT=DATE*n* the panel is redisplayed with the &SYSMSG variable set to:

```
INVALID DATE
```

For EDIT=DSN the panel is redisplayed with the &SYSMSG variable set to:

```
INVALID DATASET NAME  or INVALID MEMBER NAME
```

For EDIT=TIME*n* the panel is redisplayed with the &SYSMSG variable set to:

```
INVALID TIME
```

In all cases the terminal alarm sounds and the cursor is positioned to the field in error. If a #ERR statement has been included in the panel definition, processing of the error condition is performed as defined in that statement.

If &CONTROL PANELRC is in effect, control is returned to the NCL procedure for error handling instead of being handled totally by Panel Services. In this case, &SYSFLD contains the name of the field in error and &SYSMSG the error message text.

Note: Use of the EDIT operand might also require the use of the BLANKS operand to ensure that entered data does not include imbedded blanks. Regardless, editing is performed only for the length of the data entered and not for the length of the input field. If the entire field must be entered, the BLANKS=NONE operand should be specified.

{ HLIGHT | HLITE } = { USCORE | REVERSE | BLINK | NONE }

Applies only to terminals with extended highlighting support, and determines the highlighting to be used for the field.

Because the HLIGHT operand is ignored if the terminal does not support extended highlighting, HLIGHT can be specified on panels that are displayed on terminals that do not support extended highlighting. HLIGHT can be used with the COLOR operand.

The NONE keyword is provided as a no-impact value that can be used when the highlighting of a field is being dynamically determined from the NCL procedure and set using variable substitution of the #FLD statement. When NONE is specified, the HLIGHT operand is ignored.

INTENS={ HIGH | LOW | NON }

Determines the intensity of the field when displayed.

HIGH

The field is displayed in double intensity. High intensity is normally associated with input fields and other important data and its use minimized to maintain its effectiveness.

LOW

The field is displayed in low or standard intensity.

NON

The field is displayed in zero intensity. Any data within the field is not visible to the operator. This is normally used for input fields where sensitive data such as passwords are entered. Use of this attribute for output fields is meaningless. Color or extended high-lighting attributes are ignored when used in conjunction with this attribute.

JUST={ LEFT | RIGHT | ASIS | CENTER | CENTRE }

For output fields, this determines the alignment of the data within the field after trailing blanks have been stripped. Justification is applied at a field level and should not be confused with VALIGN, which applies to the individual variable only:

- LEFT results in padding to the right
- RIGHT results in padding to the left
- ASIS is treated as JUST=LEFT for output fields
- CENTER results in padding to both the left and the right.
- For input fields justification occurs both when the data is being displayed and when the data is being processed on subsequent entry. When an input field is formatted for display (the value currently assigned to the variable defined in the input field is substituted in place of the variable's name) the data is justified to the left and padded to the right for JUST=LEFT or justified to the right and padded to the left for JUST=RIGHT. JUST=CENTER is treated like JUST=LEFT. For JUST=ASIS data is positioned exactly as defined in the variable and padding to the right is performed.
- On subsequent re-entry, trailing blanks and pad characters are stripped, unless the trailing pad character is a numeric, in which case it is not stripped:
- For JUST=RIGHT leading blanks and pads are also stripped (including numerics). Use of JUST=RIGHT for input fields can inconvenience terminal operators, as it is necessary to move the cursor to the commencement of the data in the field.
- For JUST=ASIS trailing blanks and pads are stripped, but leading blanks and pads remain intact.

MODE={ SBCS | MIXED }

Applies to IBM terminals capable of supporting DBCS data streams. If a panel is sent to such a device, input fields on the panel that use this #FLD character let the operator enter DBCS characters if MODE=MIXED is specified. IBM DBCS terminals do not allow DBCS character entry in input fields that specify MODE=SBCS (single byte character stream).

Note: This operand does not apply to Fujitsu or Hitachi terminals, which allow DBCS character entry at any time.

NCLKEYWD={ YES | NO }

Specifies whether fields that use this FLD character accept input of words that conflict with NCL keywords. The default is YES. NO causes an attempt to enter any NCL reserved keyword to be rejected.

OUTLINE={ { L R T B } | BOX }

Specifies the extended highlighting outlining option required for this field. Any combination of L (left) R (right) T (top) or B (bottom) can be coded. The field is outlined at the top or bottom with a horizontal line and at the left and right border with a vertical line according to the options specified. Alternatively the BOX option can be specified, which is equivalent to specifying LRTB. This option is terminal dependent.

PAD={ NULL | BLANK | *char* }

Applies to INPUT, OUTPUT, and SPD fields.

For output fields PAD works in conjunction with both the JUST and VALIGN operands, one of which must be specified for PAD to take effect. Determines the pad or fill character to be used when the field is displayed.

The variable substitution process substitutes the data currently assigned to any variables within the field being processed. Having completed substitution, any difference between the length of the field defined on the panel and the length after substitution (after stripping trailing blanks) is padded with the specified PAD character.

For input fields, use of the NULL character ensures that the terminal operator can use keyboard insert mode when entering data. Padding is performed either to the left or the right as specified in the JUST operand.

char

Specifies a single character that is to be the pad character (for example, PAD=-). There is no restriction on the character used including the use of any of the field characters defined on #FLD statements. Care should be taken when using numeric pad characters as their use impacts the pad character stripping process on subsequent entry.

Note: Using PAD characters with input fields invokes special processing on subsequent input to ensure that unnecessary pad characters are stripped before returning the entered data in the nominated variable.

PSKIP={ NO | PMENU }

Applies to input fields only and determines if panel skip requests are accepted in this field. A panel skip request is entered in an input field in the format =*m.m*, where *m.m* is a menu selection. When entered in an appropriate field a panel skip to the specified menu selection is performed.

NO

The input field is not scanned for panel skip requests.

PMENU

The input field is scanned for panel skip requests and actioned.

RANGE=(*min,max*)

For numeric fields, specifies the range of acceptable values. The range includes all numbers, from the minimum number (*min*) to the maximum number (*max*). Both *min* and *max* must be specified and *max* must be equal to or greater than *min*. Use of this operand forces EDIT=NUM. If the entered number falls outside the acceptable range, and &CONTROL PANELRC is not in effect, Panel Services redisplays the panel with the &SYSMSG variable set to:

```
NOT WITHIN RANGE
```

If &CONTROL PANELRC is in effect, control is returned to the NCL procedure for error handling instead of being handled totally by Panel Services. In this case, &SYSFLD contains the name of the field in error and &SYSMSG the error message text.

REQUIRED={ YES | NO }

Specifies that this is a mandatory field that must be entered by the user. If &CONTROL PANELRC is not in effect, Panel services rejects any entry by the user unless this field has been entered. If not entered, Panel Services redisplays the panel with the &SYSMSG variable set to:

```
REQUIRED FIELD OMITTED
```

The terminal alarm sounds and the cursor is positioned to the omitted field. If a #ERR statement has been included in the panel definition, processing of the error condition is performed as defined by the #ERR statement. Failure to include the &SYSMSG variable on the panel suppresses this error message. This operand can be abbreviated to REQ=.

If &CONTROL PANELRC is in effect, control is returned to the NCL procedure for error handling instead of being handled totally by Panel Services. In this case, &SYSFLD contains the name of the field in error and &SYSMSG the error message text.

SKIP={ YES | NO }

For output fields only, determines if the skip attribute is to be assigned to the field. The result of using this option is that the cursor skips to the next input field if the preceding input field is entered in full and the intervening output field is specified with the SKIP operand.

Note: This operand is NO by default, as field skipping can unexpectedly place the cursor in the wrong screen window when operating in split screen mode.

SUB={ YES | NO }

For output fields only, determines if variable substitution is to be performed. This operand can be used for fields where data contains the & character. This results in the current value of that variable being substituted, or the variable being eliminated if no value was assigned. This operand is ignored for both INPUT and SPD fields.

TYPE={ OUTPUT | INPUT | OUTVAR | SPD | NULL }

Determines if the field is to be processed as an output-only field (OUTPUT and OUTVAR), input field (INPUT), Selector Pen Detectable (SPD) field or pseudo input field (NULL).

OUTPUT

A protected field is created that does not allow keyboard entry. This field can contain a mixture of fixed data and variables. Each variable must commence with an ampersand (&). Substitution of variables is performed using the variables available to the invoking NCL procedure at the time the &PANEL statement is issued. Global variables can be referenced in an output field. Alignment and padding is performed according to the rules defined for the field.

INPUT

An unprotected field is created that allows keyboard entry. This field must contain a single variable name (without the ampersand). This single variable must immediately follow the field character. System variables and global variables cannot be used in an input field. Subsequent data entered into this field is made available to the invoking NCL procedure in this variable on return from the &PANEL statement. Specification of multiple variables or a mixture of variables and fixed data in an input field results in an error.

OUTVAR

The same as TYPE=OUTPUT except that an & is inserted by Panel Services between the field attribute and the next character. This means that you can follow a TYPE=OUTVAR field character with a variable name without the ampersand. This facility makes it easy to create fields that switch between input and output under NCL control. For example, a panel could contain the statements:

```
#FLD $ TYPE=&INOUT
+ Record Key .....$RKEY +
```

An NCL procedure then sets the variable &INOUT to control if the data in the variable &RKEY is output only, or if it can be modified by an operator:

```
&INOUT = OUTVAR   -* the value is output only.
&INOUT = INPUT    -* the Operator can modify the
                   -* field.
```

Note: A similar effect can be achieved using &ASSIGN OPT=SETOUT.

SPD

A protected field is created in selector pen detectable format. This enables the terminal operator to select the field using either a LIGHT PEN or the CURSOR SELECT key. SPD field characters must be immediately followed by one of the three designator characters (?, &, or a blank), which can in turn optionally be followed by one or more blanks. A single variable with no other fixed data must also be defined within the field. This single variable must be defined omitting the ampersand (&) and cannot be a system or global variable. If selected by the user, the variable nominated in the SPD field is set to the value SELECTED on return to the NCL procedure. If not selected, the variable is set to a null value.

NULL

An unprotected field is created that allows keyboard entry. However, the field need not contain the name of an input variable to receive data entered in the field as any data entered by the terminal operator in a **TYPE=NULL** field is ignored. Display data in this field can be in any format. The NULL option is supplied to accommodate 4 color terminals where the field attribute byte is used to determine the color in which the field is displayed (7 color terminals utilize an extended data stream to set the color). The NULL option indicates that Panel Services is to use an unprotected field attribute in conjunction with the INTENS operand value, to determine the color of the field.

VALIGN={ NO | LEFT | RIGHT | CENTER | CENTRE }

Applies to output fields only and is ignored if specified for an input field. Determines the alignment of data for an individual variable only. This should not be confused with the JUST operand, which applies to field alignment after all variable substitution has been completed. The VALIGN operand is designed to facilitate tabular output without the need to specify many individual field characters on the panel.

The substitution process substitutes the data assigned to a variable in place of the variable name. No additional blanks are created or removed during this process. Thus, if the data being substituted is shorter than the name of the variable itself (for example, the variable &OUTPUTDATA is currently set to 5678) then data following the variable name is shifted left to occupy the area remaining after the removal of the variable name. This destroys any tabular alignment where the length of the data for each variable differs. The VALIGN option ensures that the data to the right of the variable is not shifted to the left if the data being substituted is shorter than the variable name. The length of the variable name (including the ampersand) is the important factor and determines the number of character positions to be preserved during the substitution process.

However, data truncation is not performed and if the data being substituted is longer than the variable name, the data to the right is moved to accommodate all the substituted data. VALIGN works in conjunction with the pad character specified on the PAD operand. The pad character is used to fill any differences between the data being substituted and the length of the variable name being replaced.

VALIGN=NO

No alignment or padding is performed.

VALIGN=LEFT

Data is aligned to the left and padded to the right. An abbreviation of L is acceptable.

VALIGN=RIGHT

Data is aligned to the right and padded to the left. An abbreviation of R is acceptable.

VALIGN=CENTER

Data is centered (or one position to the left for an odd number of characters) and padded both to the left and to the right. An abbreviation of C is acceptable.

Examples:

```
#FLD   #   TYPE=INPUT REQ=YES EDIT=NUM COLOR=RED RANGE=(1,3)
#FLD   #   BLANKS=TRAIL PAD=_
#FLD   #   TYPE=OUTPUT COLOR=&COLOR HLIGHT=&HLIGHT
#FLD   (   TYPE=INPUT INTENS=HIGH EDIT=DATE4
#FLD   @   HLIGHT=BLINK
#FLD   /   TYPE=SPD
#FLD   %   JUST=R PAD=- -* supplementing default output char
#FLD   _   JUST=ASIS    -* supplementing default input char
#FLD + VALIGN=RIGHT JUST=CENTER
         -* null pad assumed
```

Notes:

- Multiple #FLD statements can be used for the same field character if insufficient space is available on a single statement.
- Symbolic variables can be included in a #FLD statement. Variable substitution is performed prior to processing the statement, using variables available to the NCL procedure at the time the &PANEL statement is issued.
- The default field characters can be altered using the DEFAULT operand of the #OPT control statement.
- The #ERR control statement can be used to greatly simplify the redisplay of a panel to indicate a field in error.
- The &CONTROL PANELRC operand can be used to specify that the NCL procedure receives control for further processing when internal validation detects an error in data entered by the operator. When this technique is used the procedure can determine the field in error (from the &SYSFLD variable) and the error message to be issued (from the &SYSMSG variable) and alter its processing accordingly, including altering the text of the error message in the &SYSMSG variable if required.

#NOTE Statement

The #NOTE statement provides a means of placing documentation within a panel definition. The #NOTE statement is not processed and is ignored. Multiple #NOTE statements can be specified. However, as with #FLD, #ERR, and #OPT statements, all #NOTE statements must precede the start of the panel. The start of the panel is determined by the first line that is not a control statement or #NOTE statement.

This statement has the following format:

`#NOTE` *`any text`*

any text

Any free form user text.

Examples:

```
#NOTE This panel is used by the Network Error Log System
#NOTE INWAIT=60 CURSOR=&CURSORFLD
```

Note: As shown in the example above the #NOTE statement can provide a simple means of temporarily nullifying another control statement allowing for easy reinstatement when required.

#OPT Statement

This statement defines panel processing options.

Before parsing, the #OPT statement is scanned and any variables are substituted. This makes it possible to dynamically tailor any of the operands on the statement.

Multiple #OPT statements can be specified. However, as with #FLD, #ERR and #NOTE statements, all #OPT statements must precede the start of the panel, which is determined by the first line that is not a control statement.

This statement has the following format:

```
#OPT [ ALARM={ YES | NO } ]
     [ BCAST={ YES | NO } ]
     [ CURSOR={ varname | row,column } ]
     [ DEFAULT={ hlu | X'xxxxxx' } ]
     [ ERRFLD=varname ]
     [ FMTINPUT={ YES | NO } ]
     [ IPANULL={ YES | NO } ]
     [ INWAIT=ss.th ]
     [ PREPARSE={ (c,S) | (c,D) } ]
     [ UNLOCK={ YES | NO } ]
     [ MAXWIDTH={ YES | NO } ]
```

ALARM={ YES | NO }

Determines whether the terminal alarm is to be rung when the panel is displayed. Dynamic control of the alarm can be achieved by changing the value of the ALARM operand using a variable set prior to issuing the &PANEL statement.

If internal validation has detected an error, and the panel is being redisplayed to indicate the error, this operand is ignored and the terminal alarm rung. The #ERR statement can be used to alter the processing performed when an error condition is detected.

BCAST={ YES | NO }

Specifies that the panel is to be redisplayed automatically if a broadcast is scheduled. By default, the only panels that are redisplayed automatically are those that contain one or more of the special broadcast variables, &BROLINE*n*. If BCAST=YES is coded, a broadcast causes the panel to be redisplayed even if it does not contain any of the &BROLINE*n* variables.

CURSOR={ *varname* | *row,column* }

Specifies the name of a variable in either an INPUT or SPD field where the cursor is to be positioned. Alternatively, the precise co-ordinates for the cursor can be defined as *row,column*.

The value of *varname* should be the variable name WITHOUT the ampersand, just as used in the INPUT or SPD field (for example, CURSOR=FIELD5).

Where co-ordinates are specified, *row* must be specified in the range 1 to 62 and column in the range 1 to 80. The row and column values are always relative to the start of the current window and therefore remain unchanged when operating in split screen mode. The &CURSROW and &CURSCOL system variables can be used to determine the location of the cursor on input to the system.

Dynamic positioning of the cursor can be achieved by using a variable or variables (including the ampersand) in place of *varname* or *row*,*column*. The invoking NCL procedure can set the variables to the name of the field to contain the cursor or the coordinates prior to issuing the &PANEL statement.

If internal validation detects an error, and the panel is being redisplayed to indicate the error, the CURSOR operand is ignored and the cursor is positioned to the field in error.

Specifying varname with a name other than the name of a variable used in an INPUT or SPD field results in an error. If coordinates are used, and they lie outside the dimensions of the window currently displayed, the cursor is positioned in the upper left corner of the window.

DEFAULT={ *hlu* | X'*xxxxxx*' }

This operand alters the three standard default field characters. If the #OPT statement is omitted, or the DEFAULT operand not used, three standard field characters are provided for use when defining the panel. They are as follows:

%

protected, high-intensity

+

protected, low-intensity

–

unprotected, high-intensity

It might be necessary to select alternative field characters, for example if the underline character is required within the body of the panel for some reason.

The DEFAULT=*hlu* operand must always specify three characters. The characters chosen must be non-alpha and non-numeric, that is, any special character except ampersand (&), which is reserved for variables. They must not duplicate another field character, except one already defined as a default. The order of the characters is significant, as the attributes of the standard default characters apply in the order described above.

Therefore specification of DEFAULT=*+/ results in:

*****

protected, high-intensity

+

protected, low-intensity

/

unprotected, high-intensity

You can also specify the default field characters in hexadecimal in the following format:

DEFAULT=X'*xxxxxx*'

Each *xx* pair represents a hexadecimal number in the range X‘00’ to X‘FF’. All numbers except X‘00’ (null), and X‘40’ (blank) or X‘50’ (ampersand), are valid. This even allows alphanumeric characters to be used as field characters. For example, if you specify X‘C1’, any occurrence of the letter A in the panel definition is treated as the default character. (It is advisable to use hexadecimal values that do not correspond to alphanumeric characters.)

For example, specification of DEFAULT=‘010203’ results in the following:

X‘01’

protected, high-intensity

X‘02’

protected, low-intensity

X‘03’

unprotected, high-intensity

ERRFLD=*varname*

Specifies the name of a variable in an INPUT field that is in error and for which error processing is to be invoked as defined on a #ERR statement. Use of this operand without including a #ERR statement within the panel definition results in an error. The ERRFLD operand provides a simple way of informing Panel Services that the field identified by varname is in error. Panel Services displays the panel using the options defined on the #ERR statement. The #ERR statement could indicate that the error field is to be displayed in reverse-video, colored red and the terminal alarm sounded. Use of the ERRFLD operand can be accompanied by the assignment of some error text into a variable appearing on the screen that identifies the nature of the error.

The operand can be specified with the name of a variable (including the &) that is set to null unless an error occurs, in which case the NCL procedure sets the variable to the name of the field in error prior to issuing the &PANEL statement to display the panel.

ERRFLD provides the panel designer with a simple means of changing the attributes of a field (such as color and high-lighting) without the need to resort to dynamic substitution of #FLD statements.

Consider the case where an input field &INPUT1 is found to be in error and the #OPT statement has been defined with ERRFLD=&INERROR. The NCL procedure assigns the name of the variable used to identify the input field, in this case INPUT1 (minus the &), into &INERROR, and then redisplays the panel.

```
&INERROR = INPUT1
&SYSMSG  = &STR THIS FIELD IS WRONG
&PANEL  MYPANEL
```

Note: In this example, the text that identifies the nature of the error has been assigned into the variable &SYSMSG, which is defined somewhere on the panel.

The same effect can be achieved by using the &ASSIGN OPT=SETERR verb. This allows more than one field to be marked as in error.

Note: ASSIGN OPT=SETERR is only effective if &CONTROL FLDCTL is in effect.

FMTINPUT={ YES | NO }

Determines if input fields are to be formatted when a panel is displayed. This is a specialized option that is designed to be used in conjunction with INWAIT. When processing with INWAIT, the time interval might expire at the instant when data is being entered by the operator. If the same panel is redisplayed to update the screen contents, the data entered by the operator is lost as the new panel is written. FMTINPUT can be used to bypass formatting of input fields and hence when the panel is redisplayed only output fields are written. The value of this operand can be assigned to a variable from within the NCL procedure and changed between YES and NO as required (#OPT FMTINPUT =&YESNO). Care must taken when using this facility, as incorrect use of FMTINPUT=NO can result in validation errors. Ideally, a panel should be displayed initially with FMTINPUT=YES and only when the INWAIT timer expires is it redisplayed with FMTINPUT=NO.

IPANULL={ YES | NO)

The default YES, specifies that if the panel is displayed with the INWAIT option and the time specified on the INWAIT expires so that control is returned to the procedure without any panel input, or input is caused by a PA key, all variables associated with the panel input fields are to be set to null value.

If you do not want input field variables to be erased, if INWAIT completes, or a PA key is pressed, specify IPANULL=NO.

INWAIT=*ss.th*

Specifies the time in seconds and/or parts of seconds that Panel Services is to wait for input from the terminal prior to returning control to the NCL procedure following the &PANEL statement. By default the system waits indefinitely for input having displayed a panel. This might not always be desirable, as is the case where a terminal is performing a monitoring function where input might be infrequent or never occur. If INWAIT is utilized and the specified time elapses, control is returned to the NCL procedure with all input or SPD variables set to null. Should input be made during the time interval, the time period is canceled and standard processing will proceed.

The maximum value that can be specified for INWAIT is 86400.00 seconds (24 hours).

Specification of part seconds is possible. For example:

```
INWAIT=.5
INWAIT=20.5
INWAIT=.75
```

Any redisplay of a panel, for example, by using the clear key, causes the panel to be redisplayed and the time interval reset.

Specification of internal validation options, such as REQUIRED=YES are ignored if the time interval expires before input is received.

Specification of INWAIT=0 or INWAIT=0.00 indicates that no input is to be accepted and control is returned to the NCL procedure immediately after the panel has been displayed. In this case the period that the panel remains displayed is determined by subsequent action taken by the procedure.

The invoking NCL procedure can determine if the INWAIT time elapsed or if data was entered by testing the &INKEY system variable. &INKEY is set to the character value of the key pressed by the operator to enter the data (for example, Enter or F1). If the INWAIT time interval elapsed, and no entry was made, the &INKEY variable is set to null. If processing with &CONTROL PANELRC is in effect, &RETCODE is set to 12 to indicate that the INWAIT timer has expired.

Note: INWAIT is ignored for asynchronous panels.

PREPARSE={ (c,S) | (c,D) }

Preparsing provides a means for dynamically modifying the location of field characters in a panel. The position of field characters (as defined by the #FLD control statement) is determined when the panel is created by Panel Services and remains fixed until the panel is modified.

Although the attributes of each field character (such as the color of the field) can be modified by the use of variables in the #FLD statement, this technique is limited in the number of variations that can be achieved.

The PREPARSE operand requests that Panel Services perform a preliminary substitution scan of each panel line prior to processing the line for field characters. The PREPARSE operand specifies a substitution character *c* that is to be used to determine where substitution is to take place. This character is processed in exactly the same manner as an ampersand (&) is processed during standard substitution.

The ability to specify a character other than an ampersand means that preparsing does not impact standard substitution when it is performed following preparsing. Preparsing can be used to alter a field character that appears in a particular position, thereby allocating a new set of attributes to the field or to create entire new fields (or complete lines) that in themselves contain the required field characters.

(*c*,S)

Indicates that the character c is to be used as the preparse character for the panel, but that the Static Preparse Option is to apply during preparse processing. This prevents the movement of preparse or field characters during the substitution process. This option is useful when panels are being dynamically modified to hold data that can vary in length but is to be displayed in columns. If necessary, substituted data can be truncated if it is too long to fit into its target field without overwriting the next occurrence of a preparse or field character on the same line.

(c,D)

Indicates that the character c is to be used as the preparse character for the panel, but that the Dynamic Preparse Option is to apply during preparse processing. The dynamic option allows the movement of preparse or field characters to left or right of their original position to accommodate differing lengths of data being substituted into the panel.

UNLOCK={ YES | NO }

Determines if the terminal keyboard is to be unlocked when the panel is displayed. Specification of UNLOCK=NO prevents entry of data by the terminal operator, and can be used in conjunction with the INWAIT operand where a panel was being displayed for a short period, prior to progressing to some other function.

MAXWIDTH={ YES | NO }

Specify MAXWIDTH=YES to indicate the display is to be wider than the standard 80 character width. MAXWIDTH=YES means that panel services will use the number of columns available on the terminal (&ZCOLS).

Examples:

```
#OPT DEFAULT=#$%
#OPT INWAIT=60 CURSOR=&CURSORFLD
#OPT CURSOR=IN1 ALARM=YES
#OPT ALARM=&ALARM PREPARSE=($,D)
#OPT ERRFLD=&INERROR
#OPT INWAIT=.5 UNLOCK=NO PREPARSE=($,S)
#OPT CURSOR=5,75
#OPT CURSOR=&ROW,&COLUMN FMTINPUT=&FMT
```

Notes:

- Multiple #OPT statements can be used if required.
- Symbolic variables can be included in a #OPT statement. Variable substitution is performed prior to processing the statement.
- Panel redisplay following use of the CLEAR key is automatic. Control is not returned to the invoking NCL procedure. The attributes of the standard default characters can be modified using a #FLD statement that adds additional attributes (such as color) or alters existing attributes.

#TRAILER Statement

This statement provides a means of placing specified lines at the end of the screen, regardless of screen size.

The #TRAILER statement can be used to position function key prompts at the bottom of the screen.

Indicate the start of the trailer lines with a #TRAILER START statement. Then enter the lines to appear at the bottom of the screen, followed by a #TRAILER END statement.

This statement has the following format:

```
#TRAILER [ START | END ]
         [ POSITION={ YES | NO } ]
```

START

Indicates the start of the lines to be placed in the trailer. Each line following this line until a #TRAILER END statement or another control statement such as #FLD is placed in the trailer.

END

Indicates that this is the end of the lines to be placed into the trailer. There must have been a #TRAILER START statement earlier in the panel definition. No other operands can be specified on a #TRAILER END statement.

POSITION={ YES | NO }

Specifies if the trailer lines are to be displayed. The values available are as follows:

YES

The trailer lines are displayed on the final lines of the physical screen.

NO

This value can be used to suppress the display of the trailer lines, even though they remain in the panel definition.

Examples:

```
#TRAILER START
%This appears on the last line of the panel
#TRAILER END
```

Notes:

- The #TRAILER statements must appear before the first panel line in the definition. If you want to preparse the lines, you must place the #TRAILER statements after the #OPT PREPARSE= statement.
- The field attribute characters that you use in the trailer lines can be defined before or after the trailer lines in the panel.
- The trailer lines cover any panel lines that would otherwise have been displayed.
- The trailer lines are positioned so that they end at the bottom of the physical screen if the window starts at the top of the screen.

Maintaining Menus

This section describes the Common Application Services (CAS) Menu facility.

CAS builds and presents menus, manages user interaction with menus, and provides you with facilities for maintaining menu definitions.

A menu definition contains the information required to build a menu and display it to the user.

A menu definition consists of the following:

- A menu description that includes the menu identifier, its title and other presentation settings
- A series of menu options to be displayed on the menu and their corresponding actions
- A series of menu input fields when data needs to be entered on a menu panel in support of a particular option

Adding a Menu Definition

Defining a menu requires you to fill in three panels. The first of these contains menu identification and presentation details; the second defines the menu options and their corresponding actions; and the third specifies the input fields required for each menu option.

The first panel is the CAS : Menu Description panel.

```
PROD---------------------- CAS : Menu Description -----------------Page 1 of 3
Command ===>                                                    Function=Update

Appl ID ...................+ TST
Menu Number ................ 001

Menu Title ................. Test Application Menu_________________________________

Is This a Primary Menu? .... YES (YES or NO)
Display Userid Info Box? ... NO_ (YES or NO)
Menu Shortcut ..............
Menu Service Procedure ..... ________

Top Left Corner Display .... SOLVPROD____
Top Right Corner Display ... TST001______

Menu Input Field Attributes:
Mandatory Input ............ _
Optional Input ............. _
High Intenstity Output ..... _
Low Intensity Output ....... _

 F1=Help      F2=Split     F3=File      F4=Save
                           F9=Swap                 F11=Page 2   F12=Cancel
```

For information about the fields displayed on the panel, press F1 (Help).

After completing the menu definition details, press F11 (Page 2). The CAS : Menu Options panel is displayed. Specify menu options using this panel.

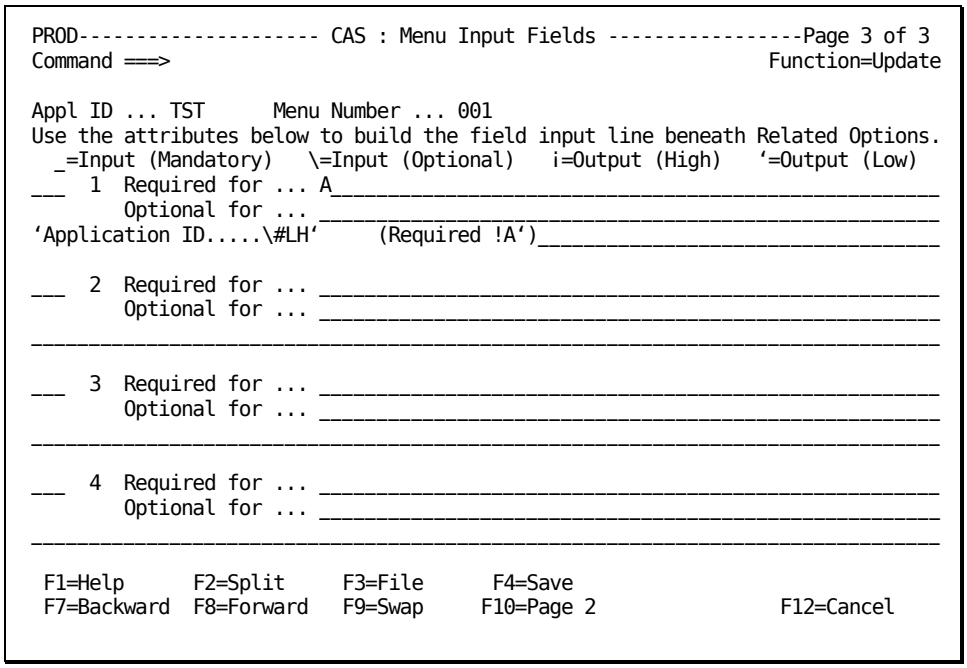
```
PROD----------------------- CAS : Menu Options --------------------Page 2 of 3
Command ===>                                                  Function=Update

Appl ID ... TST      Menu Number ... 001
       Opt     Description                                      Shortcut
___    1 A__ ... Application Register_____________________________
         Shrvars NONE_____________________________________________________
         Action  EXEC $CACALL OPT=ACTION CLASS=MENU ACTION=DISPLAY_________
                 NAME='APPL=TST MENU=20'___________________________________
                 _________________________________________________________
                 _________________________________________________________

___    2 C__ ... Common Application Services Maintenance___________
         Shrvars NONE_____________________________________________________
         Action  EXEC $CACALL OPT=ACTION CLASS=MENU ACTION=DISPLAY_________
                 NAME='APPL=TST MENU=30'___________________________________
                 _________________________________________________________
                 _________________________________________________________



 F1=Help      F2=Split     F3=File      F4=Save
 F7=Backward  F8=Forward   F9=Swap     F10=Page 1   F11=Page 3   F12=Cancel
```

Enter the following four fields once for each option to be displayed on the menu. (You can define up to 15 options; use F8 (Forward) and F7 (Backward), to scroll between them.)

For information about the fields displayed, press F1 (Help).

If you do not want any input fields to appear on the menu, then the definition is complete when you have finished specifying the menu options. Press F3 (File) to save the menu definition.

If any of the options on your menu require data from the user, you need to define input fields. Press F10 (Page 3) to go to the CAS : Menu Input Fields panel.



Enter the following data on the CAS : Menu Input Fields panel to define up to 15 input fields on the menu (use F8 (Forward) and F7 (Backward) to scroll between them).

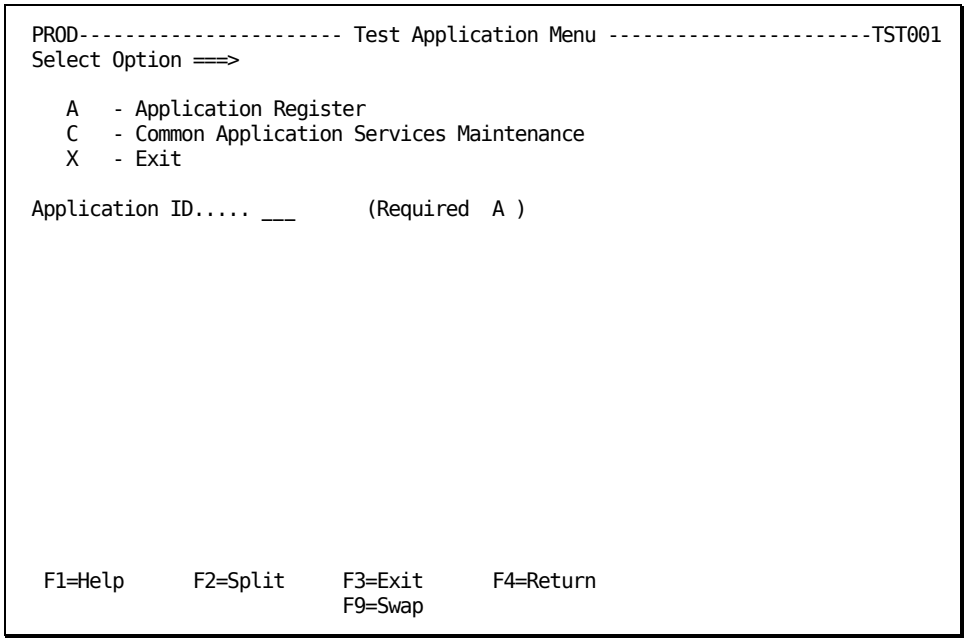
For information about the fields displayed, press F1 (Help).

The line editor commands available in the CAS : Menu Options panel are available on this panel.

After specifying the menu's input fields, press F3 (File) to add the menu definition. To cancel the menu specification, press F12 (Cancel).

Viewing a Menu Definition

Option V displays the menu as the user will see it.



Maintaining Lists

This section describes the Common Application Services (CAS) List facility and how to create and maintain list definitions.

Lists (see page 25) are defined and then stored on a database. They can be recalled to display a series of items as required on a panel. These items are typically objects and are identified with a series of attributes that relate to the object. For example, they may be identified with the object's identifier and description.

The list facility provides a generalized method for displaying and allowing selection of list items. The same list definition can be used to display the following lists:

Action List

Displays a series of objects against which actions can be applied

Single Select

Displays a series of items from which one is selected

Multiple Select

Displays a series of items from which one or more are selected

Numbered List

Displays a series of items that are numbered—an item is selected by entering its corresponding number

The type of list that is created from a list definition is determined by the call to $CACALL, the CAS interface (see page 155).

To create a list, specify a list definition. A list definition comprises the following:

- Identifying information about the list and a specification of its behavior
- The source of the data to be displayed on the list
- A service procedure that is used to retrieve items to be included in the list
- A criteria definition to be used to determine an item's eligibility for inclusion on the list
- A sort expression to determine how items in the list are sorted
- Line entry presentation attributes that let you highlight specific records in a list based on a condition that you specify
- The format of the list

Defining a List

To define a list, complete information on the following panels:

- List Description panel
- List Criteria panel
- List Entry Line Presentation Attributes panel
- List Format panel
- List Entry Line Fields panel

List Description Panel

The CAS : List Description panel, shown below, is the first panel that appears.

```
PROD---------------------- CAS : List Description -----------------Page 1 of 4
Command ===>                                                      Function=Add

Appl ID .............+ ___
List Type ............ _______   (PUBLIC or PRIVATE)
Userid ............... ________  (Userid if PRIVATE)
List Name ............ ________
Description .......... _____________________________________
Title ................ ____________________________________________________
Status ............... ACTIVE__        Group .............+ ____________
Service Procedure ..... ________        Data Source ......... ____________
Get All Entries? ...... YES             Exit Name ........... ________
Add Allowed? .......... YES             Help Name ........... ____________
Default Mnemonic ...... B__             Select Mnemonic ..... S__
Entry Msg Position .... 2__             Entry Msg Length .... ___
Present Empty List? ... YES             Auto Refresh Rate ... _____
                                        Heading Sub Char .... ___
Comments .............. ____________________________________________________
                        ____________________________________________________
                        ____________________________________________________
                        ____________________________________________________

 F1=Help      F2=Split     F3=File      F4=Save
              F8=Forward   F9=Swap                            F12=Cancel
```

The list description panel is used to specify identifying information about the list and its behavior.

You can specify whether a list is active or inactive. Inactive list definitions cannot be used and do not appear on a list of lists. A list can also be defined as private to a specific user—unlike a public list that is available to all users.

Lists can, optionally, be defined as belonging to a group. A group is a logical collection of lists and represents a convenient way of displaying only those lists that are relevant when a list of lists is displayed. For example, you could assign all lists relating to problem management to a group.

The list service procedure retrieves list items and processes requests to perform actions against list items. A list service procedure is dependent on the list's data source.

You can specify the identifier of the help definition to be used when help is requested in the list panel. In addition you can specify the name of a list exit procedure to perform application specific processing.

For information about the fields and options displayed, press F1 (Help).

List Criteria Panel

After entering the list description details, press F8 (Forward) to specify the list criteria on the CAS : List Criteria panel.

```
PROD------------------------ CAS : List Criteria ------------------Page 2 of 4
Command ===>                                                      Function=Add

Appl ID ............... TST
List Type ............. PUBLIC    (PUBLIC or PRIVATE)
Userid ................            (Userid if PRIVATE)
List Name ............. TST001

Criteria Appl ID .....+ ___
         Type ......... ________  (PUBLIC or PRIVATE or FREEFORM)
         Userid ....... ________  (Userid if PRIVATE)
         Name ........+ ________

Sort Expression ....... ____________________________________________________
                        ____________________________________________________

Format List Appl ID ..+ ___
            Type ...... _______   (PUBLIC or PRIVATE)
            Userid .... ________
            Name .....+ ________


 F1=Help      F2=Split     F3=File      F4=Save
 F7=Backward  F8=Forward   F9=Swap                              F12=Cancel
```

This panel is used to optionally specify a criteria, sort expression and existing list format to be used by the list definition.

A criteria can be associated with a list (that is used by the service procedure) to filter items to be included on the list. This criteria can be an existing criteria definition or a freeform criteria in which the user enters the details of the criteria at run time.

You can specify a sort expression to determine the order that items are displayed. The sort expression is used by the list service procedure and thus the syntax and complexity of this expression is dependent on this procedure. If a sort expression is not specified, items are displayed in the order in which they are retrieved by the service procedure.

This panel also lets you specify the identifier of an existing list definition from which the format and presentation attributes for the current list are drawn.

For example, if you want to define several lists that share the same format, you can define a list to be used as a template. This list should be set to INACTIVE so that it cannot be used, and does not appear on a list of lists. You can then define subsequent lists in the same format as the template list.

For information about the fields and options displayed, press F1 (Help).

List Entry Line Presentation Attributes Panel

After entering the list criteria details, press F8 (Forward). The CAS : Entry Line Presentation Attributes panel appears.

```
PROD-------- CAS : List Entry Line Presentation Attributes -----Page 3 of 4
Command ===>                                                    Function=Add

Appl ID ... TST      Type.Userid ... PUBLIC                Name ... TST001

Attr Test
HRN  &PRIORITY = 1_____________________________________________________________
HPN  &PRIORITY = 2_____________________________________________________________
HWN  &PRIORITY = 3_____________________________________________________________
HTN  &PRIORITY = 4:5___________________________________________________________
HBN  &PRIORITY = 6:7___________________________________________________________
___  _____________________________________________________________________________
___  _____________________________________________________________________________
___  _____________________________________________________________________________
___  _____________________________________________________________________________
___  _____________________________________________________________________________
___  _____________________________________________________________________________
___  _____________________________________________________________________________
___  _____________________________________________________________________________
___  _____________________________________________________________________________
___  _____________________________________________________________________________

 F1=Help      F2=Split     F3=File      F4=Save
 F7=Backward  F8=Forward   F9=Swap                          F12=Cancel
```

This panel lets you specify presentation attributes (color and highlighting) for specific records based on a condition (a boolean expression) that you specify. This means that you can highlight particular records of interest in a list to make them easier to identify.

For example, consider a list of problem records. You can specify a condition to identify all problem records that have the status of OPEN and set the display attributes to show those records in, for example, red, flashing text. You need to enter a code representing the presentation attributes and a matching boolean expression for the record that you want highlighted. Enter up to 15 presentation attribute / boolean test pairs to highlight the specified entries within the list.

In the example shown, records with a priority of 1 are shown in high intensity red, with priority 2 are shown in high intensity pink, with priority 3 in high intensity white, with priority 4 or 5 in high intensity turquoise, and those with a priority of 6 or 7 in high intensity blue.

Note: If you specified that the format for the list be drawn from an existing list definition, you cannot define entry line presentation attributes.

A text editor (see page 347) is included that contains commands that you can use.

For information about the fields and options displayed, press F1 (Help).

List Format Panel

After entering the Entry Line Presentation Attributes, press F8 (Forward) to specify the list format on the CAS : List Format panel.

```
PROD------------------------- CAS : List Format -------------------Page 4 of 4
Command ===>                                    Function=Add Scroll ===> PAGE

Appl ID ... TST      Type.Userid ... PUBLIC                 Name ... TST001

LINE <---+----10---+----20---+----30---+----40---+----50---+----60---+----70---
**** ****************************** TOP OF DATA ******************************
0001 A>S/B=Browse U=Update C=Copy P=Print D=Delete
0002 M>S/=Select
0003 S>S/=Select (one only)
0004 Number    Description
0005 &PROBID   &DESCRIPTION
0006 Number    Severity Status     Occurred Date/Time
0007 &PROBID   &SEV     &STATUS    &OCCURDATE  &OCCURTIME
0008 Number    Created     Last Updated
0009 &PROBID   &CRTDATE    &UPDDATE    &UPDTIME  &UPDUSERID
**** ***************************** BOTTOM OF DATA ****************************




 F1=Help      F2=Split     F3=File      F4=Save      F5=Fields    F6=Change
 F7=Backward  F8=Forward   F9=Swap     F10=Left     F11=Right    F12=Cancel
```

A list's format determines how list items are displayed, the other attributes that are displayed about each item, and static heading and option text. This panel displays a text editor that is used to define the format of a list.

A list format consists of:

- Up to four comment lines; one for each list type
- Up to 10 screen formats. Each screen format defines a screen that is displayed when the list is presented to a user (that is, a list can be presented over ten screens). F11 (Right) and F10 (Left) can be used to scroll between the screens.

A screen format consists of:

- Up to 10 heading lines
- An entry line, which contains the list data. (Each line of a displayed list is referred to as an entry line.)

At the top of the CAS : List Format panel text area, you can enter up to four comment lines, one for each type of list, to display specific instructions to the user. When the list is displayed, the appropriate comment line is shown on line 4 of the list panel. Comment lines are optional.

Use the following codes at the beginning of your comment line as follows:

A>

Defines instructions for an Action List.

M>

Defines instructions for a Multiple Select List.

S>

Defines instructions for a Single Select List.

N>

Defines instructions for a Numbered List.

Comment lines can contain NCL variables. For instance, you might want a comment line for an Action List to describe only those actions for which a user is authorized. You could achieve this by setting the entire comment line through a variable set by the list's service or exit procedure.

Any NCL variables defined in the comment line (except for system variables) must be set by the service procedure or list exit. The variable data in the comment line is then substituted by the system when a list is displayed using the list definition.

After entering the comment lines, to display the actual list entries you must define one or more screen formats (up to a maximum of 10). Each of these screen formats consists of a heading and an entry line.

Notes:

- If you specified that the format for the list be drawn from an existing list definition, you cannot define a list format.
- A heading (up to 10 lines long) can contain constant or variable data, or both. Constant data is static, while variable data is set by the service procedure or list exit and substituted into the expression when the heading is displayed. For variables to be recognized, a Heading Sub Char must be specified; for example, if the NCL variable &USERID is used in the heading definition and the Heading Sub Char has been defined as &, the variable is replaced by the user's user ID when the heading is displayed.

  If you wanted the & to appear as a constant in the heading, you would have to define a different Heading Sub Char; therefore, if you specify the ~ character as the Heading Sub Char, for the previous example you would use ~USERID instead of &USERID.
- An entry line consists of entry line fields (NCL variables), each of which is replaced by an attribute of the list item when the list is displayed. An entry line can also contain constant data. Note, however, that the first non-blank character on the line must be an ampersand (&).

  Entry line fields can be the following:

- Real variables
- Aliases for real variables

List Entry Line Fields Panel

To use aliases, press F5 (Fields). The CAS : List Entry Line Fields panel is displayed. This panel displays all the entry line fields defined on the CAS : List Format panel.

```
PROD-------------------- CAS : List Entry Line Fields -------------------------
 Command ===>                                      Function=Add Scroll ===> PAGE

 Appl ID ... TST      Type.Userid ... PUBLIC                Name ... 001

 Entry Line Field  Real Field
 A                 TSTID
 B                 TSTTYPE
 C                 TSTTIME
 D                 TSTDATE
 E                 TSTGROUP
 F                 NAME
 G                 PHONE
 CREDAT            CREDAT
 CRETIME           CRETIME
 CREUSER           CREUSER
 UPDTDATE          UPDTDATE
 UPDTTIME          UPDTTIME
 UPDTUSER          UPDTUSER
 **END**


  F1=Help      F2=Split     F3=File      F4=Save      F5=Format
  F7=Backward  F8=Forward   F9=Swap                              F12=Cancel
```

This panel is used to assign aliases to attribute identifiers used on the list format panel.

Entry line fields can sometimes be longer than the data to be displayed, meaning the NCL variable to be defined in the entry line might not fit.

To overcome this, you can define a shorter variable name in the entry line, and then assign the name of the real field that is to be substituted when you display a list using the definition.

The value of an entry line field can be the same as that of the associated real field (when its length does not cause formatting difficulties).

For information about the fields and options displayed, press F1 (Help).

After entering the list definitions, press F3 (File). To cancel the list specification, press F12 (Cancel).

Resetting the List Cache

When a list definition is recalled for use it is loaded into a VARTABLE for optimum performance. This is referred to as the List Cache. As different lists are used they are added to this cache.

Use F12 (ResCache) to reset the list cache and clear all list definitions from the cache. If the MODS file is shared the list cache needs to be reset on all systems except the system being used to maintain the list definitions. It also needs to be done if lists have been moved, copied, or deleted using the Definition Utility. Reset the list cache only after you have changed, moved, copied, or deleted list definitions on the maintenance system.

If you do not perform this action, the modified list definitions may not take effect on the other systems until the next time the product region is started up.

Note: You do not need to reset the list cache on the maintenance system after you delete or update a list definition. In either action, the list definition is deleted from the list cache on the maintenance system automatically (but not on other systems). The next time you access the list definition on the maintenance system, an updated copy is retrieved from the database.

Maintaining Help

This section describes the Common Application Services (CAS) Help facility, which is used to define and maintain online help for applications.

This facility provides a flexible means of defining context sensitive help at various application levels.

More information:

Help (see page 26)

Help Definitions

When a user presses F1 (Help), one or several panels are displayed that contain help information. The help that is displayed is dependent on where the user pressed F1 (Help). For example, if the user requests help while the cursor is located in an input field then the help associated with that field is displayed.

Note: Not all fields are supported by the CAS Help facility.

Help text can be structured in panel format, in scrollable text format, or using a combination of both. Facilities for constructing menus, help indexes and tutorials within the help file are also available. Help files can be merged or copied, both during maintenance and while being displayed.

A text editor is available to define and maintain help text. Optional embedding of control codes is provided to make control of text color, highlight, intensity and formatting as easy as possible. Facilities for browsing (which displays the help file) and viewing (which displays the help as the user sees it) are available.

Help files can be added and maintained at either application level, function level, window level, or field level.

Printing Help Files

Use option P to print the contents the help file in the format that a user sees when they request help.

More information:

MODS Component Reports (see page 137)

Viewing Help Files

Use option V to view the help file in the format that a user sees when they request help.

Maintaining Function-Level Help

Function-level help describes a particular function within an application. Typically, each function-level help file is associated with one of the application's panels.

Function-level help can also include the following:

Tutorial

A tutorial for the application can consist of just a single help file, or it can refer to other help files through the .CP and .MU macros. If you define a function-level help file called TUTORIAL, CAS automatically assigns a function key to access the tutorial in the Function Key Area on the screen, and display your tutorial file when the user presses that key.

Help Index

A help index can consist of a menu of all help available for the application (using the .MU macro). If you define a function-level help file called INDEX, CAS automatically assigns a function key to access the index in the Function Key Area on the screen, and display your index file when the user presses that key.

You have to write your own help index file. The help index file typically contains a number of .MU macros that display a menu of help topics from which a user may select. For an example of a help index file, view the INDEX function-level help for the $CA application ID.

Use option H (Help) on the CAS : Maintenance Menu (/CAS) and specify an application ID to access function-level help. The CAS : Function Level Help List for the selected application appears. This panel lets you maintain function-level help or list field and window-level help for a help function.

For information about the fields and actions on this panel, press F1 (Help).

Adding a Function-Level Help File

Use F4 (Add) to create a new function-level help file. A panel appears where you must enter the name of the new function.

Note: The application ID appears and may not be changed.

Press F3 (File) to display the CAS : Function Level Help Definition panel. Enter a description of the help file in the Help Description field, then the text of the file.

A text editor (see page 347) is available to edit the help files.

Listing Function-Level Help Files

Use option H (Help) on the CAS : Maintenance Menu to display a selection list of function-level help files.

When specifying this option, the Appl ID field must also be specified.

You can optionally specify a prefix in the Name Prefix field to limit the selection of records to be displayed on the list. For example, to list all function-level help whose Function Name starts with the $ character, specify $ in the Name Prefix field.

Maintaining Window-Level Help

Window-level help describes a particular window on a panel. Use option LW (List Windows) beside an entry on the CAS : Function Level Help List to access window-level help.

For information about the fields on this panel, press F1 (Help).

Adding a Window-Level Help File

Use F4 (Add) to create a new window-level help file. A panel appears where you must enter the coordinates of the new window.

Note: The application ID and function name appear and may not be changed.

Press F3 (File) to display the CAS : Window Level Help Definition panel. Enter a description of the help file in the Help Description field, then the text of the file.

A text editor (see page 347) is available to edit the help files.

Maintaining Field-Level Help

Field-level help defines help for a specified field on a panel. Use option LF (List Fields) beside an entry on the CAS : Function Level Help List to access field-level help.

For information about the input fields, press F1 (Help).

Adding a Field-Level Help File

Use F4 (Add) to create a new field-level help file. A panel appears where you must enter the field name.

Note: The application ID and function name appear and may not be changed.

Press F3 (File) to display the CAS : Field Level Help Definition panel. Enter a description of the help file in the Help Description field, then the text of the file.

A text editor (see page 347) is available to edit the help files.

Facilities for Help Text Editing and Formatting

The text contained within help files can be modified using the help text editor (see page 347). There are also text formatting macros available that you can embed in the help text to control the appearance of the text when displayed to the user.

Help Macros

Help macros can be embedded in help files to control the appearance of text when displayed on a panel. They must start in column 1, and there can be only one macro per line.

You can use the following macros to perform actions:

.AT

Define a display attribute.

.BX

Draw a box around trailing text.

.CE

Center trailing text.

.CH

Center a heading.

.CM

Add a comment.

.CP

Copy a help file.

.CT

Control help.

.DE

Selectively display help text depending on terminal types .

.LI

Selectively display help text depending on licensed features.

.LN

Draw a line across the screen.

.MU

Define a menu line.

.NP

Skip to a new page.

.OP

Selectively display help text depending on the operating system.

.PH

Define a primary heading.

.RA

Remove a display attribute.

.SH

Define a sub heading.

.SP

Skip lines.

.TI

Define the title line.

Note: For full descriptions, see the online help.

Display Attributes

Some of the macros allow characters to be specified that are used as display attributes (for example, to control color and highlighting).

The following predefined attribute characters are provided:

! (hexadecimal value X'5A')

This display attribute is used to highlight important help text—for example, the name of a command. It is based on the system variable &ZPOUTHIC.

‘ (hexadecimal value X'79')

This display attribute is used for normal help text. It is based on the system variable &ZPLABELC, which is the default text display attribute.

| (hexadecimal value X'6A')

This display attribute is used for headings when you are not using the .PH and .CH macros. It is based on the system variable &ZPSUBTLC.

Note: This is the EBCDIC broken bar character.

You should use this predefined set of display attributes to ensure that help text is consistent in appearance and conforms to installation standards.

If you need to use one of the above attribute characters as a real character, the .RA help macro can be used to remove a predefined attribute character definition.

Note: Do not use the asterisk (*) or ampersand (&) characters as display attributes—these characters are reserved by the system.

Maintaining Messages

This section describes the Common Application Services (CAS) Message facility.

Messages are items of text that are displayed on a panel in response to specific events (such as error conditions). Use information in this section to define and maintain message definitions.

More information:

Messages (see page 28)

Message Definitions

A message definition contains message identifying information as well as the actual message text, an explanation of its purpose and a description of system and user action taken.

Because message definitions are not defined as belonging to particular applications, any defined message can be used by all applications.

To assist in maintaining message definitions, a full-screen text editor is provided.

Defining Message Definitions

Defining message definitions requires you to complete two panels. The first of these defines the message ID, text, and explanation; the second specifies the Action and User Action.

The first panel is the CAS : Message Text/Explanation panel.

```
PROD------------------ CAS : Message Text/Explanation ------------- Page 1 of 2
Command ===>                                                    Function=Add

Message ID ......... ___________
Substitution Char .. &__



Message Text

  ____________________________________________________________________________
  ____________________________________________________________________________
  ____________________________________________________________________________

Message Explanation

  ____________________________________________________________________________
  ____________________________________________________________________________
  ____________________________________________________________________________
  ____________________________________________________________________________
  ____________________________________________________________________________

 F1=Help      F2=Split     F3=File      F4=Save
              F8=Forward   F9=Swap                 F11=Edit     F12=Cancel
```

You can extend the Message Explanation text by tabbing the cursor to any of its data entry lines and pressing F11 (Edit).

A text editor (see page 347) is available to edit the text.

After completing this panel, press F8 (Forward). The CAS : Message System/User panel is displayed.

```
PROD---------------- CAS : Message System/User Action ------------- Page 2 of 2
Command ===>                                                    Function=Add

Message ID ......... ____________
Substitution Char .. &__



System Action

  ____________________________________________________________________________
  ____________________________________________________________________________
  ____________________________________________________________________________
  ____________________________________________________________________________

User Action

  ____________________________________________________________________________
  ____________________________________________________________________________
  ____________________________________________________________________________
  ____________________________________________________________________________

 F1=Help      F2=Split     F3=File      F4=Save
 F7=Backward               F9=Swap                 F11=Edit     F12=Cancel
```

You can extend the System Action or User Action text by tabbing the cursor to any of the action's data entry lines and pressing F11 (Edit).

For information about the field and options on this panel, press F1 (Help).

After completing the message definition, press F3 (File). To cancel the definition, press F12 (Cancel).

Viewing a Message Definition

Select option V to display a message definition in the format that a user would see the message help. Use F7 (Backward) and F8 (Forward) to scroll up and down and use F6 (NewMsg), F10 (Prevmsg) and F11 (Nextmsg) to view other message definitions.

Maintaining Tables

This section describes the Common Application Services (CAS) Table facility. Tables contain sets of table entries that are used to validate data entry, and how to create and maintain table definitions and table entries.

A table (see page 29) consists of a table definition and a series of table entries. It defines the location and type of data against which input for a field is validated. Table definitions can be created, updated, browsed, deleted, copied or listed using the CAS : Table Definition List.

Entries can be stored as data within a MODS control file. The entries can also be CA SOLVE:InfoMaster field names or values of an CA SOLVE:InfoMaster field.

Note: Table entries can only be added to tables that have an Edit type of TABLE.

More information:

Maintaining Application Groups (see page 50)

Defining and Maintaining Table Definitions

Defining a table requires you to fill in two panels. The first of these defines the actual table, the second specifies all the fields required for each entry in the table.

```
PROD---------------------- CAS : Table Description -----------------Page 1 of 2
 Command ===>                                                    Function=Add

 Appl ID .................. ___   Field name ............... ___________
 Field description ........ ____________________
 Edit type ................ TABLE (TABLE, OSATT, OSFLD, IMFLD or IMREC)
 For Edit type = TABLE:
    Validation exit ................. ________
    Sequence numbers ................ ___ (YES or NO)
    Load table? ..................... ___ (YES or NO)
    Max abbreviation length ......... ___ (3 - 8 or blank if none)
    Max full value length ........... ___ (3 - 20)
    Max description length .......... ___ (3 - 38 or blank if none)
 For Edit type = IMFLD or IMREC:
    InfoMaster category ............. ____
 For Edit type = IMREC:
    InfoMaster field ................ ___________
    InfoMaster description field ... ___________
 For Edit type = OSATT or OSFLD:
    Object Services Class ID ........ ___________
 For Edit type = NDBFL:

  F1=Help      F2=Split     F3=File      F4=Save
               F8=Forward   F9=Swap                            F12=Cancel
```

For information about the fields and options on this panel, press F1 (Help).

If the table has an Edit type of TABLE, you have the option of specifying additional input fields to appear on the Table Entry panel:

- If you do not require any extra data for the table entries, press F3 (File) to create the table definition. To cancel the create, press F12 (Cancel).
- To define extra input fields for the Table Entry panel, press F8 (Forward). The CAS : Table Text Fields panel is displayed.

```
PROD---------------------- CAS : Table Text Fields -----------------Page 2 of 2
 Command ===>                                                  Function=Update

 Appl ID ................... TST
 Field name ................ TST001


                  Field   Text description        Length(4-38)

                    1     I/M Application_____    4___
                    2     Service Procedure___    8___
                    3     System Name_________    24__
                    4     Record Category_____    24__
                    5     ____________________    ____
                    6     ____________________    ____
                    7     ____________________    ____
                    8     ____________________    ____
                    9     ____________________    ____
                   10     ____________________    ____


  F1=Help      F2=Split     F3=File      F4=Save                  F6=Entries
  F7=Backward               F9=Swap                             F12=Cancel
```

For information about the fields and options on this panel, press F1 (Help).

After defining the validation table, press F3 (File). To cancel the table specification, press F12 (Cancel).

Maintaining Table Entries

Table entries represent the valid values for a data entry field. You must define a table definition before you can define its table entries.

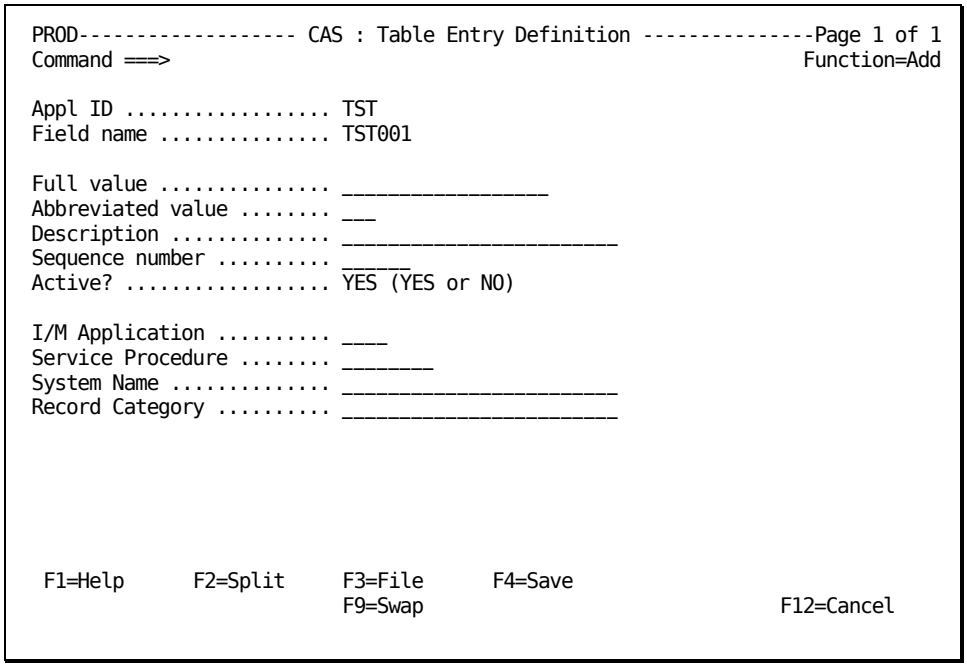
Note: Table entries are only entered for tables that are defined with an edit type of Table. Entries for other types are validated against the external source, such as an InfoMaster field.

To maintain table entries, use option LE (List Entries) beside an entry on the CAS : Table Definition List. The CAS : Table Entry for Field (*field name*) appears. This list provides options to maintain table entries.

Adding a Table Entry

Use F4 (Add) to create a new table entry. A panel appears where you must enter the full value of the new table entry.

Note: The application ID and field name appear and may not be changed.



Note: Only Appl ID, Field name, Full value, and Active? fields appear on every Table Entry Definition panel. The other fields depend on how the table is defined.

For information about the fields and options on this panel, press F1 (Help).

Any additional fields that are specified in the table definition also appear on the Table Entry Definition panel. Complete these fields as required.

After specifying the table entry, press F3 (File).

Reloading Validation Tables

Whenever table entries or table definitions are added, updated or deleted (from a table that has already been loaded), a reload must be performed. This means that all table definitions for the application are loaded from the control file into memory.

If a table is flagged for loading and has an Edit type of TABLE, all its entries are also loaded into memory.

A reload can be initiated from a list of tables by selecting option R. A confirmation message is displayed when the reload is complete.

Maintaining Criteria

This section describes the Common Application Services (CAS) Criteria facility, and how to define and maintain criteria definitions.

Criteria are rules that can be used in a test, for example, the expression on an &BOOLEXPR statement or the scan expression on an &NDBSCAN statement. They are used as a condition for the occurrence of some event—such as the inclusion of a record in a list, for example.

More information:

Criteria (see page 30)

Criteria Definitions

A criteria definition for a set of criteria consists of an identifier, a description of its behavior, a run time panel, a help name, a criteria exit, a data source, exit parameters, and the criterion condition.

A set of criteria can simply compare static values (for example, numbers or field names) or can be complex, combining numerous operators, attributes, connectors, parentheses, and values.

Criteria values can be variables (for example, the current date). When a set of criteria is recalled to perform a test, variable values are supplied either interactively by the user, using a run time panel, or by an exit procedure.

Criteria are used for the following purposes:

- Selection of objects for inclusion in a list
- Inclusion of objects in a report when using Report Writer
- The condition for display of a panel within a panel domain

Criteria definitions can be public or private. Public definitions are available to all users, while private definitions are available only to their owner.

Criteria can be predefined and reused. A criteria definition specifies the set of criteria and is named with a unique identifier. Criteria are stored on a database so that they can be recalled to provide the test that they define.

Run Time Panel

A criteria definition can include the identifier of a run time panel, to enable interactive entry of variable data for the criteria. This panel must request entry of data for variables within the set of criteria.

Criteria Exit

You can specify a criteria exit, an NCL procedure to be executed during the process of building the set of criteria. The purpose of the criteria exit is to enable the initialization and editing of the run time panel and specification of variable data that is to be included in the set of criteria.

More information:

Criteria Exit Procedures (see page 245)

Data Source

You can specify a data source for the criteria definition. The purpose of this is to allow the criteria exit to determine the type of data to which the criteria are applied. This enables the definition of a general purpose criteria exit that can be used by multiple criteria definitions (since the exit determines the type of data and processes accordingly).

Exit Parameters

Exit parameters are passed to the criteria exit when it is executed in the process of building a set of criteria. A criteria exit can base its behavior on the exit parameters defined within a criteria definition. Thus it is possible to modify the behavior of a set of criteria by changing the exit parameters without having to change the exit procedure.

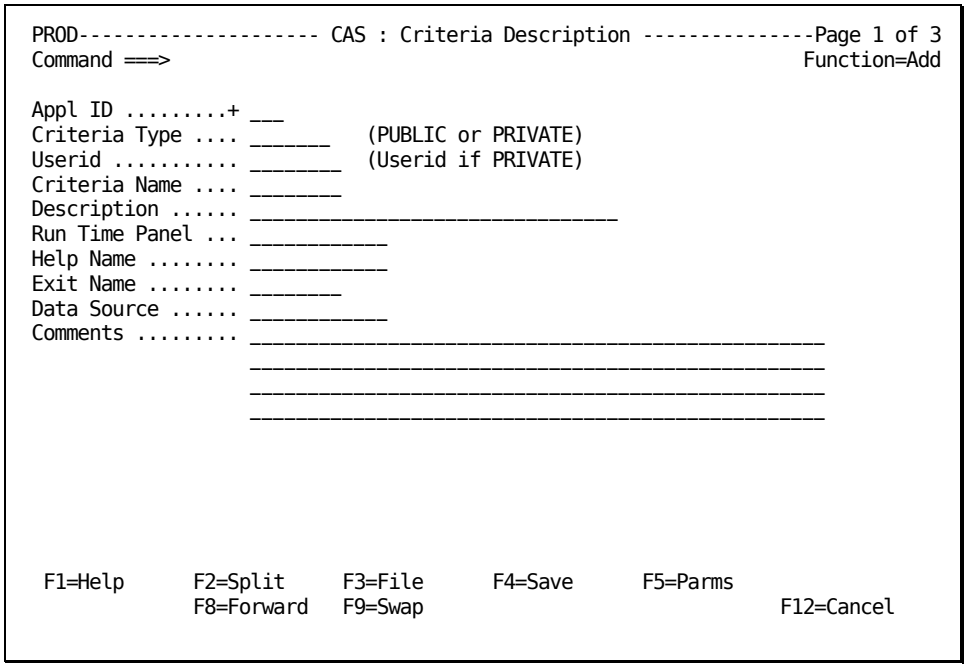
Substituting Variable Data

The NCL built-in function &ZSUBST is used to substitute variable data into the criteria. The ampersand character (&) is used as the substitution character. The variables defined within the criteria can be set by the user through entry of data on a run time panel or by the criteria exit procedure.

Defining Criteria

Defining a criteria requires you to complete a description panel, which identifies the criteria, and then specifying the actual criteria expression on a second panel.

You can also, optionally, define a series of exit parameters on a third panel if your criteria makes use of the criteria exit.



For information about the fields and options on this panel, press F1 (Help).

After completing the criteria description, press F8 (Forward) to specify the actual criteria on the Criteria Text panel.

```
PROD------------------------ CAS : Criteria Text ------------------Page 2 of 3
 Command ===>                                  Function=Add Scroll ===> PAGE

 Appl ID ... ZPR      Type.Userid ... PUBLIC                Name ... ASSIGN1

 LINE <---+----10---+----20---+----30---+----40---+----50---+----60---+----70-->
 **** ****************************** TOP OF DATA ******************************
 0001 STATUS NE ALL 'CLOSED','CANCELLED' AND ASSIGNEE EQ '&USERID'
 **** ***************************** BOTTOM OF DATA *****************************









  F1=Help      F2=Split     F3=File      F4=Save      F5=Find      F6=Change
  F7=Backward  F8=Forward   F9=Swap                              F12=Cancel
```

The criteria text panel is used to define the condition (or rule) for criteria. For example, the expression on an &BOOLEXPR statement or the scan expression on an &NDBSCAN statement.

The simplicity or complexity of the criteria and its syntax is dependent on what it is to be used for. For example, if it is to be used as the scan expression on an &NDBSCAN statement then it must obey the syntax rules for an &NDBSCAN scan expression.

Criteria can consist of constant and variable data. Constant data never changes, for example, the identifier of an attribute. Variable data can change, for example, the current date. Variable data in the criteria is represented by an NCL variable, for example, &USERID. The NCL variables defined in the criteria, except system variables, must be set by the criteria exit or by the user entering data on the run time panel. The variable data in the criteria is substituted by the system when the criteria is built.

In the panel shown before, the criteria specifies all problems that are not closed or canceled that belong to the current user.

If the criteria uses an exit procedure, you can define exit parameters to be used by that procedure. To achieve this, press F5 (Parms) in the CAS : Criteria Description panel.

After the above action, the CAS : Criteria Exit Parameters panel is displayed.

```
PROD------------------- CAS : Criteria Exit Parameters -------------Page 3 of 3
Command ===>                                    Function=Add Scroll ===> PAGE

Appl ID ... ZPR      Type.Userid ... PUBLIC                 Name ... ASSIGN1

LINE <---+----10---+----20---+----30---+----40---+----50---+----60---+----70-->
**** ****************************** TOP OF DATA ********************************
     SEVERITY=50
     FINALDAY=MON
                                                                              |
 F1=Help      F2=Split     F3=File      F4=Save      F5=Find      F6=Change
 F7=Backward  F8=Forward   F9=Swap                               F12=Cancel
```

This panel lets you enter parameters for the exit procedure in the format required by the exit.

After specifying the criteria, press F3 (File).

Maintaining Commands

This section describes the Common Application Services (CAS) Commands facility. This facility provides you with the ability to define commands to perform your own specialized functions.

More information:

Commands (see page 30)

Command Definitions

A command definition contains identifying information about the command and specifies an action that occurs when the command is executed.

Unlike the other CAS components, commands are not defined as belonging to a particular application. This means that all commands defined to CAS can be used by all applications.

Define a Command

You must specify the following information when defining a command:

- Command name
- A brief description
- The action to perform

To define a command

1. On the CAS : Maintenance Menu, select Option C.

   The Command Definition List panel appears.

2. Press F4 (Add).

   The Command Definition panel appears.

   Note: For information about the fields, press F1 (Help).

3. Specify the required information, and press F3 (File).

   The command is defined.

Example: Command Definition

This example shows a MYCMD command that executes a procedure.

```
PROD--------------------- CAS : Command Definition ---------------- Page 1 of 1
Command ===>                                                      Function=Add

Command ID ... MTCMD___

Description .. Execute the named procedure_______

Comments ..... ________________________________________________
               ________________________________________________
               ________________________________________________
               ________________________________________________

Action ....... EXEC &$CMPARM1_____________________________________________________
               ____________________________________________________________________
               ____________________________________________________________________
               ____________________________________________________________________
```

Using the Action Field

The action field specifies the action to be taken when this command is entered.

An action consists of any of the following commands, or a word recognized by the calling procedure, together with any required parameters.

CMD [*command*]

Enter the Command Entry facility and prime the command field with the *command*.

EXEC *procname* [*parameters*]

Execute the procedure *procname* with specified parameters.

DISCONN

Disconnect the session.

HELP

Display online Help information.

KEYS [PRI | ALT | OFF | SET]

PRI

Build and return Primary function key area.

ALT

Build and return Alternate function key area.

OFF

Turn function key display off.

SET

Lets user set function keys 13-24.

The default is to toggle between PRI and ALT.

LOCK

Lock the session.

NOTEPAD

Invokes the CAS Notepad facility.

PASSWORD

Lets user change password and/or user details.

PQUEUE [user ID]

List queued print requests for *user ID.* Defaults to your user ID and can be a generic ID if terminated by an asterisk.

PSKIP [*options*]

Jump to the specified panel (equivalent to the '=' panel skip command).

RETRIEVE [? prefix]

Retrieve the last command. ? is for prompt support. If followed by a prefix, obtains a list of commands starting with that prefix.

SPLIT

Split the window at the current cursor position.

START *procname [parameters]*

Start NCL procedure *procname* with specified *parameters*.

SWAP

Swap to the other window.

WHERE

Display current NCL procedure details.

When the command is entered and passed to CAS (through $CACALL) by the controlling NCL procedure, CAS either:

- Performs the required action (if the action is from the table below)
- Returns the unknown command (and any parameters) to the calling procedure

The parameters can include constant or variable data, or both. Variable data consists of NCL variables that are set by CAS when the command is processed.

Valid variables are as follows:

&$CMPARMS

Contains all data entered by the user following the command.

&$CMPARM1 ... &$CMPARM64

Contains the command parameters (delimited by a space) entered by the user following the command.

After completing the command definition, press F3 (File).

Reloading Command Definitions

Use F12 (Reload) to reload command definitions when you make a change to one or more commands.

A reload should be performed when any command maintenance is carried out—if a reload is not performed, the changes do not take effect until the next time your product region is started.

Maintaining Maps

This section describes the facilities for creating, maintaining and compiling map definitions for ASN.1 source code.

Map Definitions

Map definitions comprise a number of source statements, and are compiled to a loadable form. Map source can be kept in any convenient source library in a similar fashion to NCL procedures.

All compiled maps are lodged in the OSCNTL file that serves as the Map Library. From this library the loadable form of a map can be accessed on demand. A map is loaded only when specifically requested for use by an NCL procedure.

Map Library Structure

The Map Library is a keyed file that contains:

- Map registration records
- ASN.1 source records
- Load module records

For each map defined to the system there is one map registration record. This registration record contains global information that registers the unique map name, and other identification data. It also contains information about the system data set where the source statement file can be located.

One or more load module records exist for each map. These records are compiled from the map source statements, and represent the amalgamation of the logical and physical data structure information into a form understandable by the Map loader.

The source records contain the ASN.1 source code from the last successful compile of the map.

Creating and Maintaining the Map Source

A map must first be registered to the system before it can be used by Mapping Services. The process of registration defines some of the global attributes of the map that let it co-exist as a unique entity in the system.

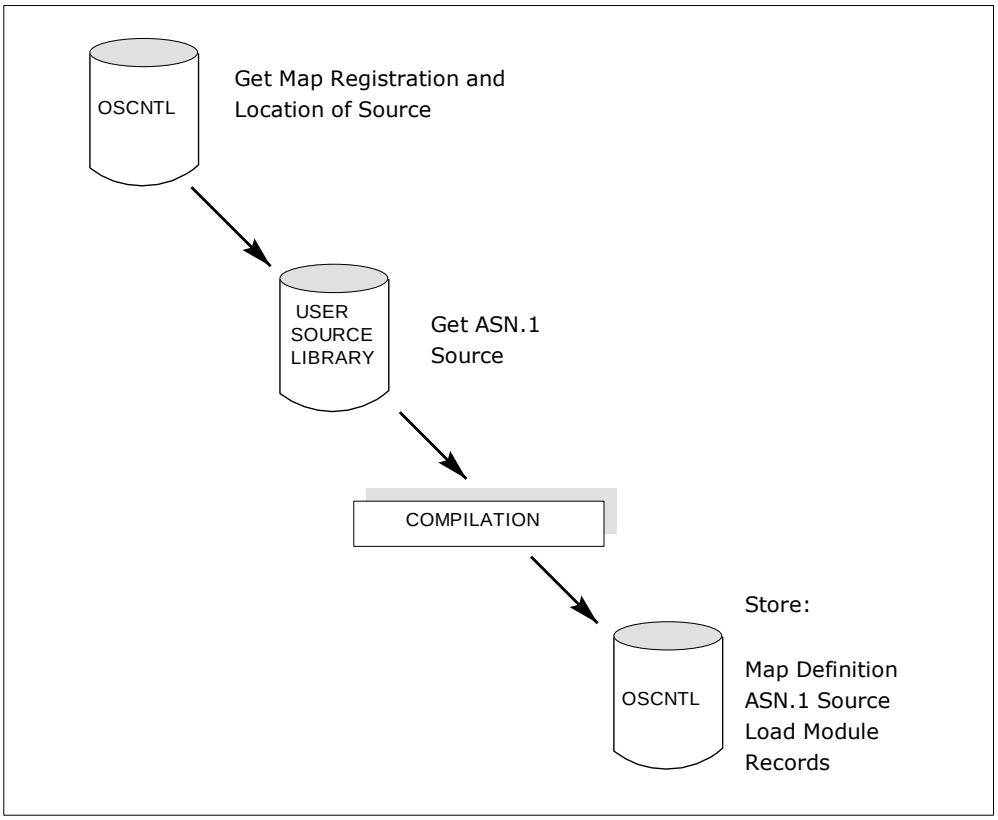
Before adding maps to the Map Library, you should consider the restrictions (see page 304) placed on the ASN.1 language by the Mapping Services implementation.

Compiling a Map

After a map is registered and its ASN.1 source location defined, the map can be compiled to produce the map load module. This module is a series of records kept in the Map Library that are accessed by the Mapping Services Loader when a map is loaded for use in the system.

If the compilation is successful, the source is stored in the Map Library for reference purposes.

The process of registering and compiling a map is as follows:



Loading a Map

During the load phase the contents of the load module records are reduced to an internal format that can be used by Mapping Services. Since references to imported definitions are resolved on load, the load module containing the imported definitions must be previously compiled.

Any inconsistencies in either the ASN.1 logical structuring, or the Mapping Services local form definitions, causes the map load to fail.

Once compiled, maps can be loaded from the Map Library to validate that they load without problems. Maps are normally loaded automatically when they are first referenced, requiring no action on your part. You might, however, want to load a map, using the LOAD MAP command, to determine if it can be loaded successfully.

The UNLOAD MAP command may be used to remove an old copy of a map from storage. The SHOW MAPS command displays a list of maps that are currently loaded.

Defining a Map

You can define a new map by selecting the appropriate option from the Mapping Services Primary Menu or from a list of existing map definitions (by pressing F4 (Add)). You can also create a new map by copying an existing map definition.

Mapping Services Primary Menu

To access the Mapping Services : Primary Menu, enter **/MAPS.** This panel lets you maintain map definitions, and to compile ASN.1 source code for the map.

```
PROD------------------- Mapping Services : Primary Menu ------------------/MAPS
 Select Option ===>

    A   - Add Map
    B   - Browse Map
    BS  - Browse Map Source
    U   - Update Map
    D   - Delete Map
    C   - Copy Map
    P   - Print Map
    L   - List Maps
    CM  - Compile Map
    V   - View Map Structure
    X   - Exit

 Map Name ...__________________________________ ( Required B BS U D C P V
                                                  Optional A CM L )




  F1=Help      F2=Split     F3=Exit      F4=Return
                            F9=Swap
```

The Map Name field lets you enter the identifier of a map in support of the relevant menu option (for instance if you select the Update Map option you must specify the map that you want to update.)

Adding a Map Definition

Select option A from the Mapping Services : Primary Menu to display the Mapping Services : Map Definition panel. Use this panel to define a new map definition.

```
PROD----------------- Mapping Services : Map Definition ----------- Page 1 of 1
 Command ===>                                                  Function=Add

 Map Name ................. ____________________________________

 Description .............. ____________________________________

 Comments ................. ________________________________________________
                            ________________________________________________
                            ________________________________________________
                            ________________________________________________

 Generated by Appl ID .....
 Source Member Name ....... ________          Number of Source Lines ...
 Source DDNAME ............ ________

 Modification Level .......

 Last Compiled On .........


  F1=Help      F2=Split     F3=File      F4=Save
                            F9=Swap                          F12=Cancel
```

For information about the fields displayed on the Mapping Services : Map Definition panel, press F1 (Help).

After completing the Mapping Services : Map Definition panel, press F3 (File) to create the map.

Maintaining Map Definitions

This section describes the maintenance facilities available for existing map definitions.

Listing Map Definitions

Select option L to display map definitions that are already defined. If you make an entry (or partial entry) in the Map Name field on the menu panel, the list is constrained to map definitions that generically match your entry.

Map definitions are displayed over four panels. Use F11 (Right) and F10 (Left) to scroll between the panels of the list.

```
PROD-------------- Mapping Services : Map Definitions List --------------------
Command ===>                                                  Scroll ===> PAGE

       S/B=Browse BS=BrowSrc U=Update D=Delete C=Copy P=Print CM=Compile V=View

    Map Name                        Description
    $ACMPGNR                        Disconnect data for VTAM Generic Resource
    $ACMPMSD                        Global Session Criteria Definition Map
    $ACMPMSL                        Global Session List Map
    $ACMPMSR                        Global Session Criteria List Map
    $ACMPSDD                        SOLVE:Access - Individual session definit
    $ACMPSDL                        SOLVE:Access - Session Definition List re
    $ACMPSRQ                        SOLVE:Access : Session Replay Request
    $AMACPRM                        Alert Monitor - Action Parameter
    $AMACTN                         Alert Monitor - Action Template Class
    $AMACTNA                        Alert Monitor - Actions for the Action Cl
    $AMALCLS                        Alert Monitor - Alert Class (metaclass)
    $AMALERT                        Alert Monitor Map
    $AMALRUL                        Alert Monitor - Alert Rule Object
    $AMAPPL                         Alert Monitor - Application
    $AMFILTR                        Alert Monitor - Filter
    $AMHISTM                        Alert Monitor - History Logging Config Re
  F1=Help      F2=Split     F3=Exit      F4=Add       F5=Find      F6=Refresh
  F7=Backward  F8=Forward   F9=Swap                  F11=Right
```

For information about the fields and actions on the panels, press F1 (Help).

Browsing the ASN.1 Source Code for a Map

Option BS displays the ASN.1 source code for the map (from the last successful compile).

```
PROD--------------- Mapping Services : ASN1 Map File Source ----Columns 001 072
 Command ===>                                    Function=Browse Scroll ===> PAGE

 Map Name ...... $OSSDU                           Source Member ... $OSSDU
 Description ... Service Data Unit                Source DDNAME ... COMMANDS

 LINE ----+----10---+----20---+----30---+----40---+----50---+----60---+----70---
 **** ***************************** TOP OF DATA ********************************
 0001 --     ++INCLUDE #OSVERSD
 0002 --*************************************************************************
 0003 --                                                                        *
 0004 -- NAME        : $OSSDU                                                   *
 0005 --                                                                        *
 0006 -- DESCRIPTION : ASN.1 map for Service Data Unit used in calling          *
 0007 --               Object Services                                          *
 0008 --                                                                        *
 0009 -- CREATE DATE : 31-OCT-1991                                              *
 0010 --                                                                        *
 0011 --*************************************************************************
 0012
 0013 --     ++INCLUDE #NMCOPYD
 0014
  F1=Help      F2=Split     F3=Exit      F4=Return    F5=Find
  F7=Backward  F8=Forward   F9=Swap     F10=Left     F11=Right
```

Printing a Map Definition and Its ASN.1 Source Code

Option P displays the PSM : Confirm Printer panel for you to enter the required printing details. Press F6 (Confirm) to confirm the action.

More information:

MODS Component Reports (see page 137)

Compiling a Map's ASN.1 Source Code

Select option CM from the Mapping Services : Primary Menu to compile the map.

The map definition is displayed. Check (and modify if necessary) the Source Member and DDName fields. You can use the BROWMEM and BROWSRC keys to browse the member definition and source definition, respectively, from this panel.

Press F6 (Action) to compile the map.

If any warnings or errors occur, the Mapping Services : Compiler Messages screen is displayed.

```
PROD----------------- Mapping Services : Compiler Messages ---------------------
 Command ===>                                                  Scroll ===> PAGE

                                                                          S/H=Help
     Error and/or Warning Messages for ASN.1 Compile
     DD0904 THE FOLLOWING MESSAGES WERE ISSUED BY COMPILER FOR MAP $OSSDU
     DD0361 Warning: line 147 - unnamed component
     **END**

  F1=Help      F2=Split     F3=Exit                  F5=Find      F6=Refresh
  F7=Backward  F8=Forward   F9=Swap                 F11=Right
```

Viewing a Map Structure

Select option V to display the compiled map structure of the map definition that you specify in the Map Name field.

```
PROD---------------- Mapping Services : View Map Structure --Line 1 to 17 of 17
Command ===>                                    Function=Browse Scroll ===> CSR

Map Name ...... $AMMPFD

******************************** TOP OF DATA ********************************
NAME
DESCRIPTION
CREATED.DATE
CREATED.TIME
CREATED.USERID
LASTUPDATED.DATE
LASTUPDATED.TIME
LASTUPDATED.USERID
FILTER.{*}.LPAREN
FILTER.{*}.LPARENVAL
FILTER.{*}.ATTR
FILTER.{*}.OPER
FILTER.{*}.VALUE
FILTER.{*}.GENERIC
FILTER.{*}.BOOLEAN
FILTER.{*}.RPAREN
 F1=Help      F2=Split     F3=Exit      F4=Return    F5=Find
 F7=Backward  F8=Forward   F9=Swap     F10=Left     F11=Right
```

Printing MODS Component Reports

This section describes how to print MODS component reports by using option REP from the MODS : Primary Menu.

MODS Component Reports

The report types that can be obtained using option REP are as follows:

- Application definitions
- CAS components, including:
- Messages
- Help
- Menus
- Lists
- Criteria
- Tables
- Commands
- Print Services Manager (PSM) definitions
- Report Writer report definitions

Note: Help text and message text can also be printed by other means, for example, by printing help text (see page 107) in either of two different formats by using the various CAS : Help Definition menus, and by printing message text using the CAS : Message Definition Menu.

You cannot use option REP to print the following:

- Map definitions (see page 134) and their ASN.1 source code
- Panel definitions (see page 56)

Printing Components Using the REP Option

You can print MODS component reports by selecting option REP from the MODS : Primary Menu. The Report Writer : Report List panel is displayed. The panel lists the component reports that you can print. For each report, you can select the definitions to be included.

There are two report formats for each component: detail and summary. A detail report contains exact copies of the selected definitions within a component. A summary report contains single line descriptions of the selected definitions within a component.

If you need to restrict the definitions that are listed when the report is generated, you can provide test criteria on the following panel.

Note: You must know the fields (variable names) that can be specified. To determine which fields apply to each report, enter LF next to the report and make a note of the required fields names before continuing.

```
PROD-------------------- Report Writer : Report List ------------------------26
 Command ===>                                                  Scroll ===> PAGE

                                        S/=Select I=Information LF=List Fields
     Description
     Application Definition Detail
     Application Definition Summary
     Command Definition Detail
     Command Definition Summary
     Criteria Definition Detail
     Criteria Definition Summary
     Default Printer Assignment Detail
     Default Printer Assignment Summary
     Form Definition Detail
     Form Definition Summary
     Help Definition Detail
     Help Definition Summary
     List Definition Detail
     List Definition Summary
     Menu Definition Detail
     Menu Definition Summary
     Message Definition Detail
     Message Definition Summary
     Printer Definition Detail
     Printer Definition Summary
     Report Definition Detail
     Report Definition Summary
     Setup Definition Detail
     Setup Definition Summary
     Table Definition Detail
     Table Definition Summary
     **END**



  F1=Help      F2=Split     F3=Exit                  F5=Find      F6=Refresh
  F7=Backward  F8=Forward   F9=Swap
```

To select a report for printing, type **S** or **/** next to the report and press Enter. The PSM : Confirm Printer panel is displayed. If necessary, change the values in the fields, then press F6 (Conform).

The MODS : Definition Report Search panel is displayed.

```
PROD------------------- MODS : Definition Report Search -----------------------
Command ===>                                                  Function=Search


Appl ID Prefix ..  ___
Test ............. _____________________________________________________________
                   _____________________________________________________________
                   _____________________________________________________________
                   _____________________________________________________________
                   _____________________________________________________________
                   _____________________________________________________________
                   _____________________________________________________________
                   _____________________________________________________________
                   _____________________________________________________________
                   _____________________________________________________________









 F1=Help      F2=Split     F3=Exit                              F6=Action
                           F9=Swap
```

Use the panel to restrict the definitions you want to include in the report. If you do not fill in any fields, all definitions of the selected component are printed.

Note: Depending on the selected component, the panel displayed might have a different title and first field. For example, the title is MODS : Command Definition Report Search and the field is Command ID Prefix for command definitions.

Use the prefix field to restrict the definitions to be included in the report for a particular MODS component: for example, $LH indicating the CAS list definitions. Use the Test fields to further restrict the definitions by a Boolean expression. For the syntax of Boolean expressions, see the *Network Control Language Reference Guide*. The expression tests for certain values within a definition. The following example restricts the report to private list definitions belonging to USER01:

```
&$LHLDTYPE='PRIVATE' AND &$LHLDUSERID='USER01'
```

To identify the variables that hold these values, use the LF=List Fields action.

Note: If reports for help or messages are selected, the panel will contain a language code field. The language code is that of the records for which you want to produce the report. If this field is left blank, the report will be produced for English language help or messages.

After you specify the restrictions, press F6 (Action) to print the report.

If the report is on hold, you can use the **PQ**[UEUE] command to access the PSM output queue and view the report.

Chapter 4: MODS Administration

This section contains the following topics:

Maintaining Panel Libraries

A panel library contains a series of panel definitions. This section describes the facilities for:

- Defining and maintaining panel libraries
- Copying panels between libraries on different paths

More information:

Concepts and Terminology (see page 51)

Panel Libraries

Panels are stored in VSAM data sets for fast retrieval and update. A VSAM data set containing panel definitions is called a panel library. Multiple panel libraries are applied.

Individual panel definitions (see page 50) are referred to as library members. A library can be used as the sole source of panel definitions, or it can be concatenated with other libraries defined to the system. A concatenation of libraries is called a panel path. Each user can be defined to use a different path. The default path is called PANELS. The PANELLIBS parameter group specifies the libraries and the paths.

For information about panel paths, see the *Administration Guide* for your product.

Accessing the Panel Library Maintenance Facilities

The panel library maintenance facilities are accessed through the MODS : Panel Library Maintenance Menu (/MODSAD.P).

```
PROD---------------- MODS : Panel Library Maintenance Menu ---------------------
 Select Option ===>

    C   - Copy Panel(s)
    L   - Library Definitions
    X   - Exit


 'From' Library .........+ ________     ( Required C )

 'To' Library ...........+ ________     ( Required C )

 Panel Name ............. ____________ ( Blank, Full or Generic name,
                                          e.g. '*'  for all panels, or
                                               'D*' for all starting with D )

 Replace Like-Named Panels? NO_         ( Required C YES or NO )

 Copy All Matching Panels?  NO_         ( Required C YES or NO )



  F1=Help      F2=Split     F3=Exit      F4=Return
                            F9=Swap
```

The MODS : Panel Library Maintenance Menu lets you:

- Copy panel definitions from one library to another (regardless of which paths the libraries are in), as described below
- Access the MODS : Library Definition Menu to maintain library definitions

Copying Panels Between Libraries

To copy a panel library, select option C from the MODS : Panel Library Maintenance Menu. For information about the fields, see the online help.

After completing the fields on the MODS : Panel Library Maintenance Menu, press Enter. The Copy is performed as follows:

- If you specified one particular panel in the Panel Name field, or entered YES in the Copy All Matching Panels field, the Copy is performed when you press Enter.
- If you left the Panel Name blank, or specified a generic panel name and entered NO in the Copy All Matching Panels field, the MODS : Panel Copy List is displayed when you press Enter. Use this list to select the actual panels to be copied from the From Library.

Selecting Panels to Copy

The MODS : Panel Copy List displays panels in the From Library that generically match the Panel Name you specified (or all panels in the From Library, if you left Panel Name blank).

The top left of the display shows the name of the From Library (LODGED) and the To Library (TEST).

```
LODGED TO TEST------------- MODS : Panel Copy List ------------------------COPY
Command ===>                                                  Scroll ===> PAGE

                                                 S/C=Copy R=Replace B=Browse
    Name                 Created      Modified          Size  Mlev     Id
    $ADEX20MC            20-NOV-1992  20-NOV-1992 15.08   30     6    USER01
    $DDCOMPP             26-NOV-1992  26-NOV-1992 15.09   62     0    USER01
    $DDEQUEP             26-NOV-1992  26-NOV-1992 15.09   62     0    USER01
    $DDIMPEP             26-NOV-1992  26-NOV-1992 15.09   62     0    USER01
    $DDMAPEP             26-NOV-1992  26-NOV-1992 15.09   62     0    USER01
    $DDTAGEP             26-NOV-1992  26-NOV-1992 15.09   62     0    USER01
    $DDTYPEP             26-NOV-1992  26-NOV-1992 15.09   62     0    USER01
    $EASIUPD2            01-NOV-1992  25-NOV-1992 11.31   35     1    USER01
    $EDTEXTEDIT          01-NOV-1992  19-NOV-1992 12.04   88     3    USER01
```

For information about the fields and actions, press F1 (Help).

You can select panels for copying by placing an **S**, **C**, or **R** beside them and pressing Enter.

After copying all required panels, press F3 (Exit). The MODS : Panel Library Maintenance Menu is displayed with a message telling you how many panels were copied and replaced.

Maintaining Library Definitions

Option L from the MODS : Panel Library Maintenance Menu displays the MODS : Library Definition Menu.

This menu can be used to define a library temporarily (for example, if you want to copy panels to or from a panels data set defined to another system). If you require a permanently available library, you should define it by using the PANELLIBS parameter group.

To make a library available for use, the data set must be allocated to the product region, opened, and defined as a library. These steps can be performed using the ALLOC, UDBCTL OPEN, and LIBRARY commands respectively.

The MODS : Library Definition Menu, shown in the following sample, provides facilities for performing these steps, and for reversing them, without the need for you to issue any commands.

```
PROD-------------------- MODS : Library Definition Menu -----------------------
 Select Option ===>

    A   - Allocate, Open and Define Library
    O   - Open and Define Library
    D   - Define Library
    R   - Remove Library Definition
    C   - Remove Library Definition and Close
    U   - Remove Library Definition, Close and Unallocate
    X   - Exit

 Library Name ... ________

 File ID ........ ________ ( Optional A O D )

 DD Name ........ ________ ( Optional A O )

 Edit Allowed ... YES      ( Required A O D YES or NO, Default YES )

 Description .... DEFINED BY USER01_____________________ ( Required A O D )

 Dataset Name ... ____________________________________________ ( Required A )




  F1=Help      F2=Split     F3=Exit      F4=Return
                            F9=Swap
```

For information about the fields and options on this panel, press F1 (Help).

Maintaining the MODS Control File

This section discusses the facilities available for maintaining Managed Object Development Services (MODS) control files.

The facilities for control file maintenance let you do the following:

- Copy data from one control file to another (for example, from TEST to PRODUCTION).
- Move data from one control file to another.
- Delete data from a control file.
- Browse a MODS definition of a control file as if browsing the definition through the MODS primary menu.
- Reset / Clear all MODS definitions from the MODS cache table.

Control Files

A MODS control file contains component definitions for use by NCL applications. It contains both application and common components as follows:

- Application dependent components:
  - $AR—Application Definitions
  - $CR—Criteria
  - $HM—Help
  - $LD—Language Services (Presentation Elements and String Definitions)
  - $LH—Lists
  - $MH—Menus
  - $MS—Messages
  - $RW—Reports
  - $VM—Tables
- Common (non-application dependent) components:
  - $CM—Commands
  - $LD—Language Services (Language Definitions)
  - $PS—Print Services Manager (PSM) Definitions

The components contained within a control file consist of the following sub components and entities Subcomponents preceded with the square bullet can be acted upon as individual entities, while those preceded with a hyphen (–) are acted upon automatically when the parent subcomponent is affected.

$AR - Application Register

Contains the following subcomponent:

- Application Identifiers

$CR - Criteria

Contains the following subcomponent:

- Criteria Definitions

$CM - Commands

Contains the following subcomponent:

- Command Definitions

$HM - Help

Contains the following subcomponents:

- Function-level help
- Window-level help
- Field-level help

$LD - Language Services (Language Definitions)

Contains the following subcomponent:

- Language Definitions

$LD - Language Services (Presentation Elements and String Definitions)

Contains the following subcomponents:

- Presentation Elements
- String Definitions

$LH - Lists

Contains the following subcomponent:

- List Definitions

$MH - Menus

Contains the following subcomponent:

- Menu Definitions

$MS - Messages

Contains the following subcomponent:

- Message Definitions

$PS - Print Services Manager (PSM) Definitions

Contains the following subcomponents:

- Printer Definitions
- Form Definitions
- Setup Definitions
- Default Printer Assignments

$RW - Reports

Contains the following subcomponents:

- Report Definitions
  - Report Descriptions
  - Report Headers
  - Page Headers
  - Control Break Headers
  - Data Formats
  - Control Break Trailers
  - Page Trailers
  - Report Trailers
  - Sort Fields

$VM - Tables

Contains the following subcomponents:

- Table Definitions
  - Table Entries

Important! Due to the logical relationships between the records in a control file, you must not perform maintenance using REPRO or other file utilities. These utilities should not be used for any purpose other than to perform backups as they cannot take account of these logical relationships and could corrupt the control file with unpredictable consequences.

Accessing the Control File Maintenance Facilities

The control file maintenance facilities are accessed through the MODS : Entity Administration Menu (/MODSADE).

```
PROD----------------- MODS : Entity Administration Menu ---------------/MODSADE
Select Option ===>

   B   - Browse Definitions                                -
   C   - Copy Definitions                                  -
   M   - Move Definitions                                  -
   D   - Delete Definitions                                -
   AL  - View Audit Log                                    -
   X   - Exit

From File ID ..+                                             Opt ( B C M D )
     or DSN ....                                             Opt ( B C M D )
To File ID ....+                                             Opt ( C M )
   or DSN ......                                             Opt ( C M )
Entity Type ...+                                             Opt ( B C M D )
Application ID +                                             Opt ( B C M D )
Audit Log? ..... NO                                          Opt ( C M D )




  F1=Help      F2=Split     F3=Exit      F4=Return
                            F9=Swap
```

For information about the options available, press F1 (Help).

Note: The MODS : Entity Administration Menu does not support the WebCenter system. You cannot use these utilities to maintain WebCenter files.

Browsing Definitions in a Control File

Option B is used to browse definitions from a control file. This function does the following:

- Specifies a source (From) control file as either a File ID (DD name) or data set name.
- Browses selected definitions from the source file.

The From control file is specified as one of the following:

- The File ID (DD name) of a MODS Control File that has been allocated and opened.
- The data set name of another MODS Control File that must already exist. The data set is temporarily allocated and opened, components browsed, and the data set then closed and unallocated.

A list of definitions that exist in the From file is displayed. This is the Entity List and is used to select the definitions to browse. The Entity Type and Application ID fields provide filters to limit the definitions displayed. Each of these fields support field prompts.

Note: We recommend that you specify the Entity Type field. When specified, the Entity List is specific to the selected entity type and the response time is fast. If the Entity Type field is not specified, the Entity List is generic and the response time is slow.

To browse a definition, select (S) the definition from the Entity List.

Copying and Moving Definitions Between Control Files

Options C and M are used to copy or move definitions from one control file to another. These two functions do the following:

- Specify a source (From) control file as either a File ID (DD name) or data set name.
- Specify a target (To) control file as either a File ID (DD name) or data set name.
- Copy selected definitions from the source to the target
- Delete the selected components from the source Control File (Move option)
- Optionally maintain an Audit Log of the definitions that are copied or moved. See Using the Audit Log below.

The From and To control files are each specified as either:

- the File ID (DD name) of a MODS Control File that has been allocated and opened. The target (To) File ID must be opened for output use. The source (From) File ID must also be open for output operation if components are being moved but may be open input-only for the copy operation
- the data set name of another MODS Control File that must already exist. The data set is temporarily allocated and opened, components copied or moved, and the data set then closed and unallocated.

A list of definitions that exist in the From file is displayed. This is the Entity List and is used to select the definitions to copy or move. The Entity Type and Application ID fields provide filters to limit the definitions displayed. Each of these fields support field prompts.

Note: We recommend that you specify the Entity Type field. When specified, the Entity List is specific to the selected entity type and the response time is fast. If the Entity Type field is not specified, the Entity List is generic and the response time is slow.

To copy or move a definition, select one of the following definitions from the Entity List:

- The default option S (Noreplace) only moves or copies the definition if it does not exist in the target file.
- Option SR (Replace) replaces the definition if it exists in the target file.

Deleting Definitions From a Control File

Option D is used to delete definitions from a control file. This function does the following:

- Specifies a source (From) control file as either a File ID (DD name) or data set name.
- Deletes selected definitions from the source file.
- Optionally maintains an Audit Log of the definitions that are deleted. See Using the Audit Log below.

The From control file is specified as one of the following:

- The File ID (DD name) of a MODS Control File that has been allocated and opened. The File ID must be opened for output use.
- The data set name of another MODS Control File that must already exist. The data set is temporarily allocated and opened, components deleted, and the data set then closed and unallocated.

A list of definitions that exist in the From file is displayed. This is the Entity List and is used to select the definitions to delete. The Entity Type and Application ID fields provide filters to limit the definitions displayed. Each of these fields support field prompts.

Note: We recommend that you specify the Entity Type field. When specified, the Entity List is specific to the selected entity type and the response time is fast. If the Entity Type field is not specified, the Entity List is generic and the response time is slow.

To delete a definition, select (S) the definition from the Entity List.

Using the Audit Log

When definitions are copied, moved, or deleted, it may be useful to provide an audit trail of the definitions that were selected. The Audit Log provides this function.

The Audit Log is enabled by specifying YES in the Audit Log? field on the MODS : Entity Administration Menu at the same time as the From and To files are specified. When enabled, entries are written to the Audit log for the following:

- Each menu option selection showing the action type (Move, Copy, Delete) and control file details (From or To DD name, DSN)
- Each definition selected. The entry contains multiple lines, one for each physical entry and action. For example, if a Help Function is defined with three field-level help members and it is moved from one file to another, there will be entries for 'Added to Target' for the function and each field, followed by 'Deleted from Source' for the function and each field.

The Audit Log may be browsed using one of the following methods:

- Option AL on the MODS : Entity Administration Menu
- F12 (=AuditLog) on the Entity List

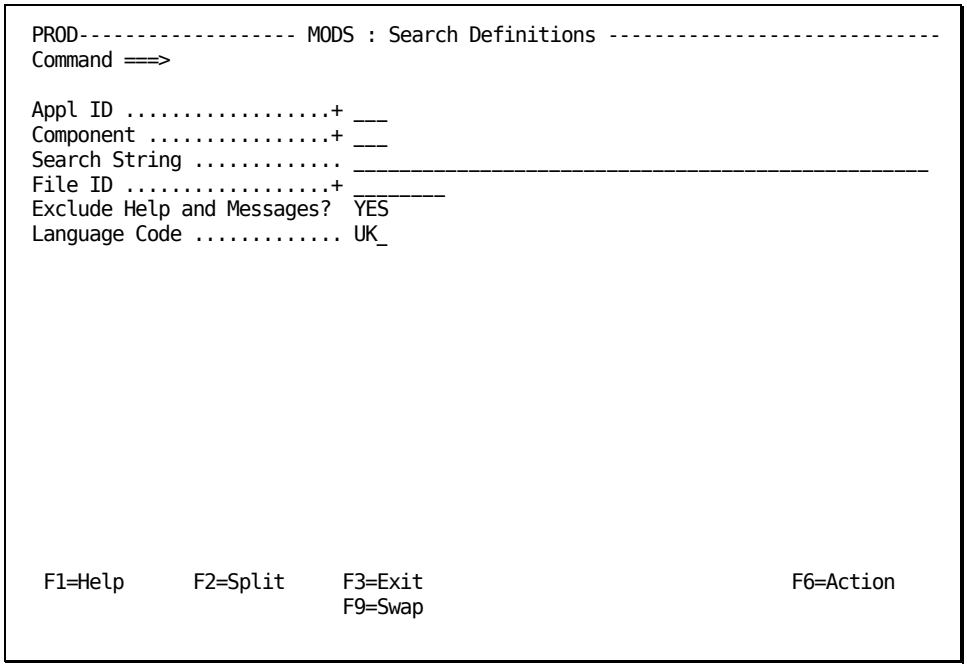
The following commands are supported when the Audit Log is browsed:

- PRINT - Prints the Audit log using PSM. The PSM: Confirm Printer panel is displayed to allow printer specification and other options.
- CLEAR - Clears all entries from the Audit Log. The Audit Log is automatically deleted on exit from the MODS : Entity Administration Menu.

Searching the Control File

Option S from the MODS : Administration Menu displays the MODS : Search Definitions panel.

This panel specifies the control file to search, and the character string to search for. Each instance of the string found by the search is listed in a PSM report.



For information about the fields on the panel, press F1 (Help).

The search results in a PSM report that lists MODS records (if found) containing the specified string of characters.

To start the search, press F6 (Action). The PSM : Confirm Printer panel is displayed.

If necessary, change the values in the fields, then press F6 (Confirm) to start the search. If the number of records to be searched is 100 or more, a panel appears on your screen to advise you of the progress of the search. When the search is complete, a message appears on your screen to advise you the success of failure of the search.

If the report is on hold, you can use the **PQ**[UEUE] command to access the PSM output queue and view the report.

Considerations

When you study your search results, consider the following:

- The search result lists only the first instance of a found string in a record.
- Record keys longer than 34 characters are truncated.
- During the search, a MODS record is accessed in chunks. The search is performed on each chunk and *not* across chunks: that is, if a string straddles two chunks, the string is *not* found. Shortening the search string can minimize this problem.

Resetting the MODS Cache

Select option RC from the MODS : Administration Menu to reset the MODS cache, the list cache, and the report cache.

This clears all the MODS definitions from the MODS cache, the list definitions from the list cache, and the report definitions from the report cache. Select option RC after you have used the MODS definition utility, so that any changes you make take effect. If you do not do this, the changes you have made may not take effect until the next time the product region starts.

Chapter 5: CAS Programming Interface ($CACALL)

This section contains the following topics:

CAS Interface Overview

The $CACALL procedure is the application program interface (API) to Common Applications Services (CAS).

When $CACALL (see page 157) is invoked, you specify an action and a class to determine which CAS function is invoked. For example, to display a menu, you invoke $CACALL with the action set to DISPLAY and the class set to MENU.

More information:

CAS Programming Interface (see page 24)

$CACALL Functions

$CACALL gives you access to the following functions:

BUILD CRITERIA

Builds and returns the specified criteria.

BUILD FKA

Builds the specified function key area.

BUILD IDTEXT

Builds lines of text containing details of the specified user ID together with the current date and time.

BUILD MESSAGE

Builds a message from the specified parameters.

DISPLAY DATA

Displays text in browse mode.

DISPLAY HELP

Displays the help text applicable to the user's current context.

DISPLAY LIST

Builds and displays a list from the specified list definition.

DISPLAY MENU

Builds and displays a menu from the specified menu definition.

DISPLAY MESSAGE

Displays the specified message.

EDIT DATA

Displays text in edit mode and returns the edited text to the caller.

EXECUTE COMMAND

Executes the specified command.

GET TENTRY

Retrieves one or all entries from a table.

LOAD COMMAND

Loads the command table.

LOAD TABLE

Loads the specified table.

LOAD PDOMAIN

Loads the specified panel domain.

NAVIGATE PDOMAIN

Determines the next panel to be displayed.

VALIDATE DATA

Validates data according to the specified parameters.

$CACALL API

The syntax for invoking $CACALL is as follows:

```
-EXEC $CACALL   OPT=ACTION
                ACTION={BUILD | DISPLAY | EDIT | EXECUTE | GET |
                           LOAD | NAVIGATE | VALIDATE}
                CLASS={COMMAND | CRITERIA | DATA | FKA |
                HELP | IDTEXT | LIST | MENU | MESSAGE |
                PDOMAIN | TABLE | TENTRY}
               [NAME='attribute1=value1 attribute 2=value2 ...
                      attributen=valuen']
               [PARMS='parm1=value1 parm2=value2 ... parmn=valuen']
```

OPT=ACTION

Specifies an optional parameter indicating the option required.

ACTION={BUILD | DISPLAY | EDIT | EXECUTE | GET | LOAD | NAVIGATE | VALIDATE}

Specifies a required parameter indicating the action to perform.

CLASS={COMMAND | CRITERIA | DATA | FKA | HELP | IDTEXT | LIST | MENU | MESSAGE | PDOMAIN | TABLE | TENTRY}

Specifies a required parameter that identifies the class on which the action is to be performed.

NAME='*attribute1=value1 attribute2=value2 ... attributen=valuen*'

Specifies an optional parameter that provides the attribute IDs and values that identify the object to process.

PARMS='*parm1=value1 parm2=value2 ... parmn=valuen*'

Specifies optional keyword parameters. These parameters are documented with the individual calls.

Note: If any value (that is, value1..*n*) specified in the NAME or PARMS operand contains embedded blanks, quotes, or double quotes, then the value and the entire operand must be quoted as described in the &ZQUOTE verb description in the *Network Control Language Reference Guide*.

Input Variables

Any required input variables are described for the individual calls.

Return Variables

Any variables set by $CACALL are described for the individual calls.

Return Codes

$CACALL sets one of the following return codes:

0

OK

8

Error

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

0

No feedback code set

1

Definition not found

2

No data found

3

Canceled by exit

4

Canceled by user

5

End of sequence

6

Unable to obtain lock

7

User not authorized

8

Processing error

9

Return requested

10

Nesting level exceeded

11

Definition not eligible for processing

Action=BUILD Class=CRITERIA

This statement builds a criteria using the criteria definition. You can supply values for the variables used in the definition. Items can be tested against the built criteria.

This statement has the following format:

```
&CONTROL SHRVARS=($CR,pref,...pref)
-EXEC $CACALL     OPT=ACTION
                  ACTION=BUILD
                  CLASS=CRITERIA
                  NAME='APPL=application id
                       [TYPE={PUBLIC | PRIVATE | FREEFORM}]
                       [USER=userid]
                       NAME=criteria name'
                  PARMS='MESSAGE=message'
```

APPL=*application id*

A required parameter (not applicable for TYPE=FREEFORM) giving the application identifier of the criteria.

TYPE={PRIVATE | PUBLIC | FREEFORM}

An optional parameter giving the type of the criteria. Valid values are as follows:

PUBLIC

Public criteria—available for general use.

PRIVATE

Private criteria—owned by a specific user ID.

FREEFORM

Free-form criteria—presents a panel for the user to enter criteria.

Note: If you do not specify TYPE or USER, the function attempts to find a PRIVATE definition owned by the invoking user ID first. If unsuccessful, the function uses a PUBLIC definition.

USER=*userid*

An optional parameter (not applicable for TYPE=PUBLIC and TYPE=FREEFORM) giving the user ID of the user owning the criteria. Default is the user ID of the user invoking the function.

NAME=*criteria name*

A required parameter (not applicable for TYPE=FREEFORM) giving the name of the criteria.

MESSAGE=*message*

A message to be displayed on the criteria panel, if one is defined.

Input Variables

Variables with prefixes as specified in SHRVARS.

Return Variables

This statement contains the following return variables:

&$CRCRIT*nnnn*

The criteria lines.

&$CRCRITTOTAL

The total number of criteria lines built (up to 9999). The value gives the number of *&$CRCRITnnnn* variables.

&$CRPANEL

Set to YES or NO indicating whether a run time panel is defined.

&$CRPANELCMD

Set to EXIT or ACTION indicating the command executed to terminate the run time panel, if a run time panel is defined, otherwise set to null.

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

1

Definition not found

3

Canceled by exit

8

Processing error

10

Nesting level exceeded

Example

The following statements build the public criteria ZPRPR001 for the application identified by the ID ZPR:

```
&CONTROL SHRVARS=($CR,ZPR)
-EXEC $CACALL     OPT=ACTION +
                  ACTION=BUILD +
                  CLASS=CRITERIA +
                  NAME='APPL=ZPR +
                          TYPE=PUBLIC +
                          NAME=ZPRPR001'
```

Action=BUILD Class=FKA

This statement builds a function key area. You can build the function key area by using a predefined one or from scratch. Your procedure should call the BUILD FKA function before displaying a panel.

This statement has the following format:

```
&CONTROL SHRVARS=($FK)
-EXEC $CACALL   OPT=ACTION
                ACTION=BUILD
                CLASS=FKA
                [NAME=' FKA=function key area']
```

FKA=*function key area*

An optional parameter identifying the predefined function key area (see page 165) on which to base the new function key area.

Input Variables

This statement contains the following input variables:

&$FK1...&$FK24

The function key actions for any or all of the keys F1 to F24. Specify NOACT to inactivate and remove a function key from the function key area.

&$FKLAB1...&$FKLAB24

The *labels* that are displayed in the function key area (the bottom two lines of the displayed screen). Each label can be up to eight characters in length. If not specified, the label for a key defaults to the first word of the key's action.

&$FKS1...&$FKS24

These variables are used internally by the function. Do not alter the values of these variables.

&$FKSLAB1...&$FKSLAB24

These variables are used internally by the function. Do not alter the values of these variables.

&$FKOPTS

This variable is used internally by the function. Do not alter the value of this variable.

Return Variables

This statement contains the following return variables:

&$FK1...&$FK24

The function key actions for keys F1 to F24.

&$FKLAB1...&$FKLAB24

The function key labels for keys F1 to F24.

&$FKS1...&$FKS24

These variables are used internally by the function. Do not alter the values of these variables.

&$FKSLAB1...&$FKSLAB24

These variables are used internally by the function. Do not alter the values of these variables.

&$FKOPTS

This variable is used internally by the function. Do not alter the value of this variable.

&$FKA1

Function key area line 1.

&$FKA2

Function key area line 2.

Note: Depending on the user's current function key area display option, &$FKA1 and &$FKA2 can contain F1 through F12, F13 through F24, or nothing.

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

8

Processing error

10

Nesting level exceeded

Example

The following statements build a function key area by using the predefined function key area $CAADD, explicitly setting F5 to HISTORY and F6 to TEXT, and disabling F7 and F11:

```
&$FK5 = HISTORY
&$FK6 = TEXT
&$FK7 = NOACT
&$FK11 = NOACT
&CONTROL SHRVARS=($FK)
-EXEC $CACALL   OPT=ACTION +
                ACTION=BUILD +
                CLASS=FKA +
                NAME='FKA=$CAADD'
```

Predefined Function Key Areas

The function keys described in this topic apply to the following predefined function key areas:

- $CAADD
- $CABRO
- $CADEL
- $CAUPD
- $IMADD
- $IMBRO
- $IMDEL
- $IMUPD

You can use the following function keys in these areas:

F1 (HELP)

Displays Help.

F2 (SPLIT)

Creates a new session.

F3 (FILE, EXIT, or CANCEL)

In $CAADD, $CAUPD, $IMADD, and $IMUPD

Files your work.

In $CABRO, $IMBRO, and $IMDEL

Exits or cancels without saving.

F4 (SAVE)

Saves your work (only on $CAADD and $CAUPD).

F5 (HISTORY)

Displays history (only on $IMBRO and $IMUPD).

F6 (TEXT)

Displays text (only on $IMADD, $IMBRO, and $IMUPD).

F7 (BACKWARD)

Displays previous page in multi-page panels.

F8 (FORWARD)

Displays next page in multi-page panels.

F9 (SWAP)

Toggles between multiple sessions.

F10 (CANCEL)

Cancels without saving (only on $IMADD and $IMUPD).

F12 (CANCEL)

Cancels without saving.

Action=BUILD Class=IDTEXT

This statement builds the ID text that contains information about a user.

This statement has the following format:

```
&CONTROL SHRVARS=($CAID)
-EXEC $CACALL      OPT=ACTION
                   ACTION=BUILD
                   CLASS=IDTEXT
                   [NAME='USER=userid']
                   [PARMS='[BORDER={YES | NO}]
                           [LENGTH={80 | length}]
                           [DATE=date]
                           [TIME=time] ']
```

USER=*userid*

An optional parameter giving the ID of the user whose ID text is to be built. The default is the current user ID.

BORDER={YES | NO}

An optional parameter specifying whether there are borders at the top and bottom of the built ID text. The default is YES.

LENGTH={80 | *length*}

An optional parameter specifying the length of the horizontal border lines if BORDER=YES. The default is 80, and the range is 1 through 255.

DATE=*date*

An optional parameter specifying the date (in DATE3 format) to be included in the built ID text. The default is the current date.

TIME=*time*

An optional parameter specifying the time (in *hh.mm.ss* format) to be included in the built ID text. The default is the current time.

Input Variables

This statement has no input variables.

Return Variables

This statement contains the following return variables:

&$CAIDTX*n*

Contains the built ID text strings.

&$CAIDTXTOT

Contains the number of ID text strings built. The value gives the number of &$CAIDTX*n* variables.

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

8

Processing error

10

Nesting level exceeded

Example

The following statements build the ID text for the current user. The text contains the current date and time, and has 80-character borders at the top and bottom.

```
&CONTROL SHRVARS=($CAID)
-EXEC $CACALL   OPT=ACTION +
                ACTION=BUILD +
                CLASS=IDTEXT +
                NAME='USER=&USERID' +
                PARMS='BORDER=YES +
                       LENGTH=80'
```

Action=BUILD Class=MESSAGE

This statement builds a message using a message definition. The BUILD MESSAGE function returns the built message in &SYSMSG.

This statement has the following format:

```
&CONTROL NOSHRVARS
-EXEC $CACALL    OPT=ACTION
                 ACTION=BUILD
                 CLASS=MESSAGE
                 NAME='MESSAGE=message id'
                 PARMS='PROCNAME=procedure name
                        [P1=parameter1... P10=parameter10]'
```

MESSAGE=*message id*

A required parameter giving the identifier of the message to be built.

PROCNAME=*procedure name*

A required parameter giving the name of the procedure that requested the message to be built.

P1=*parameter1*...P10=*parameter10*

Optional parameters providing data values to be substituted into the message definition.

Input Variables

This statement has no input variables.

Return Variables

This statement contains the following return variables:

&SYSMSG

System message. Contains either:

- The requested message (for return code 0)
- The error message (for return code 8)

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

1

Definition not found

8

Processing error

10

Nesting level exceeded

Example

The message definition for message SAMP01 is as follows:

```
VSAM ERROR ON FILE ~P1 RC=~P2 FDBK=~P3
```

Use the following statements to build the SAMP01 message:

```
&CONTROL NOSHRVARS
-EXEC $CACALL    OPT=ACTION
                 ACTION=BUILD
                 CLASS=MESSAGE
                 NAME='MESSAGE=SAMP01'
                 PARMS='PROCNAME=&0
                        P1=&FILEID
                        P2=&FILERC
                        P3=&VSAMFDBK'
```

&0 contains the name of your procedure; &FILEID, &FILERC, and &VSAMFDBK contain, respectively, the values for ~P1, ~P2, and ~P3 in the message definition.

Action=DISPLAY Class=DATA

This statement displays text in browse mode. The DISPLAY DATA function provides text browsing facilities for up to 9999 lines of text. When a user requests help, help text associated with the specified application ID and function name is displayed.

This statement has the following format:

```
&CONTROL SHRVARS=($ED)
-EXEC $CACAL        OPT=ACTION
                    ACTION=DISPLAY
                    CLASS=DATA
                    PARMS='APPL=application id
                           FUNC=function name
                           [TITLE=title]
                           [LINECNT={0 | n}]
                           [LINELEN={256 | n}]
                           [LINENUM={YES | NO}]
                           [INDENT={0 | n}]
                           [TOPEXIT={YES | NO}]
                           [MESSAGE=message]
                           [USERFUNC=function]
                           [LINETOP={1 | n}]
                           [CANCEL={YES | NO}]
                           [LMARGIN={1 | n}]
                           [RMARGIN=n]'
```

APPL=*application id*

A required parameter giving the application identifier.

FUNC=*function name*

A required parameter indicating the function being performed.

TITLE=*title*

An optional parameter giving the title to be displayed at the top of the panel. The default is CAS : Text Editor.

LINECNT={0 | *n*}

A required parameter giving the total number of lines of text. The default is 0, and the range is 0 through 9999.

LINELEN={256 | *n*}

An optional parameter indicating the maximum line length. The default is 256, and the range is 1 through 256.

LINENUM={YES | NO}

An optional parameter that indicates whether to display line numbers. The default is YES.

INDENT={0 | *n*}

An optional parameter that indicates the number of positions to indent the text lines. The default is 0, and the range is 0 through 256 minus the value in LINELEN.

TOPEXIT={YES | NO}

An optional parameter indicating whether executing the BACKWARD command—when the display is positioned at the top of the text—ends the text display and causes control to be returned to the caller. The default is NO.

MESSAGE=*message*

An optional parameter specifying a message to be displayed on line 3 of the panel, on initial entry.

USERFUNC=*function*

An optional parameter specifying the logical function being performed (for example, Browse). If specified, the function is displayed on line 4 of the panel as Function=*function*. *function* must not be longer than eight characters.

LINETOP={1 | *n}*

An optional parameter specifying the number of the line to be displayed as the first line of text, on initial entry. The default is 1, and the range is 1 to the value in LINECNT.

CANCEL={YES | NO}

An optional parameter specifying whether the CANCEL command is supported. The default is NO.

LMARGIN={1 | *n*}

The left margin (in characters) used by the text editor. The default is 1, and the range is 1 through the value in LINELEN.

RMARGIN=*n*

The right margin (in characters) used by the text editor. The default is the lesser of the value in LINELEN and the logical screen width. The range is the value of LMARGIN plus 20 to the value in LINELEN.

Input Variables

This statement contains the following input variables:

&$EDFK1...&$EDFK24

The function key actions for any or all of the keys F1 to F24. Specify NOACT to inactivate and remove a function key from the function key area.

&$EDFKLAB1...&$EDFKLAB24

The *labels* that are displayed in the function key area (the bottom two lines of the displayed screen). Each label can be up to eight characters in length. If not specified, the label for a key defaults to the first word of the key's action.

&$EDCOMMENT*n*

Contains the comment lines to be displayed (up to nine) above the text lines.

&$EDLINE*nnnn*

Contains the text lines to be displayed (up to 9999).

&$EDEXITCMDS=*command1*, *command2*, ...

Specifies commands that are accepted as valid exit commands. That is, if a specified command is executed, control is returned to the caller.

Return Variables

This statement contains the following return variables:

&$EDCMDEXIT

The command which was entered by the user to exit (normally EXIT).

&$EDCMDPARMS

The parameters for the exit command in &$EDCMDEXIT.

&$EDCOMMAND

The entire contents of the exit command including parameters.

&$EDLINETOP

The line number at the top of the display on exit.

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

8

Processing error

10

Nesting level exceeded

Example

The following statements display the contents of &$EDLINE1 and &$EDLINE2 in two lines on a panel titled Problem Text:

```
&$EDLINE1 = &STR Problem text line 1
&$EDLINE2 = &STR Problem text line 2
&CONTROL SHRVARS=($ED)
-EXEC $CACALL   OPT=ACTION +
                ACTION=DISPLAY +
                CLASS=DATA +
                PARMS='APPL=ZPR +
                       FUNC=BROWSE +
                       TITLE="Problem Text" +
                       LINECNT=2'
```

Action=DISPLAY Class=HELP

This statement displays online help. You determine the application, the function, the field, and the window on which the user requires help, and supply the values to the DISPLAY HELP function to display the appropriate help. If the required help has not been added, the function displays the next more general help in the order field-level help, window-level help, function-level help, and application-level help.

This statement has the following format:

```
&CONTROL NOSHRVARS
-EXEC $CACALL   OPT=ACTION
                ACTION=DISPLAY
                CLASS=HELP
                NAME='APPL=application id
                       [FUNC=function name]
                       [FIELD=field name]'
                [PARMS='CROW=cursor row
                        CCOL=cursor column'
                        MODE={VIEW | BROWSE}]
```

APPL=*application id*

A required parameter giving the identifier of the application for which help is to be displayed.

FUNC=*function name*

An optional parameter (required if either the FIELD=, CROW= or CCOL= keywords are specified) giving the name of the function for which help is to be displayed.

FIELD=*field name*

An optional parameter giving the name of the field for which help is to be displayed.

CROW=*cursor row*

An optional parameter giving the cursor row position when help was requested. This defaults to the value of &CURSROW.

CCOL=*cursor column*

An optional parameter giving the cursor column position when help was requested. This defaults to the value of &CURSCOL.

Input Variables

This statement has no input variables.

Return Variables

This statement contains the following return variables:

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

8

Processing error

10

Nesting level exceeded

Example

The following statements display the field-level help for the ZPRSTATUS field for the UPDATE function in the application identified by the identifier ZPR if the help is available; otherwise, the next more general help is displayed:

```
&CONTROL NOSHRVARS
-EXEC $CACALL    OPT=ACTION +
                 ACTION=DISPLAY +
                 CLASS=HELP +
                 NAME='APPL=ZPR +
                       FUNC=UPDATE +
                       FIELD=ZPRSTATUS'
```

Action=DISPLAY Class=LIST

This statement displays a list.

This statement has the following format:

```
&CONTROL SHRVARS=($LH)
-EXEC $CACALL       OPT=ACTION
                    ACTION=DISPLAY
                    CLASS=LIST
                    NAME='APPL=application id
                           [TYPE={PUBLIC | PRIVATE}]
                           [USER=userid]
                           NAME=list name'
                    [PARMS='[FORMAT={ACTION | MSELECT |
                                     SSELECT | NUMBERED}]
                           [CRITERIA=criteria]
                           [MAXSEL={9999 | nnnn}]
                           [AUTOSEL={YES | NO}]']
```

APPL=*application id*

A required parameter giving the application identifier of the list.

TYPE={PRIVATE | PUBLIC}

An optional parameter giving the type of list. Valid values are as follows:

PUBLIC

Public list—available for general use

PRIVATE

Private list—owned by a specific user ID

Note: If you do not specify TYPE or USER, the function attempts to find a PRIVATE list owned by the invoking user ID first. If unsuccessful, the function uses a PUBLIC list.

USER=*userid*

An optional parameter (if TYPE is not PUBLIC) giving the user ID of the user owning the list. Default is the user ID of the user invoking the function.

NAME=*list name*

A required parameter giving the name of the list.

FORMAT={*ACTION* | MSELECT | SSELECT| NUMBERED}

An optional parameter indicating the format in which the list is to be displayed. Valid values are as follows:

ACTION

Action list

MSELECT

Multiple selection list

SSELECT

Single selection list

NUMBERED

Numbered list

The default is ACTION.

CRITERIA=*criteria*

A criteria statement used to select items to go in the list. The format must conform to that required by the service procedure specified in the list definition. If specified, this criteria overrides any criteria specified in the list definition, and no &$LHCRIT*nnnn* variables can be set.

MAXSEL={9999 | *nnnn*}

The maximum number of items that can be chosen from a list in format MSELECT (multiple selection list). The default is 9999, and the range is 1 through 9999.

AUTOSEL={YES | *NO*}

Determines whether an entry in a list is automatically selected if it is the only entry in the list. The default is NO.

Input Variables

This statement contains the following input variables:

&$LHCRIT*nnnn*

Criteria variables (up to 9999 variables can be given). Cannot be specified if the **CRITERIA** operand is specified. The format must conform to that required by the service procedure specified in the list definition.

Return Variables

This statement contains the following return variables:

&$LHENTTOTAL

The total number of entries selected (up to 9999).

&$LHENTID*nnnn*

The IDs of the list entries selected.

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

1

Definition not found

2

No data found

3

Canceled by exit

4

Canceled by user

8

Processing error

9

Return requested

10

Nesting level exceeded

11

Definition not eligible for processing

Example

The following statements display action list ZPRPRALL in the application identified by the ID ZPR. A private list is displayed in preference to a public list.

```
&CONTROL SHRVARS=($LH)
-EXEC $CACALL    OPT=ACTION +
                 ACTION=DISPLAY +
                 CLASS=LIST +
                 NAME='?APPL=ZPR +
                 NAME=ZPRPRALL' +
                 PARMS='FORMAT=ACTION'
```

Action=DISPLAY Class=MENU

This statement displays a menu.

This statement has the following format:

```
&CONTROL NOSHRVARS
-EXEC $CACALL    OPT=ACTION
                 ACTION=DISPLAY
                 CLASS=MENU
                 NAME='APPL=application id
                        MENU=menu id'
                 [PARMS='PSKIP=xxx.xxx']
```

APPL=*application id*

A required parameter giving the application identifier.

MENU=*menu id*

A required parameter giving the menu identifier.

PSKIP=*xxx.xxx*

Panel skip setting.

Input Variables

This statement has no input variables.

Return Variables

This statement contains the following return variables:

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

8

Processing error

9

Return requested

10

Nesting level exceeded

Example

The following statements display menu 030 in the application identified by the ID ZPR:

```
&CONTROL NOSHRVARS
-EXEC $CACALL    OPT=ACTION +
                 ACTION=DISPLAY +
                 CLASS=MENU +
                 NAME='APPL=ZPR +
                       MENU=030'
```

Action=DISPLAY Class=MESSAGE

This statement displays a message and its associated explanation, system action and user action.

This statement has the following format:

```
&CONTROL NOSHRVARS
-EXEC $CACALL    OPT=ACTION
                 ACTION=DISPLAY
                 CLASS=MESSAGE
                 NAME='MESSAGE=message id'
```

MESSAGE=*message id*

A required parameter giving the identifier of the message to be displayed.

Input Variables

This statement has no input variables.

Return Variables

This statement contains the following return variables:

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

1

Definition not found

8

Processing error

10

Nesting level exceeded

Example

The following statements display information about message PV7014:

```
&CONTROL NOSHRVARS
-EXEC $CACALL   OPT=ACTION +
                ACTION=DISPLAY +
                CLASS=MESSAGE +
                NAME='MESSAGE=PV7014'
```

Action=EDIT Class=DATA

This statement displays text in edit mode. The EDIT DATA function provides editing facilities for up to 9999 lines of text, each of up to 256 characters in length. When a user requests help, help text associated with the specified application ID and function name is displayed.

This statement has the following format:

```
&CONTROL SHRVARS=($ED)
-EXEC $CACALL      OPT=ACTION
                   ACTION=EDIT
                   CLASS=DATA
                   PARMS='APPL=application id
                          FUNC=function name
                          [TITLE=title]
                          [LINECNT={0 | n}]
                          [LINELEN={256 | n}]
                          [MAXLINES={9999 | n}]
                          [EDTLINES=n]
                          [TOPEXIT={YES | NO}]
                          [MESSAGE=message]
                          [USERFUNC=function]
                          [LINETOP={1 | n}]
                          [LMARGIN={1 | n}]
                          [RMARGIN=n]'
```

APPL=*application id*

A required parameter giving the application identifier.

FUNC=*function name*

A required parameter indicating the function being performed.

TITLE=*title*

An optional parameter giving the title to be displayed at the top of the panel. The default is CAS : Text Editor.

LINECNT={0 | *n*}

A required parameter giving the number of lines of text. The default is 0, and the range is 0 through 9999.

LINELEN={256 | *n*}

An optional parameter indicating the maximum line length. The default is 256, and the range is 1 through 256.

MAXLINES={9999 | *n*}

An optional parameter indicating the maximum number of lines of text that can exist. The default is 9999, and the range is 1 through 9999.

EDTLINES=*n*

An optional parameter giving the number of lines that can be edited. If specified, the first *n* lines can be edited. If not specified, the number of lines that can be edited defaults to the value of the LINECNT parameter—that is, all lines can be edited. The range is 0 to the value in LINECNT.

TOPEXIT={YES | NO}

An optional parameter indicating whether executing the BACKWARD command—when the display is positioned at the top of the text—ends the text display and causes control to be returned to the caller. The default is NO.

MESSAGE=*message*

An optional parameter specifying a message to be displayed on line 3 of the panel, on initial entry.

USERFUNC=*function*

An optional parameter specifying the logical function being performed (for example, Update). If specified, the function is displayed on line 4 of the panel as Function=*function*. *function* must not be longer than eight characters.

LINETOP={1 | *n*}

An optional parameter specifying the number of the line to be displayed as the first line of text, on initial entry. The default is 1, and the range is 1 through the value in LINECNT.

LMARGIN={1 | *n*}

The left margin (in characters) used by the text editor. The default is 1, and the range is 1 through the value in LINELEN.

RMARGIN=*n*

The right margin (in characters) used by the text editor. The default is the lesser of the value in LINELEN and the logical screen width. The range is the value of LMARGIN plus 20 to the value in LINELEN.

Input Variables

This statement contains the following input variables:

&$EDFK1...&$EDFK24

The function key *actions* for any or all of the keys F1 to F24. Specify NOACT to inactivate and remove a function key from the function key area.

&$EDFKLAB1...&$EDFKLAB24

The *labels* that are displayed in the function key area (the bottom two lines of the displayed screen). Each label can be up to eight characters in length. If not specified, the label for a key defaults to the first word of the key's action.

&$EDCOMMENT*n*

Contains the comment lines to be displayed (up to nine) above the text lines.

&$EDLINE*nnnn*

Contains the text lines to be displayed for edit (up to 9999).

&$EDEXITCMDS=*command1*, *command2*, ...

Specifies commands which are accepted as valid exit commands. That is, if a specified command is executed, control is returned to the caller.

Return Variables

This statement contains the following return variables:

&$EDLINE*nnnn*

The edited text lines.

&$EDLINECNT

The number of edited text lines. The value gives the number of &$EDLINE*nnnn* variables.

&$EDMODIFIED

Whether any lines were modified (YES or NO).

&$EDCMDEXIT

The command that was entered by the user to exit (normally FILE or SAVE).

&$EDCMDPARMS

The parameters for the exit command in &$EDCMDEXIT.

&$EDCOMMAND

The entire contents of the exit command including parameters.

&$EDLINETOP

The line number at the top of the display on exit.

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

4

Canceled by user

8

Processing error

10

Nesting level exceeded

Example

The following statements display the contents of &$EDLINE1 and &$EDLINE2 in two lines on a panel titled Problem Text for editing:

```
&$EDLINE1 = &STR Problem text line 1
&$EDLINE2 = &STR Problem text line 2
&CONTROL SHRVARS=($ED)
-EXEC $CACALL   OPT=ACTION +
                ACTION=EDIT +
                CLASS=DATA +
                PARMS='APPL=ZPR +
                       FUNC=UPDATE +
                       TITLE="Problem Text" +
                       LINECNT=2'
```

Action=EXECUTE Class=COMMAND

Executes a command if it is known. When a user requests help, help text associated with the specified application ID, function name, and message ID (if specified) is displayed.

This statement has the following format:

```
&CONTROL SHRVARS=($FK,$CM)
-EXEC $CACALL      OPT=ACTION
                   ACTION=EXECUTE
                   CLASS=COMMAND
                   [NAME='COMMAND=command']
                   [PARMS='APPL=application id]
                            [FUNC=function name]
                            [KEY=key]
                            [MESSAGE=messageid]
                            [CURSFLD=cursorfield value] ']
```

APPL=*application id*

An optional parameter giving the application identifier (required for the HELP command).

FUNC=*function name*

An optional parameter indicating the function currently being performed (required for the HELP command).

COMMAND=*command*

An optional parameter giving the command to be executed (required if the KEY keyword is not used). If you use both the COMMAND and KEY keywords, and a command is assigned to the function key specified in KEY, the command assigned to the function key takes precedence.

The following commands (or any command defined using the CAS Command Definition facility) can be specified:

CMD

Invokes the Command Entry facility.

EX[EC]

Executes an NCL procedure.

DISC[ONN]

Disconnects the session.

HELP

Display help information.

KEYS

Switches FKA display.

LOCK

Locks the session.

NOTEPAD

Displays the CAS Notepad.

PASSWORD

Sets the user password.

PQ[UEUE]

Displays the PSM print queue.

PSKIP

Skips panels.

RETRIEVE

Retrieves the last command.

SPLIT

Splits the window.

START

Starts an NCL procedure.

SWAP

Swaps the window.

WHERE

Displays NCL procedure details.

KEY=*key*

An optional parameter giving the last function key (&INKEY value) that was pressed (required if the COMMAND keyword is not used). If you use both the COMMAND and KEY keywords, and a command is assigned to the function key specified in KEY, the command assigned to the function key takes precedence.

MESSAGE=*message id*

An optional parameter giving the message identifier of the currently displayed message (for example, when a user requires help on a message).

CURSFLD=*cursorfield value*

An optional parameter containing the value of the field in which the cursor is located.

Input Variables

This statement has no input variables.

Return Variables

This statement contains the following return variables:

&$FK1...&$FK24

The function key actions for keys F1 to F24.

&$FKLAB1...&$FKLAB24

The function key labels for keys F1 to F24.

&$FKA1

Function key area line 1.

&$FKA2

Function key area line 2.

&$CMDI

Set to **Y** or **N** to indicate if an action was performed. For example, a command is not executed if it is not in the command table.

&$CMDS

Set to **C**, **K**, or **N** indicating the source of the action (Command, Key, or None).

&$CMD

The first word of the action.

&$CMDPARMS

The remaining operands or parameters of the action.

&$CMDR

The retrieved command (when COMMAND=RETRIEVE).

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

7

User not authorized

8

Processing error

9

Return requested

10

Nesting level exceeded

Example

The following statements execute the command in variable &COMMAND or assigned to the key indicated in variable &INKEY; the APPL, FUNC, and MESSAGE keywords are specified and are referenced if the command is HELP.

```
&CONTROL SHRVARS=($FK,$CM)
-EXEC $CACALL    OPT=ACTION +
                 ACTION=EXECUTE +
                 CLASS=COMMAND +
                 NAME='COMMAND=&COMMAND' +
                 PARMS='APPL=ZPR +
                        FUNC=UPDATE +
                        KEY=&INKEY +
                        MESSAGE=PV7014'
```

Action=LOAD Class=COMMAND

Loads the command table into memory (normally during system initialization). You must perform the LOAD COMMAND function before you can execute a defined command.

This statement has the following format:

```
&CONTROL NOSHRVARS
-EXEC $CACALL    OPT=ACTION
                 ACTION=LOAD
                 CLASS=COMMAND
```

Input Variables

This statement has no input variables.

Return Variables

This statement contains the following return variables:

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

2

No data found

6

Unable to obtain lock

8

Processing error

10

Nesting level exceeded

Example

The following statements load the command table:

```
&CONTROL NOSHRVARS
-EXEC $CACALL    OPT=ACTION +
                 ACTION=LOAD +
                 CLASS=COMMAND
```

Action=GET Class=TENTRY

Retrieves one or all entries from a table.

This statement has the following format:

```
&CONTROL SHRVARS=($VM)
-EXEC $CACALL       OPT=ACTION
                    ACTION=GET
                    CLASS=TENTRY
                    NAME='APPL=application id
                            FIELD=fieldName
                            [VALUE=fullValue]'
                    [PARMS='ACTIVE={YES | NO | ANY}']
```

APPL=*application id*

A required parameter giving the identifier of the application.

FIELD=*fieldName*

A required parameter giving the name of the table.

VALUE=*fullValue*

An optional parameter giving the full value of the entry to be retrieved. If omitted, all entries in the table are returned.

ACTIVE={YES | NO | ANY}

An optional parameter that indicates which entries are to be retrieved from the table if the VALUE parameter is omitted.

YES

Only active table entries are to be retrieved. This is the default.

NO

Only inactive table entries are to be retrieved.

ANY

All table entries are to be retrieved.

Input Variables

This statement has no input variables.

Return Variables

This statement contains the following return variables:

&SYSMSG

System message. Contains the error message (for return code 8).

If a full value is supplied, the following variables are returned:

$VMABBR

The abbreviated value of the table entry.

$VMFULL

The full value of the table entry.

$VMDESC

The description of the table entry.

$VMTEXT*n*

The text fields of the table entry.

If a full value is not supplied, the following variables are returned:

$VMTOTAL

The total number of entries in the table.

$VMABBR*n*

The abbreviated value for each table entry.

$VMFULL*n*

The full value for each table entry.

$VMDESC*n*

The description of each table entry.

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

2

No data found

8

Processing error

10

Nesting level exceeded

Example

The following statements retrieve the entry $NDSYS from the Report Writer table:

```
&CONTROL SHRVARS
-EXEC $CACALL    OPT=ACTION +
                 ACTION=GET +
                 CLASS=TENTRY +
                 NAME='APPL=$RW +
                       FIELD=APPL +
                       VALUE=$NDSYS'
```

Action=LOAD Class=PDOMAIN

Loads panel domains (normally during the initialization of an application). You must perform the LOAD PDOMAIN function before you can use the NAVIGATE PDOMAIN function.

This statement has the following format:

```
&CONTROL NOSHRVARS
-EXEC $CACALL    OPT=ACTION
                 ACTION=LOAD
                 CLASS=PDOMAIN
                 NAME='APPL=application id
                        [TYPE={PUBLIC | PRIVATE}]
                        [USER=userid]
                        [NAME=domain name]'
```

APPL=*application id*

A required parameter giving the identifier of the application.

TYPE={PUBLIC | PRIVATE}

An optional parameter giving the type of panel domain. Valid values are as follows:

PUBLIC

Public domain—available for general use.

PRIVATE

Private domain—owned by a specific user ID.

USER=*userid*

An optional parameter (if TYPE is not PUBLIC) giving the user ID of the user owning the panel domains.

NAME=*domain name*

An optional parameter giving the name of the panel domain.

Note: The LOAD PDOMAIN function loads all panel domains matching the supplied operands.

Input Variables

This statement has no input variables.

Return Variables

This statement contains the following return variables:

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

8

Processing error

10

Nesting level exceeded

Example

The following statements load the panel domains for the application identified by ID ZPR:

```
&CONTROL NOSHRVARS
-EXEC $CACALL    OPT=ACTION +
                 ACTION=LOAD +
                 CLASS=PDOMAIN +
                 NAME='APPL=ZPR'
```

Action=LOAD Class=TABLE

This function loads the tables for an application (typically during system initialization). You perform the LOAD TABLE function before you can use the tables to validate data.

This statement has the following format:

```
&CONTROL NOSHRVARS
-EXEC $CACALL OPT=ACTION
              ACTION=LOAD
              CLASS=TABLE
              NAME='APPL=application_id
                   [FIELD=field_name]'
             [PARMS='RELOAD={YES | NO}']
```

APPL=*application_id*

Specifies the identifier of the application.

FIELD=*field_name*

(Optional) Specifies the name of the table if you only want to load one table.

Default: All tables for the specified application are loaded.

RELOAD={YES | NO}

(Optional) Specifies whether you want to reload previously loaded tables for the application.

Default: NO

Input Variables

This statement has no input variables.

Return Variables

This statement contains the following return variables:

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

8

Processing error

10

Nesting level exceeded

Example

The following statements load the tables for the application identified by ID ZPR. Any tables already loaded are reloaded.

```
&CONTROL NOSHRVARS
-EXEC $CACALL      OPT=ACTION +
                   ACTION=LOAD +
                   CLASS=TABLE +
                   NAME='APPL=ZPR' +
                   PARMS='RELOAD=YES'
```

Action=NAVIGATE Class=PDOMAIN

Determines the next panel to be displayed. You determine the panel by specifying the direction of movement (backward or forward) and the number of panels to skip, or by specifying the name of the panel domain element.

This statement has the following format:

```
&CONTROL SHRVARS=($PV,pref,...pref)
-EXEC $CACALL      OPT=ACTION
                   ACTION=NAVIGATE
                   CLASS=PDOMAIN
                   NAME='APPL=application id
                          [TYPE={PUBLIC | PRIVATE}]
                          [USER=userid]
                          NAME=domain name'
                   [PARMS='[CURRENT=element name]
                           [NEXT={element name / *}]
                           [SKIP={1 | n}]
                           [LOAD={YES | NO}]']
```

APPL=*application id*

A required parameter giving the application identifier of the panel domain.

TYPE={PRIVATE | PUBLIC}

An optional parameter giving the type of panel domain. Valid values are as follows:

PUBLIC

Public domain—available for general use.

PRIVATE

Private domain—owned by a specific user ID.

Note: If you do not specify TYPE or USER, the function attempts to find the panel domain definition owned by the invoking user ID. If unsuccessful, the function uses a public panel domain definition.

USER=*userid*

An optional parameter (if TYPE=PRIVATE) giving the user ID of the user owning the PRIVATE panel domain. Default is the user ID of the user invoking the function.

NAME=*domain name*

A required parameter giving the name of the panel domain.

CURRENT=*element name*

An optional parameter indicating the current element. If this keyword is blank, the current element are assumed to be ##TOP## (if DIR=FORWARD) or ##END## (if DIR=BACKWARD).

Note: DIR or SKIP is mutually exclusive to NEXT. If you do not specify DIR, NEXT, and SKIP, the default is DIR=FORWARD and SKIP=1.

DIR={*FORWARD* | BACKWARD}

An optional parameter indicating the direction in which navigation is to occur. The default is FORWARD.

NEXT={*element name* | *}

An optional parameter indicating the next element to progress to. If blank, the next element is determined based upon the defined paths (from the current element) and the direction of travel. If an element name is specified, navigation occurs to this element if it is currently eligible. If '*****' is specified, a pick list of all eligible elements in the domain (excluding the element specified in the CURRENT= keyword) is presented.

SKIP={1 | *n*}

An optional parameter indicating the number of times to perform a panel navigation. The default is 1, and the range is 1 through 9999. Use this parameter when the user specifies a numeric or MAX scroll amount.

LOAD={*YES* | NO}

An optional parameter determining whether to load a domain if it is not currently loaded. The default is YES.

Input Variables

This statement contains variables with prefixes as specified in SHRVARS.

&$PVNEST*nnn*

Tracking variables used by CAS to maintain the last *nnn* elements visited in the current direction (up to 999 elements). These variables are supplied by CAS. Do not modify these variables.

&$PVCRIT*nnnn*

Criteria variables containing the criteria used to specify eligible panel domain element types.

&$PVCRITTOTAL

The total number of criteria variables. The range is 0 through 9999.

Return Variables

This statement contains the following return variables:

&$PVELEMENT

The name of the next element to progress to.

&$PVPANEL

The name of the panel associated with the next element.

&$PVDESC

The description of the next element.

&$PVHELP

The name of the function-level help associated with the next element.

&$PVTYPE

The type (PANEL or TEXT) of the next element.

&$PVTITLE

The title of the next element (if type is TEXT).

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

1

Definition not found

4

Canceled by user

5

End of sequence

6

Unable to obtain lock

8

Processing error

10

Nesting level exceeded

Example

The following statements identify the next panel for display following the panel in &$PVELEMENT. The panel domain is ZPRPROB in the application identified by ID ZPR. A private panel domain is used in preference to a public panel domain.

```
&CONTROL SHRVARS=($PV,ZPR)
-EXEC $CACALL    OPT=ACTION +
                 ACTION=NAVIGATE +
                 CLASS=PDOMAIN +
                 NAME='APPL=ZPR +
                 NAME=ZPRPROB' +
                 PARMS='CURRENT=&$PVELEMENT'
```

Action=VALIDATE Class=DATA

This statement validates data against predefined values or edit rules.

This statement has the following format:

```
&CONTROL SHRVARS=(field name,$VM)
-EXEC $CACALL     OPT=ACTION
                  ACTION=VALIDATE
                  CLASS=DATA
                  PARMS='[APPL=application id]
                         FIELD=field name
                         [LIST=n]
                         [TEXT={YES | NO}]
                         [ACTIVE={YES | NO | ANY}]
                         DESC=description
                         EDITS=edit number 1,edit number 2,...
                         [IMSYS=system name]
                         [PROMPT={YES | NO}]'
```

APPL=*application id*

An optional (required only for EDITS=17) parameter giving the application identifier.

FIELD=*field name*

A required parameter giving the name of the field that is to be validated. This field name must also be included in the SHRVAR**=** operand.

LIST=*n*

An optional parameter (required only for EDITS=18) that gives the number of valid values supplied using the &$VMFULL*nnn* variables. The &$VMABBR*nnn* and &$VMDESC*nnn* variables can also be defined. The range is 1 through 999.

TEXT={YES | *NO*}

An optional parameter (applicable only for EDITS=17) that indicates whether the text fields associated with the selected or supplied field value are to be returned. The default is NO.

ACTIVE={*YES* | NO | ANY}

An optional parameter (applicable only for EDITS=17) that indicates which entries from the table defined for the field in FIELD are to be considered as valid values. Valid values are as follows:

YES

Only active table entries are to be considered as valid values. This is the default.

NO

Only inactive table entries are to be considered as valid values.

ANY

All table entries are to be considered as valid values.

DESC=*description*

A required parameter (optional for EDITS=17) that gives the description of the field for use in selection list headings, help panels and error messages.

EDITS=*edit number1,edit number 2,...*

A required parameter giving the edit numbers that indicate how the field is to be validated (multiple numbers can be specified, separated by commas). Valid values are as follows:

1(*l*)

YES/NO—The field can only contain YES (or a string beginning with Y) or NO (or a string beginning with N). If the length parameter (*l*) is coded, the field is set to that length. For example, 1(1) returns Y or N only.

2

Unsigned Integer—The field can only contain a positive number with no sign or decimal point.

3

Date—The field can only contain the date format *dd-mmm-yyyy* or a shorthand date format (see page 355).

4

Time—The field can only contain the time format *hh.mm* or a shorthand time format (see page 355).

5

Hexadecimal—The field can only contain values in expanded hexadecimal.

6

Signed Numeric—The field can contain a signed or unsigned number with no decimal point.

7

Real Numbers—The field can only contain real numbers. This includes (signed and unsigned) integers, numbers containing a decimal point, and numbers expressed in scientific notation within the range -1E-70 to +1E+70.

8

Name—The field can only contain numbers, alphabetic characters (upper case), and the characters @, #, and $. The first character in the field cannot be a number.

9(*l*:*h*)

Range—The field can only contain a value within the defined range, where *l* is the lowest valid value and *h* is the highest. If *l* is omitted, the value is only checked for being less than or equal to *h*; if *h* is omitted, the value is only checked for being greater than or equal to *l*, for example:

- 9(1:10) accepts values from 1 to 10
- 9(:3) accepts values less than or equal to 3
- 9(12:) accepts values greater than or equal to 12

10

Alphanumeric—The field can only contain numbers and alphabetic characters. Lower case characters are converted to upper case.

11

Alphabetic—The field can only contain alphabetic characters. Lower case characters are converted to upper case.

12

National—The field can only contain numbers, alphabetic characters, and the characters @, # and $. Lower case characters are converted to upper case.

13

Data Set Name—The field can only contain a valid data set name, with no quotes. Lower case characters are converted to upper case.

14

No Embedded Blanks—The field cannot contain any imbedded blanks.

15(*l*:*h*)

Length Of The Field—The length of the field must be within the defined range, where *l* is the lowest possible length and *h* is the highest. If *l* is omitted, the length is only checked for being less than or equal to *h*; if *h* is omitted, the length is only checked for being greater than or equal to *l*. The default value for *l* is 1.

16

Scroll Amounts—The field value must be one of the following valid scroll amounts:

- MAX (or a string beginning with M)
- CSR (or a string beginning with C)
- DATA (or a string beginning with D)
- PAGE (or a string beginning with P)
- HALF (or a string beginning with H)
- A number between 1 and 99999 (if the string starts with a number, it is truncated at the first non-numeric character)

17

Table—The field is validated against the values held in the table defined for the field in FIELD.

18

List—The field is validated against the values given in &$VMFULL*nnn*, &$VMABBR*nnn*, and &$VMDESC*nnn*.

19

NCL Keyword—The field must not contain an NCL keyword.

20

Time—The field can only contain the time format *hh*.*mm*.*ss* or a shorthand time format (see page 355).

21

Hexadecimal Characters—The field can only contain hexadecimal characters, that is, 0–9 and A–F.

22

IP Address—The address must be a valid IP address, in the format A.B.C.D, where each of A, B, C, and D have a valid range of 0 to 255.

23

Time—The field must contain the time format *hh*:*mm*:*ss* or a shorthand time format (see page 355). The returned string is in the format *hh:mm*:*ss*.

24

Time—The field can only contain the time format *hh*:*mm* or a shorthand time format (see page 355). The returned string is in the format *hh:mm*.

IMSYS=*system name*

An optional parameter (required for EDITS=17 where the table type is IMFLD or IMREC) that gives the CA SOLVE:InfoMaster name.

PROMPT={YES | NO}

An optional parameter (applicable only for EDITS=17 or 18) which determines whether a list of valid value is to be displayed if the field value contains a question mark character (?). Specifying NO lets values be defined which contain a question mark (?), and also lets validation occur from a non-full screen environment.

Input Variables

This statement contains the following input variables:

field name

The data to be validated

&$VMFULL*nnn*

The *full values* of the values (up to 20 characters) against which the field is to be validated (applicable only for EDITS=18). Up to 999 *full values* can be specified.

&$VMABBR*nnn*

The *abbreviations* of the values (up to eight characters) against which the field is to be validated (applicable only for EDITS=18). Up to 999 *abbreviations* can be specified.

&$VMDESC*nnn*

The *descriptions* of the values (up to 38 characters) against which the field is to be validated (applicable only for EDITS=18). Up to 999 *descriptions* can be specified.

Return Variables

This statement contains the following return variables:

field name

The validated data.

&$VMSELABBR

The abbreviated value of the selected value (applicable only for EDITS=17 or 18).

&$VMSELDESC

The description of the selected value (applicable only for EDITS=17 or 18).

&$VMTEXT1..10

The text associated with the value (applicable only for EDITS=17 and returned only if TEXT=YES).

&SYSMSG

System message. Contains the error message (for return code 8).

Feedback Codes

If a return code of 8 is set, then additional information is available as one of the following feedback codes, set in &$CAFDBK:

8

Processing error

10

Nesting level exceeded

Example

The following statements validate the &PCODE field against edit numbers 2 (unsigned integer) and 9 (range). The value of &PCODE must be an unsigned integer in the range 1 through 10. In the example, because &PCODE has a value of 12, an error message is returned in &SYSMSG.

```
&PCODE = 12
&DESC = &STR Priority Code
&CONTROL SHRVARS=(PCODE,$VM)
-EXEC $CACALL   OPT=ACTION
                ACTION=VALIDATE
                CLASS=DATA
                PARMS='APPL=$ML
                       FIELD=PCODE
                       DESC=&DESC
                       EDITS=2,9(1:10)'
```

Chapter 6: Menu Service Procedure Interface

This section contains the following topics:

Menu Service Procedures

A menu service procedure interface is available for Common Application Services (CAS) menus.

Menu service procedures are installation written NCL procedures that are available for performing specialized processing on menus.

Menu service procedures provide a convenient means of extending the functionality of menus.

Menu service procedures are optional. You specify the menu service procedure when you define a menu.

A menu service procedure performs site-specific menu processing at defined points during invocation and processing of a menu:

- On initial entry to the menu
- After a user makes a selection but before acting on it
- After a user acts on a selection
- Prior to exiting the menu
- When a user enters an unrecognized command
- If the menu times out when an INWAIT value is specified

You write your own menu service procedures by using the variables described in the following sections.

The variables can be divided into two groups as follows:

- The first group of variables provide information to the procedure and are *not* modifiable (read-only). In particular, check the value of &$MHOPT to determine the stage of menu processing for you to implement the required special processing.
- The second group of variables are modifiable and enable the procedure to return information to the system.

You can use your own variables in the menu service procedure. However, do not use variable names that start with #MH. The menu service procedure is called with NOSHRVARS=(#MH). After you specify your own variables, these variables are persistent through the different stages of menu processing. That is, if you specify &A = *variable_value*, the value of this variable is available on each subsequent call to the procedure.

Menu Service Procedure Statements

This section contains descriptions for the menu service procedure statements.

$MHOPT=ENTRY Statement

This statement indicates initial entry into a menu. When the value of &$MHOPT is ENTRY, you can use the menu service procedure to perform any special processing that is required before the display of the menu.

Read-Only Variables

This statement contains the following read-only variables:

&$MHOPT

This variable is set to ENTRY.

&$MHAPPLID

This variable is set to the ID of the application to which the menu belongs.

&$MHMENUNUM

This variable is set to the menu number.

Modifiable Variables

This statement contains the following modifiable variables:

&$MHAUTHMAP

Use this variable to control the access to menu options. This variable can be set to a map (15 characters in length) with each character corresponding to a menu option. Valid values for each character are as follows:

1

Access to the menu option is allowed.

0

Access to the menu option is not allowed. If this variable is set, it is used as the menu map for the user. If it is not set, access is allowed to all menu options.

&$MHINWAIT

Use this variable to set a time-out when the menu is not receiving any input. This variable can be set to an INWAIT value (between 0 and 86400 seconds) that is used when displaying the menu. If this value is zero, the menu does not time out. If the INWAIT time expires, the service procedure is called with &$MHOPT set to TIMEOUT.

&$MHDOUBLESPC

Use this variable to specify whether a blank line should be inserted before each input line on the menu. If this variable is set to YES (the default), a blank line is inserted before each input field line on the menu. If this variable is set to NO, input field lines are displayed as defined for the menu.

&$MHINPUTJUST

Use this variable to specify the value of the JUST keyword on the #FLD panel statement. Valid values are LEFT, RIGHT, ASIS, and CENTER. The default is LEFT.

&$MHINPUTMODE

Use this variable to specify the value of the MODE keyword on the #FLD panel statement. Valid values are SBCS and MIXED. The default is SBCS.

&$MHCOMMAND

Use this variable to enable the processing of commands that are not known to the system. If this value is YES, the service procedure is called with &$MHOPT set to COMMAND.

&$MHFK1..24

Use these variables to override the standard function key actions.

&$MHFKLAB1..24

Use these variables to override the standard function key labels.

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the menu service procedure to one of the following return codes:

0

Proceed; display the menu

8

Terminate the menu

&$MHOPT=SELECT Statement

This statement indicates that the user has selected a menu option. When the value of &$MHOPT is SELECT, you can use the menu service procedure to perform any special processing that is required upon a menu option being selected.

Read-Only Variables

This statement contains the following read-only variables:

&$MHOPT

This variable is set to SELECT.

&$MHAPPLID

This variable is set to the ID of the application to which the menu belongs.

&$MHMENUNUM

This variable is set to the menu number.

&$MHSELECT

This variable is set to the option the user selects from the menu.

&$MHSELECTNUM

This variable is set to the relative numeric position of the selected option on the menu, as defined on the CAS : Menu Options panel.

Modifiable Variables

This statement contains the following modifiable variables:

&$MHERRLIST

Use this variable to return the names of the input fields that contain incorrect values to the system. This variable can be set to the input fields that are in error in the form *fieldname1, fieldname2...fieldnameN.* The fields are placed in error status on the panel, and the cursor is located in the first field that is in error.

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the menu service procedure to one of the following return codes:

0

Proceed; execute the action

8

Error; redisplay the menu

&$MHOPT=RETURN Statement

This statement indicates that the action for a selected menu option has completed. When the value of &$MHOPT is RETURN, you can use the menu service procedure to perform any special processing that is required on the completion of the action.

Read-Only Variables

This statement contains the following read-only variables:

&$MHOPT

The variable is set to RETURN.

&$MHAPPLID

This variable is set to the ID of the application to which the menu belongs.

&$MHMENUNUM

This variable is set to the menu number.

&$MHSELECT

This variable is set to the menu option for which the action has completed.

Modifiable Variables

This statement contains the following modifiable variables:

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the menu service procedure to one of the following return codes:

0

Proceed; display the menu

1

Terminate the menu and return to the previous primary menu

4

If &SYSMSG is set, redisplay the menu with the value of &SYSMSG as the error message; if &SYSMSG is not set, terminate the menu and return to the previous primary menu

All other return codes are treated as errors, and &SYSMSG must be set. The menu is redisplayed with the value of &SYSMSG as the error message.

&$MHOPT=EXIT Statement

This statement indicates that a menu is terminating. When the value of &$MHOPT is EXIT, you can use the menu service procedure to perform any special processing that is required on the termination of the menu.

Read-Only Variables

This statement contains the following read-only variables:

&$MHOPT

This variable is set to EXIT.

&$MHAPPLID

This variable is set to the ID of the application to which the menu belongs.

&$MHMENUNUM

This variable is set to the menu number.

Modifiable Variables

This statement contains the following modifiable variables:

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the menu service procedure to one of the following return codes:

0

Proceed; exit from the menu

16

Do not exit; redisplay the menu

&$MHOPT=COMMAND

This statement indicates that a command that is not known to the system has been issued from the menu. When the value of &$MHOPT is COMMAND, you can use the menu service procedure to perform any special processing that is required to process the command.

Read-Only Variables

This statement contains the following read-only variables:

&$MHOPT

This variable is set to COMMAND.

&$MHAPPLID

This variable is set to the ID of the application to which the menu belongs.

&$MHMENUNUM

This variable is set to the menu number.

&$MHCMD

This variable is set to the command.

&$MHCMDPARMS

This variable is set to the command parameters.

Modifiable Variables

This statement contains the following modifiable variables:

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the menu service procedure to one of the following return codes:

0

Command processed successfully

4

Command in error; if &SYSMSG is not set, the error message MH0003 is displayed

&$MHOPT=TIMEOUT Statement

This statement indicates that the menu has timed out. When the value of &$MHOPT is TIMEOUT, you can use the menu service procedure to perform any special processing that is required when the menu times out.

Read-Only Variables

This statement contains the following read-only variables:

&$MHOPT

This variable is set to TIMEOUT.

&$MHAPPLID

This variable is set to the ID of the application to which the menu belongs.

&$MHMENUNUM

This variable is set to the menu number.

Modifiable Variables

This statement contains the following modifiable variables:

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the menu service procedure to one of the following return codes:

0

Redisplay the menu

1

Terminate the menu, and return to the previous primary menu

4

Terminate the menu

Chapter 7: List Service Procedure Interface

This section contains the following topics:

List Service Procedures

A list service procedure interface is available for Common Application Services (CAS) lists.

List service procedures are installation-written NCL procedures that are available for performing specialized processing on lists.

A list service procedure must be present for the CAS list component to work. You specify the list service procedure when you define a list.

CAS executes the list service procedure at the following processing points:

- On initial entry to the list—&$LHOPT=INIT
- To retrieve a list entry—&$LHOPT=GET
- To process an action against a selected entry—&$LHOPT=ACTION
- To process an add request—&$LHOPT=ADD
- To process the ALL command—&$LHOPT=ALL
- Prior to terminating the list—&$LHOPT=TERM
- To process a command not recognized by CAS—&$LHOPT=COMMAND

You write your own list service procedures by using the variables described in the following sections.

The variables can be divided into the following groups:

- The first group of variables provide information to the procedure and are *not* modifiable (read-only). In particular, check the value of &$LHOPT to determine the stage of list processing for you to implement the required special processing.
- The second group of variables are modifiable and enable the procedure to return information to the system. Some of these variables are set with values already, but the procedure can change those values.

You can use your own variables in the list exit procedure. However, do not use variable names that start with #LH. The list service procedure is called with NOSHRVARS=(#LH). Once you specify your own variables, these variables are persistent through the different stages of list processing. That is, if you specify &A = *variable_value*, the value of this variable is available on each subsequent call to the procedure.

List Service Procedure Statements

This sections contains descriptions of the List Service Procedure statements.

&$LHOPT=ACTION Statement

This statement indicates that a list entry has been selected for actioning. When the value of &$LHOPT is ACTION, you can use the list service procedure to perform any special processing that is required to apply an action on the list entry.

Read-Only Variables

This statement contains the following read-only variables:

&$LHOPT

This variable is set to ACTION.

&$LHLISTFMT

This variable is set to indicate the format of the list. This variable is set to one of the following values:

ACTION

An action list

MSELECT

A multiple select list

SSELECT

A single select list

NUMBERED

A numbered list

&$LHAPPLID

This variable is set to the ID of the application to which the list belongs.

&$LHLISTTYPE

This variable is set to the type of the list. This variable is set to one of the following:

PUBLIC

The list is a public list.

PRIVATE

The list is a private list.

&$LHLISTUSER

This variable is set to the user ID of the user who owns the list, if it is a private list.

&$LHLISTNAME

This variable is set to the name of the list.

&$LHDESC

This variable is set to the description of the list.

&$LHDATASRC

This variable is set to the data source defined for the list.

&$LHCRITTOTAL

This variable is set to the number of &$LHCRIT variables that contain criteria. The value of this variable is in the range 0 to 9999.

&$LHCRIT*n*

These variables are set to the criteria which the service procedure uses to determine the entries to be included in the list, if variable &$LHCRITTOTAL is greater than zero; *n* is in the range 1 to the value of &$LHCRITTOTAL.

&$LHSORT

This variable is set to the sort expression defined for the list.

&$LHFMTFLD*n*

These variables are set to the names of the real fields defined for the list; *n* is in the range 1 to the number of real fields defined.

&$LHATB*

These variables are set to the panel attributes that can be used to set the intensity, color, and highlighting for data within an entry line.

&$LHACTION

This variable is set to the mnemonic entered by the user next to the entry to be actioned.

&$LHENTID

This variable is set to the identifier of the entry selected by the user for actioning.

&$LHENTD

This variable is set to the data associated with the entry selected by the user for actioning.

Modifiable Variables

This statement contains the following modifiable variables:

&$LHSPD*c*

These variables contain service procedure data as set by the caller of $CACALL; *c* is between 0 and 5 alphanumeric and/or national characters. These variables are never set or cleared by the system and must be completely managed by your installation-written NCL procedures.

&$LHREBUILD

Use this variable to indicate whether the entry line is to be rebuilt. This variable can be set to YES or NO. If set to YES, all the attributes to be used to rebuild the entry line must be set to the appropriate values. The default is NO. This variable is cleared by the system before calling the service procedure for ACTION processing.

&$LHENTMSG

Use this variable to set a message that is to overlay the entry line. The offset and length used are those defined in the list definition. This variable is cleared by the system before calling the service procedure for ACTION processing.

&$LHREFRESH

Use this variable to indicate whether the list is to be refreshed. This variable can be set to YES or NO. If set to YES, variables &$LHREBUILD and &$LHENTMSG are ignored. The default is NO. This variable is cleared by the system before calling the service procedure for ACTION processing.

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the list service procedure to one of the following return codes:

0

Continue processing

4

Request denied or error; redisplay the list with the mnemonic set in error

8

An error occurred; terminate the list

&$LHOPT=ADD Statement

This statement indicates that the user has executed the ADD command. When the value of &$LHOPT is ADD, you can use the list service procedure to perform any special processing that is required to process the ADD command.

Read-Only Variables

This statement contains the following read-only variables:

&$LHOPT

This variable is set to ADD.

&$LHLISTFMT

This variable is set to indicate the format of the list. This variable is set to one of the following values:

ACTION

An action list

MSELECT

A multiple select list

SSELECT

A single select list

NUMBERED

A numbered list

&$LHAPPLID

This variable is set to the ID of the application to which the list belongs.

&$LHLISTTYPE

This variable is set to the type of the list. This variable is set to one of the following:

PUBLIC

The list is a public list.

PRIVATE

The list is a private list.

&$LHLISTUSER

This variable is set to the user ID of the user who owns the list, if it is a private list.

&$LHLISTNAME

This variable is set to the name of the list.

&$LHDESC

This variable is set to the description of the list.

&$LHDATASRC

This variable is set to the data source defined for the list.

&$LHCRITTOTAL

This variable is set to the number of &$LHCRIT variables that contain criteria. The value of this variable is in the range 0 through 9999.

&$LHCRIT*n*

These variables are set to the criteria which the service procedure uses to determine the entries to include in the list, if variable &$LHCRITTOTAL is greater than zero; *n* is in the range 1 to the value of &$LHCRITTOTAL.

&$LHSORT

This variable is set to the sort expression defined for the list.

&$LHFMTFLD*n*

These variables are set to the names of the real fields defined for the list; *n* is in the range 1 to the number of real fields defined.

&$LHATB*

These variables are set to the panel attributes (see page 359) that can be used to set the intensity, color, and highlighting for data within an entry line.

Modifiable Variables

This statement contains the following modifiable variables:

&$LHSPD*c*

These variables contain service procedure data as set by the caller of $CACALL; *c* is between 0 and 5 alphanumeric and/or national characters. These variables are never set or cleared by the system and must be completely managed by your installation-written NCL procedures.

&$LHREFRESH

Use this variable to indicate whether the list is to be refreshed. This variable can be set to YES or NO. The default is NO. This variable is cleared by the system before calling the service procedure for ADD processing.

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the list service procedure to one of the following return codes:

0

Continue processing

4

Request denied or error; redisplay the list with the error message

8

An error occurred; terminate the list

&$LHOPT=ALL Statement

This statement indicates that the user has executed the ALL command. When the value of &$LHOPT is ALL, you can use the list service procedure to perform any special processing that is required for ALL command processing.

Read-Only Variables

This statement contains the following read-only variables:

&$LHOPT

This variable is set to ALL.

&$LHLISTFMT

This variable is set to indicate the format of the list. This variable is set to one of the following values:

ACTION

An action list

MSELECT

A multiple select list

SSELECT

A single select list

NUMBERED

A numbered list

&$LHAPPLID

This variable is set to the ID of the application to which the list belongs.

&$LHLISTTYPE

This variable is set to the type of the list. This variable is set to one of the following:

PUBLIC

The list is a public list.

PRIVATE

The list is a private list.

&$LHLISTUSER

This variable is set to the user ID of the user who owns the list, if it is a private list.

&$LHLISTNAME

This variable is set to the name of the list.

&$LHDESC

This variable is set to the description of the list.

&$LHDATASRC

This variable is set to the data source defined for the list.

&$LHCRITTOTAL

This variable is set to the number of &$LHCRIT variables that contain criteria. The value of this variable is in the range 0 to 9999.

&$LHCRIT*n*

These variables are set to the criteria which the service procedure uses to determine the entries to be included in the list, if variable &$LHCRITTOTAL is greater than zero; *n* is in the range 1 to the value of &$LHCRITTOTAL.

&$LHSORT

This variable is set to the sort expression defined for the list.

&$LHFMTFLD*n*

These variables are set to the names of the real fields defined for the list; *n* is in the range 1 to the number of real fields defined.

&$LHATB*

These variables are set to the panel attributes (see page 359) that can be used to set the intensity, color, and highlighting for data within an entry line.

&$LHACTION

This variable is set to the mnemonic entered by the user as the parameter on the ALL command.

Modifiable Variables

This statement contains the following modifiable variables:

&$LHSPD*c*

These variables contain service procedure data as set by the caller of $CACALL; *c* is between 0 and 5 alphanumeric and/or national characters. These variables are never set or cleared by the system and must be completely managed by your installation-written NCL procedures.

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the list service procedure to one of the following return codes:

0

Continue processing

4

Request denied or error; redisplay the list with the error message

8

An error occurred; terminate the list

&$LHOPT=COMMAND Statement

This statement indicates that the user has executed a command not recognized by the system. When the value of &$LHOPT is COMMAND, you can use the list service procedure to perform any special processing that is required to process the command.

Read-Only Variables

This statement contains the following read-only variables:

&$LHOPT

This variable is set to COMMAND.

&$LHLISTFMT

This variable is set to indicate the format of the list. This variable is set to one of the following values:

ACTION

An action list

MSELECT

A multiple select list

SSELECT

A single select list

NUMBERED

A numbered list

&$LHAPPLID

This variable is set to the ID of the application to which the list belongs.

&$LHLISTTYPE

This variable is set to the type of the list. This variable is set to one of the following:

PUBLIC

The list is a public list.

PRIVATE

The list is a private list.

&$LHLISTUSER

This variable is set to the user ID of the user who owns the list, if it is a private list.

&$LHLISTNAME

This variable is set to the name of the list.

&$LHDESC

This variable is set to the description of the list.

&$LHDATASRC

This variable is set to the data source defined for the list.

&$LHCRITTOTAL

This variable is set to the number of &$LHCRIT variables that contain criteria. The value of this variable is in the range 0 through 9999.

&$LHCRIT*n*

These variables are set to the criteria which the service procedure uses to determine the entries to include in the list, if variable &$LHCRITTOTAL is greater than zero; *n* is in the range 1 to the value of &$LHCRITTOTAL.

&$LHSORT

This variable is set to the sort expression defined for the list.

&$LHFMTFLD*n*

These variables are set to the names of the real fields defined for the list; *n* is in the range 1 to the number of real fields defined.

&$LHATB*

These variables are set to the panel attributes (see page 359) that can be used to set the intensity, color, and highlighting for data within an entry line.

Modifiable Variables

This statement contains the following modifiable variables:

&$LHSPD*c*

These variables contain service procedure data as set by the caller of $CACALL; *c* is between 0 and 5 alphanumeric and/or national characters. These variables are never set or cleared by the system and must be completely managed by your installation-written NCL procedures.

&$LHCOMMAND

This variable is set to the entered command. This variable can be set to a value that is to be displayed in the Command field on the list when &$LHSETCMDFLD is set to YES.

&$LHSETCMDFLD

Use this variable to indicate whether the value of &$LHCOMMAND is to be set in the Command field on the list when &RETCODE is set to 4. This variable can be set to YES or NO. The default is NO.

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the list service procedure to one of the following return codes:

0

Continue processing

4

Invalid command or error; redisplay the list with the error message

8

An error occurred; terminate the list

&$LHOPT=GET Statement

This statement indicates that a list entry is to be retrieved. When the value of &$LHOPT is GET, you can use the list service procedure to perform any special processing that is required to get a list entry.

Read-Only Variables

This statement contains the following read-only variables:

&$LHOPT

This variable is set to GET.

&$LHLISTFMT

This variable is set to indicate the format of the list. This variable is set to one of the following values:

ACTION

An action list

MSELECT

A multiple select list

SSELECT

A single select list

NUMBERED

A numbered list

&$LHAPPLID

This variable is set to the ID of the application to which the list belongs.

&$LHLISTTYPE

This variable is set to the type of the displayed list. This variable is set to one of the following:

PUBLIC

The list is a public list.

PRIVATE

The list is a private list.

&$LHLISTUSER

This variable is set to the user ID of the user who owns the list, if it is a private list.

&$LHLISTNAME

This variable is set to the name of the list.

&$LHDESC

This variable is set to the description of the list.

&$LHDATASRC

This variable is set to the data source defined for the list.

&$LHCRITTOTAL

This variable is set to the number of &$LHCRIT variables that contain criteria. The value of this variable is in the range 0 to 9999.

&$LHCRIT*n*

These variables are set to the criteria which the service procedure uses to determine the entries to be included in the list, if variable &$LHCRITTOTAL is greater than zero; *n* is in the range 1 to the value of &$LHCRITTOTAL.

&$LHSORT

This variable is set to the sort expression defined for the list.

&$LHFMTFLD*n*

These variables are set to the names of the real fields defined for the list; *n* is in the range 1 to the number of real fields defined.

&$LHATB*

These variables are set to the panel attributes (see page 359) that can be used to set the intensity, color, and highlighting for data within an entry line.

&$LHTIMEOUT

This variable is set to indicate whether the list panel timed out. This variable is set to one of the following values:

NO

The list panel did not time out.

YES

The list panel timed out.

&$LHRFRESHCMD

This variable is set to indicate whether the REFRESH command was executed by the user. This variable is set to one of the following values:

NO

The REFRESH command was not executed.

YES

The REFRESH command was executed.

&$LHDIRECTION

This variable is set to indicate the retrieval direction. This variable is set to one of the following values:

FWD

Get an entry in a forward direction.

BKWD

Get an entry in a backward direction.

LOCATE

Get the entry with an identifier that matches the value set in &$LHSKIP, if not found, get the first entry which has an identifier less than the value set in &$LHSKIP.

If the value of this variable is null, get the entry identified in &$LHENTID.

&$LHGETALL

This variable is set to indicate whether all entries for the list are retrieved during initialization processing for the list. This variable is set to one of the following values:

NO

Entries are retrieved as required for display on the list.

YES

All entries for the list are retrieved during initialization processing for the list.

&$LHSUPPRESS

This variable is set to indicate whether entries will be suppressed from the list by this procedure when getting entries, by the list exit procedure during entry processing, or by both procedures.

&$LHSKIP

This variable is set to indicate the number of entries to be skipped if &$LHDIRECTION is set to FWD or BKWD. If &$LHDIRECTION is set to LOCATE, it is set to the identifier of the entry to be located, that is, the locate string entered by the user.

&$LHGETFWD#

This variable is set to indicate the number of entries that are to be retrieved in a forward direction until the list is displayed.

Modifiable Variables

This statement contains the following modifiable variables:

&$LHSPD*c*

These variables contain service procedure data as set by the caller of $CACALL; *c* is between 0 and 5 alphanumeric and/or national characters. These variables are never set or cleared by the system and must be completely managed by your installation-written NCL procedures.

&$LHENTTOTAL

Use this variable to specify the number of list entries to get for display. This variable can be set to the total number of entries to be displayed on the list when &$LHOPT is set to GET and &$LHRFRESHCMD or &$LHREFRESH is set to YES. The value of this variable is displayed in the top right corner of the list.

&$LHENTID

This variable is set to the identifier of the entry from which to start reading. If &$LHDIRECTION and &$LHSKIP are null, get the entry that has an identifier that matches the value of this variable and if not found return NOT FOUND condition. If this variable is null and &$LHDIRECTION is set to FWD or BKWD, get the first or last entry respectively. You must set this variable to the identifier of the entry returned.

&$LHENTD

This variable is set to the data associated with the entry identified in &$LHENTID. This variable can be set to the data to be associated with the entry returned.

&$LHENTPOS

Use this variable to specify the position of the entry being returned within the list. If &$LHSUPPRESS is not set to YES and the Get all Entries field in the list definition is set to NO, the value of this variable is displayed in the top right corner of the list in front of the total.

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the list service procedure to one of the following return codes:

0

Continue processing

4

Entry not found

8

An error occurred; terminate the list

&$LHOPT=INIT Statement

This statement indicates initialization of a list. When the value of &$LHOPT is INIT, you can use the list service procedure to perform any special processing that is required before the display of the list.

Read-Only Variables

This statement contains the following read-only variables:

&$LHOPT

This variable is set to INIT.

&$LHLISTFMT

This variable is set to indicate the format of the list. This variable is set to one of the following values:

ACTION

An action list

MSELECT

A multiple select list

SSELECT

A single select list

NUMBERED

A numbered list

&$LHAPPLID

This variable is set to the ID of the application to which the list belongs.

&$LHLISTTYPE

This variable is set to the type of the list. This variable is set to one of the following:

PUBLIC

The list is a public list.

PRIVATE

The list is a private list.

&$LHLISTUSER

This variable is set to the user ID of the user who owns the list, if it is a private list.

&$LHLISTNAME

This variable is set to the name of the list.

&$LHDESC

This variable is set to the description of the list.

&$LHDATASRC

This variable is set to the data source defined for the list.

&$LHCRITTOTAL

This variable is set to the number of &$LHCRIT variables that contain criteria. The value of this variable is in the range 0 through 9999.

&$LHCRIT*n*

These variables are set to the criteria which the service procedure uses to determine the entries to include in the list, if variable &$LHCRITTOTAL is greater than zero; *n* is in the range 1 to the value of &$LHCRITTOTAL.

&$LHSORT

This variable is set to the sort expression defined for the list.

&$LHFMTFLD*n*

These variables are set to the names of the real fields defined for the list; *n* is in the range 1 to the number of real fields defined.

&$LHATB*

These variables are set to the panel attributes (see page 359) that can be used to set the intensity, color, and highlighting for data within an entry line.

Modifiable Variables

This statement contains the following modifiable variables:

&$LHSPD*c*

These variables contain service procedure data as set by the caller of $CACALL; *c* is between 0 and 5 alphanumeric and/or national characters. These variables are never set or cleared by the system and must be completely managed by your installation-written NCL procedures.

&$LHSUPPRESS

This variable is set to indicate whether entries will be suppressed from the list by this procedure when getting entries, by the list exit procedure during entry processing, or by both procedures. Set this variable to YES if you want to suppress entries; otherwise, set this variable to NO (the default).

&$LHADDALLOW

This variable is set to the Add Allowed setting defined for the list. It is set to YES if the list supports the ADD command, and it is set to NO if the list does not support the ADD command. The value of this variable can be modified to YES or NO. This variable is ignored if the list is a single or multiple select, or numbered list.

&$LHGETALL

This variable is set to the Get All Entries setting defined for the list. It is set to YES if all entries for the list are retrieved during initialization processing for the list. It is set to NO if entries are retrieved as required for display on the list. The value of this variable can be modified to YES or NO.

&$LHBUILDBKWD

This variable is set to indicate whether entries for the list can be built in a backwards direction when processing the BACKWARD command—this means that the entries are retrieved in a backwards direction. Set this variable to YES if entries can be built in a backwards direction; otherwise, set this variable to NO. The default value is YES.

&$LHACTIONS

This variable must be set to the mnemonics of the supported actions in the format *mmm,mmm,mmm....*

&$LHCONFIRM

This variable can be set to the mnemonics of the actions that are to be confirmed before the action occurs, in the format *mmm,mmm,mmm....*

&$LHENTTOTAL

Use this variable to specify the total number of entries to be displayed on the list. The value of this variable is displayed in the top right corner of the list if &$LHSUPPRESS is not set to YES.

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the list service procedure to one of the following return codes:

0

Continue processing

4

No entries found; terminate the list

8

An error occurred; terminate the list

&$LHOPT=TERM Statement

This statement indicates the termination of a list. When the value of &$LHOPT is TERM, you can use the list service procedure to perform any special processing that is required for terminating the list.

Read-Only Variables

This statement contains the following read-only variables:

&$LHOPT

This variable is set to TERM.

&$LHLISTFMT

This variable is set to indicate the format of the list. This variable is set to one of the following values:

ACTION

An action list

MSELECT

A multiple select list

SSELECT

A single select list

NUMBERED

A numbered list

&$LHAPPLID

This variable is set to the ID of the application to which the list belongs.

&$LHLISTTYPE

This variable is set to the type of the list. This variable is set to one of the following:

PUBLIC

The list is a public list.

PRIVATE

The list is a private list.

&$LHLISTUSER

This variable is set to the user ID of the user who owns the list, if it is a private list.

&$LHLISTNAME

This variable is set to the name of the list.

&$LHDESC

This variable is set to the description of the list.

&$LHDATASRC

This variable is set to the data source defined for the list.

&$LHCRITTOTAL

This variable is set to the number of &$LHCRIT variables that contain criteria. The value of this variable is in the range 0 to 9999.

&$LHCRIT*n*

These variables are set to the criteria which the service procedure uses to determine the entries to be included in the list, if variable &$LHCRITTOTAL is greater than zero; *n* is in the range 1 to the value of &$LHCRITTOTAL.

&$LHFMTFLD*n*

These variables are set to the names of the real fields defined for the list; *n* is in the range 1 to the number of real fields defined.

&$LHSORT

This variable is set to the sort expression defined for the list.

&$LHATB*

These variables are set to the panel attributes (see page 359) that can be used to set the intensity, color, and highlighting for data within an entry line.

Modifiable Variables

This statement contains the following modifiable variables:

&$LHSPD*c*

These variables contain service procedure data as set by the caller of $CACALL; *c* is between 0 and 5 alphanumeric and/or national characters. These variables are never set or cleared by the system and must be completely managed by your installation-written NCL procedures.

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the list service procedure to one of the following return codes:

0

Continue processing

8

An error occurred

Chapter 8: List Exit Procedure Interface

This section contains the following topics:

List Exit Procedures

A list exit procedure interface is available for Common Application Services (CAS) lists.

List exit procedures are installation-written NCL procedures that are available for lists to perform specialized processing.

List exit procedures are optional. You specify the list exit procedure when you define a list (see page 25). CAS executes the list exit procedure at three processing points: initialization, entry processing, and termination.

You write your own list exit procedures by using the variables described in the following sections.

The variables can be divided into two groups as follows:

- The first group of variables provide information to the procedure and are *not* modifiable. In particular, check the value of &$LHOPT to determine the stage of list processing for you to implement the required special processing.
- The second group of variables are modifiable and enable the procedure to return information to the system. Some of these variables are set with values already, but the procedure can change those values.

You can use your own variables in the list exit procedure. However, do not use variable names that start with #LH. The list exit procedure is called with NOSHRVARS=(#LH). Once you specify your own variables, these variables are persistent through the different stages of list processing. That is, if you specify &A = *variable_value*, the value of this variable is available on each subsequent call to the procedure.

List Exit Procedure Statements

This section contains descriptions of the List Exit Procedure statements.

&$LHOPT=INIT Statement

This statement indicates initialization of a list. When the value of &$LHOPT is INIT, you can use the list exit procedure to perform any special processing that is required to initialize the list.

Read-Only Variables

This statement contains the following read-only variables:

&$LHOPT

This variable is set to INIT.

&$LHLISTFMT

This variable is set to indicate the format of the list. This variable is set to one of the following values:

ACTION

An action list

MSELECT

A multiple select list

SSELECT

A single select list

NUMBERED

A numbered list

&$LHAPPLID

This variable is set to the ID of the application to which the list belongs.

&$LHLISTTYPE

This variable is set to the type of the list. This variable is set to one of the following:

PUBLIC

The list is a public list.

PRIVATE

The list is a private list.

&$LHLISTUSER

This variable is set to the user ID of the user who owns the list, if it is a private list.

&$LHLISTNAME

This variable is set to the name of the list.

&$LHDESC

This variable is set to the description of the list.

&$LHDATASRC

This variable is set to the data source defined for the list.

&$LHFMTFLD*n*

These variables are set to the names of the real fields defined for the list; *n* is in the range 1 to the number of real fields defined.

&$LHATB*

These variables are set to the panel attributes (see page 359) that can be used to set the intensity, color, and highlighting for data within an entry line.

Modifiable Variables

This statement contains the following modifiable variables:

&$LHCRITTOTAL

This variable is set to the number of &$LHCRIT variables that contain criteria. The value of this variable is in the range 0 to 9999.

&$LHCRIT*n*

These variables are set to the criteria which the service procedure uses to determine the entries to be included in the list, if variable &$LHCRITTOTAL is greater than zero; *n* is in the range 1 to the value of &$LHCRITTOTAL.

&$LHSUPPRESS

Use this variable to specify whether entries are to be suppressed from the list by this procedure during entry processing. Set this variable to YES if you want to suppress entries; otherwise, set this variable to NO (the default).

&$LHADDALLOW

This variable is set to the Add Allowed setting defined for the list. It is set to YES if the list supports the ADD command, and it is set to NO if the list does not support the ADD command. The value of this variable can be modified to YES or NO. This variable is ignored if the list is a single or multiple select, or numbered list.

&$LHGETALL

This variable is set to the Get All Entries setting defined for the list. It is set to YES if all entries for the list are retrieved during initialization processing for the list. It is set to NO if entries are retrieved as required for display on the list. The value of this variable can be modified to YES or NO.

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the list exit procedure to one of the following return codes:

0

Continue processing

8

An error occurred; terminate the list

&$LHOPT=ENTRY Statement

This statement indicates entry processing. When the value of &$LHOPT is ENTRY, you can use the list exit procedure to perform any special entry processing that is required for the list.

Read-Only Variables

This statement contains the following read-only variables:

&$LHOPT

This variable is set to ENTRY.

&$LHLISTFMT

This variable is set to indicate the format of the list. This variable is set to one of the following values:

ACTION

An action list

MSELECT

A multiple select list

SSELECT

A single select list

NUMBERED

A numbered list

&$LHAPPLID

This variable is set to the ID of the application to which the list belongs.

&$LHLISTTYPE

This variable is set to the type of the list. This variable is set to one of the following:

PUBLIC

The list is a public list.

PRIVATE

The list is a private list.

&$LHLISTUSER

This variable is set to the user ID of the user who owns the list, if it is a private list.

&$LHLISTNAME

This variable is set to the name of the list.

&$LHDESC

This variable is set to the description of the list.

&$LHDATASRC

This variable is set to the data source defined for the list.

&$LHCRITTOTAL

This variable is set to the number of &$LHCRIT variables that contain criteria. The value of this variable is in the range 0 to 9999.

&$LHCRIT*n*

These variables are set to the criteria which the service procedure uses to determine the entries to be included in the list, if variable &$LHCRITTOTAL is greater than zero; *n* is in the range 1 to the value of &$LHCRITTOTAL.

&$LHFMTFLD*n*

These variables are set to the names of the real fields defined for the list; *n* is in the range 1 to the number of real fields defined.

&$LHATB*

These variables are set to the panel attributes (see page 359) that can be used to set the intensity, color, and highlighting for data within an entry line.

Modifiable Variables

This statement contains the following modifiable variables:

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the list exit procedure to one of the following return codes:

0

Continue processing

4

Suppress the entry from the list

8

An error occurred; terminate the list

$LHOPT=TERM Statement

This statement indicates termination of a list. When the value of &$LHOPT is TERM, you can use the list exit procedure to perform any special processing that is required to terminate the list.

Read-Only Variables

This statement contains the following read-only variables:

&$LHOPT

This variable is set to TERM.

&$LHLISTFMT

This variable is set to indicate the format of the list. This variable is set to one of the following values:

ACTION

An action list

MSELECT

A multiple select list

SSELECT

A single select list

NUMBERED

A numbered list

&$LHAPPLID

This variable is set to the ID of the application to which the list belongs.

&$LHLISTTYPE

This variable is set to the type of the list. This variable is set to one of the following:

PUBLIC

The list is a public list.

PRIVATE

The list is a private list.

&$LHLISTUSER

This variable is set to the user ID of the user who owns the list, if it is a private list.

&$LHLISTNAME

This variable is set to the name of the list.

&$LHDESC

This variable is set to the description of the list.

&$LHDATASRC

This variable is set to the data source defined for the list.

&$LHCRITTOTAL

This variable is set to the number of &$LHCRIT variables that contain criteria. The value of this variable is in the range 0 to 9999.

&$LHCRIT*n*

These variables are set to the criteria which the service procedure uses to determine the entries to be included in the list, if variable &$LHCRITTOTAL is greater than zero; *n* is in the range 1 to the value of &$LHCRITTOTAL.

&$LHFMTFLD*n*

These variables are set to the names of the real fields defined for the list; *n* is in the range 1 to the number of real fields defined.

&$LHATB*

These variables are set to the panel attributes (see page 359) that can be used to set the intensity, color, and highlighting for data within an entry line.

Modifiable Variables

This statement contains the following modifiable variables:

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the list exit procedure to one of the following return codes:

0

Continue processing

8

An error occurred

Chapter 9: Criteria Exit Procedure Interface

This section contains the following topics:

Criteria Exit Procedures

A criteria exit procedure interface is available for Common Application Services (CAS) criteria.

Criteria exit procedures are installation-written NCL procedures that are available for performing specialized processing when criteria are built.

Criteria exit procedures are optional. You specify the criteria exit procedure when you define a set of criteria. A criteria exit procedure performs installation-specific criteria processing at defined points during the building of a set of criteria:

- On initialization
- When a user performs one of the following from a run-time criteria panel:
- Press Enter
- Issue the ACTION command
- Issue the EXIT command

You write your own criteria exit procedures by using the variables described in the following sections.

The variables can be divided into two groups as follows:

- The first group of variables provide information to the procedure and are *not* modifiable (read-only). In particular, check the value of &$CROPT to determine the stage of criteria processing for you to implement the required special processing.
- The second group of variables are modifiable and enable the procedure to return information to the system.

You can use your own variables in the criteria exit procedure. However, do not use variable names that start with #CR. The criteria exit procedure is called with NOSHRVARS=(#CR). Once you specify your own variables, these variables are persistent through the different stages of criteria processing. That is, if you specify &A = *variable_value*, the value of this variable is available on each subsequent call to the procedure.

More information:

Criteria (see page 30)
Criteria Exit (see page 119)

Criteria Exit Procedure Statements

This section contains descriptions of the Criteria Exit Procedure statements.

&$CROPT=INIT Statement

This statement indicates initialization processing on the criteria. When the value of &$CROPT is INIT, you can use the criteria exit procedure to perform any special initialization processing on the criteria.

Read-Only Variables

This statement contains the following read-only variables:

&$CROPT

This variable is set to INIT.

&$CRAPPLID

This variable is set to the ID of the application to which the criteria set belongs.

&$CRCRITTYPE

This variable is set to one of the following types:

PUBLIC

The criteria set is public.

PRIVATE

The criteria set is private.

&$CRCRITUSER

If the criteria set is private, this variable is set to the ID of the user to whom the criteria set belongs. If the criteria set is public, this variable is not set.

&$CRCRITNAME

This variable is set to the name of the criteria.

&$CRDESC

This variable is set to the description of the criteria.

&$CRPANEL

This variable is set to the name of the run-time panel defined for the criteria.

&$CRDATASRC

This variable is set to the data source defined for the criteria.

&$CREPARMTOT

This variable is set to the number of exit parameter lines defined for the criteria in the range 0 to 9999.

&$CREPARM*n*

This range of variables are set to the exit parameters defined for the criteria if &$CREPARMTOT is greater than zero; *n* is in the range 1 to the value of &$CREPARMTOT.

&$CRFKA1

This variable is set to the first line of the function key area if a run-time panel is defined for the criteria. If a run-time panel is defined, this variable must be specified as an output variable on the first line of the #TRAILER statement in the panel definition.

&$CRFKA2

This variable is set to the second line of the function key area if a run-time panel is defined for the criteria. If a run-time panel is defined, this variable must be specified as an output variable on the second line of the #TRAILER statement in the panel definition.

Modifiable Variables

This statement contains the following modifiable variables:

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the criteria exit procedure to one of the following return codes:

0

Continue processing

8

An error occurred; terminate building the criteria

&$CROPT=TERM Statement

This statement indicates one of the following occurred from a run-time criteria panel:

- Enter key pressed
- ACTION command issued
- EXIT command issued

This call is not applicable if there is no run-time criteria panel. When the value of &$CROPT is TERM, you can use the criteria exit procedure to perform any special processing on the run-time criteria.

Read-Only Variables

This statement contains the following read-only variables:

&$CROPT

This variable is set to TERM.

&$CRAPPLID

This variable is set to the ID of the application to which the criteria set belongs.

&$CRCRITTYPE

This variable is set to one of the following types:

PUBLIC

The criteria set is public.

PRIVATE

The criteria set is private.

&$CRCRITUSER

If the criteria set is private, this variable is set to the ID of the user to whom the criteria belongs. If the criteria set is public, this variable is not set.

&$CRCRITNAME

This variable is set to the name of the criteria.

&$CRDESC

This variable is set to the description of the criteria.

&$CRPANEL

This variable is set to the name of the run-time panel defined for the criteria.

&$CRDATASRC

This variable is set to the data source defined for the criteria.

&$CREPARMTOT

This variable is set to the number of exit parameter lines defined for the criteria in the range 0 through 9999.

&$CREPARM*n*

This range of variables are set to the exit parameters defined for the criteria if &$CREPARMTOT is greater than zero; *n* is in the range 1 to the value of &$CREPARMTOT.

&$CRCURSOR

This variable is set to the current cursor position if a run-time panel is defined for the criteria. If a run-time panel is defined, this variable must be specified on the CURSOR parameter of the #OPT statement in the panel definition.

&$CRCOMMAND

This variable is set to the command executed from the run-time panel. The value is null if the Enter key was pressed and no command was entered. This variable must be used as the input field for the Command field on the run-time panel. This variable, when set, contains one of the following commands:

ACTION

The ACTION command was executed to allow the specification of variable data from the run-time panel to be included in the criteria.

EXIT

The EXIT command was executed to terminate the run-time panel.

Modifiable Variables

This statement contains the following modifiable variables:

&$CRALARM

Use this variable to control the terminal alarm if a run-time panel is defined for the criteria. Valid values are YES to turn the alarm on and NO otherwise. This variable must be specified in the ALARM parameter of the #ERR statement in the panel definition.

&$CRERRFLDS

Use this variable to return the names of the input fields that contain incorrect values to the system. This variable can be set to the input fields that are in error in the form *fieldname1, fieldname2...fieldnameN.*

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the criteria exit procedure to one of the following return codes:

0

Continue processing

4

Redisplay the run-time panel (not valid if &$CRCOMMAND is set to EXIT)

8

An error occurred; terminate building the criteria

Chapter 10: Table Entry Validation Exit Procedure Interface

This section contains the following topics:

Table Entry Validation Exit Procedures

A table entry validation exit procedure interface is available for Common Application Services (CAS) tables.

Table entry validation exit procedures are installation-written NCL procedures that are available for performing specialized processing when a table entry is being maintained.

Table entry validation exit procedures are optional. You specify the table entry validation exit procedure when you define a table. The table entry validation exit procedure is called during table entry maintenance functions Add, Update, and Delete.

You write your own table entry validation exit procedures by using variables described in the following sections.

The variables can be divided into two groups as follows:

- The first group of variables provide information to the procedure and are *not* modifiable (read-only). In particular, check the value of &$VMEXFUNC to determine the operation being processed for you to implement the required special processing.
- The second group of variables are modifiable and enable the procedure to return information to the system. Some of these variables are set with values already, but the procedure can change those values.

Table Entry Validation Exit Procedure Statements

This sections contains descriptions of the Table Entry Validation Exit Procedure statements.

&$VMEXFUNC=ADD Statement

This statement indicates that a table entry is being added. When the value of &$VMEXFUNC is ADD, you can use the table entry validation exit procedure to perform any special processing that is required when adding a table entry.

Read-Only Variables

This statement contains the following read-only variables:

&$VMEXFUNC

This variable is set to ADD.

&$VMEXAPPL

This variable is set to the application ID of the table definition.

&$VMEXFIELD

This variable is set to the field name of the table definition.

Modifiable Variables

This statement contains the following modifiable variables:

&$VMEXFULL

This variable is set to the full value of the table entry.

&$VMEXABBR

This variable is set to the abbreviated value of the table entry.

&$VMEXDESC

This variable is set to the description of the table entry.

&$VMEXSEQ

This variable is set to the table entry sequence number.

&$VMEXACTIVE

This variable is set to indicate whether the table entry is active (YES or NO).

&$VMEXTXT1…10

These variables are set to additional information about the table entry.

&$VMEXERRFLD

Use this variable to identify the name of the variable in the table entry definition that is in error.

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the table entry validation exit procedure to one of the following return codes:

0

Continue processing

8

Do not complete the operation; redisplay the panel

&$VMEXFUNC=DELETE Statement

This statement indicates that a table entry is being deleted. When the value of &$VMEXFUNC is DELETE, you can use the table entry validation exit procedure to perform any special processing that is required when deleting a table entry.

Read-Only Variables

This statement contains the following read-only variables:

&$VMEXFUNC

This variable is set to DELETE.

&$VMEXAPPL

This variable is set to the application ID of the table definition.

&$VMEXFIELD

This variable is set to the field name of the table definition.

&$VMEXFULL

This variable is set to the full value of the table entry.

&$VMEXABBR

This variable is set to the abbreviated value of the table entry.

&$VMEXDESC

This variable is set to the description of the table entry.

&$VMEXSEQ

This variable is set to the table entry sequence number.

&$VMEXACTIVE

This variable is set to indicate whether the table entry is active (YES or NO).

&$VMEXTXT1…10

These variables are set to any additional information about the table entry.

Modifiable Variables

This statement contains the following modifiable variables:

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the table entry validation exit procedure to one of the following return codes:

0

Continue processing

8

Do not complete the operation; redisplay the panel

&$VMEXFUNC=UPDATE Statement

This statement indicates that a table entry is being updated. When the value of &$VMEXFUNC is UPDATE, you can use the table entry validation exit procedure to perform any special processing that is required when updating a table entry.

Read-Only Variables

This statement contains the following read-only variables:

&$VMEXFUNC

This variable is set to UPDATE.

&$VMEXAPPL

This variable is set to the application ID of the table definition.

&$VMEXFIELD

This variable is set to the field name of the table definition.

&$VMEXFULL

This variable is set to the full value of the table entry.

Modifiable Variables

This statement contains the following modifiable variables:

&$VMEXABBR

This variable is set to the abbreviated value of the table entry.

&$VMEXDESC

This variable is set to the description of the table entry.

&$VMEXSEQ

This variable is set to the table entry sequence number.

&$VMEXACTIVE

This variable is set to indicate whether the table entry is active (YES or NO).

&$VMEXTXT1…10

These variables are set to additional information about the table entry.

&$VMEXERRFLD

Use this variable to identify the name of the variable in the table entry definition that is in error.

&SYSMSG

Use this variable to return a message. If an error occurs during special processing, this variable must be set to an error message.

Return Codes

The variable &RETCODE must be set by the table entry validation exit procedure to one of the following return codes:

0

Continue processing

8

Do not complete the operation; redisplay the panel

Chapter 11: Report Writer

This section contains the following topics:

Understanding Report Writer

Report Writer lets users easily define report layouts and generate defined reports immediately or at specified times. Report Writer is a facility within Managed Object Development Services (MODS).

Report Writer is designed to operate totally independently of the database in which the data to be used to generate a report is contained—it interfaces with the application, to access that application's data from the database. The User Access Maintenance Subsystem (UAMS) facility secures data against illegal access.

Report Writer interfaces with Print Services Manager (PSM) for the management of the report output.

Report Writer Facilities

Report Writer lets you do the following:

- Draw a report layout using an editor.
- View a report layout on screen.
- Generate a report on request by application or user.
- Generate reports automatically based on a schedule.
- Perform specialized processing using user exit points, such as field formatting, complex arithmetic, and conditional suppression of data during report generation.
- Integrate Report Writer with other NCL-based components.
- Look at reports in progress in the system.
- Interact with the user using System and Security services panels.
- Get information about any panel using the online help facility.

Defining a Report

A Report Definition contains all the information required to format a report. It is created using a specialized editor which lets you draw the layout of a report on the screen. The report definition is then stored on the MODS file and can be recalled at any time to produce a report. It can also be recalled to perform maintenance functions, such as updating, copying, or deleting.

Viewing a Report Layout

The View Report Layout function lets you see how the report will appear before it is generated.

Generating Reports

A report can be generated in the following ways:

- Requested by a user or application
- Generated automatically by Report Writer according to a user-defined schedule

For a report to be generated on request, you must pass the report name and the printer destination to the generator.

For a report to be automatically generated, you must define a schedule that Report Writer uses to determine which report is required and when and where it is to be generated.

Printing Reports

Report printing and validation of printer names is controlled by PSM. You can choose to print a report as soon as the printer is available or to hold the report output on the PSM print spool where it can be viewed before printing.

Report Exit Procedures

A Report Exit is an NCL procedure that lets you do specialized processing of data while a report is being generated. Functions that can be performed by a Report Exit procedure are as follows:

- Initialization processing, for example, opening files and defining vartables
- Specialized field formatting, for example, complex arithmetic and conditional suppression of data
- Item processing, for example, retrieving data from another source
- Termination processing, for example, closing files and deleting vartables

Reports in Progress

A list of reports currently in progress in the system can be displayed. From this list you can view report output on the screen, or cancel processing or purge the report output.

Security

Report Writer uses the User Access Maintenance Subsystem (UAMS) facility of management services to control user access to data.

Security is based on user IDs. A user ID defines the function and privilege level that a particular person is entitled to when they sign on to the system. It is associated with a secret password, known only to the user.

To secure Report Writer from unauthorized use, users must be defined to the security system with the required Report Writer resource keys. In general, all users can browse any public reports, and their own private reports. Depending upon access privileges, users can be prevented from adding, browsing, updating, deleting or generating any, or all, reports.

Report Definitions

Before a report can be produced, it must first be defined to Report Writer. Report definitions are created using a specialized editor which is similar to the industry-standard ISPF editor. This editor lets you draw the layout of a report on the screen. The report definition is then stored on a database and can be recalled at any time to produce the report or for maintenance functions.

A report definition contains the following components:

- Report description
- Sort fields
- Report header
- Page header
- Data formats
- Control break headers
- Control break trailers
- Page trailer
- Report trailer

Report Description

The report description contains the following control information about the report:

- Report application
- Report type
- User ID
- Report name
- Group name
- Description
- Status
- Report width
- Suit single record indicator
- Criteria identifier
- Report exit name
- Comments

Report Application

The report application is the ID of the report application to which the report belongs. The report application defines the service procedure and whether or not it supports sort fields, and if so how many.

A report application ID must begin with an application ID that is defined in the Application Register (see page 48). The report application must be defined in a $RWAPPL table for the specified application. For example, given report application YKZ123, the application YKZ must be defined in the Application Register and YKZ123 must be an entry in the YKZ.$RWAPPL table.

Report Type

Each report must be assigned a report type. Valid types are PUBLIC and PRIVATE. The report type is used to secure reports from illegal access. Your UAMS definition defines whether you can access PUBLIC or PRIVATE reports.

User ID

The user ID is used to define the owner of PRIVATE reports.

Report Name

The report name identifies the report within a report application.

Group Name

Groups are defined by your installation and are used to define different groups within your organization, with different reporting requirements. Group names can be used to simplify the administration of reports and maintaining report definitions. Valid group names are stored in a $ADGROUP table for the application to which the report belongs.

Description

The report description indicates the use of the report (for example the CA SOLVE:InfoMaster Category Summary report). This description appears in the report selection list to assist in the selection of reports.

Status

The report status indicates whether or not a report is disabled. When it is set to ACTIVE the report can be generated and appears on selection list displays. When set to INACTIVE, these functions are disabled.

Report Width

The report width indicates the maximum number of columns that can be printed per page for the report.

Suit Single Record Indicator

The suit single record field indicates that the report definition is suitable for printing a single record when set to YES, and that it is not suitable for printing a single record when set to NO.

Criteria Identifier

The criteria identifier is the identifier of a Common Application Services (CAS) criteria definition or the value FREEFORM. When the report is generated, CAS is called to build the criteria or to present the CAS Criteria panel when FREEFORM is defined. The criteria received from CAS is then passed to the service procedure and the report exit procedure. This enables criteria to be shared between different reports and for ad-hoc inquiries to be easily handled.

Report Exit Name

A report exit is an installation-written NCL procedure which can be used if specialized processing of data is to be carried out before printing.

Comments

The comments give a more detailed description of the report and any associated information.

Sort Fields

The sort fields are the fields on which data records will be sorted before printing. These field names are stored with details of how sorting will be performed (for example, whether in ascending or descending order).

Sort fields are assigned a number to allow data to be sorted on several fields within a record. For example, if you want to sort Problem file records on severity within priority, the severity field would be defined as sort field 1 and the priority field would be defined as sort field 2.

The maximum number of sort fields that can be defined depends on the report application for which the report is defined.

For each sort field, there can be defined a control break header and/or control break trailer. These will be printed on a control break, that is, when the value of the sort field changes. A control break header could be used, for example, to print column headings that describe the data being printed. A control break trailer may be a total line, showing the total values of the data in the reported group of records.

Control break headers and trailers are linked to the sort field by the sort field number, which is stored as part of their definition. So, for example, when the value of sort field number 1 changes, the control break trailer assigned to it (if one is defined) will be printed before processing continues.

Format Items

The remaining report components are the format items, which are the lines that will be printed on the report. Each format item may consist of any number of lines, made up of both constant and variable data.

Constant data is printed exactly as it is entered or drawn on the screen. It is mainly used to define headings and subheadings. Variable data is retrieved by the service procedure from a database, or from a report exit procedure, to be displayed on the report.

Variable data is prefixed by an ampersand (&) to indicate it is a database field, or an exclamation mark (!) to indicate it is a system field. Database field names are validated against the field names in the Data Fields Table for the report application. System field names are validated against the field names in the System Fields Table.

Report Header

The report header is printed once at the beginning of a report. Report Headers can be used to explain what the report is about and to indicate the beginning of a new report.

Page Header

The page header is printed at the top of every page (including the report header and trailer pages). It can consist of one or several lines of constant and/or variable data. A page header may contain a description of the use of the report, for example, Weekly Network Error Warning System (NEWS) Attention Summary. A page header may also contain the date and page number.

Data Formats

Data Formats are the details that are printed for each record that is passed to Report Writer by the service procedure. Any number of data formats can be defined. A report exit NCL procedure can be used to determine which data format or group of data formats is to be printed for each individual record. If there is no report exit, all data formats will be printed in ascending order.

Control Break Headers

Control Break Headers are the details which are printed as a heading above a group of records. If defined, a control break header will be printed each time the sort field to which it is assigned, changes value.

Control Break Trailers

Control Break Trailers are the details which are printed as a trailer below a group of records. If defined, a control break trailer will be printed each time the sort field to which it is assigned, changes value. Control break trailers are most commonly used to print sub-total and total lines.

Page Trailer

The Page Trailer is printed as a footer at the bottom of each page (including the report header and trailer pages). It can contain, for example, your company name or the page number.

Report Trailer

The Report Trailer is printed at the end of the report.

Report Layout

The View Report Layout panel can be displayed at any time during the definition of a report. This lets you see on the screen the layout of the report as it will be when it is printed.

Defining a Report Application

Report applications are stored as entries in the $RWAPPL table for the specified application—the application is indicated by the first three characters of the report application. New report applications must be added using CAS.

Data fields for each application must be defined to Report Writer by defining a Data Fields Table, using CAS.

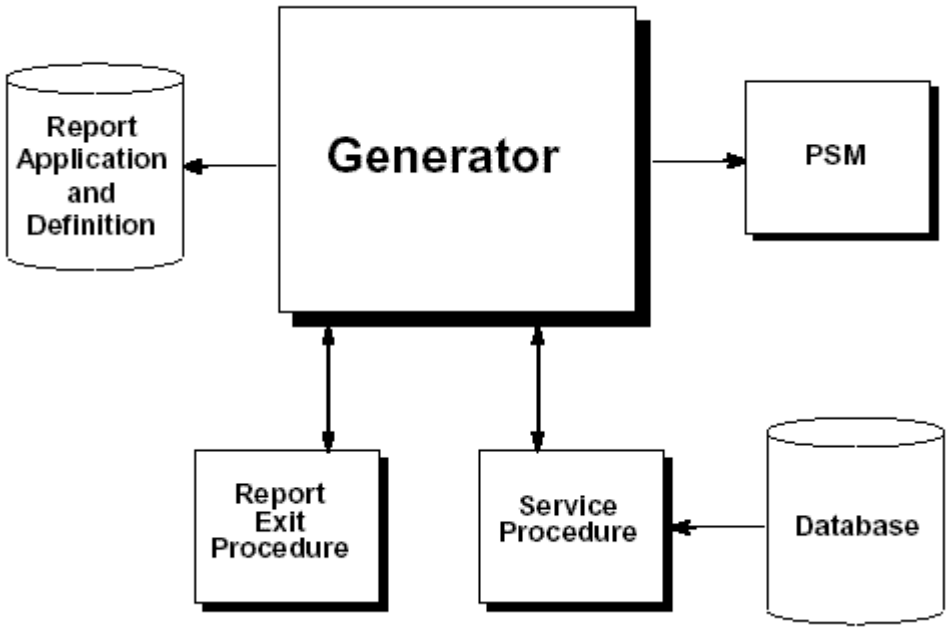
If the data input to your report needs to be sorted before printing, the fields on which the data will be sorted must be defined to Report Writer during definition of the report. The sort fields are stored in a Sort Fields Table which is controlled by CAS.

Report Generator

The Report Writer Report Generator controls the generation of reports. Reports can be generated in two ways:

- Directly, on request by a user or application
- Automatically by the Schedule facility of Report Writer

The report generator reads the control information and report layout from the MODS file and calls the service procedure to access data from the application database. If there is a report exit associated with this report definition, the generator calls it to do any specialized processing of data, then passes the formatted data to PSM for printing.



Service Procedure

The service procedure is an NCL procedure whose purpose is to provide the generator with data to be used to generate a report. The service procedure knows the database from which the data is to be retrieved and the format of the data. The name of the service procedure is defined in the report application to which the report belongs and is executed by the generator.

Functions of the service procedure are as follows:

- Initialization processing, for example, opening files and searching the database
- Get the next record to be processed
- Get the value of the sort fields for the next record to be processed
- Termination processing, for example, closing files

Report Exit

The purpose of a report exit procedure is to let the user do specialized processing, based on installation requirements, while the report is being generated. For example, data can have arithmetic performed upon it or it can be conditionally suppressed.

The report exit procedure knows the format of the data, that is, the names of the variables that contain the data and the format of that data. The name of the report exit procedure is defined in the report definition and is executed by the generator.

Functions performed by the report exit procedure are as follows:

- Initialization processing, for example, define vartables
- Item processing, for example, complex arithmetic
- Termination processing, for example, delete vartables.

Defining a Schedule

Report Writer can automatically generate reports according to a user-defined schedule. To do this you must first define the schedule.

A Schedule Definition consists of the following information:

- The name, report type, owner and associated information about the report to be generated
- The intervals at which the report will be generated
- The printer name where the report will be printed
- Whether the report is required to be printed immediately the printer is available or held on the PSM print spool until released for printing

- Whether the report is to be kept on the print spool after it has been printed, or deleted
- The number of copies to be printed

The Schedule function of Report Writer passes report control information to the generator each time the report is scheduled to be generated.

A report can be scheduled to run, for example, on the first day of every month at midday or at 10 a.m. on Mondays, Tuesdays, and Fridays.

NCL Interface

This section describes the NCL interface for Report Writer.

The NCL interface for Report Writer performs the following functions:

- $RWCALL OPT=GENERATE - generate a report
- $RWCALL OPT=INFO - return report definition information, optionally presenting a Report List
- $RWCALL OPT=MENU - present a Report Writer menu
- $RWCALL OPT=STATUS - present Reports in Progress

Notational Conventions

Each NCL call is described on a separate page under the following section headings, where applicable:

Function:

Purpose of interface

Use:

General description of interface use

Operands:

Description of operands

Variables:

Fields used to pass variable data to the service procedure or report exit procedure

Return Codes:

Return code options set on completion of interface function, with an explanation

Examples:

Examples of interface syntax

Notes:

Any further information or special aspects

Interface Syntax

The precise syntax for each interface is defined in a box towards the top of each page. For example:

```
&CONTROL NOSHRVARS

-EXEC $RWCALL   OPT=STATUS
              [ USERID=userid ]
```

On the left is the interface name ($RWCALL), and to the right are the permissible operands. The syntax used observes the following guidelines:

- UPPERCASE characters
  Interface names or operands consisting of uppercase characters must be entered as shown, but can be entered in upper or lower case.
- *Italic* characters
  These are variables that show the kind of information, rather than the exact information that must be supplied. The actual entry replaces the italic description. Valid types of data are described for each interface within the operands section.
- Underscored values
  Indicates the defaulted value that is assumed for an operand if one is not specified in the interface.
- {Braces}
  These indicate the available options for an operand. One of the alternatives described must be selected. Do not include braces when entering a specification.
- [Square brackets]
  Indicate optional specifications. Do not include square brackets when entering a specification.
- The Or Sign |
  This separates options for an optional or mandatory specification. If a group of options is enclosed by square brackets, and each is separated by an Or sign, none of the options have to be chosen. If none are coded, the default value (underscored) is used.
- Commas and Equals signs
  Commas and equal signs must be entered as shown. If commas or equal signs appear within brackets, they are optional and used only if the accompanying optional operand is used.

$RWCALL OPT=GENERATE

This function generates a report. Optionally, the function presents the Generate a Report panel on which the report details can be entered, or the Report List.

This function has the following formats:

```
&CONTROL SHRVARS=($RW)

-EXEC $RWCALL   OPT=GENERATE
              [ MODE= { DEFGEN | GENERATE | PRTGEN } ]
              [ APPL= repapplid ]
              [ TYPE= { PUBLIC | PRIVATE } ]
              [ USERID=userid ]
              [ NAME=name ]
              [ PRINTER=printer ]
              [ OWNER=userid ]
              [ HOLD= { NO | YES } ]
              [ KEEP= { NO | YES } ]
              [ COPIES=n ]
              [ WAIT= { NO | YES } ]
              [ SYSTEM=system ]
              [ RECCAT=record category ]

-EXEC $RWCALL   OPT=GENERATE
                MODE=LIST
                APPL=repapplid
              [ GROUP=group ]
```

Operands

OPT=GENERATE

Generates a report.

MODE= { DEFGEN | GENERATE | PRTGEN | LIST }

Specifies the mode of operation.

DEFGEN

Displays the Generate a Report panel.

GENERATE

Generates the report.

PRTGEN

Generates the report and displays the PSM Confirm Printer panel for the specification of printer details.

LIST

Displays a Report List in the form of an action list. The list contains all public and private reports for the current user which belong to the report application set in the APPL operand. Only those reports with a status of ACTIVE are listed.

APPL=*repapplid*

Specifies the ID of the report application to which the report belongs.

TYPE={ PUBLIC | PRIVATE }

Specifies the type of report.

PUBLIC

Specifies that the report is a public report.

PRIVATE

Specifies that the report is a private report.

USERID=*userid*

Specifies the user ID of the user who owns the report if it is a private report.

Default: Value of &USERID if TYPE is set to PRIVATE

Note: If MODE is GENERATE or PRTGEN, and TYPE and USERID are not specified, the system looks for a PRIVATE report owned by the invoking user ID (that is, the value of &USERID) with the name specified. If not found, the system looks for a PUBLIC report with the name specified.

NAME=*name*

Specifies the name of the report.

PRINTER=*printer*

Specifies the name of the printer. The printer must have previously been defined to PSM.

Default: Printer assigned to the owner of the report as their default printer

OWNER=*userid*

Specifies the user ID of the user who is to own the report. This user ID is passed to PSM as the owning user ID for the report. The default is the value of &USERID.

HOLD={ NO | YES }

Specifies whether PSM assigns the report a status of HELD when added to the Print Spool File.

KEEP={ NO | YES }

Specifies whether PSM leaves the report on the Print Spool File after being printed.

COPIES=*n*

Specifies the number of copies of the report to print.

Default: 1

Limits: 1 through 255

WAIT={ NO | YES }

Specifies whether control is returned to the requester immediately after the report is started or on completion of the report.

SYSTEM=*system*

Specifies the system name to use instead of the name defined in the table entry for the specified report application.

RECCAT=*record category*

Specifies the record category to use instead of the category defined in the table entry for the specified report application.

GROUP=*group*

Specifies the name of the group to which reports belong, to be included in the Report List when the MODE operand is set to LIST.

Variables

&$RWCRIT*n*

These variables can be set to the data criteria that is to be used by the service procedure and/or report exit procedure in determining the data to be included in the report when the MODE operand is not set to LIST. The valid value for *n* is in the range 1 to 99999. The first blank variable indicates the end of the data criteria.

&$RWUSRD*c*

These user data variables can be set to user data that is to be used by the service procedure and/or the report exit procedure when the MODE operand is not set to LIST. The variable *c* is between 0 and 5 alphanumeric and/or national characters. These variables are not set or cleared by the system, therefore must be completely managed by your installation-defined NCL procedures.

Return Codes

&RETCODE = 0

$RWCALL completed successfully. $RWFDBK is set as follows:

1

RETURN command entered or function key pressed

&RETCODE = 4

$RWCALL completed successfully. Request denied. &SYSMSG is set to an error message and &$RWFDBK is set to one of the following values:

1

User not authorized for the request

8

Report not defined

10

No report defined within the specified range

&RETCODE = 8

An error occurred. &SYSMSG is set to an error message.

Examples

```
&CONTROL SHRVARS=($RW)
-EXEC $RWCALL OPT=GENERATE MODE=DEFGEN

&CONTROL SHRVARS=($RW)
-EXEC $RWCALL OPT=GENERATE MODE=DEFGEN APPL=$SAIMPB TYPE=PUBLIC

&CONTROL SHRVARS=($RW)
&$RWCRIT1 = &STR $PBSEVERITY=1
&$RWCRIT2 = &STR AND
&$RWCRIT3 = &STR $PBFIXED=NO
-EXEC $RWCALL OPT=GENERATE MODE=GENERATE APPL=$SAIMPB +
    TYPE=PUBLIC NAME=OPENPROBLEMS

&CONTROL SHRVARS=($RW)
-EXEC $RWCALL OPT=GENERATE MODE=GENERATE APPL=$SAIMPB +
    TYPE=PRIVATE USER=USER01 NAME=TEST1

&CONTROL NOSHRVARS
-EXEC $RWCALL OPT=GENERATE MODE=LIST APPL=ZPRPROB
```

Notes

When MODE is set to DEFGEN, the APPL, TYPE, USERID, NAME, PRINTER, OWNER, HOLD, KEEP and COPIES operands are used to initialize fields on the Generate a Report panel.

When MODE is set to LIST, the variables &$RWCRIT*n* and &$RWUSRD*n* are ignored.

When MODE is set to PRTGEN, the PRINTER, HOLD, KEEP and COPIES operands are ignored. Instead, the values entered on the Confirm Printer panel are used to generate the report.

$RWCALL OPT=INFO

This function returns report definition information. Optionally, the function presents a list of reports from which a selection can be made.

This function has the following format:

```
&CONTROL SHRVARS=($RW)

-EXEC $RWCALL   OPT=INFO
              [ INFO=REPORT ]
              [ APPL={ repapplid | prefix? | ? } ]
              [ TYPE={ PUBLIC | PRIVATE } ]
              [ USERID={ userid | prefix? | ? } ]
              [ NAME={ name | prefix? | ? } ]
              [ GROUP={ group | prefix? | ? } ]
              [ SINGLE={ YES | NO } ]
              [ ORDER={ ID | DESC } ]
              [ STATUS={ ACTIVE | INACTIVE} ]
              [ AUTOSEL={ YES | NO } ]
```

Use

To validate report details entered by a user on a panel defined by your installation and provide help on a panel defined by your installation by presenting a Report List from which a selection can be made.

Operands

OPT=INFO

Specifies definition information is to be returned.

INFO=REPORT

Specifies report definition information is to be returned.

APPL={ *repapplid* | *prefix*? | ? }

Specifies the ID of the report application to which the report belongs. If a prefix followed by a question mark (?)is specified, a Report List is presented from which a selection can be made. The list will contain all reports that belong to report applications with IDs starting with the prefix specified, and which match the other criteria specified. If a question mark is specified without a prefix, all reports which match the other criteria specified are listed on the Report List.

TYPE={ PUBLIC | PRIVATE }

Specifies the type of report. PUBLIC indicates that the report is a public report and PRIVATE indicates that the report is a private report.

USERID={ *userid* | *prefix*? | ? }

Specifies the user ID of the user who owns the report if it is a private report. If a prefix followed by a question mark (?) is specified, a Report List is presented from which a selection can be made. The list will contain all private reports owned by users whose user ID starts with the prefix specified, and which match the other criteria specified. If a question mark is specified without a prefix, all reports that match the other criteria specified are listed on the Report List.

NAME={ *name* | *prefix*? | ? }

Specifies the name of the report. If a prefix followed by a question mark (?) is specified, a Report List is presented from which a selection can be made. The list will contain all reports with names starting with the prefix specified, and which match the other criteria specified. If a question mark is specified without a prefix, all reports that match the other criteria specified are listed on the Report List.

GROUP={ *group* | *prefix*? | ? }

Specifies the group to which the reports belong that are to be presented in the Report List. A Report List is presented from which a selection can be made. If a prefix followed by a question mark (?) is specified, the list will contain all reports that belong to groups with names starting with the prefix specified. If a question mark is specified without a prefix, all reports that match the other criteria specified are listed on the Report List.

SINGLE={ YES | NO }

Specifies the setting of the Suit Single Record field for reports that are to be included in the Report List. If not specified, reports with a Suit Single Record setting of YES or NO are included in the list.

ORDER={ ID | DESC }

Specifies the order in which the Report List will be presented.

ID

Reports will be listed in user ID order.

DESC

Reports will be listed in description order.

STATUS={ ACTIVE | INACTIVE }

Specifies the status of reports to be included on the Report List. If not specified, both active and inactive reports are included in the list.

AUTOSEL={ YES | NO }

Allows the automatic selection of a report, when set to YES, instead of displaying a selection list containing only one report.

Return Codes

&RETCODE = 0

$RWCALL completed successfully. The variables returned are as follows:

&$RWREPAPPL

Report Application

&$RWREPTYPE

Report type, PUBLIC or PRIVATE

&$RWREPUSERID

User ID of owner if it is a private report

&$RWREPNAME

Report name

&$RWREPDESC

Brief description of report

&$RWREPGROUP

Group

&RETCODE = 4

Request denied. &SYSMSG is set to an error message and &$RWFDBK is set to one of the following:

1

User not authorized for the request

8

Report not defined

10

No reports defined within the specified range

11

Report not selected from Report List

&RETCODE = 8

An error occurred. &SYSMSG is set to an error message.

Examples

```
&CONTROL SHRVARS=($RW)
-EXEC $RWCALL OPT=INFO INFO=REPORT NAME=?

&CONTROL SHRVARS=($RW)
-EXEC $RWCALL OPT=INFO INFO=REPORT TYPE=PUBLIC +
NAME=SUMMARY?

&CONTROL SHRVARS=($RW)
-EXEC $RWCALL OPT=INFO INFO=REPORT APPL=$SAIMPB +
TYPE=PUBLIC NAME=OPENPROBLEMS
```

Notes

Either the APPL, USERID, NAME, GROUP, SINGLE or STATUS operand must be specified.

$RWCALL OPT=MENU

This function presents a Report Writer menu.

This function has the following format:

```
&CONTROL NOSHRVARS

-EXEC $RWCALL   OPT=MENU
              [ MENU={ PRIMARY | REPORT | SCHEDULE } ]
```

Use

To present a Report Writer menu when a user selects an option from a menu defined by your installation or enters a command written by your installation.

Operands

OPT=MENU

Specifies that a Report Writer menu is to be presented.

MENU={ PRIMARY | REPORT | SCHEDULE }

Specifies which Report Writer menu is to be presented.

PRIMARY

Presents the Report Writer Primary Menu.

REPORT

Presents the Report Definition Menu.

SCHEDULE

Presents the Schedule Definition Menu.

Return Codes

&RETCODE = 0

$RWCALL completed successfully. &$RWFDBK can be set to the following:

1

RETURN command entered or function key pressed

&RETCODE = 4

$RWCALL completed successfully. Request denied. &SYSMSG is set to an error message and &$RWFDBK is set to the following:

1

User not authorized for the request

&RETCODE = 8

An error occurred. &SYSMSG is set to an error message.

Examples

```
&CONTROL NOSHRVARS
-EXEC $RWCALL OPT=MENU

&CONTROL NOSHRVARS
-EXEC $RWCALL OPT=MENU MENU=PRIMARY

&CONTROL NOSHRVARS
-EXEC $RWCALL OPT=MENU MENU=REPORT
```

$RWCALL OPT=STATUS

This function presents the Reports in Progress list.

This function has the following format:

```
&CONTROL NOSHRVARS

-EXEC $RWCALL   OPT=STATUS
              [ USERID= userid ]
```

Use

To present Reports in Progress when a user selects an option from a menu defined by your installation or enters a command written by your installation.

Operands

OPT=STATUS

Specifies the Reports in Progress panel is to be presented.

USERID=*userid*

Specifies the user ID of the user whose reports are to be displayed in the Reports in Progress list.

Return Codes

&RETCODE = 0

$RWCALL completed successfully. &$RWFDBK can be set to the following:

1

RETURN command entered or function key pressed

&RETCODE = 4

$RWCALL completed successfully. Request denied. &SYSMSG is set to an error message and &$RWFDBK is set to the following:

1

User not authorized for the request

&RETCODE = 8

An error occurred. &SYSMSG is set to an error message.

Examples

```
&CONTROL NOSHRVARS
-EXEC $RWCALL OPT=STATUS
&CONTROL NOSHRVARS
-EXEC $RWCALL OPT=STATUS USERID=USER01
```

Report Exit Procedure

This section describes the variables that are passed to the report exit procedure and the return codes and variables it can set.

Function

The purpose of the report exit procedure is to let the user do specialized processing, based on installation requirements, while a report is being generated. For example, data can be conditionally suppressed from being processed or printed.

The report exit procedure must know the format of the data, that is, the name of the variables that contain the data and the format of that data. The name of the report exit procedure is defined in the report definition and is executed by the generator.

Functions performed by report exit procedures are as follows:

- Initialization processing, for example, open files or define vartables.
- Item processing, where an item is a report header or trailer, page header or trailer, control break header or trailer or record (that is, data format or sequence of data formats). For example, retrieve additional data from another source, and set it in variables that are defined in the report or are to be used in later processing.
- Termination processing, for example, close files or delete vartables.

Variables

The share variables facility of NCL is used to pass data to the report exit procedure and to let it pass data back to the generator. The report exit procedure is executed with &CONTROL NOSHRVARS=(#RW) specified. All variables not starting with the characters #RW can be accessed and set by the report exit procedure and variables containing data (data fields) can be modified. The variables set by the generator indicating the status of the current environment are as follows:

&$RWOPT

This variable is set to indicate the processing that is to be performed by the report exit. This variable can be set to one of the following variables:

INIT

Initialization processing is to be performed

TERM

Termination processing is to be performed

ITEM

Item processing is to be performed

&$RWAPPL

This variable is set to the ID of the report application to which the report being generated belongs.

&$RWTYPE

This variable is set to the type of report being generated. Type can be one of the following:

PUBLIC

The report is a public report

PRIVATE

The report is a private report

&$RWUSERID

This variable is set to the user ID of the user who owns the report if it is a private report.

&$RWNAME

This variable is set to the name of the report being generated.

&$RWDESC

This variable is set to the brief description of the report being generated.

&$RWITEM

This variable is set to indicate the item that is being processed when &$RWOPT is set to ITEM. Item can be set to one of the following values:

RH

Report header

PH

Page header

CH

Control break header

DF

Data format or sequence of data formats

CT

Control break trailer

PT

Page trailer

RT

Report trailer

&$RWITEMKEY

This variable is set to the sort field number of the control break header or trailer when &$RWITEM is set to CH or CT.

&$RWFMTSEQ

This variable is set to a list of all the data format numbers separated by commas when &$RWITEM is set to DF. This variable can be modified to contain a list of data format numbers that are to be printed, in the order they are to be printed, separated by commas. To process but not print the record (that is, to perform totalling) set this variable to null.

&$RWSYSTEM

This variable is set to the system name that is defined in the table entry for the report application to which the report belongs. The system name can be overridden on the GENERATE call to $RWCALL.

&$RWRECCAT

This variable is set to the record category that is defined in the table entry for the report application to which the report belongs. Record category can be overridden on the GENERATE call to $RWCALL.

&$RWOWNER

This variable is set to the user ID of the user who is to own the report. This user ID is passed to PSM as the owning user ID for the report.

&$RWPAGES

This variable is set to the number of pages that have been printed.

&$RWLINES

This variable is set to the number of lines that have been printed.

&$RWRECS

This variable is set to the number of records that have been processed.

&$RWCRITTOTAL

This variable is set to the number of &$RWCRIT*n* variables that contain data criteria. The value of this variable is in the range 0 to 99999.

&$RWCRIT*n*

These variables are set to data criteria if the &$RWCRITTOTAL variable is greater than zero. The variable *n* is a number in the range 1 to the value of &$RWCRITTOTAL. The data criteria is used by the service procedure to determine which data is to be included in the report. The format of these variables is dependent on the service procedure.

&$RWUSRD*c*

This is user data and is as set by the caller of $RWCALL or the service procedure. The variable *c* is between 0 and 5 alphanumeric and/or national characters. These variables can be used to pass user data through the generator to the report exit procedure. They can also be accessed by the service procedure. These variables can also be modified by the report exit procedure. The variables are never set or cleared by the system and must be completely managed by your installation-defined NCL procedures.

Return Codes

&RETCODE = 0

Indicates successful completion, continue processing. The system variable &$RWFMTSEQ may be set to a list of data format numbers that are to be printed, in the order they are to be printed, separated by commas. To process but not print the record (that is, to perform totalling) set this variable to null.

&RETCODE = 2

Indicates that the item is not to be printed. When &$RWITEM is set to DF also, do not process the record, get the next record.

&RETCODE = 8

Indicates that an error occurred, terminate processing. &SYSMSG may be set to an error message.

Notes

The generator executes the report exit procedure to perform item processing immediately before the item is to be printed.

If there is more than one data format, the report exit procedure is executed once, before all the data formats are printed for a record.

Service Procedure

This section describes the variables that are passed to the service procedure and the return codes and variables it can set.

Function

The purpose of the service procedure is to provide the generator with data to be used to generate a report. The service procedure must know the database from which the data is to be retrieved and the format of the data. The name of the service procedure is defined in the table entry for the report application to which the report belongs and is executed by the generator.

Functions of the Service Procedure are as follows:

- Initialization processing, for example, open files, search database (&NDBSCAN)
- To get the next record to be processed (that is, set the data fields)
- To get the value of the sort fields for the next record to be processed
- Termination processing, for example, close files

Variables

The share variables facility of NCL is used to pass data to the service procedure and to let it pass data back to the generator. The service procedure is executed with &CONTROL NOSHRVARS=(#RW) specified. All variables not starting with the characters #RW can be accessed and set by the service procedure. The variables set by the generator are as follows:

&$RWOPT

This variable is set to indicate the processing that is to be performed by the service procedure. This variable is set to one of the following values:

INIT

Initialization processing is to be performed

TERM

Termination processing is to be performed

GET

Get the next record to be processed by the generator

GETSF

Get the value of the sort fields for the next record to be processed by the generator. The values must be returned in the variables &$RWSFVAL*n* and the data fields for the previous record must not be modified. The service procedure is called to do GETSF processing before each GET call only if control break headers or trailers are defined in the report definition.

&$RWAPPL

This variable is set to the ID of the report application to which the report being generated belongs.

&$RWTYPE

This variable is set to the type of report being generated.

PUBLIC

The report is a public report

PRIVATE

The report is a private report

&$RWUSERID

This variable is set to the user ID of the user who owns the report if it is a private report.

&$RWNAME

This variable is set to the name of the report that is being generated.

&$RWDESC

This variable is set to the brief description of the report being generated.

&$RWSYSTEM

This variable is set to the system name that is defined in the table entry for the report application to which the report belongs. System name can be overridden on the GENERATE call to $RWCALL.

&$RWRECCAT

This variable is set to the record category that is defined in the table entry for the report application to which the report belongs. Record category can be overridden on the GENERATE call to $RWCALL.

&$RWOWNER

This variable is set to the user ID of the user who is to own the report. This user ID is passed to PSM as the owning user ID for the report.

&$RWPAGES

This variable is set to the number of pages that have been printed.

&$RWLINES

This variable is set to the number of lines that have been printed.

&$RWRECS

This variable is set to the number of records that have been processed.

&$RWCRITTOTAL

This variable is set to the number of &$RWCRIT*n* variables that contain data criteria. The value of this variable is in the range 0 to 99999.

&$RWCRIT*n*

These variables are set to data criteria if &$RWCRITTOTAL is greater than zero. The variable *n* is a number in the range 1 to the value of &$RWCRITTOTAL. These variables are to be used by the service procedure in determining the data to be included in the report. The format of these variables is dependent on the service procedure.

&$RWUSRD*c*

This is user data and is as set by the caller of $RWCALL or the report exit procedure. The variable *c* is between 0 and 5 alphanumeric and/or national characters. These variables are used to pass user data through the generator to the service procedure and can also be modified by the service procedure. They can also be modified by the report exit procedure. These variables are never set or cleared by the system and must be completely managed by your installation defined NCL procedures.

&$RWSFFLD*n*

This variable is set to the name of the data field defined as a sort field in the report. The variable *n* is a number in the range 1 to 10, corresponding to the sort field number assigned to the sort field.

&$RWSFDIR*n*

This variable is set to the order defined in the report for the corresponding sort field. The variable *n* is a number in the range 1 to 10. This variable may be as follows:

A

Ascending order

D

Descending order

&$RWSFSTART*n*

This variable is set to the start offset defined in the report for the corresponding sort field, if the records are not to be sorted using the full value of the field. The range is 1 to 255. The variable *n* is a number in the range 1 to 10.

&$RWSFEND*n*

This variable is set to the end offset defined in the report for the corresponding sort field, if the records are not to be sorted using the full value of the field. The range is the start offset to 255. The variable *n* is a number in the range 1 to 10.

Return Codes

&RETCODE = 0

Indicates successful completion, continue processing. When &$RWOPT is set to GETSF the following variable must be set: &$RWSFVAL*n* must be set to the value of the sort fields for the next record that will be retrieved for processing. Where *n* is the sort field number.

&RETCODE = 4

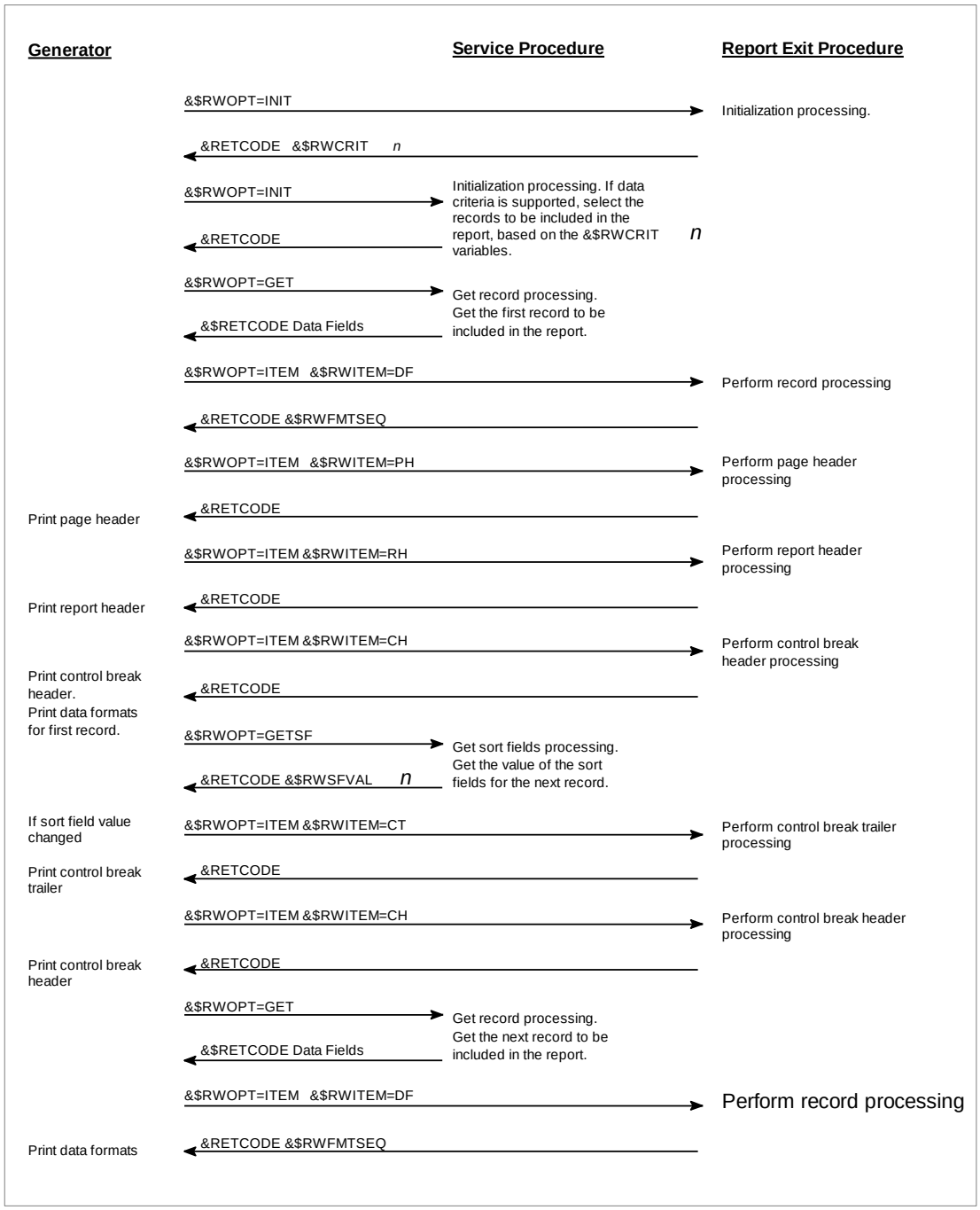
Indicates the end of data if &$RWOPT is set to INIT or GET.

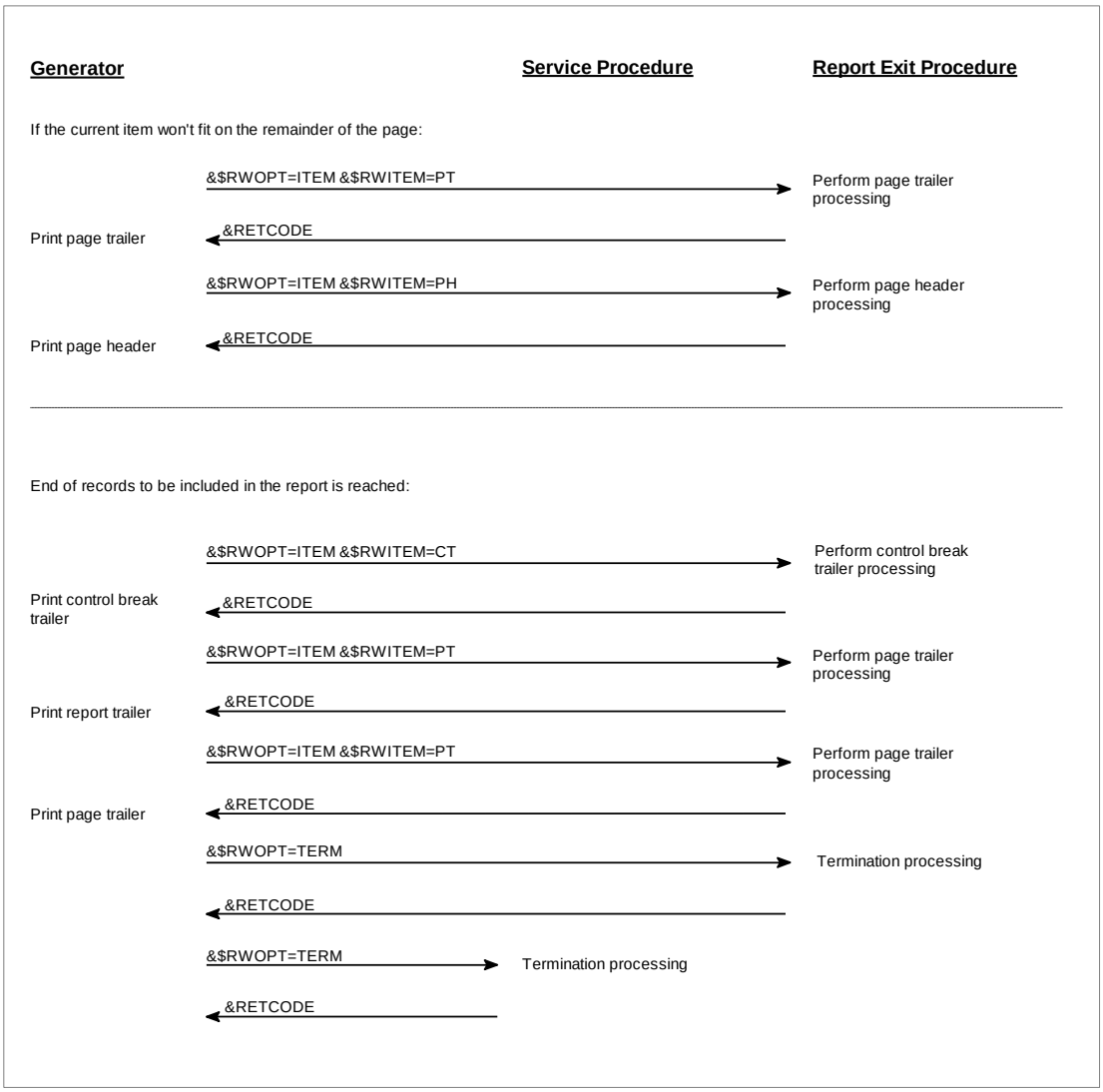
&RETCODE = 8

Indicates that an error occurred, terminate processing. &SYSMSG may be set to an error message.

Generator Logic Flow

The diagrams in this section illustrate the flow of logic as the generator passes control to the service procedure and the report exit procedure during generation of a report that consists of all component types.





Distributed Service Procedures

A Report Writer report application consists of several attributes, one of these being the name of a service procedure. The purpose of the service procedure is to provide the generator with the data that is to be used to generate a report.

Some components of system services provide a Report Writer service procedure that can be used to generate reports for that feature. This appendix describes the service procedures that are distributed with your product.

Note: These service procedures must not be modified.

Distributed Service Procedures

The names of the distributed service procedures are as follows:

$ADRW50Z

MODS Reports

$NDRW01Z

NDB Reports

$NWRW01Z

NEWS Reports

$IMRW27Z

CA SOLVE:InfoMaster Application Reports

$IMRW29Z

CA SOLVE:InfoMaster System Reports

$OSRW85Z

Object Services Application Reports

$UARW01Z

UAMS Reports

Report applications, stored as entries in $RWAPPL tables, are also distributed with your product. These report applications have one of the procedures listed as their service procedure. Each distributed service procedure has particular values that must be specified for each field in a report application when the procedure is specified as the service procedure.

Following is a section on each procedure describing the values that must be specified in the report application.

MODS Reports

The service procedure for MODS reports is $ADRW50Z. When defining a report application using this service procedure, the text fields must be set as follows:

I/M Application?

Set to NO.

System Name

Null.

Record Category

Set to the application ID followed by the category of the MODS records on which the reports are to be based.

Maximum Sort Fields

Set to 0.

Sort Order Support

Set to A.

Sort Offset Support

Set to NO.

The IDs of the distributed report applications that use this service procedure, all begin with the characters $AD.

This service procedure supports data criteria. The data criteria must be in the form of a boolean expression. For details on the syntax of a boolean expression, see the description of the &BOOLEXPR verb in the *Network Control Language Reference Guide.*

NDB Reports

The service procedure for NDB reports is $NDRW01Z. When defining a report application using this service procedure, the text fields must be set as follows:

I/M Application?

Set to NO.

System Name

Set to the file identifier of the NDB on which the reports are to be based.

Record Category

Null.

Maximum Sort Fields

Set to 0.

Sort Order Support

Set to A.

Sort Offset Support

Set to NO.

The ID of the distributed report application that uses this service procedure is $NDSYS. This service procedure does not support data criteria, meaning it ignores the &$RWCRIT*n* variables.

NEWS Reports

The service procedure for NEWS reports is $NWRW01Z. When defining a report application using this service procedure, the text fields must be set as follows:

I/M Application?

Set to NO.

System Name

Set to the file identifier of the NEWS file on which the reports are to be based.

Record Category

Set to ATTN for attention records, EVENT for event records, RTM for response time monitor statistics and TRAFFIC for traffic or error statistics.

Maximum Sort Fields

Set to 1.

Sort Order Support

Set to A.

Sort Offset Support

Set to NO.

The IDs of the distributed report applications that use this service procedure are $NWATTN, $NWEVENT, $NWRTM and $NWTRAF.

This service procedure does not support data criteria, meaning it ignores the &$RWCRIT*n* variables.

CA SOLVE:InfoMaster Application Reports

The service procedure for CA SOLVE:InfoMaster application reports is $IMRW27Z. When defining a report application using this service procedure, the text fields must be set as follows:

I/M Application?

Set to YES.

System Name

Set to the name of the CA SOLVE:InfoMaster system identifier on which the reports are to be based.

Record Category

Set to the name of the category on which the reports are to be based.

Maximum Sort Fields

Set to 7.

Sort Order Support

Set to MIXED.

Sort Offset Support

Set to YES.

The IDs of the distributed report applications that use this service procedure are $SAIMCF, $SAIMCH and $SAIMPB.

This service procedure supports data criteria. The data criteria must be in the format of an &NDBSCAN scan-expression. For an explanation of the &NDBSCAN scan-expression, see the *Network Control Language Reference Guide*. If data criteria is not specified, all the records defined in the category specified in the Record Category field are read from the database.

CA SOLVE:InfoMaster System Reports

The service procedure for CA SOLVE:InfoMaster system reports is $IMRW29Z. When defining a report application using this service procedure, the text fields must be set as follows:

I/M Application?

Set to NO.

System Name

Null.

Record Category

Null.

Maximum Sort Fields

Set to 0.

Sort Order Support

Set to A.

Sort Offset Support

Set to NO.

The ID of the distributed report application that uses this service procedure is $IMSYS.

This service procedure does not support data criteria, meaning it ignores the &$RWCRIT*n* variables.

UAMS Reports

The service procedure for UAMS reports is $UARW01Z. When defining a report application using this service procedure, the text fields must be set as follows:

I/M Application?

Set to NO.

System Name

Null.

Record Category

Null.

Maximum Sort Fields

Set to 0.

Sort Order Support

Set to A.

Sort Offset Support

Set to NO.

The ID of the distributed report application that uses this service procedure is $UASYS. This service procedure does not support data criteria, meaning it ignores the &$RWCRIT*n* variables.

Chapter 12: Mapping Services Facility

This section contains the following topics:

Mapping Services

Mapping Services is a facility that gives NCL access to complex data structures. It lets NCL procedures deal with the logical relationships and usage of data (the data protocol), while the system manages and maintains the physical representation of the data (the data format).

More information:

Map Library (see page 35)

Abstract Syntax Notation One

ISO 8824 defines the Abstract Syntax Notation One (ASN.1) language as the standard mechanism for describing abstract data structures. The Mapping Services facility has adopted this language as the means to describe the logical (or abstract) data items within a data structure. The physical representation of the data can be defaulted from the ASN.1 definitions, or can be explicitly described by the inclusion of implementation specific definitions. This aspect is discussed in more detail later.

ASN.1 Type Assignments

ASN.1 defines data in terms of types. There are ASN.1 defined basic types that describe both simple data items, and constructed data items. By using these types as a foundation, ASN.1 allows the definition of user types. By combining data components of various types to form a new data type, complex data structures can be described.

The basic instrument of definition in the ASN.1 language is the type assignment. This lets the programmer specify the name of a user type, and assign to that name one of the ASN.1 basic types (or perhaps another user defined type). If the type assigned is an ASN.1 constructed type, the definition is expanded to describe each of the components that comprise the structure. The constructed types are as follows:

- SEQUENCE
- SEQUENCE OF
- SET
- SET OF

If a type is defined as a SEQUENCE (for example), then the components that comprise that sequence are listed as part of the type assignment. Each component in the constructed type is identified by a component name and its type.

A similarly useful type which is normally constructed is the type CHOICE which allows the definition of a number of alternate data components, one of which is chosen to complete an instance of data within a larger data structure.

As well as these five types there are a number of simple types, including INTEGER, BOOLEAN and various character string types. These are introduced later.

Type definitions can be reused to describe the underlying characteristics of different data items. For example, a data component named *errorCount* and another named *dayOfWeek* can both be defined as INTEGER, although they should not be confused by a programmer. Hence it is not the type that is significant in referring to the data items, but rather the component names.

Defining the Logical Structure of Data

Mapping Services uses ASN.1 as the source language for describing the logical structure of data. Any number of ASN.1 type assignments can be combined to form a map. Such a map is capable of describing a single data unit, or any number of differing data units. However the component that is associated with the map itself (and not structures defined within the map) must be defined as a single user type.

This is best explained by way of an example. Consider a Customer Order file that contains two types of record. One record type is called CustomerDetail and it contains customer specific information, such as name, address, contact, and so on. For each record of this type there are zero or more records of a type called OrderDetail, each of which contains specific order details related to the customer. The CustomerDetail and OrderDetail records are quite different so you could choose to use a separate map for each, but equally you can specify them in a single map as follows.

The map for the file is defined as being of type CustomerOrder. This is a user defined ASN.1 type, so a type assignment for CustomerOrder must form part of the source definition for the map. It need not be the first type assignment in the source definition but it is usually convenient in understanding the map definition if it is.

Since, in the above example, there are two types of possible record on the file, the CustomerOrder type assignment is best described by the ASN.1 type CHOICE. Its ASN.1 language definition might look like:

```
CustomerOrder ::= CHOICE {
        customer  CustomerDetail
        order     OrderDetail }
```

This means that the type CustomerOrder is a choice of either a component named *customer* which is of type CustomerDetail, or a component named *order* which is of type OrderDetail. Hence, although we have a map which consists of a single type, it immediately diverges into the two possible different record types that the file can contain. Subsequent type assignments specify the nature of CustomerDetail and OrderDetail. These are likely to be structures themselves, for example:

```
CustomerDetail ::= SEQUENCE {
        name      GraphicString
        address   Address
        openOrdersINTEGER }

Address ::= SEQUENCE {
        number    INTEGER
        street    GraphicString
        suburb    GraphicString }
```

and hence can contain either simple data items, such as *openOrders* and *name*, that are defined as the ASN.1 basic types INTEGER and GraphicString, or constructed data items such as *address*, which is of a user defined type Address. Eventually, through further type assignments, such as the one for the type Address shown, all structures are defined down to their elementary components.

Referencing Logical Data Structures from NCL

An NCL programmer can connect a data structure to a map in a number of ways. For example, during retrieval from a file:

```
&FILE GET ID=CUSTFIL MDO=NEXTREC MAP=CUSTOMER
```

The user supplies a local name to the MDO (in this case, NEXTREC) when reading it from the file, and nominates the map to be used to define the record contents (in this case, CUSTOMER). If you continue with the sample map described above, this record is defined as a CHOICE type. This means that you can refer to either NEXTREC.CUSTOMER, if the record is of type CustomerDetail, or NEXTREC.ORDER, if the record is of type OrderDetail.

You might know what type to expect (for example by understanding the key used to retrieve the record) and can immediately reference the inner structure. Otherwise, assuming that the records can be distinguished by Mapping Services due to some physical characteristic in the record itself, the component name of the choice within a particular record can be determined by using a query function.

In either case, reference to subsequent structures can proceed. The rules are simple. If a component is defined as a constructed type, (one of SEQUENCE, SEQUENCE OF, SET, SET OF, or CHOICE), then the one or more components within that construction are referenced by using the constructed component name, followed by a period, then the inner component names.

Hence in the example, the following logical components of a CustomerDetail record can be referenced from NCL:

```
NEXTREC.CUSTOMER.NAME
NEXTREC.CUSTOMER.ADDRESS.NUMBER
NEXTREC.CUSTOMER.ADDRESS.STREET
NEXTREC.CUSTOMER.ADDRESS.SUBURB
NEXTREC.CUSTOMER.OPENORDERS
```

Defining the Physical Structure of Data

A Mapped Data Object is simply a series of bytes. A subset sequence of these bytes can compose a logical structure. Some structures can be explicitly identified by the presence of enclosing lengths and some form of identifier that prefaces the data. Other logical structures might have no explicit identification but are defined only by external knowledge of the meaning of some sequence of bytes. Whatever the case, the physical representation of the data must be known to Mapping Services so that it can translate any NCL reference of a logical structure into the correct access technique corresponding to the physical data construction.

The system provides default rules for structuring data. If an MDO is created and always processed by NCL, you can choose to accept the default physical structure provided by the system. However, if an external data source is to be processed, or if a specific format needs to be devised for communication with some other program, then control of the physical representation is necessary.

Component Tags

Mapping Services allows the physical representation to be described by defining, for each unique component name referenced in a map, a unique *tag* (alternatively known as an identifier, or key), which is an integer that can be placed within the data to identify a logical component. If the component is constructed, the way in which its contents are represented can also be defined. This ensures that for a constructed component, each of the separate items within it can be isolated according to a common access technique and each can be independently identified by their unique tag values.

Local Form

The definitions required to support the physical representation of data are not part of the defined ASN.1 language. This is because ASN.1 does not define how data ought to be represented, but only its abstract syntax. An implementation of ASN.1 represents data in a local form that is convenient for subsequent processing in the local environment.

This allows the physical representation, and hence the local form, to be customized, and subsequently ensures that the data conforms to this defined view.

To allow the customization of local form data, the ASN.1 Compiler recognizes a sequence of characters (which by definition are comments to ASN.1) and interprets the contents as implementation specific directives. It is these directives that can be used to instruct Mapping Services on the physical nature of the underlying data.

Data Interchange Between Open Systems

The other important aspect of ASN.1 is that the data it defines can be encoded for transmission. The product region supports the encoding and decoding of data using the Basic Encoding Rules (BER), ISO 8825. BER is specifically designed to interact with the logical structure as defined by ASN.1.

Independently of local form (that is, regardless of the physical representation of the data used for any MDO), the BER encoder can be used to build a data stream corresponding to the ASN.1 definition. This data stream can be understood by any BER decoder that has the same ASN.1 definition, even if it is a completely different ASN.1 platform.

For details on BER encoding and decoding, see the NCL &ENCODE and &DECODE verb descriptions in the *Network Control Language Reference Guide*.

Map Source Definitions

In the following sections the way in which ASN.1 is implemented is explored, however it is not the intention to give a detailed explanation of all aspects of the ASN.1 language. You should refer to the official ISO documents for a complete understanding.

Only those aspects important and useful in understanding the mechanics of the Mapping Services implementation are discussed in depth.

Maps and ASN.1 Modules

A set of ASN.1 source definitions that is compilable is termed an ASN.1 module. ASN.1 modules can be registered in the Map Library as a map. Modules are identified within the ASN.1 source, and this name must correspond to the registered map name. An ASN.1 module can also have a registered and unique identifier, called an object identifier, that officially identifies this module amongst all registered objects.

In general ASN.1 modules have a very free syntax, but Mapping Services imposes some restrictions on this for practical reasons.

Mapping Services Considerations

Mapping Services restricts all names, both component and type identifiers, to 32 characters in length. ASN.1 defines that a component name must start with a lowercase alphabetic character, and a type name with an uppercase alphabetic. Remaining characters can be chosen from the set of alphanumeric characters plus the hyphen. The Mapping Services compiler does not permit a hyphenated name, but instead allows the underscore (_) character.

The ASN.1 compiler does not support ASN.1 macros, nor does it support complex subtyping. When introducing ASN.1 source containing macros or complex subtypes you must edit the source to resolve macros manually, and remove complex subtyping (possibly employing new code to perform value checks).

The ASN.1 compiler does support simple subtyping such as sizes, integer and real value ranges and character set constraints.

The ASN.1 compiler ignores value assignments. The compiler checks the assignments, but generate no output. However, named values appearing after a type assignment are supported.

Support is provided for the ASN.1 import facility, however imports are resolved during map load, and not during compilation. Any type definition can be imported from another module, provided it is marked for EXPORT within that module. The compiler produces an IMPORT list of types and the containing maps. All such maps must be compiled for the import to be successful on map load. Because importing takes place at map load time and not at compile time it is possible for a map load to fail due to incomplete or inconsistent definitions.

ASN.1 Module Layout

An ASN.1 module has a layout depicted as follows:

Note: Uppercase values are entered as shown. Lowercase values are expanded in the actual source. Angle brackets (<>) denote optional definitions and are not part of the source. The vertical bar (|) denotes alternatives and is not part of the source. Underlined values are defaults.

```
module identifier < module oid >
DEFINITIONS < EXPLICIT TAGS | IMPLICIT TAGS > :: =
BEGIN
< EXPORTS export list >
< IMPORTS import list >
< type assignments >
< value assignments >
END
```

ASN.1 Comments

Comments can be used anywhere within the module source. They begin with a sequence of two adjacent hyphens (--) and terminate with another pair, or at the end of the current line, for example:

```
-- this is an ASN.1 comment that ends here --
-- this is an ASN.1 comment that extends to the end of the line
```

Module Identifier

The module identifier names the module and can optionally provide the unique object identifier under which the module is registered.

Module Definitions

Following the DEFINITIONS keyword the default tagging options can be set to EXPLICIT TAGS (the default) or IMPLICIT TAGS. Tagging is discussed in more detail later. The assignment sequence (::=) is then followed by the BEGIN keyword. The BEGIN and END keywords bracket the module body.

Exports

The module body begins with an optional export sequence. This declares those definitions from within the module that are externally referencable. It takes the form:

```
EXPORTS symbol < ,symbol,symbol,...,symbol >
```

where each symbol is a *typeReference* or *valueReference* from the module body.

Imports

Exports are followed by an optional import sequence. This declares those external definitions that are required to complete the definitions within this module. It takes the form:

```
IMPORTS symbol < ,symbol,symbol,...,symbol > FROM module
     <  symbol < ,symbol,symbol,...,symbol > FROM module >
```

where symbol is a *typeReference* or *valueReference* defined in the identified module which is a module identifier.

Type Assignments

The main portion of the module body usually comprises a number of type assignments. Each type assignment has the form:

```
typeReference ::= typeDefinition
```

where the *typeReference* is the name of a user defined type. The *typeDefinition* expands this user type according to any of the ASN.1 options which are discussed later.

Value Assignments

The module body can contain a number of value assignments. Each value assignment has the form:

```
valueReference type ::= valueDefinition
```

where the *valueReference* is the name of a user defined value, of the *type* referenced, and is assigned the *valueDefinition* referenced.

ASN.1 Compiler's Interpretation of the ASN.1 Module

This section describes the characteristics of the ASN.1 compiler.

Use of Comments

The compiler ignores ASN.1 comments that are not interpreted as implementation specific directives. It accepts Mapping Services directives, for manipulating the local form data, embedded in ASN.1 comments that start with the sequence (--<) and terminate with the sequence (>--). The Mapping Services directive start sequence must be explicitly terminated with the end sequence:

```
--< directive1,directive2,...,directiven >--
```

The allowable directives, and where they can appear in the module source, are explained in more detail later.

Module Identifier

The ASN.1 module identifier is checked by the compiler to be the same as the map name that is being compiled. (This check is not case sensitive, only the uppercase version of the names must agree). If an object identifier is included, all integer values must be explicitly present for each portion of the registration arc.

Default Tagging

The compiler accepts the default tagging information as supplied.

Exports

The compiler accepts any EXPORT sequence. Only typeReferences and valueReferences named in the export sequence can be imported by other maps. Imports and exports are resolved at map load time.

Imports

The compiler accepts any IMPORT sequence, but the definitions are not imported during compilation. The list of imports are remembered in the compiled output, and the requested definitions imported only during map load in their compiled version. Hence compilation is performed without resolving that all type definitions exist for the module. However, the load fails unless resolution is complete following the import phase.

When a *typeReference* is imported, all contained component definitions are imported, plus all supporting typeReferences. Thus importing represents a convenient technique for rationalizing definitions such that several specific maps can include a set of common definitions. Import resolution is discussed further in the section dealing with map loading. In the map being used for importing, the export sequence must contain the name of the Type Reference that is being imported.

Type Assignments

The compiler processes all type assignments, producing compiled output for each type and its constituent components. All ASN.1 base types are supported, except the EXTERNAL type. (If required, it can be defined as a user defined type). Each type is fully syntax checked, but cross-checking between individual components and their defined types is not performed. During map load the fully resolved map definitions are validated further to be logically consistent.

Value Assignments

The compiler accepts value assignment definitions but produces no output. These definitions are not used in NCL.

Map Components

Components are the data entities defined in a map that can be referenced by NCL. Every component has some data type associated with it. Data types give components their properties, but the types themselves are not referenced by NCL. A component's type can be a user defined type, which is defined by a type assignment within the source, or it could be a primitive ASN.1 type. Even where a component's type is a user defined one, by going to that type assignment (and possibly repeating this process), the type assignment specifies a primitive ASN.1 type. This primitive type is referred to as the base type for the component.

Component Definition

Although each component is of some type, the components themselves can only be defined within a type assignment that has a base type which is a constructed type, that is, one of SEQUENCE, SEQUENCE OF, SET, SET OF, or CHOICE. Hence to define any components within a map at least one type assignment referencing one of the constructed types is required.

For a data component to be usable, ASN.1 only requires it to have a type. The compiler requires that the component is named (except in some circumstances discussed later). The component name must be unique (in its uppercase form) within the construct in which it is defined, and is the name used by NCL to reference that particular data entity.

Mapped Data Object as a Component

When an NCL procedure creates an MDO, or receives one through an NCL verb, it connects explicitly or implicitly to a map. It is convenient to think of the MDO itself as a component. However, the MDO name is supplied by the user and is not defined within the map.

This component name (the MDO name) forms the first name segment when referencing any component defined within the MDO. As for other components, this first component is given a type, called the map type, which is the first typeReference encountered in the module body.

Component Name Hierarchy

When, as is typical, the map type is a constructed type, then it contains the definitions of one or more components that do (or can) comprise the data elements for that type. These components are referenced from NCL by concatenating each component name to the MDO name separated by a period, for example:

```
MDO=problem.number
```

where the MDO created by the user is the component problem, and the component number is defined within the component list for the map type. Likewise, if a named component itself has a type which is constructed, then the components defined within its type form the next level in the component name hierarchy.

Map Closure

A map is given a map type by the compile process. This map type is the first type definition encountered in the source. As described above, this type is typically a constructed type, containing the definitions of its contained components. Each of these components form the next level of the name hierarchy within the MDO. If they are constructed, their components form the next level of the name hierarchy, and so on.

Overall, the relationship between components and their types maps out a complete hierarchical structure where the type names form the linkage between component names.

If a module contains type references which cannot be reached by following the reference linkages, starting from the map type and working down, then these types are not usable within this map. Such isolated definitions can be the subject of an import statement from some other map, and hence such orphaned types are allowable. However, it is possible that they were intended to form part of the module in which they reside, and an unreachable reference is an indication that one of the following has occurred:

- The wrong type was selected as the map type. (That is, a branch further down the tree than intended was the first type reference and has become the map type, leaving higher level types unreachable). This problem can be resolved by ensuring that the correct type reference is the first in the module.
- The definitions as entered have more than one possible entry type. (That is, there are a number of separate type and component hierarchies defined such that they do not completely overlap). This problem can be resolved by introducing a superior type containing components that refer to each of the possible entry types (for example, by making it a CHOICE of these types).

This process is termed map closure, and helps ensure the completeness and consistency of the supporting map definitions for use by NCL.

Data Tagging

This section describes the data tags used in Mapping Services.

ASN.1 Tags

Maps can be developed from ASN.1 for the purpose of driving a transfer syntax, such as BER, in order to communicate in an open systems environment. The ASN.1 definitions describe how the data is serialized for transmission, and how each data component is tagged so that data items can be distinguished.

An ASN.1 tag consists of a tag class, and a tag number. There are the following ASN.1 tag classes:

- **UNIVERSAL** tags—are defined by ISO bodies.
- **APPLICATION** tags—are unique throughout an application.
- **PRIVATE** tags—are defined by private agreement.
- **CONTEXT-SPECIFIC** tags—have meaning in the immediate context only.

The tag number is an integer value greater than 0.

When an ASN.1 tag is encoded both the class and number are used to create a unique value such that the class and number remain separately decipherable.

Explicit and Implicit Tagging

Each ASN.1 base type has a predefined tag number in the UNIVERSAL tag class. Since all data components must be of a base ASN.1 type a default tag value exists for all data items. However ASN.1 allows these tags to be overridden and data to be explicitly tagged with any tag value. In fact, any data component can be tagged more than once, the tags being applied left to right in the order defined.

For example:

```
component1  [ APPLICATION 23 ]  [ 3 ]  INTEGER
```

produces the following tags for *component1*:

- APPLICATION class, tag number 23
- CONTEXT-SPECIFIC (the default) class, tag number 3
- UNIVERSAL class, tag number 2 (for INTEGER)

In general, where a data component is explicitly tagged, it is followed by either more explicit tags, or its UNIVERSAL tag for the underlying base type as shown in the example above. However, use of the IMPLICIT keyword before a tag, or before a typeReference, indicates that the next tag is understood (implied), and is not encoded.

For example:

```
component1 [ APPLICATION 23 ] IMPLICIT INTEGER
```

produces just one tag: APPLICATION class, tag number 23.

The INTEGER tag is implied and not encoded due to the presence of the IMPLICIT keyword.

Note that following the DEFINITIONS keyword in the module header the tagging defaults can be set for the module. The default is EXPLICIT TAGS indicating all tags are produced unless the IMPLICIT keyword is used to override. The IMPLICIT TAGS options can be used to indicate that only the first tag encountered for a component is used, the rest being implicit (except for the types CHOICE and ANY DEFINED BY, when all tags apply).

MDO Tags

To differentiate between data items maintained internally in their local form, Mapping Services also requires that all components have a recognizable tag.

Because a tag value prefixes most components in the MDO local form, the MDO is a self-defining structure that allows, for example, an MDO to be stored and later retrieved while the data within remains separately identifiable even where the map might have altered slightly. As long as map maintenance does not change the relationship between tag values and the components they represent, maps can be extended and modified without affecting the ability to process an MDO defined under a previous map version.

Mapping Services ignores tag classes, and simply uses an integer value as a tag. However, the MDO tag numbers must be unique within a logical structure in the same way that component names must be unique within that structure. Hence, within any structure (such as a sequence or a set), all components must have unique names (enforced by ASN.1) and corresponding to each component a unique MDO tag number (enforced by Mapping Services).

These MDO tags can be set explicitly and independently of ASN.1, or can be generated by the compiler. However, the compiler can be directed to use the ASN.1 tag values as MDO tag values. This is likely to be the case where the map being developed has no requirement for BER encoding, hence all tag values are open to interpretation by Mapping Services only.

Note: By defining the tags explicitly in the source data, the user is afforded a greater level of protection against change. MDOs that are created using one version of the map can often continue to be compatible with higher versions of the map, even though new components are added. However, if the compiler generates the tags, this is unlikely to be the case.

Mapping Directives

This section describes the directives available within Mapping Services.

MDO Tag Generation

Before the first ASN.1 statement in a module is encountered, one of the following directives can be used to indicate how MDO tags are to be generated.

```
--< TAGS(ANY) >--      compiler is to generate tags
--< TAGS(EXPLICIT) >-- tags are explicitly coded in the source
```

ANY is the default, meaning that the compiler generates a unique tag value for each component reference in a module. Even where a component name occurs more than once in the module it is given a separate tag. In general this means that the actual tag values used are unpredictable across compiles. If there is no desire to control any MDO tag values, and no requirement to store MDOs in their local form, then this is the easiest way to define MDO tagging.

If EXPLICIT tagging is used, the compiler assumes that the first explicit ASN.1 tag found for a component is in fact the MDO tag value, regardless of its tag class. If no explicit tag is found for a component, the compiler generates a tag value for it as described above for TAG(ANY). Note that this means that the compiler does not use the default tagging for ASN.1 base types as they are likely to lead to ambiguities.

When defining explicit tags care must be taken to ensure that a tag value is used only once within a given structure. However, this tagging technique is very useful where control over tag values is desirable but there is no requirement to BER encode the data, and hence the tags used throughout the module need only be thought of as being MDO tags.

Alternatively, where it is necessary to BER encode the data, but it is also desired to control the MDO tags, the need might arise to use explicit MDO tags that are not part of the ASN.1 source. This can be done by using the following compiler directive following the component name being tagged:

```
--< [ nn ] >--
```

This directive follows the component name, but precedes any explicit ASN.1 tag or type information, for example:

```
userid        --< [ 10 ] >- [ PRIVATE 23 ] GraphicString
```

In this example, the *userid* component has an MDO tag value of 10, but if BER encoded, it is tagged with a private tag value of 23, and possibly the GraphicString universal tag, depending upon whether the tag option is set to IMPLICIT or EXPLICIT. The tag option immediately follows the word DEFINITIONS in an ASN.1 module.

Setting the MDO tag value this way always overrides any other MDO tag generation option.

Local Form Control

Mapping Services is capable of maintaining MDOs according to a number of local form rules. By default it assumes all components are variable length items consisting of a tag (or key), a length, and the data itself. The following Mapping Services directives can be placed in the source to control the local form of data (*k* and *l* are integers in the range 0 to 4, where *k* is the length of the key, and *l* is the length of the length bytes):

```
--< KL0(k,l) >--tag, length, then data, length is of
                data only.

--< KL1(k,l) >--tag, length, then data, length is of
                length bytes + data.

--< KL2(k,l) >--tag, length, then data, length is of
                tag +length + data.

--< LK0(l,k) >--length, tag, then data, length is of
                data only.

--< LK1(l,k) >--length, tag, then data, length is of
                tag bytes + data.

--< LK2(l,k) >--length, tag, then data, length is of
                length + tag + data.
```

Such rules apply to structured components only. It is important to note that the rule applying to a structured component describes the manner in which its embedded components are managed and not the structure itself. The way in which the structure itself is managed depends upon the rule applying to its parent structure.

These directives can be used before any ASN.1 type assignment statement to set the compiler default for components encountered in the source module after that point, for example:

```
--< KL0(2,2) >--
CustomerOrder ::= CHOICE {
        customer   CustomerDetail
        order      OrderDetail }
```

In this example the components *customer* and *order* carry the KL0(2,2) encoding rule such that their embedded components (if any) are managed according to that rule.

This directive can also be applied to an individual component without changing the compiler default, for example:

```
--< KL0(2,2) >-
CustomerOrder ::= CHOICE {
        customer   CustomerDetail
        order  --< KL0(4,4) >--OrderDetail }
```

In this example the *customer* component carries the compiler default KL0(2,2) encoding rule, while the *order* component carries the KL0(4,4) rule.

Note that in both the above examples, the *customer* and *order* components must be managed in the same manner. This is described by the encoding rule carried in the component which is of type CustomerOrder (not shown here).

Mapping Services can be used to map existing data structures and to accommodate this it can also manage components that do not have explicit length and tags in the data. Such data is considered fixed and fixed components can only be specified within a SEQUENCE or SEQUENCE OF type definition, for example:

```
CreateDetails ::= SEQUENCE {
        userid   --< FIX(8) >-- GraphicString,
        date     --< FIX(8) >-- GraphicString,
                                 --YY/MM/DD--
        time     --< FIX(8) >-- GraphicString}
                                 --HH:MM:SS--
```

In this example the *userid* component occupies the first 8 bytes of the data, the *date* component the next 8 bytes, and the *time* component the last 8 bytes. Although tags are not used within the data, each component can still be defined with a tag value, or the compiler generates one.

```
--< FIX(x,y) >--
```

In this example, the component is of length x, and starts at offset y within its enclosing structure. If y (offset) is not specified, the offset is defaulted to the end of the preceding component, or 0 (zero) if there is no preceding component.

Default Formatting

All data types have both local form and external form defaults (see page 316). Local form refers to the format that data is kept in within an MDO; external form refers to the format of data as made available to, or supplied by, an NCL process.

However, some types have alternative formats that compiler directives can set and are explained in the following sections.

Local Form for INTEGER

The local form of INTEGER types is, by default, a 1 to 4 byte signed binary field. However, IBM packed decimal and zone decimal formats can be managed by setting the options as one of the following (to request binary, packed decimal or zone decimal local form respectively):

```
INTEGER--< BINARY >--
INTEGER--< PACK >--
INTEGER--< ZONE >--
```

External Form for REAL

The external form of REAL types is by default the NCL character representation of floating point numbers. However the external form can be modified by setting the number of integer digits and decimal places, for example:

```
REAL      --< EF(6,2) >--
```

This example requests up to six significant leading digits followed by two decimal places only.

Type Description and Formats

This section describes the types available within Mapping Services and their formats.

ASN.1 Types

The following simple types are defined by ASN.1, and most are supported by Mapping Services:

- BOOLEAN
- INTEGER
- BIT STRING
- OCTET STRING
- NULL
- OBJECT IDENTIFIER
- ObjectDescriptor
- REAL
- ENUMERATED
- NumericString
- PrintableString
- IA5String
- UTCTime

- GeneralizedTime
- GraphicString
- VisibleString
- GeneralString

One extension supported by Mapping Services is HEX STRING. This is identical to the ASN.1 type OCTET STRING except in the way it is presented to NCL, as described in a later section. In addition, the following types are supported by the compiler, but have no real application to NCL (and are supported as if they are OCTET STRINGS):

- TeletexString
- VideotexString
- ANY and ADB

There are also a number of constructed types defined by ASN.1, and all are supported by Mapping Services:

- SET
- SET OF
- SEQUENCE
- SEQUENCE OF
- CHOICE

How Mapping Services implements each of these types is discussed in the following sections.

NCL Reference, Type Checking, and Data Behavior

When referencing an MDO in an NCL procedure, Mapping Services validates that the named component is defined (according to the name hierarchy supplied), and that the data within the component is valid, according to its underlying ASN.1 type. Each ASN.1 type can contain only certain valid values. Mapping Services checks the data value when retrieving data from, or assigning data into, an MDO. An operation attempting to retrieve or assign invalid data is rejected by Mapping Services with a message indicating a type check error.

In order to perform type checking Mapping Services first determines the base ASN.1 type of the component. Where a component is of a user defined type, the base ASN.1 type of the user defined type is inherited by the component. It is possible to have a number of levels of indirection between a user defined type and its base ASN.1 type.

The valid NCL values allowed for each of the base ASN.1 types is termed the *external* form. In addition to the set of valid values for each type, a specific component can be further constrained in what values are acceptable. Such *constraints* can be the result of either ASN.1 definitions or compiler directives. Finally, when data representing a valid NCL value is accepted for a component update, it is subject to a transformation from external form to local form, which is the MDO internal representation of data. This process carries with it further constraints.

The valid external form values, and the behavior of data managed by Mapping Services, is described for each type in the following sections.

SET Type

The SET type allows the definition of a fixed number of components where order is irrelevant. Within the set each component must be given a name, unless its base type is CHOICE. When a set is transferred from one system to another the items in the SET can be sent in any order. Items in the set can be optional, as indicated by the OPTIONAL keyword, for example:

```
Contact ::= SET {
   name                [1]   GraphicString,
   title               [2]   GeneralString,
   businessNumber      [3]   NumericString,
   afterHoursNumber    [4]   NumericString OPTIONAL}
```

When referring to SET items in NCL they are referenced by name, and hence the order that the data is stored in within the set is unimportant, and is in fact arbitrary.

An item in a SET can be a CHOICE of a number of alternatives. Such an item need not be explicitly named, but in this case each alternative of the choice must be named, and each must be unique amongst all other items within the entire SET. Alternatively, the SET item that is a CHOICE might indeed be named, and in this case only that name must be unique, as the CHOICE components are pushed to the next level in the component name hierarchy. For example, the following is valid:

```
Contact ::= SET {
   name                [4]   GraphicString,
   title               [5]   GeneralString,
   businessNumber      [6]   NumericString,
   CHOICE {
           pagerNumber  [1] NumericString,
           homeNumber   [2] NumericString
          } ,
   additionalContact    [9] CHOICE {
                                    pagerNumber  [1] NumericString,
                                    homeNumber   [2] NumericString
                                   }
               }
```

This results in allowing the following component names to be referenced from NCL:

- Name
- Title
- BusinessNumber
- PagerNumber
- HomeNumber
- AdditionalContact.pagerNumber
- AdditionalContact.homeNumber

All SET items must be differentiated by an ASN.1 tag value. It is good practice to explicitly tag all SET items rather than default to tags of the ASN.1 base types.

External Form—Input and Output

All structures have an external form that is the same as their local form.

Local Form and Behavior

The local form of a SET is the data stream consisting of each set component, where each component has its leading tag and length (as determined by the parent component) and data in the local form for its type.

Named Values

Named values are not applicable to the SET type.

Constraints

Constraints are not applicable to the SET type.

SET OF Type

The SET OF type allows the definition of either an arbitrary or fixed number of items of the same type, where order is not important. The definition includes only the type of the SET item, and does not name the component, for example:

```
PokerHand ::= SET OF Cards
```

In NCL, SET OF items are referenced by index only, within their parent structure. In the example above, the following index values are used to reference the set items of a component named card of type PokerHand:

```
card.{1}
card.{2}
...
card.{n}
```

External Form—Input and Output

All structures have an external form that is the same as their local form.

Local Form and Behavior

The local form of a SET OF type is the data stream consisting of each set component, where each component has its leading tag and length (as determined by the parent component) and data in the local form for its type.

Named Values

Named values are not applicable to the SET OF type.

Constraints

An upper limit can be placed on the number of items that the set can contain by use of the SIZE keyword. For example:

```
Pokerhand ::= SET SIZE(5) OF Cards
```

It is also possible to specify a SIZE range but the lower bound is ignored. For example, the following is equivalent to the above:

```
Pokerhand ::= SET SIZE(2..5) OF Cards
```

SEQUENCE Type

The SEQUENCE type allows the definition of a fixed number of components where order is relevant. As for a set, within a sequence each component must be given a name, unless its base type is a CHOICE. When a sequence is transferred from one system to another the items in the SEQUENCE are sent in the order defined. Items in the sequence can be optional, as indicated by the OPTIONAL keyword.

Mapping Services allows items of a SEQUENCE, or a SEQUENCE OF, construct (and only these constructs) to be of a fixed length. All other components must have a variable length structure, with a tag value so that each component can be recognized. Mapping Services ensures the sequence is reflected in the local form data. A compiler directive is used to indicate fixed data items, for example:

```
Contact ::= SEQUENCE {
   name       --< FIX(20) >--  GraphicString,
   birthDate --< FIX(8) >--   GraphicString,
              -- YY/MM/DD
   age        --< FIX(3) >--   INTEGER
   sex--      --< FIX(1) >--   ENUMERATED {
                                          female(0),
                                          male(1) },
   address                [1]   Address,
   previousAddress        [2]   Address }
```

In this example the first four components are fixed, and occupy the first 32 bytes of the SEQUENCE. The last two component are variable structures, and follow in sequence after the first 32 bytes.

External Form—Input and Output

All structures have an external form that is the same as their local form.

Local Form and Behavior

The local form of a SEQUENCE is the data stream consisting of each sequence component, where each component has its leading tag and length (where applicable, as determined by the parent component) and data in the local form for its type.

Named Values

Named values are not applicable to the SEQUENCE type.

Constraints

Constraints are not applicable to the SEQUENCE type.

SEQUENCE OF Type

The SEQUENCE OF type allows the definition of either an arbitrary or fixed number of items of the same type, where order is important. The definition includes only the type of the SEQUENCE item, and does not name the component, for example:

```
Counters ::= SEQUENCE OF INTEGER
```

In NCL, SEQUENCE OF items are referenced by index only, within their parent structure. In the example above, the following index values are used to reference the sequence items of a component named *counters* of type Counters:

```
counter.{1}
counter.{2}
...
counter.{n}
```

External Form—Input and Output

All structures have an external form that is the same as their local form.

Local Form and Behavior

The local form of a SEQUENCE OF type is the data stream consisting of each set component, where each component has its leading tag and length (where applicable, as determined by the parent component) and data in the local form for its type.

Named Values

Named values are not applicable to the SEQUENCE OF type.

Constraints

An upper limit can be placed on the number of items that the sequence can contain by use of the SIZE keyword. For example:

```
Counters ::= SEQUENCE SIZE(50) OF INTEGER
```

It is also possible to specify a range, but the lower bound is ignored.

CHOICE Type

The CHOICE type allows the definition of a number of components, each of which is an alternative in the data structure. A CHOICE component can be named, such as the details component in this example:

```
Person ::= SET {
   name       GraphicString,
   birthDate  GeneralString, -- YY/MM/DD
   age        INTEGER,
   details    CHOICE {
              femaleInfo   [1]   FemaleInfo,
              maleInfo     [2]   MaleInfo } }
```

The *details* component can contain either *femaleInfo* or *maleInfo*, but not both. In this case the *details* component containing the choice can be referenced directly without first knowing which choice was made. If this is changed to the following:

```
Person ::= SET {
   name       GraphicString,
   birthDate  GeneralString, -- YY/MM/DD
   age        INTEGER,
   CHOICE {
           femaleInfo   [1] FemaleInfo,
           maleInfo     [2] MaleInfo } }
```

then both *femaleInfo* and *maleInfo* are alternatives within the SET type, and must have unique names within that set. In this case, the actual choice that is present can only be discovered by attempting to reference each possible one, and determining whether it exists or not.

All alternatives of a CHOICE must have unique tags so that they can be differentiated. It is good practice to use explicit tags for each CHOICE item.

External Form—Input and Output

This is the same as the CHOICE item.

Local Form and Behavior

This is the same as the CHOICE item.

Named Values

Named values are not applicable to the CHOICE type.

Constraints

Constraints are not applicable to the CHOICE type.

BOOLEAN Type

The BOOLEAN type is used to represent a value of *true* or *false* only.

External Form—Input

The local character strings TRUE and FALSE (not case sensitive) are accepted, but the digit 0 is interpreted as false, and the digit 1 is true.

External Form—Output

The digit 0 (false) or 1 (true) is always returned.

Local Form and Behavior

Internally, Mapping Services stores a value of X'00' for false, and X'01' for true (and accepts any value other than X'00' as true).

For an input operation, where the component is variable length its length is always set to 1. Where the component length is fixed and is greater than 1 the value occupies the first byte only (it is left aligned) and the remainder of the component's data is set to zeros.

For an output operation, where the component is located and has a length greater than 1, only the first byte is inspected as the value.

Named Values

Named values are not applicable to the BOOLEAN type.

Constraints

Constraints are not applicable to the BOOLEAN type.

INTEGER Type

The INTEGER type is used to contain any positive or negative whole numbers in the range -2,147,483,648 to 2,147,483,647 (that is, a signed 32 bit number).

External Form—Input

Valid input consists of a string of up to 15 digits optionally preceded by a plus sign (+) or a minus sign (-), which provides the sign (positive or negative) of the value. The sign is positive, if omitted. All other characters must be valid digits (that is, 0 to 9). Alternatively, if the map definition included named values for this component, the symbolic name of the named value can be supplied as external form input.

External Form—Output

Output consists of a string of one or more local characters. If the integer value is negative, the first character is a minus sign (-), otherwise the sign is omitted. All other characters are numeric characters. Leading zeroes are stripped.

Local Form and Behavior

Internally, Mapping Services can store integers in one of the following formats:

Binary

Can be up to 4 bytes in length. If the length is not fixed, the value is kept in the smallest number of bytes possible. If the length is fixed, the value is right aligned and sign extended to the left.

Packed

Can be up to 8 bytes in length. The integer value is converted to the packed decimal equivalent. If the length is not fixed, the value is kept in the smallest number of bytes possible. If the length is fixed, the value is right aligned and zero padded to the left.

Zoned

Can be up to 15 bytes in length. The integer value is converted to the packed decimal equivalent. If the length is not fixed, the value is kept in the smallest number of bytes possible. If the length is fixed, the value is right aligned and zero padded to the left.

For any format, if a value exceeds that which can be stored without loss of significance a type check results. If a named value is input, the map definition is used to determine the actual integer value.

Named Values

It is possible to specify a range of names that correspond to particular integer values in an INTEGER type definition. For example:

```
x INTEGER   {red (1),
             green (5)
             blue (7)}
```

These names can then be used as external input to represent the corresponding value. On retrieval the integer value is returned.

Constraints

It is possible to constrain the allowed set of integers for an integer type. This is done by specifying a list of integer ranges and/or single values. For example:

```
( 1..10 | 20..30 | 50 | 100 )
```

In the above example the integer values are restricted to numbers 50, 100 and the range 1 to 30. If anything else is entered, a type check error results.

BIT STRING Type

The BIT STRING type is used to contain any data where individual bit values might have meaning. Mapping Services supports two types of BIT STRING access, Standard and Boolean. These are described below.

Standard BIT STRING Access

Standard BIT STRING access deals with the string as a whole, allowing manipulation of the entire component through a single operation, as for most other types.

External Form—Input

Valid external form can be a string of one or more digits, each a zero (0) or a one (1). However, where named values are defined for the BIT STRING type, a list of named values, each separated by a plus sign (+) or a minus sign (-) sign, is an acceptable alternative. A named value preceded by a plus sign indicates that the named bit value should be set to true (the bit is set to 1), and a named value preceded by a minus sign indicates that the named bit value should be set to false (the bit is set to 0).

External Form—Output

The output format depends upon whether or not named values are defined for the BIT STRING type. Where no named values are defined the output consists of a string of zero or more (always a multiple of 8) digits, each a 0 or 1. Where one or more named values do exist the output is a character string comprising each named value where the named bit is 1 (meaning the value is true). Each name is delimited with a plus sign (+).

Local Form and Behavior

When a string of 0's and 1's is supplied as input, each digit in the input sequence is treated (left to right) as the value of the corresponding bit in that position of the local form data. If the number of bits supplied is not a multiple of 8, trailing bits are set to zero and padded to a byte boundary. If the component has a fixed length exceeding that of the input string, the value is left aligned, and all unreferenced bytes are set to X'00'. If the component cannot contain the number of input bytes supplied, the string is truncated.

When a list of named values, each preceded by a plus sign (+) or a minus sign (-), is supplied as input, only the named bits take part in the operation. Each named bit preceded by a plus sign (+) is set to 1 (true), and each named bit preceded by a minus sign (-), is set to 0 (false). All other bits in the BIT STRING are unaffected by the input operation.

When fetching the value of a BIT STRING a named value list is always returned if any named values are defined for the type, else a string of 0s and 1s is returned corresponding to the BIT STRING values. Note that when named values are defined all other bits in the BIT STRING are ignored on output regardless of their value.

Named Values

It is possible to specify a list of names corresponding to particular bit positions, starting from zero. For example:

```
{ FLAG1(0), ALLSET (1), ...}
```

It is possible to use a list of these names, preceded by a plus or minus signs, to indicate set or not set as external form input for the bit string component. When output the names of the set bits, separated by plus signs are returned, for example:

```
+FLAG1+ALLSET ...
```

Constraints

It is possible to specify a size constraint on a bit string type using the SIZE keyword. For example:

```
BIT STRING (SIZE(2.4))
```

The size refers to the maximum number of bits (not bytes) in the bit string. Internally, the minimum is rounded down, and the maximum rounded up, to the nearest multiple of 8.

Boolean BIT STRING Access

Boolean BIT STRING access deals with individual bit level access and operates only through named values. We recommend this access because program access to bits is only via their symbolic named values, thus removing from NCL the need to know relative bit positions.

For Boolean BIT STRING access to be invoked the named value of a bit is provided by NCL as an additional name segment after the BIT STRING component name. Since the BIT STRING type is primitive, the additional name in the name hierarchy is understood to be a named value, and is treated as a BOOLEAN type. No matter where the named value is in the BIT STRING the value of the bit is always 0 or 1, as for a BOOLEAN type.

External Form—Input

The local character strings TRUE and FALSE (not case sensitive) are accepted, the digit 0 is interpreted as false, and the digit 1 is true.

External Form—Output

The digit 0 (false) or 1 (true) is always returned.

Local Form and Behavior

The component name plus the named value is treated as a reference to a specific bit (the bit position within the component being defined by the named value), and that bit is set to 0 or 1 depending upon the input. No other bits in the BIT STRING component are affected. If the component is extended to accommodate the input, all other bits are set to 0.

Named Values

Named values are not applicable to Boolean BIT STRING access.

Constraints

Constraints are not applicable to Boolean BIT STRING access.

OCTET STRING Type

The OCTET STRING type is used to contain any data where no formatting is required.

External Form—Input and Output

Any data is accepted and returned unchanged.

Local Form and Behavior

Data is stored as is. If the component has a fixed length exceeding that of the input string, the data is left aligned, and all unreferenced bytes are set to X'00'. If the component cannot contain the number of input bytes supplied, the string is truncated.

Named Values

Named values are not applicable to the OCTET STRING type.

Constraints

The SIZE keyword can be used to constrain the length of an octet string to a certain range or value. Length is measured in bytes. For example:

```
OCTET STRING (SIZE(4..8))
```

If the component is variable length, a type check error occurs if the size constraints are breached.

HEX STRING Type

The HEX STRING type is an Mapping Services extension to ASN.1, but is processed as a base ASN.1 type. It is identical in all respects to the ASN.1 OCTET STRING type except for its external form representation.

External Form—Input

Valid input consists of a string of one or more local characters, each selected from the set 0123456789ABCDEF. Each pair of hexadecimal characters represents a single byte value. If an odd number of characters is supplied, the string is treated as though padded on the left with a single zero (0).

External Form—Output

Data is returned in hexadecimal characters, as for input. An even number of characters is always returned.

Local Form and Behavior

Each two hexadecimal characters of input represents the actual data to be stored in a single byte. Otherwise, behavior is as for OCTET STRING.

Named Values

Named values are not applicable to the HEX STRING type.

Constraints

The SIZE keyword can be used to limit the size of a HEX STRING type. The size refers to the length of the local form, not external form.

NULL Type

The NULL type is used where data in a component either must be null (that is, empty), or not accessible.

External Form—Input

The only valid input is a null value.

External Form—Output

A null value is always returned.

Local Form and Behavior

The component can be created by an input operation, but no contents are modified. If it already exists, no data is modified.

Named Values

Named values are not applicable to the NULL type.

Constraints

Constraints are not applicable to the NULL type.

OBJECT IDENTIFIER Type

The OBJECT IDENTIFIER type is used to contain object identifier values that uniquely identify registered objects.

External Form—Input and Output

Any sequence of characters from the set 0123456789 and the period (.), is valid provided it does not begin or end with a period (.), contains no consecutive periods, and contains at least one period. Each sequence of decimal digits punctuated by a period (.), represents a sub-identifier in the series of sub-identifiers that comprise an object identifier value.

Local Form and Behavior

The data format is as supplied for input, however truncation is not allowed. If the component is fixed length, it must be able to contain the input string, otherwise a type check results. If necessary, it is padded with blanks.

Named Values

Named values are not applicable to the OBJECT IDENTIFIER type.

Constraints

Constraints values are not applicable to the OBJECT IDENTIFIER type.

ObjectDescriptor Type

The ObjectDescriptor type is used to contain object descriptions for registered objects.

External Form—Input and Output

All values are accepted as supplied.

Local Form and Behavior

The data format is as supplied for input. If the component is fixed length, a short input string is padded with blanks, and a long input string is truncated.

Named Values

Named values are not applicable to the ObjectDescriptor type.

Constraints

A character set constraint can be specified for ObjectDescriptor types. In addition, a size constraint can be specified.

More information:

GraphicString Type (see page 342)

EXTERNAL Type

The EXTERNAL type is not supported by Mapping Services. If required, define it as a user-defined type.

REAL Type

The REAL type is used to contain floating point, or scientific notation, numbers in the range 10-70 to 1070.

External Form—Input

Format allowed is as follows, such that either (or both) *nnnnnn* and *mmmmmm* are present, and the resulting REAL number is within the allowable range:

+ or -

(optional, plus or minus sign, followed by)

nnnnnn

(optional, any number of digits, followed by)

.

(optional, decimal place, followed by)

mmmmmm

(optional, any number of digits, followed by)

E*sxx*

(optional, signed exponent power of 10, range -99 to 99)

Examples:

```
1457892345509676544283940 4
-123.567
.555
.0023E-23
3.142776589E+66
```

External Form—Output

The default is a normalized decimal real number:

+ or -

(plus or minus sign of the value, followed by)

.

(decimal place indicator, followed by)

nnnnnn

(15 significant fraction digits, followed by)

E*sxx*

(signed exponent power of 10)

Examples:

```
+.314277658900000E-10
-.123456789000000E+52
```

A compiler directive is available to specify how a REAL type is to be presented externally. For example:

```
x REAL --< EF(3,5)>--
```

The above directive gives an external form for the number +.314277658900000E-10 as 3.14277.

This has three digits before the decimal point, (including two blanks) and five digits after the decimal point.

Space padding occurs on the left if required, and zero padding occurs on the right. No truncation takes place. The external form expands to allow all of the digits that precede the decimal point in a real number to be represented in full.

Local Form and Behavior

For IBM S/370 machines, local form is a 64-bit long floating point value, and the component must be at least 8 bytes in length. Truncation is not allowed. If the component has a fixed length greater than 8 bytes, the value is left aligned and padded on the right with zero bytes.

Named Values

Named values are not applicable to the EXTERNAL type.

Constraints

Real types can be restricted to a set of real value ranges or simple real values. For example:

```
REAL ({150, 10, -2}..{150, 10, -1})
```

The above example restricts the real type to the range (1.5...15). It is possible to specify a real range using whole numbers as well, for example:

```
REAL (1..20)
```

ENUMERATED Type

The ENUMERATED type is used to constrain a component to a defined set of values. Each defined value is named using a name identifier similar to a component name. Associated with each name is a unique integer value (which can be signed), for example:

```
Color  ::= ENUMERATED {  red(0),blue(1),yellow(2),
                         green(3),black(7) }
```

External Form—Input and Output

The external form must be one of the names listed in the ENUMERATED type. The enumerated values are not allowed (that is, *red* is valid, 0 is not).

Local Form and Behavior

Internally, the ENUMERATED value is kept in the same manner, and is subject to the same local form constraints, as an INTEGER of the binary local form.

Named Values

Named values are not applicable to the ENUMERATED type.

Constraints

If a component indirectly references an enumerated type, it is possible to constrain it to a subset of the set of named values. For example:

```
X Y(ONE, FIVE, SIX)
Y ::= ENUMERATED {ONE(1), TWO(2), THREE (3), FOUR (4), FIVE (5), SIX (6)}
```

Component X is restricted to a subset (one, five, six) of the named values of type Y (one, two, three, four, five, six).

NumericString Type

The NumericString is a subset of GeneralString that comprises the following:

- 0 to 9 (numeric characters)
- Space or blank character

External Form—Input and Output

This can be any string of valid characters as described above.

Local Form and Behavior

On input, data is stored as supplied. If the component has a fixed length, a short input string is padded with blanks, and a long input string is truncated.

Named Values

Named values are not applicable to the NumericString type.

Constraints

There are three ways of constraining this type:

- **Size constraint—**the SIZE keyword can be used to constrain the number of bytes for this type to a range of values or a single value. For example:

  ```
  GRAPHICSTRING (SIZE(3..8))
  GRAPHICSTRING (SIZE(5))
  ```

- **Character set constraint**—you can constrain the valid characters that are allowed to be used on this type as external form data by using the FROM keyword to specify a character set. For example:

  ```
  GRAPHICSTRING (FROM("A"|"B"|"C"|"a"))
  ```

  In the example, only strings consisting of the specified letters are treated as valid external input.

- **Character string constraint**—a set of valid character strings can be specified for this type. For example:

  ```
  GRAPHICSTRING ("ABC"|"PaR"c|"xYz"m)
  ```

  The letter *m* can be placed after the string to indicate that case is irrelevant. The letter *c* (the default) is used to show case is relevant. If the case in the external input does not match the string, the input is rejected.

Note: References to GRAPHICSTRING in the preceding examples should be replaced with the applicable type.

PrintableString Type

The PrintableString is a subset of GeneralString that comprises the following:

- a to z (lowercase alphabetic characters)
- A to Z (uppercase alphabetic characters)
- 0 to 9 (numeric characters)
- Space or blank character
- ’ ( ) + , - . / : = ? (special characters)

External Form—Input and Output

This can be any string of valid characters as described above.

Local Form and Behavior

On input data is stored as supplied. If the component has a fixed length, a short input string is padded with blanks, and a long input string is truncated.

Named Values

Named values are not applicable to the PrintableString type.

Constraints

There are three ways of constraining this type:

- **Size constraint—**the SIZE keyword can be used to constrain the number of bytes for this type to a range of values or a single value. For example:

  ```
  GRAPHICSTRING (SIZE(3..8))
  GRAPHICSTRING (SIZE(5))
  ```

- **Character set constraint**—you can constrain the valid characters that are allowed to be used on this type as external form data by using the FROM keyword to specify a character set. For example:

  ```
  GRAPHICSTRING (FROM("A"|"B"|"C"|"a"))
  ```

  In the example, only strings consisting of the specified letters are treated as valid external input.

- **Character string constraint**—a set of valid character strings can be specified for this type. For example:

  ```
  GRAPHICSTRING ("ABC"|"PaR"c|"xYz"m)
  ```

  The letter *m* can be placed after the string to indicate that case is irrelevant. The letter *c* (the default) is used to show case is relevant. If the case in the external input does not match the string, the input is rejected.

Note: References to GRAPHICSTRING in the preceding examples should be replaced with the applicable type.

TeletexString Type

There is no use for this type in NCL.

External Form—Input and Output

All values are accepted as supplied.

Local Form and Behavior

Data is stored as is. If the component has a fixed length exceeding that of the input string, the data is left aligned, and all unreferenced bytes are set to X'00'. If the component cannot contain the number of input bytes supplied, the string is truncated.

Named Values

Named values are not applicable to the TeletexString type.

Constraints

There are three ways of constraining this type:

- **Size constraint—**the SIZE keyword can be used to constrain the number of bytes for this type to a range of values or a single value. For example:

  ```
  GRAPHICSTRING (SIZE(3..8))
  GRAPHICSTRING (SIZE(5))
  ```

- **Character set constraint**—you can constrain the valid characters that are allowed to be used on this type as external form data by using the FROM keyword to specify a character set. For example:

  ```
  GRAPHICSTRING (FROM("A"|"B"|"C"|"a"))
  ```

  In the example, only strings consisting of the specified letters are treated as valid external input.

- **Character string constraint**—a set of valid character strings can be specified for this type. For example:

  ```
  GRAPHICSTRING ("ABC"|"PaR"c|"xYz"m)
  ```

  The letter *m* can be placed after the string to indicate that case is irrelevant. The letter *c* (the default) is used to show case is relevant. If the case in the external input does not match the string, the input is rejected.

Note: References to GRAPHICSTRING in the preceding examples should be replaced with the applicable type.

VideotexString Type

There is no use for this type in NCL.

External Form—Input and Output

All values are accepted as supplied.

Local Form and Behavior

Data is stored as is. If the component has a fixed length exceeding that of the input string, the data is left aligned, and all unreferenced bytes are set to X'00'. If the component cannot contain the number of input bytes supplied, the string is truncated.

Named Values

Named values are not applicable to the VideotexString type.

Constraints

There are three ways of constraining this type:

- **Size constraint—**the SIZE keyword can be used to constrain the number of bytes for this type to a range of values or a single value. For example:

  ```
  GRAPHICSTRING (SIZE(3..8))
  GRAPHICSTRING (SIZE(5))
  ```

- **Character set constraint**—you can constrain the valid characters that are allowed to be used on this type as external form data by using the FROM keyword to specify a character set. For example:

  ```
  GRAPHICSTRING (FROM("A"|"B"|"C"|"a"))
  ```

  In the example, only strings consisting of the specified letters are treated as valid external input.

- **Character string constraint**—a set of valid character strings can be specified for this type. For example:

  ```
  GRAPHICSTRING ("ABC"|"PaR"c|"xYz"m)
  ```

  The letter *m* can be placed after the string to indicate that case is irrelevant. The letter *c* (the default) is used to show case is relevant. If the case in the external input does not match the string, the input is rejected.

Note: References to GRAPHICSTRING in the preceding examples should be replaced with the applicable type.

IA5String Type

The IA5String type provides a transparent general character set (VisibleString plus control characters).

External Form—Input and Output

All values are accepted as supplied.

Local Form and Behavior

On input data is stored as supplied. If the component has a fixed length, a short input string is padded with blanks, and a long input string is truncated.

Named Values

Named values are not applicable to the IA5String type.

Constraints

There are three ways of constraining this type:

- **Size constraint—**the SIZE keyword can be used to constrain the number of bytes for this type to a range of values or a single value. For example:

  ```
  GRAPHICSTRING (SIZE(3..8))
  GRAPHICSTRING (SIZE(5))
  ```

- **Character set constraint**—you can constrain the valid characters that are allowed to be used on this type as external form data by using the FROM keyword to specify a character set. For example:

  ```
  GRAPHICSTRING (FROM("A"|"B"|"C"|"a"))
  ```

  In the example, only strings consisting of the specified letters are treated as valid external input.

- **Character string constraint**—a set of valid character strings can be specified for this type. For example:

  ```
  GRAPHICSTRING ("ABC"|"PaR"c|"xYz"m)
  ```

  The letter *m* can be placed after the string to indicate that case is irrelevant. The letter *c* (the default) is used to show case is relevant. If the case in the external input does not match the string, the input is rejected.

Note: References to GRAPHICSTRING in the preceding examples should be replaced with the applicable type.

UTCTime Type

The UTCTime type provides date and time, as Universal Coordinated Time (year without century numbers), in the format:

***YYMMDDHHMM*[*SS*]Z**

GMT date and time (to minutes or seconds); Z indicates GMT time

YYMMDDHHMM*[*SS*]*sHHMM

Local date and time (to minutes or seconds); with signed zone offset from GMT time (*s* = + or -)

External Form—Input and Output

This can be any valid data as shown above.

Local Form and Behavior

On input data is stored as supplied. If the component has a fixed length, a short input string is padded with blanks. Truncation is not allowed.

Named Values

Named values are not applicable to the UTCTime type.

Constraints

Constraints are not applicable to the UTCTime type.

GeneralizedTime Type

The GeneralizedTime type provides date and time, as GeneralizedTime (year includes century numbers), in the format:

***YYYYMMDDHH*[*MM*[*SS*]][*.f*]Z**

GMT date and time (to hours, minutes or seconds); with optional fractional time units to any significance (hours, minutes or seconds); Z indicates GMT time

***YYYYMMDDHH*[*MM*[*SS*]][*.f*][*sHHMM*]**

Local date and time (to hours, minutes or seconds); with optional fractional time units to any significance (hours, minutes or seconds); with signed zone offset from GMT time (*s* = + or -)

External Form—Input and Output

This can be any valid data as shown above.

Local Form and Behavior

On input data is stored as supplied. If the component has a fixed length, a short input string is padded with blanks. Truncation is not allowed.

Named Values

Named values are not applicable to the GeneralizedTime type.

Constraints

Constraints are not applicable to the GeneralizedTime type.

GraphicString Type

The GraphicString type provides a transparent character set of graphic characters only.

External Form—Input and Output

All values are accepted as supplied.

Local Form and Behavior

On input data is stored as supplied. If the component has a fixed length, a short input string is padded with blanks, and a long input string is truncated.

Named Values

Named values are not applicable to the GraphicString type.

Constraints

There are three ways of constraining this type:

- **Size constraint—**the SIZE keyword can be used to constrain the number of bytes for this type to a range of values or a single value. For example:

  ```
  GRAPHICSTRING (SIZE(3..8))
  GRAPHICSTRING (SIZE(5))
  ```

- **Character set constraint**—you can constrain the valid characters that are allowed to be used on this type as external form data by using the FROM keyword to specify a character set. For example:

  ```
  GRAPHICSTRING (FROM("A"|"B"|"C"|"a"))
  ```

  In the example, only strings consisting of the specified letters are treated as valid external input.

- **Character string constraint**—a set of valid character strings can be specified for this type. For example:

  ```
  GRAPHICSTRING ("ABC"|"PaR"c|"xYz"m)
  ```

  The letter *m* can be placed after the string to indicate that case is irrelevant. The letter *c* (the default) is used to show case is relevant. If the case in the external input does not match the string, the input is rejected.

Note: References to GRAPHICSTRING in the preceding examples should be replaced with the applicable type.

VisibleString Type

The VisibleString type provides a transparent character set of graphic characters only.

External Form—Input and Output

All values are accepted as supplied.

Local Form and Behavior

On input data is stored as supplied. If the component has a fixed length, a short input string is padded with blanks, and a long input string is truncated.

Named Values

Named values are not applicable to the VisibleString type.

Constraints

There are three ways of constraining this type:

- **Size constraint—**the SIZE keyword can be used to constrain the number of bytes for this type to a range of values or a single value. For example:

  ```
  GRAPHICSTRING (SIZE(3..8))
  GRAPHICSTRING (SIZE(5))
  ```

- **Character set constraint**—you can constrain the valid characters that are allowed to be used on this type as external form data by using the FROM keyword to specify a character set. For example:

  ```
  GRAPHICSTRING (FROM("A"|"B"|"C"|"a"))
  ```

  In the example, only strings consisting of the specified letters are treated as valid external input.

- **Character string constraint**—a set of valid character strings can be specified for this type. For example:

  ```
  GRAPHICSTRING ("ABC"|"PaR"c|"xYz"m)
  ```

  The letter *m* can be placed after the string to indicate that case is irrelevant. The letter *c* (the default) is used to show case is relevant. If the case in the external input does not match the string, the input is rejected.

Note: References to GRAPHICSTRING in the preceding examples should be replaced with the applicable type.

GeneralString Type

The GeneralString type provides a transparent character set of both graphic and control characters.

External Form—Input and Output

All values are accepted as supplied.

Local Form and Behavior

On input data is stored as supplied. If the component has a fixed length, a short input string is padded with blanks, and a long input string is truncated.

Named Values

Named values are not applicable to the GeneralString type.

Constraints

There are three ways of constraining this type:

- **Size constraint—**the SIZE keyword can be used to constrain the number of bytes for this type to a range of values or a single value. For example:

  ```
  GRAPHICSTRING (SIZE(3..8))
  GRAPHICSTRING (SIZE(5))
  ```

- **Character set constraint**—you can constrain the valid characters that are allowed to be used on this type as external form data by using the FROM keyword to specify a character set. For example:

  ```
  GRAPHICSTRING (FROM("A"|"B"|"C"|"a"))
  ```

  In the example, only strings consisting of the specified letters are treated as valid external input.

- **Character string constraint**—a set of valid character strings can be specified for this type. For example:

  ```
  GRAPHICSTRING ("ABC"|"PaR"c|"xYz"m)
  ```

  The letter *m* can be placed after the string to indicate that case is irrelevant. The letter *c* (the default) is used to show case is relevant. If the case in the external input does not match the string, the input is rejected.

Note: References to GRAPHICSTRING in the preceding examples should be replaced with the applicable type.

ANY and ADB Types

The ANY and ADB types can contain any sort of data. The type can be changed dynamically by attaching a map at runtime.

External Form—Input and Output

Any data is accepted and returned unchanged.

Local Form and Behavior

Data is stored as is. If the component has a fixed length exceeding that of the input string, the data is left aligned, and all unreferenced bytes are set to X'00'. If the component cannot contain the number of input bytes supplied, the string is truncated.

Named Values

Named values are not applicable to the ANY and ADB types.

Constraints

Constraints are not applicable to the ANY and ADB types.

Appendix A: Text Editor Commands

This section contains the following topics:

Using the Text Editor Commands

On the left of each line of text is a *sequence number* field. When updating or adding text, line commands are entered into these fields to perform edit functions.

Primary commands are entered into the Command field.

More information:

Text Editor (see page 31)

Line Commands

The line on which the line command is entered is referred to as the current line in the following text.

The following text editor line commands are supported:

A*nn*

Inserts copied or moved lines after the current line, *nn* times. If *nn* is omitted, the lines will be inserted once only.

B*nn*

Inserts copied or moved lines before the current line, *nn* times. If *nn* is omitted, the lines will be inserted once only.

C*nn*

Copies *nn* lines starting from the current line. Enter A or B beside the line of text to which the lines are to be copied after or before, or O*nn* or OO to copy the text over a sequence of lines.

If *nn* is omitted, only the current line is copied.

CC

Copies a sequence of lines. Enter CC beside the first and last line to be copied and A or B beside the line of text to which the lines are to be copied after or before, or O*nn* or OO to copy the text over a sequence of lines.

COLS

Inserts a line containing column numbers after the current line.

D*nn*

Deletes *nn* lines starting from the current line. If *nn* is omitted, only the current line is deleted.

DD

Deletes a sequence of lines. Enter DD beside the first and last line to be deleted.

I*nn*

Inserts *nn* blank lines after the current line. If *nn* is omitted, one blank line is inserted.

ID

Inserts lines of text containing the current date and time, the user's ID, name and phone number.

LC*nn*

Converts alphabetic characters to lowercase on the next *nn* lines. If *nn* is omitted, only the current line is converted.

LCLC

Converts alphabetic characters to lowercase in a block of lines. Enter LCLC beside the first and last lines to convert.

M*nn*

Moves *nn* lines starting from the current line. Enter A or B beside the line of text to which the lines are to be moved after or before, or O*nn* or OO to move the text over a sequence of lines.

If *nn* is omitted, only the current line is moved.

MM

Moves a sequence of lines. Enter MM beside the first and last line to be copied and A or B beside the line of text to which the lines are to be moved after or before, or O*nn,* to move the text over a sequence of lines.

NA

Inserts the contents of the Notepad after the current line.

NB

Inserts the contents of the Notepad before the current line.

N*nn*

Appends *nn* lines to the Notepad. If *nn* is omitted, only the current line is appended to the Notepad.

NN

Appends a sequence of lines to the Notepad. Enter NN beside the first and last line to be appended to the Notepad.

O*nn*

Overlays copied or moved lines over the next *nn* lines. If *nn* is omitted, it defaults to the number of lines being moved or copied.

OO

Overlays copied or moved lines over a sequence of lines. Enter OO beside the first and last lines to be overlaid.

QA

Inserts queued lines after the current line.

QB

Inserts queued lines before the current line.

Q*nn*

Queues *nn* lines. If *nn* is omitted, only the current line is queued. This command overwrites currently queued lines.

Q+*nn*

Appends *nn* lines to the end of the queue. If *nn* is omitted, only the current line is appended.

QQ

Queues a sequence of lines. Enter QQ beside the first and last line to be queued. This command overwrites currently queued lines.

QQ+

Appends a block of lines to the end of the queue. Enter QQ+ beside the first and last lines to be appended to the queue.

R*nn*

Repeats the current line *nn* times. If *nn* is omitted, the current line is repeated once.

RR*nn*

Repeats a sequence of lines *nn* times. Enter RR*nn* beside the first and last line to be repeated. If *nn* is omitted, the sequence of lines is repeated once only.

TE

Text entry mode. Blank input lines are inserted after the current line allowing the entry of many lines of text. When the Enter key is pressed the text is flowed within the current margins and inserted.

TF

Flows text to the current right margin preserving the indentation of each line from column 1. Text is flowed from the current line to the end of the paragraph. The end of the paragraph is determined by a blank line or a change in indentation.

TF*nn*

This command is the same as TF except that it uses the right margin specified by *nn*.

TFM

Flows text to the current left and right margins preserving only the relative indentation between the current line and the following line. Text is flowed from the current line to the end of the paragraph. The end of the paragraph is determined by a blank line or a change in indentation.

TM

Merges text. The current line is split at the cursor position as for a TS command. Text Entry mode is then invoked. On exit from Text Entry mode a TF is performed from the current line through to the end of the paragraph.

TS

Splits text. A new line is inserted after the current line and the text from the cursor position to the end of the line is moved to the new line.

UC*nn*

Converts alphabetic characters to uppercase on the next *nn* lines. If *nn* is omitted, only the current line is converted.

UCUC

Converts alphabetic characters to uppercase in a block of lines. Enter UCUC beside the first and last lines to convert.

)*nn*

Shifts the data on the current line *nn* positions to the right. Any data shifted beyond the maximum record length is lost. If *nn* is omitted, the data is shifted one position to the right.

(*nn*

Shifts the data on the current line *nn* positions to the left. Any data shifted beyond column one is lost. If *nn* is omitted, the data is shifted one position to the left.

))*nn*

Shifts the data on a block of lines *nn* positions to the right. Enter *))nn* beside the first and last lines to have their data shifted. Any data shifted beyond the maximum record length is lost. If *nn* is omitted, the data is shifted one position to the right.

((*nn*

Shifts the data on block of lines *nn* positions to the left. Enter *((nn* beside the first and last lines to have their data shifted. Any data shifted beyond column one is lost. If *nn* is omitted, the data is shifted one position to the left.

Line Command Examples

The following figures show some examples of commonly-used line commands.

```
PROD----------------------- CAS : Notepad Editor -------------Line 1 to 1 of 1
Command ===>                                    Function=Update Scroll ===> PAGE

**** ****************************** TOP OF DATA ******************************
i501 To add lines, type 'i' plus the number of lines to add in left column
**** ***************************** BOTTOM OF DATA *****************************









 F1=Help      F2=Split     F3=File      F4=Save      F5=Find      F6=Change
 F7=Backward  F8=Forward   F9=Swap     F10=Left     F11=Right    F12=Cancel
```

```
PROD----------------------- CAS : Notepad Editor -------------Line 1 to 6 of 6
Command ===>                                    Function=Update Scroll ===> PAGE

**** ***************************** TOP OF DATA ******************************
0001 To add lines, type 'i' plus the number of lines to add in left column
0002
0003 In this example 'i5' was typed into the left column, adding 5 lines to
0004 the Notepad text entry area.
0005
0006
**** **************************** BOTTOM OF DATA ****************************









 F1=Help      F2=Split     F3=File      F4=Save      F5=Find      F6=Change
 F7=Backward  F8=Forward   F9=Swap     F10=Left     F11=Right    F12=Cancel
```

```
PROD----------------------- CAS : Notepad Editor -----------Line 1 to 14 of 14
Command ===>                                    Function=Update Scroll ===> PAGE

**** ***************************** TOP OF DATA ******************************
0001
d002 A letter 'd' in the first column will delete that one line
0003
dd04 To delete a block of text type 'dd' into the left column of the first line
0005 of the block to be deleted.....
dd06 .....then type 'dd' in the last line of the block and press ENTER.
0007
c008 To copy one line of text type 'c' into the left column.
0009
cc10 To copy a block of text type 'cc' in the first line of the block...
cc11 .....then type cc in the last line of the block and press ENTER.
0012
a013 The letter 'a' in the left column will insert copied text after 'this line
'
0014
b015 The letter 'b' will insert copied text before 'this line'
**** **************************** BOTTOM OF DATA ****************************


 F1=Help      F2=Split     F3=File      F4=Save      F5=Find      F6=Change
 F7=Backward  F8=Forward   F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Primary Commands

In addition to the standard product commands, the following primary commands are available to you within the text editor:

ALL *string*

Positions on the first occurrence of the specified string and displays a message indicating the total number of occurrences that are found.

CHANGE *from-string to-string* [ALL]

Modifies a string of characters in the text. Enter CHANGE (or C) followed by the character string to be changed, followed by the new character string, followed by ALL, if every occurrence of character string is to be changed. If either character string contains imbedded blanks, enclose the character string in single or double quotes. If either character string contains quotes, the entire string must be enclosed by the quote which does not appear in the string. To repeat the change enter the CHANGE command with no operands. Change processing starts from the current cursor position.

Note: If the *from-string* contains embedded blanks, the string is not changed unless it is contained entirely within a single line.

FIND *string*

Searches for a string of characters in the text. Enter FIND (or F) followed by the character string. To repeat the search press the FIND function key or enter the FIND command with no operands. Searching starts from the current cursor position.

Note: If the string being searched for contains embedded blanks, it must be enclosed in single or double quotes. In addition, the string is not found unless it is contained entirely within a single line.

FIRST *string*

Starts a search at the top of the text and finds the first occurrence of the specified string.

FLOW

Flows all text to the right margin preserving the indentation of each line from column 1. To flow text to a specific right margin enter FLOW, followed by the right margin column number. To flow text to the current right margin enter FLOW with no operands.

FLOWM

Flows all text to the left and right margins preserving only the relative indentation between the first and second lines of each paragraph. To flow text within specific margins enter FLOWM, followed by the left margin and right margin column numbers separated by a space.

LAST *string*

Starts a search at the bottom of the text and finds the first occurrence of the specified string.

LCMD *cmd*

Executes the line command *cmd*. This can be used to assign often used line commands to function keys.

LM *nn*

Sets the left margin of the text. Enter LM followed by the required left margin column number.

MARGINS

Displays/sets the left and right margins of the text. To set the margins enter MARGINS or MAR, followed by the left margin column number, followed by the right margin column number. To display the current margin settings enter MARGINS or MAR with no operands. Margin settings are only used when flowing text as a result of the FLOW or FLOWM primary command or the TF, TFM or TE line commands. The difference between the left and right margins must not be less than 20.

NEXT *string*

Searches forward through the text to find the next occurrence of the specified string. This is the default.

NULLS

Pads text lines with nulls or blanks. To pad text lines with blanks enter NULLS OFF. To pad text lines with nulls enter NULLS ON or NULLS.

NOTEPAD

Gives access to the Notepad facility which holds temporary text entered during the session.

PREV *string*

Searches backward through the text to find the previous occurrence of the specified string. This is the default.

PRINT

Prints the text on the printer of your choice.

RESET

Clears all pending line commands. This command can be abbreviated to RES.

RM *nn*

Sets the right margin of the text. Enter RM followed by the required right margin column number.

Appendix B: Shorthand Time and Date Formats

This section contains the following topics:

Shorthand Time and Date Formats

CAS data validation supports shorthand entry of times and dates.

Time abbreviations and the corresponding values that are returned to the calling procedure by CAS, are summarized in the tables in this section.

Shorthand Time Formats (HH.MM)

Abbreviation	Returns (Edit 4)	Returns (Edit 24)
=	The current time	The current time
+h	Current time + h hours	Current time + h hours
-h	Current time - h hours	Current time - h hours
h	0*h*.00	0*h*:00
hh	*hh*.00	*hh*:00
hmm	0*h.mm*	0*h:mm*
hmm	*hh.mm*	*hh:mm*
h.m or *h:m*	0*h*.0m	0*h*:0*m*
h.mm or *h:mm*	0*h.mm*	0*h:mm*
hh.m or *hh:m*	*hh*.0m	*hh*:0*m*
hh.mm or *hh:mm*	*hh.mm*	*hh:mm*

Abbreviation	Returns (Edit 4)	Returns (Edit 24)
.m or *:m*	00.0*m*	00:0*m*
.mm or *:mm*	00.*mm*	00:*mm*

Shorthand Time Formats (HH.MM.SS)

Abbreviation	Returns (Edit 20)	Returns (Edit 23)
=	The current time	The current time
h	0*h*.00.00	0*h*:00:00
hh	*hh*.00.00	*hh*:00:00
hmm	0*h.mm*.00	0*h:mm:*00
hhmm	*hh.mm*.00	*hh:mm:*00
hhmms	*hh.mm*.0*s*	*hh:mm:*0*s*
hhmmss	*hh.mm.ss*	*hh:mm:ss*
hh.mm.ss or *hh:mm:ss*	If hh or mm or ss are single digits, zero (0) is inserted before the digit. If hh or mm or ss are omitted, they are set to 00—for example, .2. becomes 00.02.00	
+ or *- hh.mm.ss* or *+* or *- hh:mm:ss*	Same rules as above except that the time is added or subtracted from the current time—for example, +1.5 is 1 hour and 5 minutes from now	

Shorthand Date Formats

Date abbreviations and the corresponding values that are returned to the calling procedure by CAS are summarized in the table below. All dates are returned in *DD-MMM-YYYY* format.

Abbreviation	Returns
=	Today's date.
+d	Today's date + *d* days.
-d	Today's date - *d* days.
d	The *dth* day of the current month and year.
dd	The *ddth* day of the current month and year.
ddd	The *dddth* Julian day of the current year.
ddmm or *mmdd*	The *dd*th day of the *mm*th month of the current year (order depends on national language code).
yyddd	The *dddth* Julian day of the year *yy*.
yy.ddd	The *dddth* Julian day of the year *yy*.
ddmmyy or *mmddyy*	The specified date (order depends on national language code).
dd/mm/yy or *mm/dd/yy*	The specified date (order depends on national language code). If *dd* or *mm* or *yy* are omitted, then they default to the current day, month or year (for example, entering //96 on the 3rd January 1992 returns 03-JAN-1996).
day-of-week	The date of *day-of-week* (MON/TUE/WED/THU/FRI/SAT/SUN) in the current week.
+day-of-week	The date of *day-of-week* in the following week.
-day-of-week	The date of *day-of-week* in the previous week.

Appendix C: List Panel Attributes

This section contains the following topics:

List Panel Attributes

The following table shows list panel attributes that can be used to modify the intensity, color, and highlighting of data within an entry line.

Variable	Intensity	Color	Highlight
&$LHATBLBN	Low	Blue	NONE
&$LHATBLGN	Low	Green	NONE
&$LHATBLPN	Low	Pink	NONE
&$LHATBLRN	Low	Red	NONE
&$LHATBLTN	Low	Turquoise	NONE
&$LHATBLWN	Low	White	NONE
&$LHATBLYN	Low	Yellow	NONE
&$LHATBHBN	High	Blue	NONE
&$LHATBHGN	High	Green	NONE
&$LHATBHPN	High	Pink	NONE
&$LHATBHRN	High	Red	NONE
&$LHATBHTN	High	Turquoise	NONE
&$LHATBHWN	High	White	NONE
&$LHATBHYN	High	Yellow	NONE
&$LHATBLBR	Low	Blue	REVERSE
&$LHATBLGR	Low	Green	REVERSE
&$LHATBLPR	Low	Pink	REVERSE
&$LHATBLRR	Low	Red	REVERSE
&$LHATBLTR	Low	Turquoise	REVERSE
&$LHATBLWR	Low	White	REVERSE

Variable	Intensity	Color	Highlight
&$LHATBLYR	Low	Yellow	REVERSE
&$LHATBHBR	High	Blue	REVERSE
&$LHATBHGR	High	Green	REVERSE
&$LHATBHPR	High	Pink	REVERSE
&$LHATBHRR	High	Red	REVERSE
&$LHATBHTR	High	Turquoise	REVERSE
&$LHATBHWR	High	White	REVERSE
&$LHATBHYR	High	Yellow	REVERSE
&$LHATBLBB	Low	Blue	BLINK
&$LHATBLGB	Low	Green	BLINK
&$LHATBLPB	Low	Pink	BLINK
&$LHATBLTB	Low	Turquoise	BLINK
&$LHATBLWB	Low	White	BLINK
&$LHATBLYB	Low	Yellow	BLINK
&$LHATBHBB	High	Blue	BLINK
&$LHATBHGB	High	Green	BLINK
&$LHATBHPB	High	Pink	BLINK
&$LHATBHRB	High	Red	BLINK
&$LHATBHTB	High	Turquoise	BLINK
&$LHATBHWB	High	White	BLINK
&$LHATBHYB	High	Yellow	BLINK

Appendix D: Web File Utilities

This section contains the following topics:

Web File System

The Web file system contains all the files used by the Web Interface. The Web file system is stored in the MODS file; however, it has no relation to the other MODS components such as the Common Application Services (CAS) functions.

In general, the Web file system is accessed only by internal servers. For diagnostic purposes, a limited range of Web file utilities are available online. This appendix describes the Web file utilities.

Accessing Web File Utilities

The Web file utilities are accessed through the CAS : Maintenance Menu, by entering **EXEC $W3DB90L ACTION=TOPLEVEL** from an Operator Console Services (OCS) or Command Entry (CMD) window. The Web File Utilities : Top Level Directory Summary panel is displayed:

```
PROD--------- Web File Utilities : Top Level Directory Summary ----------------
Command ===>                                                  Scroll ===> PAGE
W3DB9002 7 directories found, containing a total of 396 files.
                                                 (S = Display directory contents)
    Directory Name                                  # of files
    COMMON                                                 54
    INTERNAL                                                6
    JAVA                                                   14
    LOGON                                                   3
    NETSCAPEUPDA                                            1
    PUBLIC                                                 91
    TCPIP                                                 227
    **END**









 F1=Help      F2=Split     F3=Exit      F4=Return    F5=Find      F6=Refresh
 F7=Backward  F8=Forward   F9=Swap
```

Top Level Directory Summary Panel

The Web File Utilities : Top Level Directory Summary panel is a summary of all files present in the Web file system.

The display fields on the Web File Utilities : Top Level Directory Summary panel are as follows:

Directory Name

Displays the top level or first segment of the full path name of the Web file.

# of files

Displays the total number of Web files in all subdirectories contained in this top level directory.

Directory Panel for Selected Directory

To display a list of files and subdirectories contained in a top level directory, enter **S** beside the directory name. The Web File Utilities : Directory panel for the selected directory is displayed:

```
PROD-------------- Web File Utilities : Directory "COMMON" --------------------
Command ===>                                                  Scroll ===> PAGE
W3DB9105 54 files found in  "COMMON" and its subdirectories.
    (S/B=Browse File or Subdirectory, E=Edit File, D=Delete File, I=File Info.)
    File or Subdirectory Name                          Fix Lvl Type   DD
    AlertMonitor                                               Directory
    chgpwd01.esp                                       Install TEXT   MODSTST1
    ContentMenu.esp                                    Install TEXT   MODSTST1
    dataframework                                              Directory
    functions                                                  Directory
    help                                                       Directory
    javachart                                                  Directory
    logoffi.html                                       Install BINARY MODSTST1
    registration                                               Directory
    SelectSolveLinkSingle.htmlf                        Install TEXT   MODSTST1
    SolveLinksSingle.htmlf                             Install TEXT   MODSTST1
    SolveMenu.shtml                                    Install TEXT   MODSTST1
    SolveMenuApplet.esp                                Inhouse TEXT   MODSUSR
    SolveMenuApplet.shtml                              Install TEXT   MODSTST1
    SolveMenuLogo.shtml                                Install TEXT   MODSTST1



 F1=Help      F2=Split     F3=Exit      F4=Return    F5=Find      F6=Refresh
 F7=Backward  F8=Forward   F9=Swap
```

The display fields on the Web File Utilities : Directory panel are as follows:

File or Subdirectory Name

Displays the file name of an individual Web file, or the name of a subdirectory.

Fix Lvl

Displays the fix level of the Web file.

Install

Indicates that this file is still at the level that was installed from the original product tape.

NZ12345

Indicates that this file was modified by the fix identified by this fix number.

Inhouse

Indicates that this file has been modified at your site after installation.

Type (files only)

Indicates whether the data in the Web file is stored in text or binary format.

DD

Displays the DD name of the highest level in the MODS concatenation that this file is present in.

An identically named Web file can exist in more than one level in the MODS concatenation, but is only retrieved from the highest level. In general, Web files installed as directed from product or maintenance tapes are installed into the MODSDIS level. Web files modified after installation appear in the MODSUSR level.

The Web File Utilities : Directory panel contains the following options:

S/B (text files)

Browse the source of the Web file. Only files containing text can be browsed.

S/B (subdirectories)

List the individual files and subdirectories contained in the subdirectory.

E (text files)

Allow editing of the source of the Web file. Only files containing text in lines of less than 256 characters can be edited. An edited Web file, when saved, is placed in the highest level of the MODS concatenation (usually MODSUSR). You should not edit files unless asked to do so by CA Support.

D

Deletes the Web file from the DD level displayed. Only Web files in the highest level (usually MODSUSR) can be deleted.

I (files only)

Display detailed information about the Web file. Information includes the file size and the last updated details. This information is displayed separately for every DD in the MODS concatenation in which this Web file is present.

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W