Generation
Version 5.3
November 2010

Natural Construct™

Order Number: CST530-021IBU

This document applies to Natural Construct Version 5.3 and to all subsequent releases. Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Readers' comments are welcomed. Comments may be addressed to the Documentation Department at the following e-mail address: Documentation@softwareag.com.

TABLE OF CONTENTS

2. USING THE GENERATION SUBSYSTEM

9. BROWSE-SELECT MODELS

10. DRIVER MODEL

11. EXTENDABLE-INPUT MODEL

12. MAINT MODEL

13. MAP MODEL

25. JCL MODELS (MAINFRAME)

27. CREATING NATURAL COMMAND PROCESSORS

28. USE OF EXTERNAL OBJECTS

29. SUPPLIED GENERATION UTILITIES

PREFACE

This preface explains how information is presented for different platforms, as well as the structure of *Natural Construct Generation*. It includes information about the conventions used in this document and other resources you can use to learn more about Natural Construct.

This documentation assumes you have some experience with Natural. The following topics are covered:

Mainframe and Unix Platforms

The majority of the information in this documentation applies to both mainframe and Unix platforms. Differences between the two platforms are explained using the following methods:

- When a description applies to only one platform, it is indicated in parentheses. For example, (mainframe) or (Unix).
- When a minor difference exists between platforms, it is indicated by instructions in parentheses. For example, “Enter ‘ncstg’ at the Next prompt (in the Direct Command box in Unix).”
- When a significant difference exists between platforms, a note explains the difference. For example:

  Unix Note:
  You are required to enter...

- When major differences exist, separate sections or chapters are devoted to specific platforms. The platform names are displayed in the section or chapter headings in parentheses. For example:

  Invoking Natural Construct (Mainframe)

Other Platform Support

This documentation contains references to Natural for DB2, Natural for VSAM, and Natural for SQL/DS. Although these products are not available on Unix, users can develop on Unix and later port the resulting Natural source code to larger mainframes where Natural for DB2, Natural for SQL/DS, or Natural for VSAM are available.

This documentation uses illustrations from the Natural/Adabas environment. When commands, fields, or visible differences occur for DB2 (SQL/DS), IMS (DL/1), or VSAM users, mention is made within those sections.

Purpose and Structure of this Documentation

This documentation is intended for programmers who create applications using the supplied models. The chapters in this document are:

Chapter	Title	Topics
1	**Introduction to Natural Construct**, page 33	Introduces you to Natural Construct and explains how to invoke Natural Construct on each platform, as well as from Predict Case and ISPF. Minor differences for DB2, SQL/DS, VSAM, and DL/1 users are discussed.
2	**Using the Generation Subsystem**, page 57	Explains the Generation main menu, the functions available through the menu, and the use of user exits.
3	**Using the Editors**, page 91	Explains when and how to use the editors in the Generation subsystem.
4	**Implementation Methodology**, page 107	Explains the implementation methodology underlying Natural Construct. This chapter includes information about data areas and command processors. It also contains information about password checking, messaging, using PF-keys, using external action codes, and supporting multilingual applications.
5	**Design Methodology**, page 141	Explains the design methodology underlying Natural Construct. This chapter includes information about identifying objects during analysis, defining Predict relationships, generating parameter data areas, and creating object maintenance processes.
6	**General Model Specifications**, page 167	Describes the Standard Parameters panel (first specification panel) for the program models. This chapter includes information about setting up a Password file and the differences between using message numbers and message text.

Chapter	Title	Topics (continued)
7–22	Model chapters	Describe the models supplied with Natural Construct. Most models are described within their own chapter (some chapters describe a group of related models). Each chapter contains an introduction, an example of a module generated with the model, a description of model parameters and specification panels, and a list of the user exits supplied for that model (if the model supports user exit processing).
23	**User Exits for the Generation Models**, page 321	Explains how to use the user exits and User Exit editor. User exits for each model are listed in alphabetical order, along with examples of many exits.
24	**Statement Models**, page 395	Describes each statement model supplied with Natural Construct, as well as how to invoke a statement model. The statement models are listed in alphabetical order. Each section includes an example of using that model to generate a Natural statement or code block.
25	**JCL Models (Mainframe)**, page 569	Describes the JCL models supplied with Natural Construct. These models generate Job Control Language (JCL) for OS and DOS operating systems and are available for mainframe platforms only.
26	**Use of Predict in Natural Construct**, page 609	Explains how Predict is used with Natural Construct. This chapter includes model examples (user views) and their relationships to each other. Examples are from the Natural Construct demo system.
27	**Creating Natural Command Processors**, page 639	Presents an overview of the Natural SYSNCP utility and highlights the features specific to Natural Construct-generated applications.
28	**Use of External Objects**, page 651	Describes the external objects supplied with Natural Construct.
29	**Supplied Generation Utilities**, page 659	Describes the Natural Construct utilities supplied for use in the Generation subsystem, as well as their functions, and parameters.
A	**Appendix A — Glossary of Terms**, page 673	Contains a glossary of terms used throughout this documentation.

Document Conventions

Throughout this documentation, the following conventions apply:

Term	Description
Enter	Type a value in a field and press the Enter key.
Field	In general, any area on a screen where users can type information, select a value from a pop-up window, or indicate a preference by marking a box or circle.
Invoke	Activate or execute a program or menu.
Mark	Type a non-blank character in an input field (for example, an X) to select the corresponding option.
Panel	A full screen of information displayed by a program, etc.
Select	One of the following actions: • Move the cursor to a value and press the Enter key. • Scroll through a selection box and highlight a value. • Double-click on a value. • Type the name of a value in a key field and press the Enter key.
Specify	One of the following actions: • Type a value in a field. • Select a value from a selection window.
Window	A partial screen of information that overlays the current screen. A window is usually displayed with a border.

Other Resources

This section provides information about other resources you can use to learn more about Natural Construct. For more information about these documents and courses, contact the nearest Software AG office or visit the website at www.softwareag.com to order documents or view course schedules and locations. You can also use the website to email questions to Customer Support.

Related Documentation

This section lists other documentation in the Natural Construct documentation set.

Natural Construct User Documentation

For information about using Natural Construct, see:

- *Natural Construct Administration and Modeling*
  This documentation is intended for administrators who maintain the Natural Construct generation environment, as well as for developers who create new models.
- *Natural Construct Help Text*
  This documentation is intended for developers who create and maintain help text for Natural Construct-generated applications, as well as for developers who create and maintain help text for user-written models.
- *Natural Construct Getting Started Guide*
  This guide provides a quick overview of Natural Construct and its many features and capabilities. It is intended for programmers who are new to Natural Construct.

Installation Documentation

For information about installing Natural Construct, see the installation documentation for your platform.

Other Documentation

This section lists documents published by WH&O International:

- *Natural Construct Tips & Techniques*
  This book provides a reference of tips and techniques for developing and supporting Natural Construct applications.
- *Natural Construct Application Development User's Guide*
  This guide describes the basics of generating Natural Construct modules using the supplied models.
- *Natural Construct Study Guide*
  This guide is intended for programmers who have never used Natural Construct.

Related Courses

In addition to the documentation, the following courses are available from Software AG:

- A self-study course on Natural Construct fundamentals
- An instructor-led course on building applications with Natural Construct
- An instructor-led course on modifying the existing Natural Construct models or creating your own models

INTRODUCTION TO NATURAL CONSTRUCT

This chapter introduces Natural Construct and describes its subsystems. It also contains information about invoking Natural Construct and accessing the online help.

The following topics are covered:

Description of Natural Construct

Natural Construct is a set of tools for application developers. Created for the Natural/ Predict environment, Natural Construct assists Natural developers in achieving higher productivity goals than are obtainable using Natural and Predict alone. At the same time, Natural Construct helps to standardize and control the application development process.

Natural Construct is intended to be used as a Natural workbench. Application developers use the tools provided by Natural Construct to enhance the Natural environment and speed up the design and implementation of Natural systems.

Natural Construct Subsystems

Natural Construct consists of the following subsystems:

Subsystem	Description
Generation	Used by developers to generate Natural modules using models supplied with Natural Construct. It is described in this documentation.
Administration and Modeling	Used by the administrator to define custom models and maintain the models Natural Construct uses to generate modules. For information about this subsystem, refer to *Natural Construct Administration and Modeling*.
Help Text	Used by the documentor or programmer to create and maintain help text for all programs and fields. For information about this subsystem, refer to *Natural Construct Help Text*.

Each of the three subsystems has its own main menu, which is displayed by invoking the subsystem. However, as this document addresses the needs of application developers, the Generation main menu is the only Natural Construct main menu referred to. For more information about the main menu, see **Generation Main Menu**, page 58.

Generation Subsystem

Using the Generation subsystem, you can generate Natural modules using the models supplied with Natural Construct. A model is a set of code frames, parameter data areas (PDAs), and related subprograms used to capture user-supplied specifications and generate a Natural module. Natural Construct supplies more than 81 models.

Using the supplied models, you can generate the following:

- browse programs
- browse helproutines
- browse subprograms
- help text selection panels
- maintenance programs
- menu programs
- batch programs
- multi-line selection panels
- complex data object maintenance subprograms
- Natural statements
- maps (mainframe)
- parameter data areas (PDAs)
- quit programs
- startup programs
- code blocks
- JCL (mainframe)
- user exit code

Invoking Natural Construct

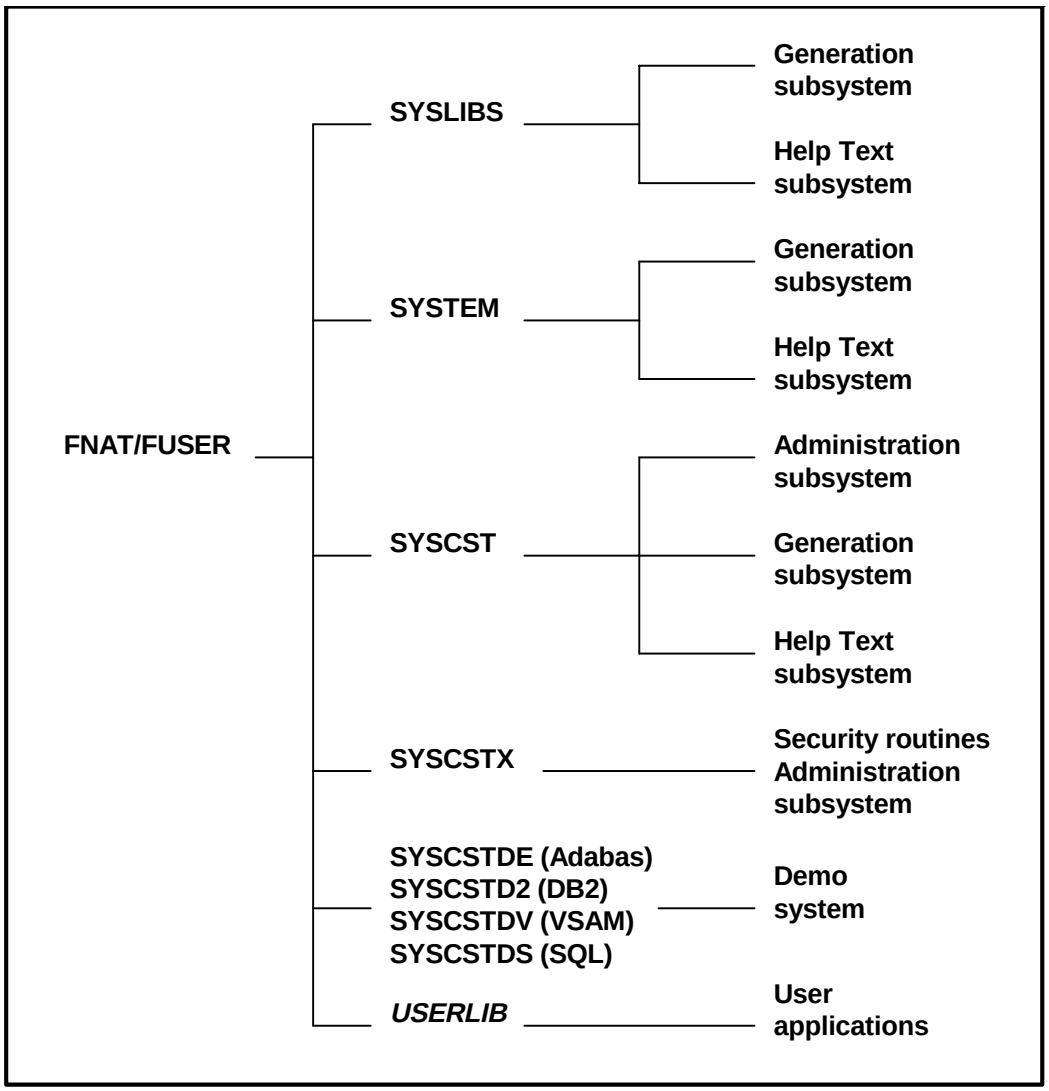
The following sections describe how to invoke Natural Construct from each library, as well as how to invoke under Predict Case, ISPF, DB2, SQL/DS, VSAM. DL/1 support is also discussed.

Note: Always terminate Natural Construct by pressing the quit PF-key or entering a period (.) in the input field on the Generation main menu. This method ensures proper cleanup of the environment.

Natural Construct Libraries

Copies of Natural Construct are stored in the libraries shown on the following page.

Note: For more information about Natural Construct libraries, see **Natural Construct Libraries**, *Natural Construct Administration and Modeling*.



Natural Construct Libraries

Each library is available to different users and contains different subsystems. The libraries are described in the following sections.

SYSLIBS Library

The SYSLIBS library contains modules used by Natural Construct. The following table indicates who can use the library, the subsystems it contains, and the command used to invoke each subsystem:

Users	Subsystems	Enter at the Next prompt:
All users	Generation	ncstg
	Help Text	ncsth

SYSTEM (FNAT) Library

The SYSTEM library contains modules used by Natural Construct-generated applications. The following table indicates who can use the library, the subsystems it contains, and the command used to invoke each subsystem:

Users	Subsystems	Enter at the Next prompt:
All users	Generation	ncstg
	Help Text	ncsth

SYSCST Library

The SYSCST library is used to modify the supplied models or create new ones. The following table indicates who can use the library, the subsystems it contains, and the command used to invoke each subsystem:

Users	Subsystems	Enter at the Next prompt:
Administrators	Administration	menu (standard mode) menut (translation mode)
	Generation	cstg
	Help Text	csth

SYSCSTX Library

The SYSCSTX library contains sample routines provided with Natural Construct. The routines can be used as is or modified as desired.

- To customize a routine, create a copy of the routine in the SYSCST library.
- To make the routine active, move the object code to the SYSLIBS library.

SYSCSTDE, SYSCSTD2, SYSCSTDV, and SYSCSTDS Libraries

These libraries contain the Natural Construct demo system for different systems. To invoke the demo system, enter “menu” at the Next prompt in the applicable library.

USERLIB Library

This library is created by Natural Construct users.

Invoking Natural Construct from Predict Case

From Predict Case, you can invoke Natural Construct from both the system function (SF) and the SF prompt in the Predict Case Navigator.

The Predict Case (PCA) Frame (FR) object type accepts a Natural Construct model as a PCA frame (referred to as PCA/CST models). To have Natural Construct models accepted as PCA frames, specify the following:

Field	Description
Tech. object type	Specify “Construct model” as the attribute in the PCA frame.
Object name	Specify the name of the PCA frame containing a Natural Construct model.

The formal specification (+F command) of a PCA/CST model should contain one line of code referencing a component type (CT), which is referred to as specification data CT. Create one specification data CT, which can be referenced by all other PCA/CST models.

Similar to other PCA frames, each PCA/CST model must be assigned a system function type (ST). Natural Construct is invoked from Predict Case to generate the system function for any system function assigned a system function type that has a PCA/CST model as its frame.

When a system function is generated by Natural Construct, both the specification data and the user exit code (supplied by users within Natural Construct) is saved as a custom component (CC) of the system function. The custom component is one instance of specification data CT and is referred to as specification data CC.

The following sections describe how to invoke and use Natural Construct from within Predict Case.

Invoking Natural Construct

To invoke Natural Construct from the Maintain System Function window in Predict Case, you can issue the following commands.

Formal Specification of a System Function (+F)

If a system function is associated with a PCA/CST model, the +F zoom command invokes the Modify Specifications (M) function in the Generation subsystem. You can scroll through all specification panels associated with the Natural Construct model as you would in Natural Construct.

You can save the specification data entered with the user exit code, in the specification data CC of Predict Case's system function. Conversely, when you invoke Natural Construct using the +F command within Predict Case, all existing data stored in the specification data CC is passed to Natural Construct.

From a Natural Construct specification panel, you can return to Predict Case or invoke the User Exit editor.

Custom Components of a System Function (+CU)

If a system function is associated with a PCA/CST model, the +CU zoom command invokes the User Exit editor (U) function in the Generation subsystem.

If specification data for the system function is not specified using the +F zoom command, the +CU command automatically invokes the +F command. This allows you to supply the specification data before the User Exit editor is displayed.

In the User Exit editor, you can create customized code using the SAMPLE command as you would in Natural Construct.

You can return the user exit code to Predict Case and store it in the specification data CC. Conversely, when you invoke Natural Construct from Predict Case using the +CU command, all user exit code stored in the specification data CC of the system function is returned to Natural Construct.

From the User Exit editor, you can either return to Predict Case or invoke the Natural Construct specification panels.

Create PG Command

If a system function is associated with a PCA/CST model, the Create PG command invokes the Natural Construct Generate (G) and Stow (W) functions in the Generation subsystem.

All specification data (supplied by using the +F command) and user exit code (supplied by using the +CU command) stored in the specification data (CC) of the system function is passed to Natural Construct for generation and compilation in the application library.

If the user exit code contains references to other PCA objects (i.e., PD, PU, IR, MA, or SF), the Create PG command invokes Natural Construct's Generate function only. The entire generated source returns to Predict Case for storage in the system function. The Predict Case Composer is then invoked to expand the Predict Case "$$" references and complete the Stow operation.

Note: When you invoke Natural Construct from Predict Case (using any of the three methods), the Generation main menu is not displayed.

Invoking Natural Construct from the Predict Case Navigator

To invoke Natural Construct from the Predict Case Navigator, enter the +F and +CU line commands beside an SF line. This is equivalent to issuing the +F and +CU zoom commands in the Maintain System Function window. For example, entering the ,W line command beside an SF prompt invokes the Natural Construct Generate and Stow functions. Similarly, entering the ,S line command invokes the Natural Construct Generate and Save functions.

Navigation Option Within Natural Construct

When you access Natural Construct from Predict Case, the PF-keys on the specification panels perform the following functions:

PF-Key	Function	Description
PF2	Return	Saves specification data and returns to Predict Case.
PF3	Quit	Returns to Predict Case without saving specification data.
PF12	Create PG	Invokes the Generate and Stow functions, as does the Create PG command. Press PF11 (userX) to invoke the User Exit editor from the last specification window.

In the User Exit editor, you can issue all commands you are familiar with in Natural Construct. The following commands are available:

Command	Description
CS-CMD F	Invokes the Natural Construct model specification panels (similar to the +F command in Predict Case).
CS-CMD PG	Invokes the Natural Construct Generate and Stow functions (similar to the Create PG command).
CS-CMD CANCEL	Returns you to Predict Case without saving your user exit code.

To terminate the User Exit editor and save your user exit code in specification data CC of the system function, enter a period (.)

Examples of Using Natural Construct Within Predict Case

➢ To use Natural Construct within Predict Case:

1 Create a specification data component type (CT) that can be used for all PCA/CST models.
The CT does not need formal specifications (+F), variable definitions (+V), or user exits:

```
17:30:49                  ***** Predict Case *****                      11-07
Plan 0                  - Maintain Component Type -                   XCTMAE00
     Object name: CONSTRUCT-SPECIFICATION-DATA          Status in process
     +Keywords:                                          Changed by DEVDN
                                                              on    11-07
 +Description                +Abstract:
--------------------------------------------------------------------------------
 +Formal specification          Tech. object type:

 +Variable definition                   User exit:









Command:
ACces CLear LET   REFre SAve  STAtu SUSpe
QUIt  . ? * +K +D +A +F +V                                F:.ADMPRST I * +
```

Predict Case Maintain Component Type Window

2 Define the System Function type (as you would in Predict Case) as follows:

```
17:33:12                 ***** Predict Case *****                      11-07
Plan 0              - Maintain System Function Type -                XSTMAE00
   Object name:   MAINTAIN-PROCESS                    Status in process
     +Keywords:                                       Changed by DEVDN
                                                             on    11-07
 +Description                +Abstract:
--------------------------------------------------------------------------------
  Has as default layout MA                  Has as default layout RP




Command:
ACces CLear LET   REFre SAve  STAtu SUSpe
QUIt  . ? * +K +D +A                                   F:.ADMPRST I * +
```

Predict Case Maintain System Function Type Window

3 Create one Predict Case frame for each model you want to use within Predict Case. Note that the frame name must be the same as the Natural Construct model name. This frame must also have the Tech. object type attribute set to Natural Construct model as follows:

```
17:34:34                 ***** Predict Case *****                      11-07
Plan 0                      - Maintain Frame -                      XFRMAE00
   Object name:   MAINT                               Status in process
     +Keywords:                                       Changed by DEVDN
                                                             on    11-07
 +Description                +Abstract:
--------------------------------------------------------------------------------
 +Formal specification                        Belongs to ST
                                              MAINTAIN-PROCESS
 +Variable definition
                                Tech. object type:   Construct Model
                                  Report Mode (x):









Command:
ACces CLear LET   REFre SAve  STAtu SUSpe
QUIt  . ? * +K +D +A +F +V                  F:.ADMPRST I * +
```

Predict Case Maintain Frame Window

4 At the command line in the Maintain Frame window, issue the +F command to display the Formal Specification window.
Specify one line of code referencing the CT CONSTRUCT-SPECIFICATION-DATA:

```
17:35:53                  ***** Predict Case *****                      11-07
Plan 0                       - Maintain Frame -                      X3LEDE0A
   Object name:   MAINT                                Status in process
  Used  1 % 1....+....2.... < Formal Specification > ....+....6....  Line 1
  $$CT CONSTRUCT-SPECIFICATION-DATA









Command:                                                              EscL , ALL
ACces CLear FPrin LET   REAd  REFre SAve  SCan  UPrin     +-H +-n + - ++ --
QUIt  . ? * +K +D +A +F +V
```

Predict Case Maintain Frame (+) Window

5 Create a system function belonging to a system function type that has a model frame type.
The following window shows how the MAINTAIN-ORDER system function uses the Maint model as its frame:

```
17:37:34                 ***** Predict Case *****                      11-07
Plan 0                - Maintain System Function -                  XSFMAE00
   Object name:   MAINTAIN-ORDER                       Status in process
     +Keywords:                                        Changed by DEVDN
                                                               on     11-07
 +Description                +Abstract:
------------------------------------------------------------------------------
  Subtype                                  Tech. object name
  dialog object                            PCMTORD
  Is of type ST                            Has as frame FR
  MAINTAIN-PROCESS                         MAINT
 +Supports FU                             +Calls conceptually SF

                                          +Uses MA / creates RP


 +DEcomposed into SF                      +Formal specification (Construct)

                                          +CUstom components (Construct)

Command:
ACces CLear Edit  EXec  LET   REFre SAve  STAtu SUSpe
QUIt  . ? * +K +D +A +S +DE +C +U +F +CU                F:.ADMPRST I * +
```

Predict Case Maintain System Function Panel

➢ To invoke the model specification panels:

1 Issue the +F command from the Maintain System Function panel.
The following example shows the first Maint model specification panel:

```
CUFMMA                          MAINT Program                          CU--MA0
Oct 21                       Standard Parameters                        1 of 3

 Module ............. PCMTORD_
 System ............. PCADEMOE_____________________________
 Global data area ... CDGDA___
 With block ......... _____________________________________

 Title ............... Maintain Order___________
 Description ........ This program is used to maintain the Order Entry_______
                      ____________________________________________________
                      ____________________________________________________
                      ____________________________________________________

 First header ....... Predict Case and Natural Construct demo_________
 Second header ...... Maintain Order Entry_____________________________

 Command ............ X
 Message numbers .... _
 Password ........... _

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                                          right crePG
```

Standard Parameters Panel for the Maint Model

On any model specification panel, you can press:

- PF2 (retrn) to save your specification data and return to Predict Case
- PF3 (quit) to return to Predict Case without saving your specification data
- PF12 (crePG) to invoke the Natural Construct Generate and Stow functions

After specifying the parameters on the model specification panels, the Predict Case Maintain System Function panel is displayed:

```
CASE0283 Changes to the object specification data were saved.
17:41:15                 ***** Predict Case *****                      11-07
Plan 0                 - Maintain System Function -                  XSFMAE00
   Object name:   MAINTAIN-ORDER                      Status in process
     +Keywords:                                       Changed by DEVLGL
                                                             on    11-07
 +Description               +Abstract:
--------------------------------------------------------------------------------
  Subtype                                  Tech. object name
  dialog object                            PCMTORD
  Is of type ST                            Has as frame FR
  MAINTAIN-PROCESS                         MAINT
 +Supports FU                             +Calls conceptually SF

                                          +Uses MA / creates RP


 +DEcomposed into SF                      +Formal specification (Construct)

                                          +CUstom components (Construct)

Command:
ACces CLear Edit  EXec  LET   REFre SAve  STAtu SUSpe
QUIt  . ? * +K +D +A +S +DE +C +U +F +CU                 F:.ADMPRST I * +
```

Predict Case Maintain System Function Panel

- To invoke the User Exit editor from this panel, issue the +CU command
- To exit the User Exit editor, enter a period (.)
  This saves your user exit code and returns you to Predict Case:

```
Program specifications read successfully.
Program name: PCMTORD                                        Line    1 of    8
      Title: Maintain Order
Cmd:                                                  Abs: X x-y: _ Dir: +
  All  ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
  0010 DEFINE EXIT AFTER-INPUT
  0020 *
  0030 * Processing to be performed immediately after the input screen.
  0040 IF NCST-ORDER-HEADER.ORDER-NUMBER > 900000
  0050   REINPUT WITH 'Order number can not greater than 900000'
  0060     MARK *NCST-ORDER-HEADER.ORDER-NUMBER ALARM
  0070 END-IF
  0080 END-EXIT
  0090
  0100
  0110
  0120
  0130
  0140
  0150
  0160
  0170
  0180
```

Natural Construct User Exit Editor

- To invoke the Predict Case Navigator for a system function, issue one of the following line commands at the SF prompt:
  - +F
  - +CU
  - ,W
  - ,S

  The following example shows the Predict Case Navigator invoked for a system function:

```
17:44:49                  ***** Predict Case *****                      11-07
Plan 0                           - Navigator -                        X00NV100
      Template:   SF-
F ----01-02-03-04-05-06-07-08-09-10-11-12-13-14------------------  Line 1
   SF MAINTAIN-ORDER
      contains
   CC :  CONSTRUCT-SPECIFICATION-DATA









Command:                                                              All
BUild CATal FLip REFre UPrin                              +-H +-n + - ++ --
QUIt  . ?                                                 F:.ADMPRST ? * +
```

Predict Case Navigator Window

All specification data specified on the model panels, as well as your user exit code, are saved in the custom component of the system function (CC):

```
17:46:05                 ***** Predict Case *****                      11-07
Plan 0                         - Navigator -                        X00NV100
      Template:   SF-
F ----01-02-03-04-05-06-07-08-09-10-11-12-13-14-------------------  Line 1
   SF MAINTAIN-ORDER
      contains
   CC :  CONSTRUCT-SPECIFICATION-DATA
      :  formal specification
      :  :  **SAG GENERATOR: MAINT                          Version: 3.2.2
      :  :  **SAG TITLE: Maintain Order
      :  :  **SAG SYSTEM: PCADEMOE
      :  :  **SAG GDA: CDGDA
      :  :  **SAG DESCS(1): This program is used to maintain the Order Entry
      :  :  **SAG HEADER1: Predict Case and Natural Construct Demo
      :  :  **SAG HEADER2: Maintain Order Entry
      :  :  **SAG DIRECT-COMMAND-PROCESS: X
      :  :  **SAG PRIME-FILE: NCST-ORDER-HEADER
      :  :  **SAG PRIME-KEY: ORDER-NUMBER
      :  :  **SAG RECORD-DESC: Record
      :  :  **SAG ACTIONS: 0101010101010000
Command:                                                              Top
BUild CATal FLip REFre UPrin                          +-H +-n + - ++ --
QUIt  . ?                                             F:.ADMPRST ? * +
```

Predict Case Specification Data Custom Component (Part 1)

```
17:47:02                 ***** Predict Case *****                      11-07
Plan 0                         - Navigator -                        X00NV100
      Template:   SF-
F ----01-02-03-04-05-06-07-08-09-10-11-12-13-14-------------------  Line 10
   SF MAINTAIN-ORDER
      :  :  **SAG HEADER2: Maintain Order Entry
      :  :  **SAG DIRECT-COMMAND-PROCESS: X
      :  :  **SAG PRIME-FILE: NCST-ORDER-HEADER
      :  :  **SAG PRIME-KEY: ORDER-NUMBER
      :  :  **SAG RECORD-DESC: Record
      :  :  **SAG ACTIONS: 0101010101010000
      :  :  **SAG MAX-WINDOWS: 01
      :  :  **SAG DEFINE EXIT AFTER-INPUT
      :  :  *
      :  :  * Processing to be performed immediately after the input screen.
      :  :  IF NCST-ORDER-HEADER.ORDER-NUMBER > 900000
      :  :    REINPUT WITH 'Order number can not greater than 900000'
      :  :      MARK *NCST-ORDER-HEADER.ORDER-NUMBER ALARM
      :  :  END-IF
      :  :  **SAG END-EXIT
Command:                                                              Bot
BUild CATal FLip REFre UPrin                          +-H +-n + - ++ --
QUIt  . ?                                             F:.ADMPRST ? * +
```

Predict Case Specification Data Custom Component (Part 2)

Invoking Natural Construct from ISPF

You can invoke the Natural Construct statement models (or any model that does not require user exits) as macros from within the ISPF editor.

➢ To invoke a model from within the ISPF editor:

1 Type “COPY MAC CSTMAC” in the Command field.

2 Mark the line where you want to insert the code.

3 Press Enter.
The Natural Construct ISPF Macro Interface window is displayed:

```
CSGISPF                    Natural Construct
Oct 21                   ISPF Macro Interface                1  of 1

Model ..............  _____________________________ *
Default Parameters .  _____________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11-
      help  retrn
```

4 Enter the name of the model you want to invoke in the Model field and the required parameters in the Parameters field.

In the following example, the VIEW statement model and the NCST-CUSTOMER parameter were entered:

```
EDIT-NAT:DEVPR(BBBBB)-Program->Struct-Free-32K ---------- >>> Member was copied
 COMMAND===>                                                 SCROLL===> CSR
****** ***************************** top of data ****************************
000010 **
000020   01 NCST-CUSTOMER VIEW OF NCST-CUSTOMER
000030     02 CUSTOMER-NUMBER
000040     02 BUSINESS-NAME
000050     02 PHONE-NUMBER
000060     02 MAILING-ADDRESS
000070       03 M-STREET
000080       03 M-CITY
000090       03 M-PROVINCE
000100       03 M-POSTAL-CODE
000110     02 SHIPPING-ADDRESS
000120       03 S-STREET
000130       03 S-CITY
000140       03 S-PROVINCE
000150       03 S-POSTAL-CODE
000160     02 CONTACT
000170     02 CREDIT-RATING
000180     02 CREDIT-LIMIT
000190     02 DISCOUNT-PERCENTAGE
000200     02 CUSTOMER-WAREHOUSE-ID
000210     02 CUSTOMER-TIMESTAMP
```

ISPF Editor Panel — After Generating Code for a VIEW Statement

Natural DB2 and SQL/DS Users

Natural DB2 users will encounter minor differences when using Natural Construct. Global differences are explained here. Local differences are explained as they occur.

Note: For information on date, time, and timestamp formats, see **DB2 Users**, page 612.

Difference	Description
Field names	The dash in field names is replaced with an underscore for DB2 users. For example, LOG-COUNTER becomes LOG_COUNTER.
Superdescriptors	To browse or maintain a DB2 table using a combination of fields, you must define a compound key (superdescriptor) in Predict before generating the browse or maintenance programs. Once the key is defined, Natural Construct uses the combination of fields to form a maintenance key or browse sort order. This superdescriptor tells Natural Construct which underlying fields form the maintenance or browse primary key.
Defining a unique field	Every Adabas record has a unique identifier called the ISN (Internal Sequence Number). This ISN allows you to scroll backwards in browse programs or recall and update secondary records in maintenance programs. Since DB2 does not have a unique record number, you must create one. Using the DB2 file maintenance options in Predict, enter “U” in the U column for the field you want to make unique (for example, PERSONNEL_ NUMBER). If no field in a file is unique, you can define a combination of fields as a superdescriptor by entering “U” in the U column of the newly-defined compound key. You can generate a browse or browse-select program using a key that is not unique. To scroll backward, the generated program creates a sequence field as its unique identifier.
Null indicators	DB2 distinguishes between a blank and a null. You can specify a type U field suppression type in Predict to define fields that support the null value. For all fields with corresponding null indictor fields in Predict, Natural Construct maintenance programs generate a null prompt beside the field input prompt.

Using a phone number as an example, the Predict file appears as follows:

Ty	L	Name	F	Length	D	U	DB	S
	1	PHONE	A	8.0	D		AA	U

When you generate a Natural Construct maintenance program, the null field appears beside each field that supports a null value. The following table lists the different options available for these fields and the result of each option:

If you enter:	The result is:
Phone: 555-1234 Null: __	Not null with phone number.
Phone: 555-1234 Null: X	Not null with phone number. (The specified phone number overrides the null indicator.)
Phone: _____ Null: __	Not null, blank phone number. The person has no phone.
Phone: _____ Null: X	Null value phone number. The person either has no phone or the phone number is unknown.

The null indicator (format I2) in the Data Definition Module (DDM) is named N@, followed by the field name. In programs generated by Natural Construct, this field is mapped to the N# logical variable, followed by the field name (for example, N#PHONE (L)).

If N#PHONE is assigned “TRUE”, N@PHONE is assigned “-1”.

Natural VSAM Users

If you are using Natural VSAM, you will encounter minor differences using Natural Construct. These differences are explained in the documentation as they occur.

DL/1 Support

All models support the generation of programs accessing DL/1 files. As with Adabas, DB/2, and VSAM, the model specifications for DBMS are independent, facilitating transparent generation within heterogeneous DBMS environments.

Since DL/1 requires the values of the parent segment keys when accessing the records of a child segment, any user view of the child segment generated by Natural Construct contains the parent segment keys. If you are using a map, include the parent segment keys on the map.

File Types Supported

The following table indicates the file types supported by DL/1 file types:

File Type	Description
Maintenance models	Only type J files are supported in the maintenance process models (all models except the Browse models). Type K files may contain many overlapping fields. Type J files are segment layouts, where all the UDF fields in each layout form a contiguous block of data imposed on the physical segment. To simulate the Adabas mechanism for extracting fields from a master file to create a user view, create another type J file with fields selected from an existing type J file with the same type I master file. Any fields in the existing type J file that are not needed can be renamed “DUMMY” in the new type J file.
Browse models	Type J and K files are supported only in the Browse models.

Features

You can specify two different types of fields that form a contiguous layout on a physical IMS segment within a type J file. These fields are:

Field Type	Description
Keys	You can extract keys (sequence fields, alternate index, and search fields) from the type I master file and include them in a type J file. Once they are included in a type J file, you can redefine these keys as usual.
UDF fields	You can specify UDF fields classified as the following types:
DUMMY fields	To denote unused areas in a segment, specify the field name of “DUMMY” at least once in the layout. When a DDM is generated for the file, the DUMMY fields are not generated into the DDM. Natural Construct ignores the DUMMY fields when generating user views or object parameter data areas (PDAs).
Normal fields	Any fields that do not overlap keys (from the type I master file) are considered non-key fields and not used as search criteria.

Field Type	Description (continued)
UDF redefinitions for key fields	Natural Construct does not generate keys with UDF redefinitions into user views, since their locations are already claimed by their UDF redefinitions. Any field with the same offset and length of a key in the type I master file is considered a UDF redefinition of a key. Any type J file that has UDF fields partially overlapping keys from the master file is rejected. The field must be totally overlapped to qualify as a UDF redefinition of a key. There are three cases of UDF redefinition of a key: • redefinition is a normal field • redefinition is a normal field that is redefined into components • redefinition is a group field In all three cases, you can specify the UDF redefinitions as keys for search criteria on Natural Construct panels, although they are not marked as descriptors in the DDM. Natural Construct recognizes the redefinitions and retrieves the appropriate key names from the DDMs to build the search criteria.

Getting Online Help

Natural Construct provides extensive online help. You can either display general help information for each panel (panel-level help) or specific help for a field on a panel (field-level help). There are two types of field-level help:

- Passive field-level help, which provides a description of valid parameters for the field
- Active field-level help, which displays a selection window listing valid parameters for a field

The following sections describe each type of online help supplied with Natural Construct.

Accessing Panel-Level Help

While using Natural Construct, you can display help information about the current panel by moving the cursor anywhere on the panel (except an input field) and pressing PF1 (help).

Note: If the cursor is in an input field when you request help, Natural Construct displays help for that field. For information, see **Accessing Field-Level Help**, page 55.

The following example shows the first page of panel-level help for the Generation main menu:

```
                          Panel Help
                     Generation Main Menu

This menu lists the available generation functions, as well as the
code you enter to invoke each function. In addition to the Function
field, other input fields are also provided. These additional input
fields are required for some of the functions listed.

To select a function from this menu:
1  Type the corresponding function code in the Function field
2  Type the name of the module in the Module name field
3  Optionally, type a panel number in the Panel field
4  Type the name of the model in the Model name field
5  Optionally, press the <<optns>> key
6  Press Enter

For translation mode details, see:
<<Generation Main Menu>>

Page ... 1  / 2
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF1
frwrd help  retrn quit                   bkwrd frwrd
Help for: P/CS/CSGMNM0/1
```

Panel-Level Help for the Generation Main Menu — Page 1

Take the following actions to move through the help windows:

- To scroll forward through the pages of help text, either enter a number in the Page field, press PF8 (frwrd), or press Enter.
- To scroll backward, either enter a number in the Page field or press PF7 (bkwrd).
- To return to the main screen, press PF2 (retrn).
- To display help about how to use the online help facility, press PF1 (help) in any help window.
- To display information about any topic enclosed within angle brackets (<< >>), move the cursor over the topic name and press Enter. A window is displayed, containing help information about the selected topic.

Accessing Field-Level Help

Natural Construct has two types of field-level help: passive and active. Passive field-level help displays a description of a field on a panel. Active help displays a selection window containing valid values for a field. Active help is indicated by an asterisk (*) to the right of the field.

Passive Field-Level Help

➢ To display passive field-level help, either:

- Move the cursor to any field that is not followed by an asterisk (*) and press PF1 (help)

  or

- Enter a question mark (?) in the first-character position of any field that is not followed by an asterisk (*)

  A help window is displayed, describing the valid parameters for the field.

Active Field-Level Help

➢ To display active field-level help, either:

- Move the cursor to any field that is followed by an asterisk (*) and press PF1 (help)

  or

- Enter a question mark (?) in the first-character position of any field that is followed by an asterisk (*)

  A selection window is displayed, listing the valid parameters and allowing you to select one.

The following example shows the selection window for the Predict view name field:

```
Position cursor or enter screen value to select
CPHFIB                 Natural Construct                  CPHFIB0
Oct 29               Select Predict View                   1 of 1

Predict view                     File type
-------------------------------- --------------------------------
NCSTDB2-CUSTOMER                 DB2 Table
NCSTDB2-ORDER_DISTRIBUTION       DB2 Table
NCSTDB2-ORDER_HEADER             DB2 Table
NCSTDB2-ORDER_INSTRUCTIONS       DB2 Table
NCSTDB2-ORDER_LINES              DB2 Table
NCSTDB2-PRODUCT                  DB2 Table
NCSTDB2-WAREHOUSE                DB2 Table
NCSTVSAM-CUSTOMER                VSAM File
NCSTVSAM-ORDER-DISTRIBUTION      VSAM File
NCSTVSAM-ORDER-HEADER            VSAM File
Predict view ..............  NCSTDB2_____________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10-
      help  retrn                         bkwrd frwrd
```

Active Field-Level Help Window

➢ To select a value from the selection window, either:

1 Move the cursor to the line containing the value.

2 Press Enter.

or

1 Enter the name of the view in the Predict view field.
The view must be currently displayed in the window. If not, the list of views is repositioned to the specified name.

You are returned to the original panel where the selected value is displayed in the field for which you requested help.

Note: You can display help about how to use the online help facility by pressing PF1 (help) in any help window.

USING THE GENERATION SUBSYSTEM

This chapter describes the Generation main menu and how to use its functions and PF-keys to generate Natural modules.

The following topics are covered:

Generation Main Menu

When you invoke the Generation subsystem (see **Invoking Natural Construct**, page 36), the Generation main menu is displayed:

```
CSGMAIN              N a t u r a l   C o n s t r u c t            CSGMNM0
Oct 21                     Generation Main Menu                    1 of 1

                  Functions
                  ----------------------------------------------
                  R  Read Specifications
                  M  Modify Specifications
                  U  Invoke User Exit Editor
                  G  Generate Source
                  T  Test Generated Source
                  E  Edit Generated Source
                  S  Save Generated Source
                  W  Stow Generated Source
                  L  List Generated Modules
                  C  Clear Edit Buffer
                  ?  Help
                  .  Return
                  ----------------------------------------------
Function ......... _  Module ............. ________ Panel ...... __
Model ............ ________________________________ Type .......
Command .......... ________________________________ Library .... CST341S
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help        quit        optns                                  lang
```

Generation Main Menu

The Functions column displays the available menu codes and the corresponding functions. Specify a function code, module name, panel number (optional), and model name in the fields near the bottom of the menu. You can specify additional options by pressing PF5 (optns).

The fields on the Generation main menu are:

Field	Description
Function	Code for the function you want to perform.
Module	Name of the module you are generating. The name must be alphanumeric, a maximum of eight characters, and contain no blanks.
Panel	Optional panel number. For models that have more than one specification panel, specify a panel number to go directly to that panel.
Note:	If you access a panel by specifying a panel number and then press Enter, you are returned to the Generation main menu.

Field	Description (continued)
Model	Name of the model used to generate the module. If you do not specify a model (or if the specified model does not exist), a selection window is displayed. For a description, see **Using the Select Model Window**, page 65.
Type	Type of Natural module generated by the specified model (program, subprogram, helproutine, etc.). Natural Construct supplies this value after you enter a model name.
Command	Command line on which you can issue Natural commands without leaving Natural Construct. For example, you can use the direct command line to invoke test modules or issue Logon commands.
Library	Name of the current Natural library. Natural Construct supplies this value.

The following table lists the function codes and their requirements:

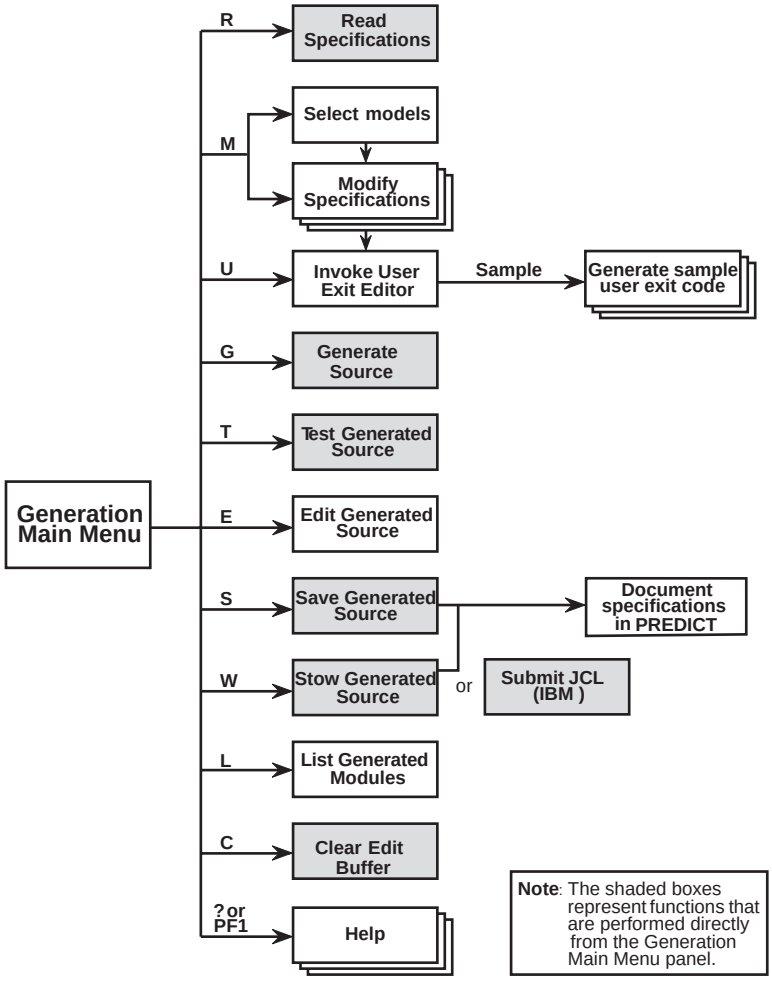
Code	Function	Requirements
R	Read Specifications	Enter the module name.
M	Modify Specifications	Enter the module name, model name, and, optionally, the panel number.
U	Invoke User Exit Editor	Enter the module name and model name.
G	Generate Source	Enter the module name and model name.
T	Test Generated Source	Perform either the Read Specifications (R) or Modify Specifications (M) function first.
E	Edit Generated Source	Perform either the Read Specifications (R) or Modify Specifications (M) function first.
S	Save Generated Source	Perform either the Read Specifications (R) or Modify Specifications (M) function first.
W	Stow Generated Source	Perform either the Read Specifications (R) or Modify Specifications (M) function first.
L	List Generated Modules	Enter a module name or the first few characters of a module name (optional).
C	Clear Edit Buffer	Enter a model name or the first few characters of a model name (optional).

The functions available on the Generation main menu are described in **Functions on the Generation Main Menu**, page 61.

Help and Return Codes on Menus

In each menu throughout the Natural Construct system, you have the option of using a question mark (?) and a period (.) as valid menu codes. Typing a question mark (?) in the Function field and pressing Enter displays help for that panel. It is equivalent to pressing PF1 (help). Typing a period (.) and pressing Enter or pressing PF2 (retrn) alone, terminates the current program and returns you to the previous menu.

Functions on the Generation Main Menu



Functions on the Generation Main Menu

Note: Additional options are available for the Generate, Save, and Stow functions. For information, see **Setting Generation Options**, page 76.

Read Specifications Function

This function reads the specification parameters and user exit code for an existing Natural Construct-generated module into Natural Construct.

Natural Construct modules are stored within the current library as are other Natural modules. Before you read a module into the edit buffer, you must first logon to the library in which it is stored.

If you do not know the name of a module, you can use the List Generated Modules (L) function to display a list of all Natural Construct-generated modules in the current library. For more information about this function, see **List Generated Modules Function**, page 74.

➢ To read a module and its specifications into the edit buffer:

1. Logon to the library where the module is stored.
2. Enter “ncstg” at the Natural Next prompt.
   The Generation main menu is displayed.
3. Type “R” in the Function field.
4. Type the name of the module in the Module field.
5. Press Enter.

Note: If you specify a model name in the Model name field, that model is used — regardless of which model generated the module. All common parameters for the original model and the specified model are included. A warning message is displayed if the parameters are not compatible.

Note: The Read Specifications function is not currently supported for Natural Construct-generated maps, data areas, or code blocks.

Modify Specifications Function

Use this function to define the specifications for a new module or to modify the specifications for an existing module.

➢ To define the specifications for a new module:

1 Type “M” in the Function field on the Generation main menu.

2 Type the name of the module in the Module field.
The module name must be alphanumeric, a maximum of eight characters in length, and cannot contain blanks.

3 Specify the model you want to use in the Model field.
You have the following options:

- Type the full name of the model.
- Type enough characters of the model name to identify it (for example, “Bat” for Batch).
- Type the model’s two-character alias to use a statement model. (The naming convention for these aliases is the one used for Natural statement help codes.)
- Type the model’s user-defined alias. For information about creating aliases, see **User Exit Subprograms**, *Natural Construct Administration and Modeling*.

If you do not specify a valid model name, the Select Model window is displayed. For more information, see **Using the Select Model Window**, page 65.

4 Press Enter.
The first model specification panel is displayed. If the model has more than one specification panel, you must specify all required information on the current panel before you can display the next panel. Press Enter or PF11 (right) to display the next panel. (For information about the model specification panels, see the chapter that describes the model you are generating.)

➢ To modify the specifications for an existing module:

1 Type “R” in the Function field on the Generation main menu.

2 Type the name of the module in the Module field.

3 Press Enter to read the module into the edit buffer.

4 Enter “M” in the Function field.
The first specification panel for the specified module is displayed.

Tip: To display a specific panel for a multi-panel model, specify the panel number in the Panel field.

Note: To use a model other than the one used to generate the original module, specify "M" in the Function field and the new model name in the Model field after the module has been read. The new model must have the same parameter data area (PDA) as the old model (the Browse, Browse-Subp, and Browse-Helpr models, for example).

The following table describes modules used in the Consumer Warehouses order entry system:

Module Name	Description
CWCOA	Parameter data area for the customer order function.
CWCOMNA	First maintenance subprogram.
CWCOMNA0	Map associated with the first maintenance subprogram.
CWCOMHA	Helproutine associated with the first maintenance subprogram.
CWCOMNB	Second maintenance subprogram.
CWCOMNB0	Map associated with the second maintenance subprogram.
CWCOMHB	Helproutine associated with the second maintenance subprogram.
CWCOBNA	First browse subprogram.
CWCOBNA0	Map associated with the first browse subprogram.
CWCOBHA	Helproutine associated with the first browse subprogram.
CWCOBNB	Second browse subprogram.
CWCOBNB0	Map associated with the second browse subprogram.
CWCOBHB	Helproutine associated with the second browse subprogram.

Using the Select Model Window

If you do not specify a valid model name for the Modify Specifications function, the Select Model window is displayed:

```
CSGDLIST            Natural Construct             CSGDLST0
Aug 29                 Select Model                 1 of 1

              Model name                     Type
         ------------------------------ ------------
         Batch                          Program
         Browse                         Program
         Browse-Helpr                   Helproutine
         Browse-Select                  Program
         Browse-Select-Helpr            Helproutine
         Browse-Select-Subp             Subprogram
         Browse-Subp                    Subprogram
         CST-Clear                      Subprogram
         CST-Document                   Subprogram
         CST-Frame                      Subprogram
         CST-Modify                     Subprogram
Model .. ______________________________ All models ... _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---
      help  retrn                       bkwrd frwrd
Position cursor or enter screen value to select
```

Select Model Window

Displaying a List of all Available Models

➢ To display a list of all available models, including the statement models:

1 Mark the All models field.

2 Press Enter.
The Select Model window displays a list of all available models.

Selecting a Model

➢ To select a model from the Select Model window, either:

1 Move your cursor to the line containing the model name.

2 Press Enter.

or

1 Type the name of the model in the Model name field at the bottom of the window.

2 Press Enter.

Repositioning the List of Models

➢ To position the list to a model name that is not currently displayed, do one of the following:

- Press PF8 (frwrd) to scroll down through the list of available models.
- Press PF7 (bkwrd) to scroll back up the list.
- Enter the name of the model in the Model field.

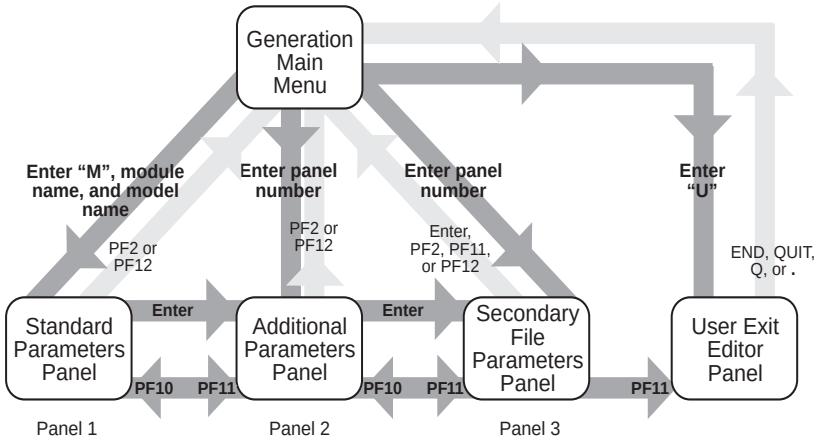
  All Natural Construct selection windows support wildcard selection. For information, see **Wildcard Selection**, page 89.

Note: Models with a CST prefix are only used in the Administrative subsystem to create new models. You cannot use these models in your application library.

Note: Due to the hotlink function available in the Help Text subsystem, the Help-Text-Select model is no longer needed. The model is supplied for developers who have used the model to generate links to help text members. For information about creating help text hotlinks, see **Hotlinks**, *Natural Construct Help Text*. For information about the Help-Text-Select model, see the online help.

Navigating the Model Specifications Panels

The following diagram illustrates how to navigate through the model specification panels (the example is for the Maint model):



- To move back and forth between specification panels, press PF10 (left) and PF11 (right). You must enter all required parameters on the current panel before going to the next panel.
- To display a particular model specification panel, enter the desired panel number in the Panel field on the Generation main menu. If you display a panel this way, you cannot press Enter to advance to the next panel. When you press Enter, you are returned to the main menu. To advance to the next panel, press PF11 (right).
- If the model supports user exits, press PF11 (userX) on the last specification panel to invoke the User Exit editor. If the model does not support user exits, press PF11 (right) on the last specification panel to display the Generation main menu.

Navigating the Specification Panels for the Maint Model

Invoke User Exit Editor Function

User exits allow you to include customized or specialized processing in a module, which is preserved when the module is regenerated. Depending on what module and model are specified on the Generation main menu, this function invokes either the User Exit editor or the User Exits panel for the specified model:

- If user exits have been defined for the module, the user exit code is displayed in the User Exit editor.
- If user exits have not been defined for the module, the User Exits panel for the specified model is displayed.

➢ To invoke the User Exit editor from the Generation main menu:

1 Type "U" in the Function field.

2 Type the name of the module in the Module name field.

3 Type the name of the model in the Model name field.

4 Press Enter.
User Exit editor is displayed.

Note: You can also invoke the User Exits panel or User Exit editor by pressing PF11 (userX) on the last model specification panel.

For information about using the User Exit editor, see **User Exit Editor**, page 96. For information about defining user exits, as well as information about each user exit available in Natural Construct, see **User Exits for the Generation Models**, page 321.

Generate Source Function

This function generates or regenerates a module using the specifications currently in the edit buffer. Perform this function:

- after you create a new module (to generate)
- after you modify the specifications for an existing generated module (to regenerate)
- when a model is modified and you want to include the changes in any previously-generated modules (to regenerate)

Note: Always save or stow the module when you generate or regenerate. For more information, see **Save Generated Source Function**, page 73 and **Stow Generated Source Function**, page 73.

Generating or Regenerating a Module

➢ To generate a new module or regenerate an existing module using the specifications currently in the edit buffer:

1 Type “G” in the Function field on the Generation main menu.

2 Press Enter.
A message is displayed, indicating whether the generation was successful.

3 Type “S” (Save Generated Source function) or “W” (Stow Generated Source function) in the Function field to save or stow the generated or regenerated module.

Note: If an error occurs during generation, you can continue or terminate the generation process, depending on the severity of the error. For example, if you include a user exit unknown to the model, a warning is displayed and the user exit is placed at the bottom of the generated module.

Modifying User Exit Code

➢ To modify the code within an existing user exit:

1 Type “R” in the Function field on the Generation main menu.

2 Type the name of the module in the Module name field.

3 Press Enter. The module is read into Natural Construct.

4 Type “U” in the Function field.
The User Exit editor is displayed.

5 Edit the code as desired.

6 Enter “.” at the > prompt in the User Exit editor.
The main menu is displayed.

7 Type “G” in the Function field to regenerate the module.

8 Type “S” (Save Generated Source function) or “W” (Stow Generated Source function) in the Function field to save or stow the module.

Note: When you use the Invoke User Exit Editor function on the Generation main menu to modify user exit code, regenerate the module. If you modify user exit code within the Natural editor, save the module before exiting the editor.

Regenerating a Module After Modifying a Model

➢ To regenerate a module after the model used to generate the module was modified:

1 Type “R” in the Function field on the Generation main menu.

2 Type the name of the module in the Module name field.

3 Press Enter.
The module is read into Natural Construct.

4 Type “G” in the Function field.

5 Press Enter to regenerate the module.
A message is displayed, indicating whether the regeneration was successful.

6 Type “S” (Save Generated Source function) or “W” (Stow Generated Source function) in the Function field to save or stow the regenerated module.

After a module is generated, it exists in the editor. To modify or display the generated code, type “E” in the Function field and press Enter. You can also edit the generated module after exiting Natural Construct by typing “E” at Next prompt (in the Direct Command box in Unix) and pressing Enter.

Note: Modifying generated code outside of the user exits is strongly discouraged because the modifications are not included in subsequent regenerations of the modules.

Test Generated Source Function

This function tests the generated module currently in the edit buffer. If the module is a program, this function runs the program. If the module is not a program (a subprogram, helproutine, or data area), this function performs a Natural syntax check.

➢ To test the generated module currently in the edit buffer:

1 Type “T” in the Function field on the Generation main menu.

2 Press Enter.

When you test a module using the Test Generated Source function, the same functionality is not available as when the module is part of a finished system. This function executes one module, not the whole system. Avoid using application-related PF-keys and direct commands because they involve inter-program communication. For information about the global variables used in Natural Construct-generated modules, see **Default Global Variables**, page 109.

If compilation errors occur due to invalid user exit code, Natural Construct invokes the Generated Code editor, which highlights the line in error and displays an error message. (If the error does not appear directly on the highlighted line, it is usually near that line.) After correcting the error, test the module in the Generated Code editor using the RUN or CHECK command.

Note: To use the Save Generated Source, Edit Generated Source, or Test Generated Source function, generate the module in the edit buffer. A message indicating that the module has not been generated is displayed if you attempt to save or test a module that has not been generated.

Edit Generated Source Function

This function invokes the Generated Code editor. It is equivalent to issuing an edit command from the Next prompt. In this editor, you can edit the generated module currently in the edit buffer. It is easier to issue an edit command from the Generation main menu than to terminate Natural Construct and issue the command from the Next prompt.

Note: To modify user exit code, do not regenerate the module. Instead, read the module into the editor using the Read Specifications function, invoke the Edit Generated Source function, edit the code, return to the Generation main menu, and stow the module using the Stow Generated Source function. (If you use the Invoke User Exit Editor function to modify user exit code, you must regenerate the module.)

➢ To edit the generated module currently in the edit buffer:

1 Type “E” in the Function field on the Generation main menu.

2 Press Enter.
The Generated Code editor is displayed:

```
>                                      > +  Program     MYMENU   Lib DEVPR
Top    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
  0010 **SAG GENERATOR: MENU
  0020 **SAG TITLE: MENU PROGRAM
  0030 **SAG SYSTEM: DEVPR
  0040 **SAG GDA: CDGDA
  0050 **SAG DESCS(1): THIS MENU IS USED IN THE DEMO SYSTEM.
  0060 **SAG LINE(01): C          Customer Subsystem
  0070 **SAG LINE(02): O          Order Subsystem
  0080 **SAG LINE(03): W          Warehouse Subsystem
  0090 **SAG HEADER1: DEMO SYSTEM
  0100 **SAG HEADER2: MAIN MENU
  0110 **SAG DIRECT-COMMAND-PROCESS: X
  0120 **SAG FUNCTION-HEADING: Functions
  0130 **SAG DYNAMIC-ATTRIBUTES: }{        #
  0140 ************************************************************************
  0150 * Program  : MYMENU
  0160 * System   : DEVPR
  0170 * Title    : MENU PROGRAM
  0180 * Generated: Apr 30,96 at 01:12 PM
  0190 * Function : THIS MENU IS USED IN THE DEMO SYSTEM.
  0200 *
       ....+....1....+....2....+....3....+....4....+....5....+... S 211  L 1
```

Generated Code Editor

Warning:
Changes made to the generated modules outside of the user exits are not saved upon re-generation of the module.

Mainframe Note:
You can specify any command in this editor as can be specified on the Natural Command line.

Unix Note:
When you invoke the vi (Generated Code) editor and specify a file name, Natural reads the module from disk and overwrites the edit buffer. Consequently, Natural Construct performs a SAVE command before you edit the module. If the module already exists, you are prompted for confirmation before the SAVE command is performed. To confirm replacement of the existing module, press Enter.

Save Generated Source Function

This function saves a generated module and its specifications (currently in the edit buffer) to the Natural source file.

➢ To save a module and its specifications:

1 Type “S” in the Function field on the Generation main menu.

2 Type the name of the module in the Module name field.

3 Press Enter.
If no module currently exists with that name, a message is displayed indicating that the module was saved under the specified name. If you try to save a module under an existing name, a message that the module already exists is displayed. (This warning is not issued when a module is saved using the name of the module that was read into the buffer.) You can either press Enter to confirm a replacement or specify a different name.

Note: Natural Construct does not save specifications used to generate maps and data areas.

Stow Generated Source Function

This function performs two different actions, depending on the generated module involved. If the generated module is JCL (mainframe), this function submits the generated JCL (which is currently in the edit buffer). For all other generated modules, this function saves and catalogs a generated module and its specifications (which are currently in the edit buffer) to the Natural source file.

➢ To submit JCL or save and catalog a generated module:

1 Type “W” in the Function field on the Generation main menu.

2 Type the name of the module in the Module name field.

3 Press Enter.

List Generated Modules Function

This function lists all Natural Construct-generated modules in the current library.

➢ To list the modules in the current library:

1 Type “L” in the Function field on the Generation main menu.

2 Press Enter.
The Select Module window is displayed:

```
CSGLIST                  Natural Construct                  CSGLIST0
Oct 28                     Select Module                      1 of 1

  Module               Model                        Title
---------- -------------------------------- ------------------------
#MENU1     Startup                          Demo System Startup Pgm
#MENU3     Startup                          Demo System Startup Pgm
ACDERN     Object-Subp                      Order Object Subprogram
ACOMENT    Object-Maint-Dialog              Order object maintenance
BCOMENT    Object-Maint-Dialog-Fix          Order object maintenance
BCOMENT1   Object-Maint-Dialog              Order object maintenance
BCOMENT2   Object-Maint-Dialog              Order object maintenance
BRHELP     Browse-Helpr                     Branch Help
CLGJHF     Object-Maint-Dialog-Subp         Journal Header Maint
CUSTBRWS   Browse-Subp                      Browse customer file
CUSTN      Object-Maint-Subp                Customer Maintenance
Module ... ________  Model .... ________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11
      help  retrn                        bkwrd frwrd
Position cursor or enter screen value to select
```

Select Module Window

Only modules that can be regenerated are listed in this window. Maps and data areas are not listed. The list begins with the first module in the file unless a starting point is specified (by entering a starting value in the Module name field on the Generation main menu). If a starting point is specified, the list begins with the first record after the specified value.

The following table describes the fields in the Select Module window:

Field	Description
Module	To reposition the list to a module that is not currently displayed, enter a module name or the first few characters in a module name. The new list begins with the specified module or the first record following the specified characters.
Model	To restrict the list to only those modules generated by a specific model, enter the name of a model.

Note: You can also use wildcard characters to limit the range of the browse. For information, see **Wildcard Selection**, page 89.

➢ To select a module that is currently displayed in the Select Module window, either:

1 Type the name of the module in the Module field.

2 Press Enter.

or

1 Move your cursor to the line containing the module name.

2 Press Enter.

The Generation main menu is displayed and the selected module is read into the edit buffer.

Clear Edit Buffer Function

This function clears the current specifications from the edit buffer and assigns initial values to the model specification parameters.

➢ To clear the current specifications from the edit buffer:

1 Type “C” in the Function field on the Generation main menu.

2 Press Enter.
The specifications for the module are cleared, but the name is still displayed in the Module name field.

The Clear Edit Buffer function is not normally required because Natural Construct clears the previous model and module specifications from the edit buffer automatically:

- prior to performing the Read Specifications function

  or

- whenever a new model type is selected

Note: If the buffer is not automatically cleared after you perform the Read Specifications function, you may be reading a model that has the same PDA as the previous model (such as the Browse and Browse-Subp models).

Setting Generation Options

Additional options are available for many of the functions on the Generation main menu. For example, you can have a window displayed during generation that details each step of the generation process or have embedded statements written to the source buffer.

➢ To set additional options for functions on the Generation main menu:

1 Press PF5 (optns) on the Generation main menu.
The Optional Parameters window is displayed:

```
CSGOPTS             Natural Construct             CSGOPTS0
May 14             Optional Parameters              1 of 1

  Status window ................. X
                  Step ....... _
                  Text ....... _
  Embedded statements .......... _
  Condition codes ............... _
  Post-generation modifications  _

  Specifications only .......... _
  Document in Predict .......... _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9-
      help  retrn
```

Optional Parameters Window

The fields in this window are:

Field	Description
Status window	Mark this option to display the Status window during generation (by default, this field is marked). Messages in the Status window indicate which module is executing at each stage of the generation process. For examples of this window, see **Examples of the Status Window**, page 77.
Step	Mark this field to “step” through the stages of the generation process by pressing Enter in the Status window.You must press Enter before you can display the next message. To continue the generation process unaided, press PF2 (retrn).
Text	Mark this field to display messages in the Status window as text (for example, “starting” and “ending”). If this field is not marked, the messages are displayed with arrows “-->” (starting) and “<--” (ending).

Field	Description (continued)
Embedded statements	Mark this field to write embedded statements to the source buffer as part of the generated module. The statements indicate where the lines of code being written originated and the name of the code frame, generation subprogram, or sample subprogram that produced the code.
Condition codes	Mark this field to display the condition codes values in the Condition Codes Trace window after the pre-generation subprogram is executed. For more information, see **Determining Which Condition Codes are Set**, page 78.
Post-generation modifications	Mark this field to display the values of the code frame substitution parameters that are identified by an apostrophe (‘) during generation in the Post-Generation Modifications window. The window is displayed after the post-generation subprogram stacks the substitution values in the code frame.
Specifications only	Mark this field to save only the current specifications and user exit code. This function is helpful if parameter edits do not allow you to complete the generation process and you want to save the current specifications and user exit code.
Document in Predict	Mark this field to document a saved generated module in the Predict data dictionary. After you press PF2 (retrn) in the Optional Parameters window, other windows are displayed. For a description of these windows, see **Documenting a Module in Predict**, page 79.

Examples of the Status Window

By default, the Status window displays arrows to indicate which module is executing during the generation process. For example:

```
CSGENPGF                    Natural Construct
Apr 25                        Status Window                    1 of 1

<-- PREGEN CUMNPR
--> FRAME CUMN9
    --> FRAME CU--B9
```

Status Window Using Arrows

The following example shows the Status window with the Text field marked:

```
CSGENPGF                    Natural Construct
Apr 25                        Status Window                        1 of 1

Ending Pre-generation Subprogram CUMNPR
Starting Code Frame CUMN9
Starting Code Frame CU--B9
```

Status Window Using Text

Determining Which Condition Codes are Set

➢ To determine which condition codes are set:

1 Mark the Condition codes field in the Optional Parameters window.

2 Press PF2 (retrn).
The Generation main menu is displayed.

3 Generate the module.

After generation is complete, the Condition Codes Trace window is displayed, showing the condition codes set for the current module. For example:

```
CSGCTRAC                 Natural Construct                  CSGCTRC0
Jan 13                Condition codes Trace                   1 of 1

   Condition codes                     Condition codes
   ----------------------------     ----------------------------
1  X MULTIPLE-ENTITIES              14   DB2-TIMESTAMP
2  X KEY-NOT-DEFINED-AS-UQ          15   CONFINED-KEY-PREFIX
3    KEY-IS-ALPHA                   16   RESET-REDEFINES
4    KEY-IS-LOGICAL                 17   LOGGING-UPDATES
5    HOLD-FIELD-EQ-TIME             18   DB2-REFERENTIAL-INTEGRITY
6  X PRIME-FILE-IS-ADABAS           19   DB2-INTRA-CASCADE-DELETE
7    PRIME-FILE-IS-DB2              20   KEY-IS-REDEFINED
8    CLAUSE-IS-DB2-SPECIFIC         21   DERIVED-DATA
9    PRIME-FILE-IS-IMS              22   NAT-REFACTORING
10   PRIME-FILE-IS-VSAM             23 X GET-BY-ISN
11   KEY-IS-A-SUPER                 24 X HASH-LOCKING
12 X USE-MSG-NR                     25   NAT4-OR-HIGHER
13   KEY-IS-PRIME
```

Condition Codes Trace Window

Tip: If the condition code is not marked, ensure that the code is being generated in a Natural V4 (Natural V6 on open systems) or higher environment and that the option is set to true in the CSXDEFLT subprogram. CSXDEFLT is installed in the SYSCSTX library. To use the code, compile the subprogram in the SYSCST library and then copy the object code to the SYSLIBS library.

Documenting a Module in Predict

➢ To document a module in Predict:

1 Mark the Document in Predict field in the Optional Parameters window.

2 Press PF2 (retrn).
The Predict Parameters window is displayed. By default, the module name is displayed in the Program ID field:

```
CPNMPR1               Natural Construct               CPNMPR10
May 01         Program MYMENU Predict Parameters         1 of 1

  Program ID ......... MYMENU_____________________________ *
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10
      help  retrn
```

Predict Parameters Window

Note: Not all models support this feature.

3 Press Enter.
The Store Predict Documentation window is displayed:

```
CPNMPR2                      Natural Construct                     CPNMPR20
May 01                  Store Predict Documentation                  1 of 1

Program ID ......... MYMENU                          Modified ... 00-01-02
Type ............... P Program                       By .........
System Name ........ TEST_____________________________ *
Keys ......... 1_ 1  _____________________________ *
                  2  _____________________________ *
Mode ......... Batch ...... _  Online ..... X  Both ....... _
Member ............. MYMENU    User system Fnr .... ___
Library ............ DEVPR___  User system DBnr ... ___
NAT-Function ....... _____________________________
      Comments                          Authors
1_ 1  MENU PROGRAM_____________________ 1_ 1  Natural Construct_____________
   2  _____________________________       2  MENU_____________________________
   3  _____________________________       3  DEVPR____________________________
   4  _____________________________          Owners
   5  _____________________________ 1_ 1  _____________________________
   6  _____________________________       2  _____________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn                         frwrd bkwrd
```

Store Predict Documentation Window

4 Type the file number in the User system Fnr field.

5 Type the database ID in the User system DBnr field.

6 Press Enter.

You can type comments in the Comments field and associate the module with existing keywords (in the Keys field) and owners (in the Owners field). You can change some information (such as the Predict system name) by typing over the data displayed.

Modifying Predict Information for a Regenerated Module

You can modify existing Predict documentation by pressing PF5 (optns) and marking the Document in Predict field while saving or stowing a regenerated module. The Modify Predict Documentation window is displayed:

```
CPNMPR2                      Natural Construct                      CPNMPR20
May 01                  Modify Predict Documentation                  1 of 1

Program ID ......... MYMENU                            Modified ...    05-01
Type ............... P Program                         By ......... **NCST**
System Name ........ TEST_____________________________ *
Keys ......... 1_ 1  _____________________________ *
                  2  _____________________________ *
Mode ......... Batch ...... _  Online ..... X  Both ....... _
Member ............. MYMENU    User system Fnr .... ___
Library ............ DEVPR___  User system DBnr ... ___
NAT-Function ....... _____________________________

      Comments                              Authors
1_ 1  MENU PROGRAM_____________________ 1_ 1  Natural Construct_______________
   2  _____________________________       2  MENU_____________________________
   3  _____________________________       3  DEVPR____________________________
   4  _____________________________          Owners
   5  _____________________________ 1_ 1  _____________________________
   6  _____________________________       2  _____________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn                           frwrd bkwrd
```

Modify Predict Documentation Window

This window contains the same fields and functionality as the Store Predict Documentation window. To modify the Predict documentation, type over the data displayed.

The extended description of the module is based on the specifications used to generate the module. (As with all other aspects of Natural Construct models, the generated descriptions can be modified to meet specific installation requirements and standards.)

Example of an extended description for a module in Predict

```
Program: MYMENU                                  Added:    05-01
Type...: Program                                 Modified:
──────────────────── Program attributes ──
Language: Natural
Mode....: Online

Implementation pointer
   Member...: MYMENU   Library.: CSTDEV   Fnr.: 7   DBnr.: 18
   NAT-Func.:
   Load Lib.:
──────────────────────── K
CONTROL-GEN,ORDER-ENTRY

──────────── Comments ────────────
DEMO SYSTEM MENU. THIS MENU       Natural Construct
ACCESSES THE CUSTOMER             MENU
SUBSYSTEM AND ALLOWS YOU TO       DEVPR
ADD, DELETE, MODIFY, DISPLAY,
CLEAR, AND SCRATCH CUSTOMERS
STORED IN THE CUSTOMER SUBSYSTEM.
YOU CAN ALSO QUERY THE CUSTOMER
SUBSYSTEM FROM THIS MENU.
──────────────────── Descr
This is the menu for the Demo system.

──────────────────────────

On this menu the user enters a particular code value in order
to select a certain function. When a valid function is
selected, this menu stacks itself and fetches the program
associated with the selected function. The following table
identifies all valid codes together with their function
description and invoking program.

 Code                  Function                    Program
─────  ──────────────────────

   M   Maintain Customers                          NCCFCUST
   Q   Query Customers                             NCCSCUST
   ?   Invoke the help facility
   .   Terminate menu and return to previous screen

  ──────────────────── PF Key Processing —

  The following table shows the PF-keys used by this menu.

    PF-Key PF-Key
    Value   Name              PF Key Function
  ──────── ────── ──────────

     PF1   help   Invoke help for current field or panel
                                                  (same as entering ?)
     PF2   retrn  Return to previous function
                                                  (same as entering .)
     PF3   quit   Quit out of the current application
     PF5   flip   Flip PF Keys to another format
     PF12  main   Exit and go to main menu

──────────────── Miscellaneous Specifications ─────

Title..................: Customer Subsystem Menu
Global Data Area.......: NCGDA
First Heading..........: DEMO SYSTEM
Second Heading.........: MAIN MENU
Input Map Name.........: CDLAYMN&
Direct Command Support : Yes
Message Numbers or Text: Text

***** End of Report *****
```

Multilingual Support in Natural Construct

Natural Construct retrieves text for field prompts and messages from SYSERR at runtime. This allows you to specify which language to use on the model specification panels, windows, and messages.

Changing the Language Displayed on Model Panels

➢ To change the language displayed on the model specification panels, windows, and messages:

1 Press PF12 (lang) on the Generation main menu.
The Language Preference window is displayed:

```
CSULPS               Natural Construct              CSULPS0
Oct 08              Language Preference              1 of 1

           Number    Languages
          ----------  ------------------------------
              1       English
              2       Deutsch (German)
              3       Francais (French)
              4       Espagnol (Spanish)
              5       Italiano (Italian)
              6       Dutch
              7       Turkish
              8       Danish
              9       Norwegian
              10      Albanian
Number ... __
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---
      help  retrn                         bkwrd frwrd
Position cursor or enter screen value to select
```

Language Preference Window

2 Select the language you want to use.

Note: English (*Language 1) is the default language. If no translation is defined for a selected language, English is used. Although other languages are listed in the Language Preference window, you must add the translations for those languages in SYSERR.

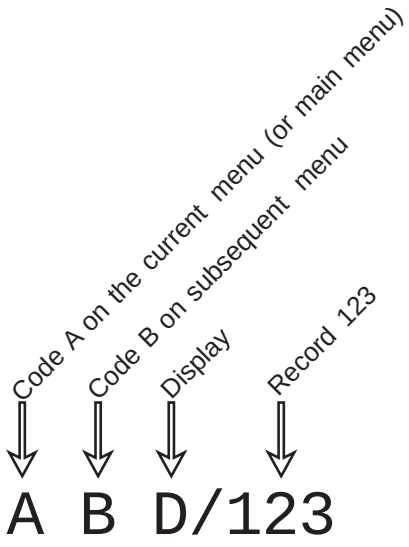
Direct Commands

You can navigate within the Generation subsystem by entering codes on menus, pressing PF-keys, or issuing direct commands. Direct commands allow you to go to any function or menu within the subsystem without using intervening menus. They are useful for experienced users who know the menu structure, the valid menu codes, and the required parameters at each menu level. A Command line is also provided on all modules generated by Natural Construct. The following figure is an example of the Command line:

```
Command: _____________________________________________________
```

The number of commands you can string together is unlimited. If one of the codes you specify is invalid on the corresponding menu, Natural Construct displays that menu so you can enter a valid code.

The following illustrates a sample command line:



Sample Command Line Entry

Command code A from the current menu (or the system's main menu if code A is not applicable to the current menu) invokes code B from the subsequent menu. Record 123 is displayed on the subsequent panel (assuming that it is a Maint or Object-Maint-Dialog program).

A command comprises the codes you specify in the successive menus. Each command must begin with a valid menu code. When specifying a command, leave a space between codes to indicate a new menu or level. To indicate parameters that are at the same level, use a slash (/) to separate them.

Note: The slash (/) is Natural Construct's default input delimiter. You can change the default input delimiter by issuing the GLOBALS ID=new-character command at the Next prompt (Direct command box for Unix) or the Natural command line.

When you enter commands on the Command line for a menu, Natural Construct first determines whether the code is a valid option on that menu. If no code on the current menu matches the first code in the command, Natural Construct checks the main menu for a match.

You can issue a command at the Natural Next prompt (Direct command box for Unix), while in the SYSCST library. For example, you can enter:

```
CSTG M/TEST/1/MENU
```

Natural Construct PF-Keys

Throughout Natural Construct, certain PF-keys have standard functions. For example, the help key invokes online help throughout all Natural Construct subsystems. The PF-keys, located at the bottom of most panels, display functions available on that panel.

PF-keys 13 to 24 are equivalent to PF-keys 1 to 12, respectively. However, only PF1 to PF12 are displayed. Your Natural Construct administrator can change the function and/or description associated with each key.

The following table describes the standard set of PF-keys. Some keys are available on all panels, such as PF1 (help) to invoke online help. However, other keys are only available when applicable, such as PF10 (left) and PF11 (right) to scroll between multiple panels.

PF-Key	Name	Function
PF1	help	Displays help for a particular panel or field. What help is displayed depends on where the cursor is when you press this PF-key: • If the cursor is not in a field, pressing PF1 displays help information for the panel. • If the cursor is in a field that is not followed by an asterisk, help information is displayed. • If the cursor is in a field that is followed by an asterisk (*), a help window is displayed to select a valid value for the field. For more information, see **Accessing Field-Level Help**, page 55, and **Accessing Panel-Level Help**, page 54.
	Note:	An asterisk is the default help indicator for Natural Construct. The help indicator for your organization may be different.
PF2	retrn	Displays the previous panel. Pressing PF2 is equivalent to entering a period (.) in the Code field on a menu.
PF3	quit	Terminates the Natural Construct session. In most cases, a confirmation window is displayed when you press PF3. Press PF3 again to complete the termination process and return to Natural.
PF4	test	Tests options on the model specification panels while you are defining parameters for the model.
PF7	bkwrd	Scrolls backward (up) through data.
PF8	frwrd	Scrolls forward (down) through data.

PF-Key	Name	Function (continued)
PF10	left	Displays the panel to the left of the current panel. If you are currently on the first panel in a series of panels, pressing PF10 displays the last panel in the series.
PF11	right	Displays the panel to the right of the current panel. If you are currently on the last panel in a series of panels, pressing PF11 displays the first panel in the series.
PF12	main	Displays the Natural Construct Generation main menu.

Note: PF5 (windw) and PF6 (attr) are described in the following sections.

Changing the Default Window Settings for Generated Modules

To change the default window settings for your generated module, press PF5 (windw). The following window is displayed:

```
CU--DWM           Natural Construct           CU--DWM0
Oct 21            Window Parameters             1 of 1

          Size .........  Height ....... ___
                          Width ........ ___

          Position .....  Line ......... ___
                          Column ....... ___

          Frame OFF .... _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---
      help  retrn quit  test
```

Window Parameters Window

The following values are used to build the DEFINE WINDOW statement:

Field		Description
Size	Height	Number of lines the window spans.
	Width	Number of columns the window spans.
Position	Line	Number of lines from the top of the panel to the top of the window.

Field	Description (continued)
Column	Number of columns from the left side of the panel to the left side of the window. The line and column values form the top left corner of the window.
Frame OFF	If this field is marked, the window is displayed without a border (frame).

Testing the Modified Window Settings

To view a test version of the window with the characteristics specified in the Window Parameters window, press PF4 (test).

Changing the Default Attribute Parameters for Generated Modules

To change the default dynamic attribute parameters for generated modules, press PF6 (attr):

```
CUSCDYM                 Natural Construct                 CU--DYM0
Oct 08             Dynamic Attribute Parameters             1 of 1

       Intensify .......... <
       Blue ............... _
       Green .............. _
       White .............. _       Special Hardware
       Pink ............... _       Blinking ........... _
       Red ................ _       Italic ............. _
       Turquoise .......... _       Underline .......... _
       Yellow ............. _       Reverse video ...... _

       Default return ..... >
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF
      help  retrn quit
```

Dynamic Attribute Parameters Window

You can specify up to four attributes, one of which must be the return to default display attribute (Default return field). For a description of these attributes and valid parameters for the fields, see the applicable Natural documentation.

To use some of the attributes listed in this window, special hardware is required. Avoid using terminal control, alphabetic, and numeric characters when defining dynamic attributes. If you are using Com-Plete or cross-generating applications to run on a platform where Com-Plete is in use, also avoid using stacking characters.

You can change the default dynamic attribute parameters for all modules generated by the Browse, Browse-Select, and Menu models. Your Natural Construct administrator can change the default dynamic attribute parameters used by other models.

Natural Construct Messages

Natural Construct sounds an alarm and displays warning messages for errors. Make sure the alarm on your terminal is set to an audible volume.

Natural Construct supports multilingual messages for your generated programs. If you use message numbers, the message text for the specified language is retrieved at execution time. If you use message text, the text for the specified language is inserted into the program at generation time.

You can change or add to these messages using the SYSERR utility.

- Messages 8000 to 8200 are stored in the SYSTEM and SYSCST libraries
- Messages 8300 to 8500 are stored in the CSTAPPL library
- Messages 1 to 9999 (error message text) are stored in the CSTMSG library
- Messages 1 to 9999 (screen prompt text) are stored in the CSTLDA library
- Messages 1 to 9999 (text for Actions) are stored in the CSTACT library
- Messages 1 to 9999 (text for PF-keys) are stored in the CSTPFK library

For all REINPUT and INPUT message numbers, you can also use the SYSERR utility to add other languages. Generation and CDUTRANS messages are stored in the CSTAPPL library.

For information about defining references, see **Defining SYSERR References**, *Natural Construct Administration and Modeling*.

Wildcard Selection

You can limit the range of records displayed in browse windows by using wildcard characters, such as an asterisk (*), a less than (<), or a greater than (>) symbol. For example, while browsing the modules file, you can enter the following values to limit the range of modules displayed:

Value Entered:	Modules Displayed:
SM *	Module names beginning with the letters “SM.” For example, the list includes SMPR, SMOR, and SMWH, but not SLPR or SNPR.
SM >	Module names greater than or equal to “SM.” For example, the list includes SMPR, SNPR, and TAPR, but not SLPR.
SM <	Module names less than or equal to “SM.” For example, the list includes ALPR, HAPR, and SLPR, but not SMPR, SNPR, or TAPR.

Automatic Upper Case Translation

Natural Construct automatically converts text from lower or mixed case to upper case. Headings are displayed exactly as entered (lower or upper case). However, if certain specification parameters must be in upper case, Natural Construct converts them. For example, all program text is automatically converted to upper case, except for comments and quoted text strings. When you exit Natural Construct, the case setting is restored to the same value as when Natural Construct was invoked.

Mainframe Note:
You must enter your teleprocessing (TP) monitor’s command for lower case. In COM-PLete, for example, the LOW command must be issued.

Storing Saved Modules

Any module generated using the default generators and saved by Natural Construct is stored as a Natural structured mode module in the current library. You can edit this module as you would any other structured mode module.

Always terminate Natural Construct by pressing the quit PF-key or entering a period in the Code field on the main menu. This method ensures proper cleanup of the environment (session).

USING THE EDITORS

This chapter discusses the two editors provided by Natural Construct for use in the Generation subsystem: the User Exit editor and the Generated Code editor.

The following topics are covered:

Introduction

Natural Construct provides two editors for the Generation subsystem — the User Exit editor and the Generated Code editor. Which editor you can use depends on what task you want to perform.

- Use the User Exit editor to insert customized or specialized processing (user exit code) in your generated modules. Information specific to the User Exit editor is presented in **User Exit Editor**, page 98.
- Use the Generated Code editor to edit a generated module. Information specific to the Generated Code editor is presented in **Generated Code Editor**, page 103.

Unix Note:
You can choose which editor you use as your Generated Code editor. Select either the Natural-provided ISPF editor or the Unix-provided Visual editor (vi), with which you may be more familiar. For more information about setting up a default editor, see the EDITOR profile parameter in the Natural documentation for Unix.

Features Common to Natural Construct Editors

This section describes the features that are common to all Natural Construct editors, such as:

- the order in which commands are executed
- how to change your PF-key settings
- how often work is saved automatically and where the work is saved

Unix Note:
The common features described in this section do not apply to Unix's vi editor. Consult the appropriate vi documentation to determine the equivalent commands for this editor.

Order of Command Execution

The Natural Construct editors execute commands in the following order:

1 Modify text.

2 Execute line commands.
You issue line commands in the text area of the editor.

3 Execute edit commands.
You issue edit commands at the greater than (>) prompt.

Changing Your PF-Keys

The PF-key settings used by the Natural Construct editors are determined in the same manner as those used by the Natural editor. If you have a profile that corresponds to your user ID, Natural Construct automatically uses those defaults.

To change PF-key settings for the current edit session, use the PROFILE command described in the following section.

Global Editor Profile Panel

➢ To change your PF-key settings for the current session:

1 Type “PROFILE” at the > prompt in the editor.

2 Press Enter.
The Global Editor Profile panel is displayed:

```
  12:56:31          ***** Natural PROFILE MAINTENANCE *****              01/05/96
                         - Global Editor Profile -

Profile Name.. SYSTEM__
PF and PA Keys
  PF1 ... HELP_____________   PF2 ... _________________   PF3 ... EXIT_____________
  PF4 ... _________________   PF5 ... _________________   PF6 ... _________________
  PF7 ... -________________   PF8 ... +________________   PF9 ... _________________
  PF10 .. SC=______________   PF11 .. _________________   PF12 .. CANCEL___________
  PF13 .. _________________   PF14 .. _________________   PF15 .. MENU_____________
  PF16 .. _________________   PF17 .. _________________   PF18 .. _________________
  PF19 .. --_______________   PF20 .. ++_______________   PF21 .. _________________
  PF22 .. _________________   PF23 .. _________________   PF24 .. _________________
  PA1 ... _________________   PA2 ... SCAN_____________   PA3 ... _________________

Automatic Functions
  Auto Renumber .. Y    Auto Save Numbers .. 0__  Source Save into .. EDITWORK

Additional Options .. N


Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit  AddOp Save  Flip                          Del  Canc
```

Global Editor Profile Panel

Use this panel to set up the PF-keys for your current edit session or change the values currently assigned to each PF-key. You can also specify the number of updates to allow before an automatic save, as well as the name of the recovery member where updates are saved. For a description of this panel, see the Editor Profile section in the Natural Utilities documentation.

Note: Auto Renumber and Additional Options relate to the Generated Code editor only.

Edit Recovery (Mainframe)

The Natural Construct editors automatically save work in the edit buffer after a certain number of updates. The number specified in the Auto Save Numbers field on the Global Editor Profile panel determines when the automatic save takes place. If this field is blank, no automatic save is performed.

You can also specify the name of the recovery member where you want work to be automatically saved. If you lose your code or forget to save it, you can restore from the last automatic save by performing the following steps:

➢ To restore from the User Exit editor:

1 Invoke the User Exit editor for the current module.

2 Read EDITWORK (the default recovery member) or the name of the recovery member specified on the Global Editor Profile panel into the edit buffer.

➢ To restore from the Generated Code editor:

1 At the Natural Next prompt, read EDITWORK (the default recovery member) or the name of the recovery member specified on the Global Editor Profile panel into the edit buffer.

2 Invoke the Generation subsystem.
Natural Construct reads the specifications into the edit buffer.

Note: Saving your work using a unique recovery name, such as your user ID, is strongly recommended. This ensures your work is not overwritten by other programmers using the same recovery name in the same library.

User Exit Editor

Use the User Exit editor to define user exits for your generated modules. User exits insert customized or specialized processing in a module, which is preserved upon subsequent regeneration of the module.

Natural Construct provides two methods to invoke the User Exit editor. Use the Invoke User Exit Editor function on the Generation main menu or press PF11 (userX) on the last model specification panel.

- For information about invoking the User Exit editor from the Generation main menu, see **Invoke User Exit Editor Function**, page 68.
- For information about invoking the editor from the model specification panels, see **Invoking the User Exit Editor**, page 323.
- For a description of the user exits supplied with Natural Construct, see **User Exits for the Generation Models**, page 321.

The following example shows the User Exit editor and CHANGE-HISTORY user exit:

```
Module name ........ MYMENU
Title ............... Menu program
>                                          > + ABS: _ X-Y: _ S    6 L    1
  All  ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
  0010 DEFINE EXIT CHANGE-HISTORY
  0020 * Changed on Feb 13,96 by SAG for release ____
  0030 * >
  0040 * >
  0050 * >
  0060 END-EXIT
                        Enter SAMPLE to generate sample user code
```

User Exit Editor

The fields in the User Exit editor are:

Field	Description
Module name	Name of the module for which you are defining user exits. The module name must be alphanumeric, cannot exceed 8 characters in length, and cannot contain blanks.
Title	Title specified on the model specification panel for this module. This title is used by the List Generated Modules function to identify modules for selection.
>	Command line prompt for the User Exit editor.

Field	Description (continued)
+	Direction indicator for edit commands. By default, plus (+) is displayed for this field. It indicates that all commands are performed in a forward direction. To perform commands in a backward direction, type minus (-) over the plus (+) displayed. For example, when you use the insert lines command (.I), lines are inserted after the line on which the command is entered if the direction is (plus) +, and before the line on which the command is entered if the direction is minus (-).
ABS	Absolute indicator. This field is used in conjunction with the SCAN and CHANGE commands. • If this field is marked, Natural Construct scans for changes made to specified characters, even if they are found within a word. • If this field is blank, Natural Construct only scans for or changes the characters if they are a separate entity (delimited by blanks or special characters) and not part of a larger word.
X-Y	Range delimiters. If this field is marked, editor commands are confined to the code within the X and Y range delimiters. Text outside the X-Y range is not affected.
S	Total number of code lines currently in the editor.
L	Line number currently displayed at the top of the editor.

User Exit Editor

The following sections describe the line commands, edit commands, and positional commands you can use in the User Exit editor.

Line Commands

You can enter line commands in the text area of the editor. Line commands must start in the first column of a line and begin with a . (period).

Note: Except for the .L command, always press Enter before using line commands on modified text.

If the direction indicator in the User Exit editor is forward (+) the command is applied in a forward direction. Note that copied, moved, or inserted lines are placed *below* the line on which the command was entered. If the direction indicator is backward (-), the command is applied in a backward direction. The copied, moved, or inserted lines are placed *above* the line on which the command was entered.

The following table lists the line commands available within the User Exit editor:

Command	Function
.C(*nn*)	Copies the current line the specified number (*nn*) of times. The default is one time.
.CX(*nn*)	Copies the line marked with X the specified number (*nn*) of times. The default is one time.
.CY(*nn*)	Copies the line marked with Y the specified number (*nn*) of times. The default is one time.
.CX-Y(*nn*)	Copies the X-Y delimited block of lines the specified number (*nn*) of times. The default is one time.
.D(*nn*)	Deletes the specified number (*nn*) of lines. The default is one line.
.G*(model, parameters)*	Invokes the specified Natural Construct statement model using the specified parameters.
.I*(member, startline, number of lines)* (mainframe)	Places a *member* from the current library onto the specified line in the editor. Optionally, you can specify a starting line and the total number of lines to include.
.I(*nn*)	Inserts the specified number (*nn*) of lines. The default is 9 lines. All Natural Construct editors, except the Help Text editor, suppress unused lines.

Command	Function (continued)
.J	Joins the next line of code to the end of the line on which the command is entered.
.L	Restores the current line to its previous state. (This command is similar to the LET edit command — but only applies to one line.)
.MX	Moves the line marked with X to the line on which the command is entered.
.MY	Moves the line marked with Y to the line on which the command is entered.
.MX-Y	Moves the range of code between the X and Y delimiters to the line on which the command is entered.
.N	Marks the line of the POINT command. For a description of the POINT command, see **Positional Commands**, page 102.
.P	Positions the current line at the top of the panel.
.S	Splits the current line. This begins at the cursor position.
.X	Marks the beginning of a range of lines for processing.
.Y	Marks the end of a range of lines for processing.

Edit Commands

Specify edit commands at the greater than (>) prompt. The following table lists the edit commands available within the User Exit editor. The underscored part of each command indicates the minimum characters required to execute the function:

Command	Function
ADD	Adds 9 blank lines to the current source.
CHANGE	Scans for and replaces the specified text. The syntax for this command is: `CHANGE 'scanvalue'replacevalue'` You can use any special character as a delimiter, providing the same character is not used within the command.
CLEAR	Clears the current contents of the edit buffer.
DX	Deletes the line marked X.
DY	Deletes the line marked Y.
DX-Y	Deletes the lines between the X and Y delimiters, inclusive.
END	Ends the edit session and returns to the previous menu.
EX	Deletes all lines before the X delimiter.
EY	Deletes all lines after the Y delimiter.
EX-Y	Deletes all lines before the X delimiter and after the Y delimiter.
LET	Restores lines to their previous state (if they were inadvertently changed). Issue this command before pressing Enter. The LET command is similar to the .L line command, but it applies to the whole panel.
PROFILE (mainframe)	Displays the Global Editor Profile panel, on which you can modify PF-key settings and edit work specifications for the duration of the edit session. For information about this panel, see **Global Editor Profile Panel**, page 94.
QUIT	Ends the edit session and returns to the previous menu.
READ	Reads Natural source into the edit buffer.
RENUMBER	Renumbers the lines in increments of 10 (if the editor has line numbers).

Command	Function (continued)
RESET	Clears the X and Y delimiters.
SAMPLE	Invokes the User Exits panel for the specified module. Natural Construct automatically issues the sample command when the User Exit editor for a model is invoked that supports user exits. This only occurs if no user exits are specified.

SCAN	Scans for specified data in the source area. The syntax for the SCAN command is:
	`SCAN 'scanvalue'`
	which scans for text within the delimiters, or:
	`SCAN scan value`
	which scans for the entire text after the SCAN keyword, including spaces. You must use delimiters when the scan value begins with a non-alphanumeric character.

Note:	The SCAN command is executed from the first line displayed in the editor to the end of the text if the direction indicator is + (forward). If the direction indicator is - (backward), the command is executed from the last line displayed in the editor to the beginning of the text. Unless the X and Y delimiters limit the range, the CHANGE command performs changes within the entire edit buffer.
SHIFT	Shifts text between the X-Y delimiters to the right or left the specified number (*nn*) of spaces. The syntax for the SHIFT command is: `SHIFT -`*nn* which shifts the text to the left the specified number (*nn*) of spaces, or: `SHIFT +`*nn* which shifts the text to the right the specified number (*nn*) of spaces. The default is +3 spaces.

Positional Commands

Positional commands are edit commands positioning the code forward or backward in the User Exit editor. Specify positional commands at the > prompt. The following table lists the positional commands available within the User Exit editor:

Command	Function
+*nnnn*	Repositions the code forward the specified number (*nnnn*) of lines.
-*nnnn*	Repositions the code backward the specified number (*nnnn*) of lines.
+H	Repositions the code forward by half a panel.
-H	Repositions the code backward by half a panel.
+P	Repositions the code forward one panel.
-P	Repositions the code backward one panel.
ENTER	Repositions the code forward one panel, if the text or code was not changed.
BOTTOM, ++	Repositions the code forward, to the end of the text.
POINT	Repositions the line on which the .N line command was entered to the top of the current panel (for a description of the .N line command, see **Line Commands**, page 98.
TOP, --	Repositions the code backward to the top of the text.
X	Repositions the code to the line marked X.
Y	Repositions the code to the line marked Y.
nnnn	Repositions the code to the specified line, where *nnnn* is the line number (if the editor has line numbers).

Generated Code Editor

Use the Generated Code editor to edit code in a generated module. The editor you use depends on the platform on which you are running Natural Construct. For information about invoking the Generated Code editor, see **Edit Generated Source Function**, page 71.

The following sections describe the Generated Code editor on mainframe and Unix platforms. For information about using the editor on your platform, consult the appropriate section.

Generated Code Editor for Mainframes

On mainframe platforms, the Generated Code editor in the Natural Construct environment is equivalent to the Program editor, Data Area editor, and Map editor in the Natural environment. The following example shows the Generated Code editor used on mainframe platforms:

```
>                                    > + Program     : INSMAINT Lib: NCSTV231
 Top    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
   0010 **SAG GENERATOR: OBJECT-MAINT
   0020 **SAG TITLE: Object Maintenance ...
   0030 **SAG SYSTEM: Natural-CONSTRUCT
   0040 **SAG GDA: CDGDA
   0050 **SAG DESCS(1): This program is used to maintain the....
   0060 **SAG HEADER1: INSURANCE SYSTEM
   0070 **SAG HEADER2: POLICY MAINTENANCE
   0080 **SAG DIRECT-COMMAND-PROCESS: X
   0090 **SAG ACTIONS: 0101010101010000
   0100 **SAG OBJECT-NAME: INSSUBP
   0110 **SAG BROWSE-PROGRAM: INSBROWS
   0120 **SAG MAX-WINDOWS: 2
   0130 **SAG MAP-NAME(1): INSMAP1
   0140 **SAG UPPER-BOUNDS(1,1): 015
   0150 **SAG SCREEN-OCCURS(1,1): 006
   0160 **SAG SCROLL-LINE(1,1,1): 015
   0170 **SAG SCROLL-COL(1,1,1): 019
   0180 **SAG SCROLL-LINE(1,1,2): 020
   0190 **SAG SCROLL-COL(1,1,2): 033
   0200 **SAG UPPER-BOUNDS(1,2): 006
        ....+....1....+....2....+....3....+....4....+....5....+... S 758  L 1
```

Generated Code Editor on Mainframe Platforms

Enter commands on the command line. These commands include any commands that can be entered at the Natural Next prompt.

The following additional commands are available in the Generated Code editor for editing generated code. The underscored part of each word indicates the minimum characters required to execute the command:

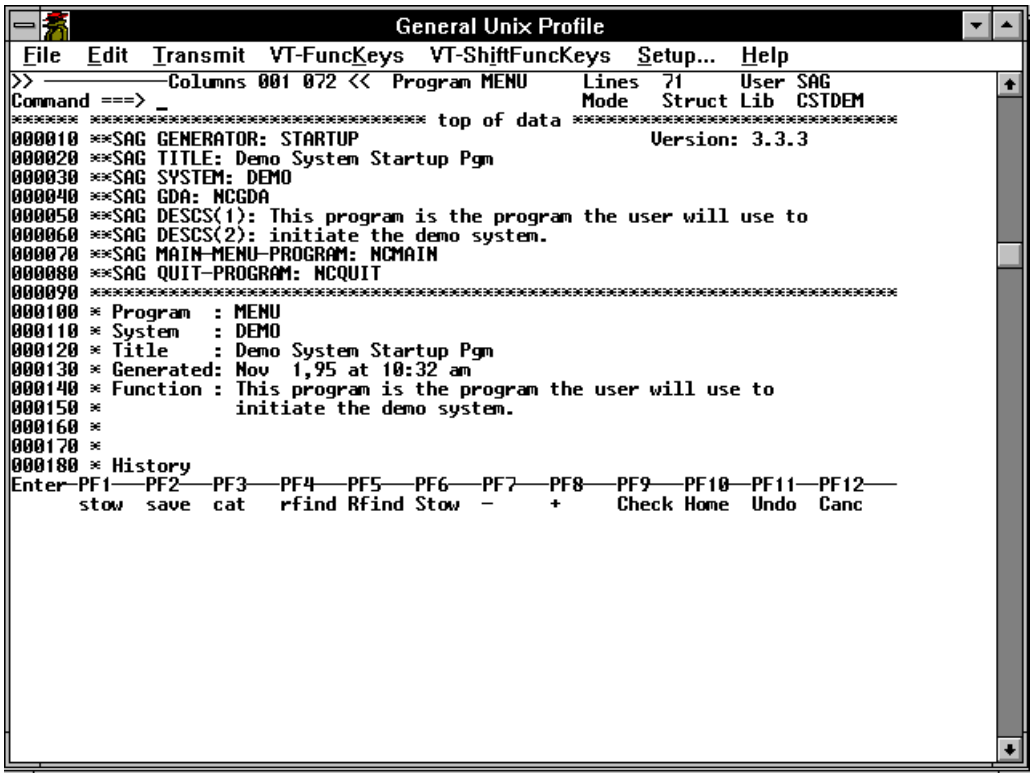
Command	Function
CHECK	Performs a Natural syntax check on the code in the edit buffer.
RUN	Performs a Natural RUN command on the code in the edit buffer (if the code is a program). Otherwise, the Natural CHECK command is performed.
SAVE (*module name*)	Saves a specification and module with the name specified. If no name is specified, the module name in the Specification/Module field is used.
STOW (*module name*)	Stows a specification and module with the specified name. If no name is specified, the module name in the Specification/Module field is used.

Generated Code Editors for Unix

On Unix, there are two Generated Code editors for Natural Construct. Use either the ISPF editor provided with Natural or the Unix Visual (vi) editor available on your system (by default, the ISPF editor). For information about setting the default editor, see the EDITOR profile parameter in the Natural documentation for Unix.

ISPF Editor

The Natural Construct ISPF editor is equivalent to the Natural Program editor — a Software AG editor tailored for the Natural environment. Use the ISPF editor to display and/or edit all Natural Construct modules. For more information, see the Natural ISPF documentation.



ISPF Generated Code Editor for Unix

vi Editor

To use Unix's vi editor, change your default editor. For more information, see the EDITOR profile parameter in the Natural documentation for Unix.

For information about using the vi editor, consult the Unix documentation.

IMPLEMENTATION METHODOLOGY

This chapter explains the implementation methodology for Natural Construct.

The following topics are covered:

Natural Construct Implementation Methodology

As a code generator, Natural Construct plays a vital role in the implementation process. Because an increasing amount of code results from Natural Construct models in a Natural Construct-generated application, and less from user exits and non-Natural Construct-generated modules, implementation techniques (methodology) used in the models becomes important to the development process.

Natural Construct models can have a similar effect on application design. When mapping business functions to computer functions, the set of available models has an important role in system design. Since Natural Construct can generate entire processes, it can generate *building blocks* that can assemble into full applications.

Natural Construct implies and uses numerous standards. These standards include not only coding practices, but also techniques to interrelate the various modules supported by Natural.

One fundamental objective of using these standards is to minimize the impact of future changes on generated applications (the result of ongoing system maintenance and development). To achieve this, Natural Construct-generated applications make extensive use of reusable application components in the form of Natural subprograms, copycode, and external data areas.

Who Uses Global Data?

Generally, generated programs (Natural object type P) access a common global data area (GDA). Generated subprograms are passed any required global variables as parameter data and do not define their own GDA. (An exception may be warranted if the subprogram requires more data storage than is remaining within the available USIZE.)

By default, Natural Construct assumes that generated helproutines do not require access to the GDA for the program. This allows helproutines to be easily shared across multiple applications. However, there are no restrictions on a helproutine's use of its own GDA. In some circumstances, the helproutine's ability to retrieve and assign global variables can enhance functionality.

Natural Construct does not supply models to generate external subroutines. Subprograms provide a more structured interface between the calling and the called modules and are less likely to cause side effects, since the parameters available to the subprogram are determined and controlled by the invoking module.

Note: As the Generation subsystem supports external subroutine generation, you can create your own subroutine models.

Default Global Variables

Natural Construct's default GDA is called CDGDA. Use this GDA as is, or copy and extend it as required for each application. All global variable names begin with two hash (#) characters, an easy way to identify global variables within generated modules. (The standard + prefix is not used as it is only supported for level 1 variables.)

Generated applications use the following default global variables:

Level	Field Name	Format	Length	Occurs
1	DIALOG-INFO			
2	##COMMAND	A	60	
2	##MAIN	A	8	
2	##QUIT	A	8	
1	MSG-INFO			
2	##MSG	A	79	
2	##MSG-NR	N	4	
2	##MSG-DATA	A	32	(1:3)
2	##RETURN-CODE	A	1	
2	##ERROR-FIELD	A	32	
2	##ERROR-FIELD-INDEX1	P	3	
2	##ERROR-FIELD-INDEX2	P	3	
2	##ERROR-FIELD-INDEX3	P	3	
1	ERROR-INFO			
2	##LAST-PROGRAM	A	8	
2	##LAST-ERROR-TIME	T		
1	PASS			
2	##USER	A	8	

Note that the supplied GDA is divided into several structures. Each structure serves a particular function within the generated application. These structures are described in the following sections.

DIALOG-INFO Structure

This structure contains information shared among all modules (programs, subprograms, and helproutines) that present the end user with an input panel.

The variables within this structure pass commands between processes and contain the names of the end user's main menu and termination programs. The values for the ##MAIN and ##QUIT variables are assigned by each application's startup program (often called "menu"). These values may vary, depending on the end user.

To generate programs that assign these global variables, use the Startup model.

MSG-INFO Structure

This structure contains information shared among all modules (programs, subprograms, and, optionally, helproutines) that need to exchange message information.

To accommodate environments using message numbers, as well as those using message text, this structure contains both a message text field (##MSG) and a message number field (##MSG-NR). We recommend you choose a single message-passing technique for all processes within an application, or ideally for all processes within an organization. The technique used (message numbers or text) varies, depending on your standards and requirements.

The advantages of using message numbers include:

- You can control and maintain messages in a central location.
- You can easily implement language-independent messages.
- You can change messages without changing programs.
- You can display messages in the language of the currently user (the value of *Language for the user at generation/test time).
- You can reduce program USIZE requirements.

The disadvantages of using message numbers include:

- The effort involved in checking the suitability of available messages. Adding new messages reduces development productivity. (Minimize the impact of this disadvantage by using statement models, such as the Reinput model.
- You must perform a database call to retrieve each message issued during execution.
- Messages are generated into the application in the language of the developer (according to the value of *Language at generation time).
- If message numbers are not accompanied by appropriate comments, the module source can be difficult to read.

Another variable within the MSG-INFO structure is ##MSG-DATA (1:3), which substitutes variable information in the supplied message. To accommodate the use of supplied data throughout the applications, code all INPUT statements with a Text option as follows:

```
INPUT WITH TEXTMSG-INFO.##MSG, (or *MSG-INFO.##MSG-NR,)
MSG-INFO.##MSG-DATA(1),
MSG-INFO.##MSG-DATA(2),
MSG-INFO.##MSG-DATA(3)
```

To associate a severity indicator with each message, the MSG-INFO structure contains a return code field. The return code values may vary between applications. However, we recommend you use the following standard:

Return Code	Meaning
blank	Informational message only.
W	Warning message; user may proceed.
E	Error message; corrections required before proceeding.

The MSG-INFO structure also contains an error field variable. This variable returns the name of the field (and occurrences where applicable) to the pertaining error message pertains. This field name must be returned by the validation or object subprogram for the dialog program to mark the variable on an input panel or error report.

ERROR-INFO Structure

This structure allows an application to easily recover from a Natural error. By default, applications generated by the supplied Natural Construct models do not contain any ON ERROR blocks within the generated code. Instead, it is best to use a Natural error transaction program to centrally control the error handling for an application (see the *ERROR-TA system variable in the Natural documentation).

When you use an error transaction program, take precautions to ensure that errors do not go undetected. These precautions are described in the following sections.

Develop Without an Error Transaction

During the development phase, do not assign an error transaction — Natural should handle all error processing. This prevents loss of error information during the error-prone implementation phase of an application. During the final implementation phase, and in the testing phase, you can set the *ERROR-TA system variable to trap errors itself and provide the user with error information.

Under Natural Security, you can achieve toggling between using and not using an error transaction by defining an error transaction to the application. Without Natural Security, you can choose to have two startup modules. However, only one sets the *ERROR-TA system variable. Create both of these startup transactions using the supplied Startup model.

Recovering from Errors

When an error occurs, Natural automatically invokes the error transaction assigned to the *ERROR-TA system variable and stacks pertinent information about the error, (such as the error number, module name, and line number). One of the global variables within the ERROR-INFO structure is called ##LAST-PROGRAM. It allows the error transaction to transparently restart the application at the last point prior to the error. ##LAST-PROGRAM contains the name of the last program (as opposed to a subprogram or helproutine) executing at level 1 prior to the error. The FETCH command should be part of the error recovery procedure.

All programs executing at level 1 and containing an input panel should assign ##LAST-PROGRAM, the value of *Program, during the module initialization process. When the program is FETCHed by the error transaction, error information is passed in the MSG-INFO structure to alert an end user that an error has occurred.

Preventing Error Cycles

The error handling mechanism uses extreme caution to prevent cycles in the error trapping flow. A cycle during error trapping is normally accompanied by an “Escape From Error Loop” message in Natural. The following table lists the types of error cycles and recommendations to prevent error cycles from occurring:

Error Cycle	Recommendations

Error in error transaction	The best way to prevent an error from occurring in your error transaction is to keep this program as short and simple as possible. We recommend the following error transaction (supplied as CDERRTA):
	`ASSIGN *ERROR-TA = ' '`
	`STACK TOP DATA *PROGRAM`
	`FETCH 'CDERROR'`
	`END`

Error Cycle	Recommendations (continued)
	Notice that the first thing the error transaction does is reset the error transaction variable. Therefore, if an error occurs during error processing, Natural's default error handling mechanism is used. The error transaction then FETCHes the name of the program responsible for recovering from the error. It then passes the program its own name on the stack and the error for recovery program to reassign *ERROR-TA as a final step in the recovery process.
Error in last program	Part of the error recovery process should be to FETCH the last active level 1 program. If the trapped error occurred in this program, Natural reinvokes the error transaction (prior to any INPUT statements within the program) without any intermediate screen communication. The /##LAST-ERROR-TIME global variable allows the error transaction to detect and recover from this cycle. If the time between the current error and the previous error is small (by default 10 seconds), the error transaction program presents the user with a panel identifying the error and does not FETCH the last program. By changing the global data area (GDA) to match the application's GDA, you can use the supplied error handling program (CDERROR) in your generated applications. You may also consider having the error transaction log errors to a file, to help uncover and prioritize production problems.

PASS Structure

The contents of this structure will vary for each application. It normally contains such things as security and profile information that is shared across all modules within an application.

Note: Only include variables within the PASS structure to pass them to all application-specific subprograms.

Using a Natural Command Processor

Most Natural Construct models have the direct command line available at the bottom of each panel. You can enter Natural command processor commands on the direct command line to navigate through a menu hierarchy without viewing each intervening menu.

To use direct commands, you must be familiar with the menu structure, menu codes, and parameters required at each menu level. For example, you can enter “MT P D,123333” on the direct command line on the main menu to invoke the Product Table maintenance panel in the Demo system. This functionally is equivalent to entering “MT” at the main menu, entering “P” at the Maintain Tables submenu, and then entering “D” and “123333” on the Maintain Product Table panel to display the specified product.

Conversely, when you enter a Natural command processor command on the direct command line, you do not need to know the menu structure details of the application. Applications designed to use processor commands make it easier for users who are unfamiliar with the application to maneuver within the system. For example, you can enter “MAINTAIN PRODUCT 123333” to reach the Product Table maintenance panel.

You create Natural command processors using the Natural SYSNCP utility (for more information, see the SYSNCP utility in the Natural documentation or **Creating Natural Command Processors**, page 639.

When you design Natural command processors for Natural Construct-generated applications, we recommend the following guidelines:

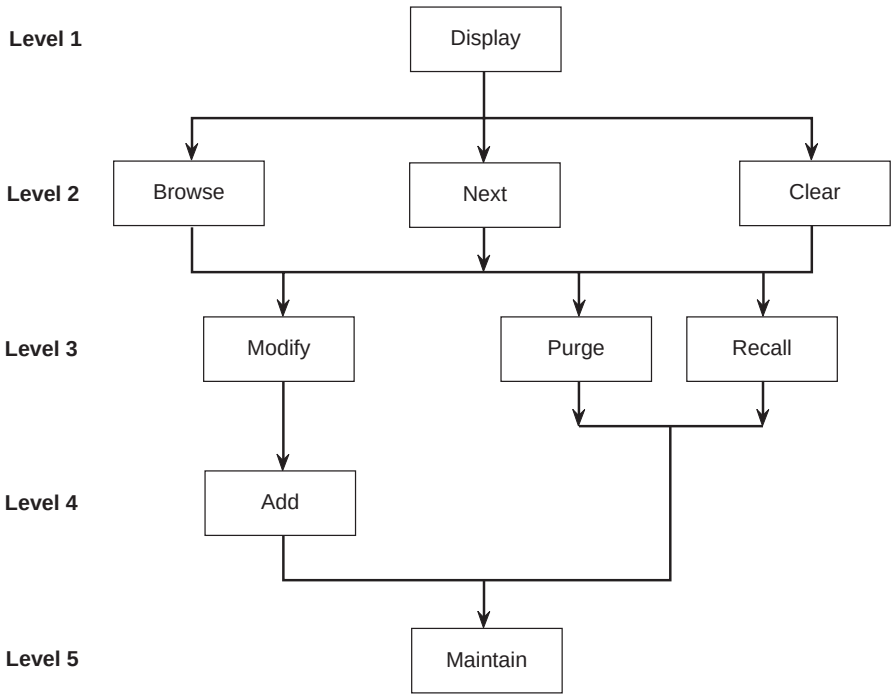
- Use object-oriented design methods when developing applications to reduce the impact of future changes.
- Implement functional security to restrict access to the dialog actions, if desired.

Implementing Object Maintenance Dialog Actions

Processor commands generally have two parts — an action and an object. For example, the DISPLAY PRODUCT, ADD PRODUCT, and MAINTAIN PRODUCT processor commands combine the Display, Add, and Maintain actions with the Product object. You can abbreviate the processor commands to a minimum unique length and/or define keyword synonyms.

Because module names are more likely to change than command names, restrict the use of module names in the runtime actions of a command processor. This reduces future maintenance requirements.

If Natural Security is installed, you can restrict user access to processor commands. While you have some flexibility establishing security levels that best fit your organization's needs, the following illustration displays the conceptual levels of security associated with the object maintenance dialog actions:



Security Levels Associated with Object Maintenance Dialog Actions

Each higher level of security inherits the actions defined for the previous levels. For example, users with access to level 3 actions (Modify or Purge and Recall) automatically have access to level 1 and 2 actions (Display, Browse, Next, and Clear). Users with access to level 5 actions automatically have access to all actions.

The diagram on the previous page illustrates two paths of inheritance. Some users have access to the Display, Browse, Next, and Clear actions (all level 1 and 2), as well as the Purge and Recall actions. Others have access to the Add and Modify actions, but not the Purge and Recall actions. Users with level 5 security privileges (Maintain action) inherit access to both paths.

The following table summarizes this security structure:

Level	Permissions	Description
1	Display	Accessible to all users.
2	Browse, Next, Clear	Usually accessible to all users and inherits level 1 permissions.
3	Modify or Purge/Recall	Accessible only to users with permission to modify records. Inherits level 1 and 2 permissions.
4	Add	Inherits level 1, 2, and 3 permissions with the exception of Purge and Recall.
5	Maintain	Accessible to users responsible for system maintenance and inherits all permissions (levels 1, 2, 3, and 4).

The Display action is the most basic action for an object maintenance dialog panel. This is explained in the following scenarios:

- All users accessing maintenance applications should have access to the Display action.
- The name of the module to FETCH should be contained in the Display action processor. Other actions can be directed to the correct module by issuing the DISPLAY command internally. Direct command processing involves issuing other direct commands.

  On the generated panel, users only view the actions they can access. Using Natural Security, you can restrict permissions by limiting the use of keywords, objects, and commands for individual users. For example, a user with level 1 permissions for the PRODUCT object only has access to the DISPLAY PRODUCT, BROWSE PRODUCT, and NEXT PRODUCT commands. Remember to include DISPLAY keyword permissions for all users who require access to the object.

  When developing Natural command processors to use with Natural Construct, we recommend using these guidelines:

- When modules are FETCHed by a processor command, either use the same GDA as the startup program or capture and restore the original GDA.
- Remember that the action codes specified in the Natural command processor runtime actions must correspond to the action codes defined in the #CODES table for the CDACT subprogram. These actions are language independent.
- When using SYSNCP to validate processor commands, define these commands as global. The processor commands are then accessible from any direct command line in the application.

  For more information about coding the actions for Natural command processors, see **Creating Natural Command Processors**, page 639.

Using Parameter Data Areas

External parameter data areas (PDAs) play a fundamental role in minimizing maintenance efforts. To ensure consistent field formats and lengths, external PDAs define both the receiving fields for a subprogram, and the local data areas (LDAs) in the calling modules.

To accommodate parameter changes in subprograms, it is important to define all PDAs with few level 1 fields (preferably a single level 1 field that matches the name of the data area). This allows the calling module to supply the list of parameters to the invoking subprogram by specifying the level 1 name(s). If the subprogram parameters are extended, all invoking modules can be corrected by recompiling them (assuming the existing parameters are upwardly compatible).

Another advantage of using PDAs with a single level 1 field is that you can easily divide the data area into input and output fields. You may also have several data areas containing the same field names, without the danger of conflicting variables.

To demonstrate the naming conventions used by supplied data areas and subprograms, assume a subprogram name of CDSAMPL for the following example.

Example of naming conventions

Level	Field Name	Format	Length
1	CDSAMPLA		
2	INPUTS		
3	VAR1	A	10
3	VAR2	A	10
2	INPUT-OUTPUTS		
3	VAR3	A	10
3	VAR4	A	10
2	OUTPUTS		
3	VAR5	A	10
3	VAR6	A	10

A typical calling module performs the following functions:

- Assigns CDSAMPLA INPUT/INPUT-OUTPUT parameters.
- CALLNATs 'CDSAMPL' CDSAMPLA MSG-INFO.
- Processes CDSAMPLA OUTPUT/INPUT-OUTPUT parameters.

Note: We highly recommend using the structure name to fully qualify all references to variables within a structure (such as parameter data and view elements). This makes it easy to rename variables globally using the SCAN command — without unintentionally changing field names.

Differentiate between parameters specific to one subprogram and those common to a class of subprograms. For example, you should not merge message and error handling information in the same PDA as variables specific to one subprogram. To provide ultimate flexibility for future maintenance, most subprograms use multiple PDAs.

For example, an online browse subprogram uses the following PDAs:

- Parameters specific to a subprogram. (For browse subprograms, this PDA contains the key value returned to the calling program. By default, you do not create an external PDA for this.)
- Parameters specific to a class of subprograms. This PDA (CDSELPDA) contains the action returned from the browse subprogram. You have an option to extend this PDA.
- Message information used by all subprograms. This PDA (CDPDA-M) contains messages, return codes, and error fields shared by all processes. It must match the MSG-INFO structure in the application's GDA.
- Dialog information used by all subprograms that have input panels. This PDA (CDPDA-D) contains dialog information, such as the direct command line and the name of the main menu program. It must match the DIALOG-INFO structure in the application's GDA.
- Information shared across all application-dependent subprograms. This PDA (CDPDA-P) contains application-specific information, such as user privileges and preferences. It must match the PASS structure in the application's GDA.

  When using multiple data areas, you can implement changes to standard application functionality (such as message passing) across all subprograms by changing a single data area.

  The technique of passing global information to subprograms that define matching PDAs allows the sharing of selected global data across the entire application, irrespective of artificial subprogram boundaries. By using the same variable names in both the PDAs and GDA, the application can share copycode members among programs and subprograms. To extend these data structures, add the new variables to the GDA structure, match the PDAs, and recompile the application. No other source changes are necessary.

Capturing and Restoring the Environment

All processes that depend on a certain environment (for example, window size or location) must capture the current global environment prior to altering any global settings. Additionally, the previous environment must be restored prior to terminating the process. Called processes, rather than the calling process, must capture and restore the previous environment since modules (such as helproutines) can be invoked without the knowledge of the invoking module.

Natural Construct supplies the CDENVIR subprogram to help you capture and restore the environment. Typically, this subprogram should be invoked (using a CALLNAT statement) by all processes that write information to the terminal. CDENVIR should be invoked once at the beginning of the process to capture the environment, and again at the end to restore it.

Using Layout Maps

Many of the Natural Construct models allow you to use maps to create input panels. You can create the map using either the Natural Map editor or the Map model. (A map generated by the Map model is especially useful for programs generated by the Object-Maint-Dialog model that contain scroll regions which must be defined.)

Some layout maps are supplied with Natural Construct. Use these maps to standardize the look and functionality of your input panels. The following table describes the layout maps provided for the models indicated:

Model	Layout Map	Description
All models	CDLAY	Use with any supplied model (generic layout map).
Browse	CDLAYSC1	Use with Browse models.
Browse-Select	CDLAYSL1	Use with Browse-Select models that support an action column.
Browse-Select	CDLAYSL2	Use with Browse-Select models that do not support an action column (required for multiple languages).
Maint	CDLAYFM1	Use with Maint model (primary file only).
Maint	CDLAYFM2	Use with Maint model (primary and secondary file or primary file with multiple-valued fields and/or periodic groups).

Model	Layout Map	Description (continued)
Maint	CDLAYFM3	Use with Maint model to support push buttons (primary file only).
Maint	CDLAYFM4	Use with the Maint model to support push buttons and scroll buttons (primary and secondary files or primary file with multiple-valued fields and/or periodic groups).
Menu	CDLAYMN1	Use with the Menu model.
Object-Maint-Dialog/Subp	CDLAYMP1	Use with the Object-Maint-Dialog or Object-Maint-Dialog-Subp models when the map is created using the Map model.
Object-Maint-Dialog/Subp	CDLAYOM1	Use with the Object-Maint-Dialog or Object-Maint-Dialog-Subp models when the map is created in the Map editor.
Object-Maint-Dialog/Subp	CDLAYOM2	Use with the Object-Maint-Dialog or Object-Maint-Dialog-Subp models to support push buttons and/or scroll buttons.

When using layout maps, the page size is 23 and the line size is 80. This is true of all layout maps except CDLAYSC1, which has a page size of 4, and CDLAYMP1, CDLAYOM1, and CDLAYSL1, which have page sizes of 25. The column shift is 1.

Note: The CDLAYFM3, CDLAYFM4, and CLAYOM1 layout maps support push buttons and scroll buttons. To tailor these maps, change the control variables and dynamic attribute fields. For information about variables (and their attributes) that can be placed on these maps, refer to the descriptions in **Object-Maint Models**, page 297.

Defining PF-Keys for Generated Applications

To provide total flexibility for PF-key assignments within generated applications, Natural Construct does not generate any hardcoded PF-key references. Instead, all supported PF-key values and names are defined in the CDKEYLDA external local data area (LDA). All PF-key checks appear in the form:

```
IF *PF-KEY = #RETURN-KEY THEN
```

Note: You can assign the same PF-key value to multiple PF-key variables, if only a single value is required in any given module.

Note: For information on defining PF-keys for object browse dialogs, see **Object-Browse Models**, page 293.

Standard PF-Keys

By default, CDKEYLDA contains the following PF-keys:

Variable Name	PF-key	Name	Function
#HELP-KEY	PF1	help	Displays help information.
#RETURN-KEY	PF2	retrn	Displays the previous menu.
#QUIT-KEY	PF3	quit	Quits (terminates) the application.
#ADD-KEY	PF4	add	Adds a record.
#FLIP-KEY	PF5	flip	Changes (or flips) the format of the lines displaying the PF-keys. You can display PF-keys in three different formats, as well as change the intensity and color of the lines.
#PLACE-KEY	PF6	place	Places the current browse window in a new location.
#PREFERENCES-KEY	PF6	pref	Preserves the contents of a maintenance panel after a maintenance action is performed.
#BACKWARD-KEY	PF7	bkwrd	Scrolls backward (up) through data.
#FORWARD-KEY	PF8	frwrd	Scrolls forward (down) through data.

Variable Name	PF-key	Name	Function (continued)
#HARDCOPY-KEY	PF9	print	Prints the output of browse and browse-select programs.
#LEFT-KEY	PF10	left	Scrolls to the panel on the left.
#RIGHT-KEY	PF11	right	Scrolls to the panel on the right.
#MAIN-KEY	PF12	main	Displays the main menu.
#CONFIRM-KEY	ENTR	enter	Confirms update actions.

Extended PF-Keys

PF-keys 25 through 48 are associated with push buttons or pull-down menus in a workstation environment. They are not displayed at the bottom of generated panels, but can be selected using a mouse, the cursor, or the PF-keys described below. Selecting the extended keys with a mouse or cursor activates the keys programatically (for more information, see the CCACTBUT and CCSCROLL copycode in the Natural documentation).

Action-Related PF-Keys

PF-keys 25 through 32 help users perform an action quickly. The following table describes their individual functions:

Variable Name	PF-key	Name	Function
#ADD-KEY2	PF25	add	Adds an action.
#BROWSE-KEY	PF26	browse	Browses an action.
#CLEAR-KEY	PF27	clear	Clears an action.
#DISPLAY-KEY	PF28	display	Displays an action.
#MODIFY-KEY	PF29	modify	Modifies an action.
#NEXT-KEY	PF30	next	Displays the next action.
#PURGE-KEY	PF31	purge	Purges the action.
#RECALL-KEY	PF32	recall	Recalls the last purged action.

Note: The order of the #ACTION-KEYS must exactly match the order of the actions in CDACTA.

Scroll-Related PF-Keys

PF-keys 33 to 40 allow users to scroll backward and forward within a single scroll region. Each panel contains up to 4 scroll regions. Use the PF-keys described below to scroll backward and forward through the records in each scroll region. The standard PF7 (bkwrd) and PF8 (frwrd) keys scroll within all regions. If a panel contains multiple pages, the standard PF-keys move backward and forward between pages.

Variable Name	PF-key	Function
#BACKWARD1-KEY	PF33	Scrolls backward in scroll region 1.
#FORWARD1-KEY	PF34	Scrolls forward in scroll region 1.
#BACKWARD2-KEY	PF35	Scrolls backward in scroll region 2.
#FORWARD2-KEY	PF36	Scrolls forward in scroll region 2.
#BACKWARD3-KEY	PF37	Scrolls backward in scroll region 3.
#FORWARD3-KEY	PF38	Scrolls forward in scroll region 3.
#BACKWARD4-KEY	PF39	Scrolls backward in scroll region 4.
#FORWARD4-KEY	PF40	Scrolls forward in scroll region 4.

Miscellaneous PF-Keys

The following table describes miscellaneous PF-keys and their functions:

Variable Name	PF-key	Function
#DIRECT-CMD-KEY	PF41	Displays the direct command line.
#RESERVED	PF42	Reserved for future use.
#RESERVED	PF43	Reserved for future use.

Changing PF-Key Display Options on Generated Panels

You can enable a flip PF-key to allow users to switch between PF-key lines or available actions on generated panels.

➢ To change the current PF-key display (lines or actions), the user can:

1 Press the flip key (PF5, by default) on the panel.
The PF-Key Display Options window is displayed:

```
CDFLIP                  Natural Construct                 CDFLIP11
May 09                PF-Key Display Options                1 of 1

   PF-key Display Formats
      _ Enter-PF1---PF2---PF3---PF4---PF5    ...   (Two line ISA)
              help  retrn quit  etc.  etc.

      _       help  retrn quit  etc.  etc.   ...   (Half display)

      _ PF1=help,PF2=retrn,PF3=quit,PF4=etc. ...   (keys 1-24 SAA)

      _ F1=help,F2=retrn,F3=quit,F4=etc.     ...   (keys 1-24 PC)

      Display Characteristics
        _ Intensify          _ Cursor sensitivity     _ Keys 1-12
        _ Reverse video      _ Color                  _ Keys 13-24

   _ Available commands
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF
apply help  retrn
```

PF-Key Display Options Window

2 Mark one of the formats listed in PF-key Display Formats.

3 Mark one or more of the Display Characteristics.
The following table describes the display characteristics:

Display Characteristic	Description
Intensify	Changes the intensity of the PF-key line. (If the line is currently intensified, marking this field turns the intensify option off.)
Cursor sensitivity	Supports cursor selection.
Keys 1-12	Displays the PF1 to PF12 keys.
Reverse video	Displays the PF-key line in reversed video.
Color	Changes the color of the PF-key line. After marking this field and pressing Enter, the PF-Key Color Specification window is displayed. For a description of this window, see **Changing the Color of the PF-Key Line**, page 126.

Display Characteristic	Description (continued)
Note:	This option is only available for terminals that support color.
Keys 13-24	Display the PF13 to PF24 keys.
Available commands	To display the list of available commands instead of the PF-key lines, mark this field.

4 Press Enter to apply the changes.

Changing the Color of the PF-Key Line

If the user marked the Color field in the PF-Key Display Options window, the PF-Key Color Specification window is displayed after the user presses Enter:

```
CDFLIP              Natural Construct             CDFLIP21
May 09          PF-Key Color Specification          1 of 1

First PF-key Line

  _ Blue         _ Green        _ White        _ Pink
  _ Red          _ Turquoise    _ Yellow       _


Second PF-key Line

  _ Blue         _ Green        _ White        _ Pink
  _ Red          _ Turquoise    _ Yellow       _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---
apply help  retrn
```

PF-Key Color Specification Window

➢ To change the color of the PF-key line, the user can:

1 Mark one color for the First PF-key Line.

2 Mark one color for the Second PF-key Line.

3 Press Enter to apply the new color and return to the panel.

Printing Hardcopy Documents from Generated Modules

You can enable a print PF-key to allow users to print a hardcopy document from information in the generated module.

➢ To print a hardcopy, the user can:

1 Press PF9 (print) on the panel.
The Hardcopy Print Options window is displayed:

```
CDHCOPY              Natural Construct              CDHCOPY1
May 09             Hardcopy Print Options             1 of 1

Please specify the following print options...

  Hardcopy device ........................
  Print lines per page ..................... 60_
  Maximum number of records to be written ... 100____
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---P
print help  retrn
```

Hardcopy Print Options Window

2 Type the name or number of the printer in Hardcopy device.
This value is the location to which documents are routed for printing.

3 Type the number of print lines per page in Print lines per page.
By default, 60 lines are printed per page.

4 Type the maximum number of records that can be output at any one time in Maximum number of records to be written.
By default, a maximum of 100 records can be written.

5 Press Enter to print the document.

Tailoring Your PF-Keys

Generated modules use the RESET INITIAL CDKEYLDA.*#variable-keyname* notation to set a PF-key, rather than a SET KEY statement. To set the return key, for example, the generated module contains the following line:

```
RESET INITIAL CDKEYLDA.#RETURN-KEY
```

The CCSETKEY copycode member performs the SET KEY commands. This method of using variable PF-key values at compile time provides the following benefits for generated applications:

- You can postpone decisions regarding PF-key standards (such as SAA) until all existing applications are converted.
- You can ensure that consistent PF-key values and names are used throughout an entire application or organization.
- You can apply changes to PF-key values or names for an entire application by changing a single data area and recompiling the application.
- Applications contain multilingual support for PF-key names.
- Applications do not need to incur the overhead of looking up externally-defined variable PF-key names at execution time.

To introduce new PF-keys or modify supplied values, simply extend the keys specified in CDKEYLDA and/or change the supplied initial values. The following section describes how to add a new PF-key.

Adding a New PF-Key

You can add standard PF-keys, which are available to all Natural Construct-generated applications, as well as non-standard PF-keys, which only apply to selected modules. The following sections describe how to add a standard and non-standard PF-key.

Adding a Standard PF-Key

➢ To add a standard PF-key:

1 Using SYSMAIN, copy the following Natural Construct modules into the current library:

Module	Description
CDKEYLDA	Local data area containing PF-key names and settings.
CCPFSTD	Copycode to assign the standard PF-keys.
CCSTDKEY	Copycode to process the standard PF-keys.
CDFLIP	Subprogram to support the Flip key function.

Note: Because Natural Construct retrieves the above modules from the current library, these changes are library/application specific.

2 Edit CDKEYLDA in the current library:

```
Local     CDKEYLDA  Library CSTDEVS                    DBID  18 FNR   4
Command                                                          > +
I T L Name                           F Leng Index/Init/EM/Name/Comment
- - - -------------------------------- - ---- -------------------------
    2 #PF-INDX                         P    3
    2 #PF-ARRAY-SIZE                   P    3 INIT<14>
    2 #KEY-INITS
    3 #HELP-KEY                        A    4 INIT<'PF1'>
    3 #RETURN-KEY                      A    4 INIT<'PF2'>
    3 #QUIT-KEY                        A    4 INIT<'PF3'>
    3 #ADD-KEY                         A    4 INIT<'PF4'>
    3 #FLIP-KEY                        A    4 INIT<'PF5'>
    3 #PLACE-KEY                       A    4 INIT<'PF6'>
    3 #BACKWARD-KEY                    A    4 INIT<'PF7'>
    3 #FORWARD-KEY                     A    4 INIT<'PF8'>
    3 #LEFT-KEY                        A    4 INIT<'PF10'>
    3 #RIGHT-KEY                       A    4 INIT<'PF11'>
    3 #MAIN-KEY                        A    4 INIT<'PF12'>
    3 #HARDCOPY-KEY                    A    4 INIT<'PF9'>
    3 #PREFERENCES-KEY                 A    4 INIT<'PF6'>
  *
  *   If you want a PF-Key to confirm
  *   update actions, change the
  *   initial values of #CONFIRM-KEY
  *   and #CONFIRM-KEY-MSG
    3 #CONFIRM-KEY                     A    4 INIT<'ENTR'>
  R 2 #KEY-INITS                           /* REDEF. BEGIN : #KEY-INITS
    3 #PFKEY                           A    4 (1:14)
  *   Initialize key names with
  *   numbers corresponding to
  *   their literal names in User
  *   error messages. Never insert
  *   unrelated fields between
  *   #KEY-NAME and #KEY-NAME-END
I   2 #KEY-NAME                        A   10 (1:14)
  R 2 #KEY-NAME                            /* REDEF. BEGIN : #KEY-NAME
    3 CDUTRANS                         A    1 (1:140)
    2 #KEY-NAME-END                    A    8 INIT<'CDKEYLDA'>
```

To add a standard key (PF9, for this example), change the initial value of #PF-ARRAY-SIZE from 14 to 15. After the line containing #CONFIRM-KEY, immediately add the following line:

```
3 #YOUR-KEY  A    4   INIT<'PF9'>
```

Change the array constants for the #PFKEY and #KEY-NAME variables from 14 to 15. Increase the array constant for CDUTRANS from 140 to 150.

3 Change the initial values of #KEY-NAME (A10/1:15).
#KEY-NAME contains the SYSERR message number Natural Construct uses to retrieve text in the current language (defined in *Language).

Edit the #KEY-NAME line of CDKEYLDA by entering ".E" (mainframe) or typing "E" (Unix) on the #KEY-NAME line. Then enter "S" (Single Value Initialization) in the Code field of the Initial Values and Edit Mask panel. Scroll to the end of the list to add the names for the new keys.

Example of changing initial values — Single Mode panel

```
      Index           #KEY-NAME(A10/1:15)
--------------------- ------------------------------------------------
(1)                   *8430/10
(2)
(3)
(4)
(5)
(6)
(7)
(8)                   *8431/10
(9)
(10)
(11)
(12)
(13)
(14)
(15)
```

By default, #KEY-NAME starts with message number *8430, which corresponds to the PF-keys for help, retrn, quit, add, flip, place, and bkwrd. At runtime, the CDUTRANS routine reads this text (in the language defined in *Language) into CDKEYLDA. It then parses it into 7 segments separated by slashes (/), and fills the corresponding *KEY-NAME(1) to *KEY-NAME(7) fields.

The /10 notation means each field is 10 characters long (for more information, refer to the CDUTRANS routine). Follow the same procedure for the other initialized value (*8431).

To initialize #YOUR-KEY name, add **nnnn* in the #KEY-NAME column beside (15), where *nnnn* is a 4-digit message number corresponding to the text in the CSTAPPL library. If the entry *nnnn* does not exist in CSTAPPL, use the SYSERR utility to add the number and name. Stow this LDA.

4 Edit CCSTDKEY and insert the following code immediately before the line containing WHEN NONE:

```
WHEN *PF-KEY EQ CDKEYLDA.#YOUR-KEY
     /* include any
     /* processing here
```

Save this copycode.

5 Edit CCPFSTD and add the following code on line 0050:

```
CDKEYLDA.#YOUR-KEY
```

Save this copycode.

6 Catalog the CDMNTPR subprogram.

Adding a Non-Standard PF-Key

To add a non-standard PF-key, which is used by one module, follow the instructions for adding a standard PF-key but omit Step 5. Instead of performing Step 5, insert the following code in the SET-PF-KEYS user exit for the program:

```
DEFINE EXIT SET-PF-KEYS
*
* Set non-standard PF-key
RESET INITIAL CDKEYLDA.#YOUR-KEY
END DEFINE
```

Confirmation Key Setup

Users are automatically asked to confirm Purge actions in programs generated by both the Maint model and the Object-Maint-Dialog model. After requesting a Purge, users are prompted to “Press enter to confirm purge.” (There is no confirmation required for Add or Modify actions.)

To change this default functionality, change two fields in the CDKEYLDA local data area: #CONFIRM-KEY and #CONFIRM-MSG-KEY.

The #CONFIRM-KEY field defines the key that must be pressed by the user to confirm the purge. The #CONFIRM-MSG-KEY field defines the message displayed informing the user of the key that must be pressed to confirm the purge.

#CONFIRM-KEY Field

By default, the value of the #CONFIRM-KEY field is ENTR. Changing the value of this field to anything other than ENTR, changes the functionality of the confirmation logic. If the #CONFIRM-KEY field is not ENTR, the Add, Modify, and Purge actions must all be confirmed with the new key defined in #CONFIRM-KEY for both the Maint and Object-Maint-Dialog models. If you change the #CONFIRM-KEY field, you must also change the #CONFIRM-MSG-KEY field so users know the correct key to press to confirm an action.

#CONFIRM-MSG-KEY Field

The value of the #CONFIRM-MSG-KEY field should match the value of the #CONFIRM-KEY field. Do not forget to change the value of this field after changing the confirm key, as users will not know what confirm key to use.

Example of changing the confirm key

To require users to confirm Add, Modify, and Purge actions for the Maint and Object-Maint-Dialog models, modify CDKEYLDA as follows:

```
#CONFIRM-MSG-KEY    INIT<‘PF15’>
#CONFIRM-KEY        INIT<‘PF15’>
```

This results in the following messages:

```
Press PF15 to confirm purge
Press PF15 to confirm modify
Press PF15 to confirm add
```

Defining Actions for Generated Applications

All Natural Construct-generated modules use externally-defined action and function codes. This functionality provides the following benefits:

- Applications use consistent action codes, making actions easier to identify for the user.
- You can modify the actions or names of an entire application by changing and recompiling the CDACTA parameter data area, CDACTL translation local data area, and CDACT subprogram.
- You can display all action names and abbreviations in the language of the user.
- You can centrally control action-level security within a single subprogram.

Note: For information on defining actions for object browse dialogs, see **Object-Browse Models**, page 293.

Add an Action

You can add up to 16 actions to the CDACTA parameter data area (PDA). This PDA can contain up to 30 actions, including one action for the flip option and 13 standard actions (#ADD, #BROWSE, #CLEAR, #DISPLAY, #MODIFY, #NEXT, #PURGE, #COPY, #RECALL, #REPLACE, #SELECT, #DETAIL, and #FORMER).

Warning: Do not remove any of the standard actions.

➢ To add an action:

1 Copy the CDACT subprogram, CDACTA parameter data area, and CDACTL translation local data area from your steplib (or SYSTEM library) into your application library.

2 In the CDACTA parameter data area, define the desired action as a redefined component of #APPL-ACTION.

3 Modify the CDACTL translation local data area in three areas:

- Increase the #USED-CODES level 2 variable to one greater than the current value (for example, from 13 to 14).
- Increase the array size of the TEXT-ARRAY level 4 variable from 130 to 140.

– Add "**nnnn*" to the end of the initialization of the #ACTION-INIT level 4 variable, where *nnnn* is a 4-digit message number corresponding to the text in the CSTAPPL library.
  If the entry *nnnn* does not exist in CSTAPPL, use the SYSERR utility to add the number and its text (in different languages, if desired). See the figure on the following page.

4 Recompile the CDACT subprogram with the updated CDACTL.

These changes only affect the current library. To apply changes to the entire installation, ask your Natural Construct administrator to change the CDACT subprogram, CDACTL translation local data area, and CDACTA parameter data area in the SYSTEM library.

Note: Since the Browse-Select models store selected actions positionally, always append new actions at the end of the CDACTA.#APPL-ACTIONS redefinition table.

For example, assume that CDACTA, CDACTL, and CDACT are in your application library and the application supports the Validate action:

1 Within CDACTA, define #VALIDATE as a redefined component of #APPL-ACTIONS and stow the modified CDACTA data area.
The modified CDACTA data area is displayed:

```
14:26:39               *****  E D I T  DATA  *****                  90-11-21
Parameter CDACTA    Library CSTDEV                            DBID  18 FNR   4
Command                                                              > +
I T L Name                            F Leng Index/Init/EM/Name/Comment
- - - --------------------------------- - ---- ---------------------------------
    3 #APPL-ACTIONS                   A    3 (1:30,1:3)
  R 3 #APPL-ACTIONS                          /* REDEF. BEGIN : #APPL-ACTIONS
    4 #ADD                            A    3 (1:3) /* Always extend this
    4 #BROWSE                         A    3 (1:3) /* list of actions at the
    4 #CLEAR                          A    3 (1:3) /* end since the browse
    4 #DISPLAY                        A    3 (1:3) /* select model stores
    4 #MODIFY                         A    3 (1:3) /* the selected actions
    4 #NEXT                           A    3 (1:3) /* as an occurrence number
    4 #PURGE                          A    3 (1:3) /* within this table.
    4 #COPY                           A    3 (1:3)
    4 #RECALL                         A    3 (1:3)
    4 #REPLACE                        A    3 (1:3)
    4 #SELECT                         A    3 (1:3)
    4 #DETAIL                         A    3 (1:3)
    4 #VALIDATE                       A    3 (1:3)
    2 OUTPUTS
  *
------------------------------------------------------------------ S 52   L 23
```

Modified CDACTA Data Area

2 Modify the CDACTL translation local data area by changing the initialization values of the #USED-CODES and #ACTION-INIT variables and increase the array size of TEXT-ARRAY.

Because message number *9000 is not in the CSTAPPL library, use the SYSERR utility to add the message number and the text, “validate”, for *Language 1. If the application is multilingual, remember to add the foreign-language texts for other supported languages.

The modified CDACTL Translation Local Data Area window is displayed:

```
Local     CDACTL    Library CST341S                          DBID  17 FNR  28
Command                                                              > +
I T L Name                            F Leng Index/Init/EM/Name/Comment
Top - ------------------------------- - ---- --------------------------------
  * * * Translation LDA for map text
  * * * used by map CDACT.
  * * * Note: this translation LDA
  * * * is being used by Construct to
  * * * display valid actions in the
  * * * Browse-Select* models.
  * * * If you want Construct to
  * * * reflect these changes then you
  * * * must recompile the modules
  * * * CDACT and CDACTCST.
    1 CDACTC
    2 #USED-CODES                     P    3 CONST<12> /* Number of codes
    1 CDACTL
    2 TEXT
    3 #FLIP-NAME                      A   10 INIT<'*8422,</10'>
    3 #ACTION-NAMES
I   4 #ACTION-INIT                    A   10 (1:#USED-CODES)
------------------------------------------------------------------ S 38   L 1
```

Modified CDACTL Translation Local Data Area

3 Stow the CDACT subprogram with the modified CDACTL.

Using Common Copycode Members

All modules use common copycode members at the beginning and end of a generated module, and after input panels. Additionally, the modules use nested copycode members to provide even greater functionality in your Natural Construct-generated modules. Using common copycode members allows you to easily incorporate future changes to your standards.

Supporting Standard Flip Key Functionality

All generated modules that have input panels support standard flip key functionality. Use the flip key processor, CDFLIP, to generate panels that allow users to change the layout and format of the PF-keys.

The invoking module can also pass a parameter to the CDFLIP subprogram, indicating that the module supports the display of a command list under the PF-key lines. This feature allows users to toggle between displaying the PF-key lines in the standard format or, displaying available commands. For more information, see **Changing PF-Key Display Options on Generated Panels**, page 125.

Implementing Multilingual Support

If you want your generated applications to support multiple languages at execution time, you must change some of the supplied components. These changes are described in the following sections.

Language-Independent Messages

To support language-independent messages, generate your modules with the message number option (available on the first model specification panel). Alternatively, your Natural Construct administrator can enforce this option by removing the choice from the panel and assigning the option in the clear subprogram associated with each model.

Note: All supplied messages are stored in the SYSTEM library, starting at message number *8000. In addition to English, these messages are provided in Danish, Dutch, French, German, and Spanish. Use the SYSERR utility to add other languages.

Language-Independent Panels

To support language-independent panels, you can create all panels as Natural maps with the correct language indicator used in the map names. These maps are then invoked using the “&” notation as the last character in the map name, and displayed according to the current value of *Language (for more information, see the Natural documentation).

Alternatively, you can create Natural maps containing only variables. These maps are then invoked using a 0 (zero) as the last character in the map name, and displayed according to the current values of the SYSERR numbers defined for each variable.

To maximize the power and productivity benefits of the WRITE, PRINT, and DISPLAY statements in your reports, as well as language independence, use the Browse-Select model instead of the Browse model. The reports are overlaid with a map supplying all panel and column headings, as well as input prompts.

Language-Independent Text, PF-Keys, and Action Codes

To provide complete language independence, Natural Construct-generated applications use text stored in certain local data areas (LDAs) and subroutines. This text is obtained at runtime from the CSTAPPL library, beginning at message number *8000. English is the default language supplied with the LDAs. Other languages provided include Danish, Dutch, French, German, and Spanish. You can add other languages using the SYSERR utility.

To support multilingual text, translate the English text corresponding to any valid message number in the CSTAPPL library to your supported language. You do not need to change any routines or LDAs in Natural Construct.

Note: To translate English text to another language, use the SYSERR utility. Log onto the SYSERR library and enter “Menu”. The list of SYSERR functions is displayed, from which you can select the Translate function.

Languages without Latin Roots

Languages that cannot handle Latin lower case (for example, Greek or Hebrew), encounter problems using the AD=T option on an INPUT statement. With these languages, you must remove all occurrences of AD=T from the generated code. To do this, include the following statement in the CSXCNAME global post-generation subprogram:

```
STACK TOP DATA FORMATTED 'AD=T' 'AD='
```

This statement is automatically substituted after generation.

For information about the languages to which this applies, see *Language in the Natural documentation.

DESIGN METHODOLOGY

This chapter explains the design methodology for Natural Construct.

The following topics are covered:

Introduction

As a code generator, Natural Construct plays an important role in the implementation process. Because more and more of the code results from Natural Construct models in a Natural Construct-generated application, and less and less from user exits and non-Natural Construct programs, the implementation techniques (methodology) used in the models becomes important to the development process.

Natural Construct models can have a similar effect on the design of an application, although it may not be as obvious. The set of available models plays an important role in system design when you are mapping business functions to computer functions. Since Natural Construct can generate entire processes, it follows that Natural Construct is generating *building blocks* which can be assembled into full applications.

Natural Construct Design Methodology

Natural Construct affects design methodology in two major areas:

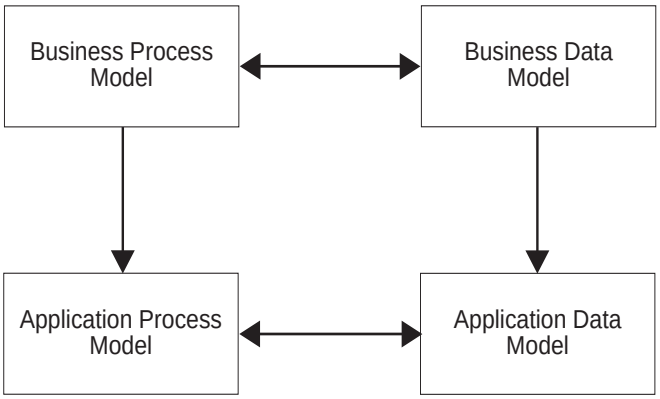
- the establishment of *objects* as a new data manipulation concept
- the building of complex processes by assembling Natural Construct-generated processes

Natural Construct and Data Design—The Object Approach

In the analysis phase of a traditional development life cycle, the Business Process model and the Business Data model are defined where the business processes are related to the business data objects:

Traditional Approach — The Analysis Phase

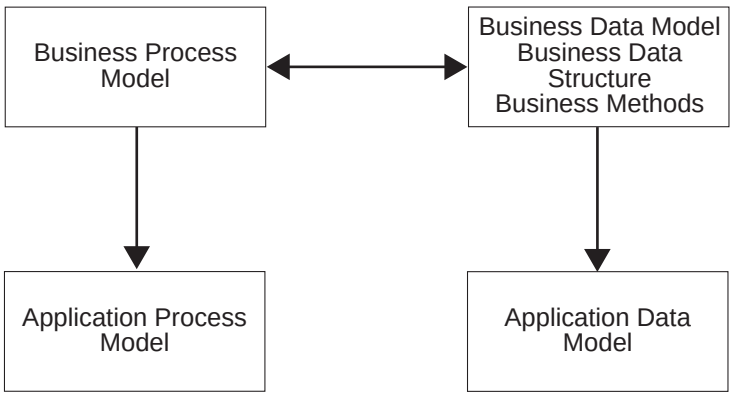
During the design phase, the Business Process model is typically mapped/decomposed into an Application Process model, and the Business Data model is mapped/decomposed into an Application Data model:



Traditional Approach — The Design Phase

The Application Process model represents the computer functions available to implement the business processes; the Application Data model represents the computer files available to implement the Business Data model. The application processes then use the computer files to access data.

In an object-oriented design, you can modify the structure as follows:



The Object-Oriented Approach

Business processes are mapped to application processes, but access the Business Data model directly. A mapping still exists between the business entities and the computer files, but it is transparent to the application processes. The Business Data model and the mapping between the Business Data model and the Application Data model must be formally defined. The definitions are part of the implementation (and not used for documentation purposes only).

The Business Data model contains information about the data structure mapping between a business object and its corresponding computer files, as well as information about how to perform actions (such as Add, Delete, Approve, etc.) on these business objects. The implementation of these actions form the data object's *methods,* which are invoked as a result of messages sent by an application process.

An application process that sends an Add message to a business data object causes the method for that object to be invoked to carry out the Add action. In this manner, a Business Data model encapsulates two distinct components: the structural mapping from business data to computer files and the semantics that surround the object's reaction to the various messages it receives in its lifetime.

The advantages of application processes accessing business data objects include:

- Creates a clearer distinction between data and processes. You can easily change application processes, since application implementations do not contain a mix of process and data-related code.
- Data objects are handled in a consistent manner. Code that is usually repeated throughout an application becomes centralized as part of an object.
- You can work at a higher level of data abstraction, as you do not have to be concerned about how an object is implemented. In the same way DBMSs increase productivity by hiding the physical data model, accessing data at the business level hides the database model from the developer.
- You can include the semantics relating to an object within the object, rather than within the application accessing the object.
- You can apply performance and tuning to the logical data model. Since the application does not know how objects are implemented, you can change the object implementation for performance purposes without concern for the application.
- You can build applications that can be easily distributed on a client-server architecture at a later date.

Defining Objects and Relationships

The following sections describe how to identify and normalize an object, the life cycle of an object (as supported by Natural Construct), and how to define object relationships and create data areas.

Identify Objects During Analysis

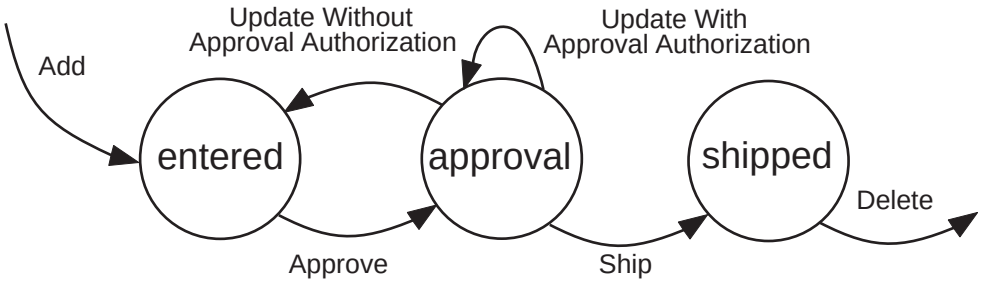
Objects are typically identified as a result of interaction with users, such as Joint Application Design (JAD) sessions. For example, the following Object/Relationship diagram might arise during a JAD session:

Object/Relationship Diagram

Relationships between objects are called *inter-object relationships*. When you delete an object, an inter-object relationship causes a block form of referential integrity. When you add an order, the corresponding customer must exist. When you delete a customer, no orders may exist for that customer.

The life cycle of an object is an important concept that defines what does or does not constitute an object. Each object goes through a series of states that represent its lifetime.

For example, an order can have a life cycle represented by the following diagram:



Object Life Cycle

An object must move through its life cycle as a single unit. An order may have many lines, but since all the lines move through the life cycle together, they are considered to be one object. Similarly, although a dialog might update both customer and order information, customer and order are considered to be two objects since they do not share the same life cycle.

Another important aspect of the life cycle is that it identifies the methods that have to be implemented. The order object is sent messages based on its life cycle: Add, Approve, Update, Ship, and Delete (both Update with Authorization and Update without Authorization can be implemented via the same method).

Along with identifying objects, you must also identify the attributes of an object. As with all aspects of analysis, interaction with business users provides the prime source of information.

Example of identifying attributes of the order object

```
01 ORDER
  02  ORDER-NUMBER (N8)
  02  ORDER-DATE (D)
  02  ORDER-LINES (1:30)
    03  PRODUCT (A5)
    03  QUANTITY-ORDERED (N5)
    03  PRICE (P7.2)
    03  ORDER-NOTE (A30)
    03  ORDER-DISTRIBUTION (1:5)
      04  ORDER-ACCOUNT (A5)
      04  ORDER-DISTRIBUTION-AMOUNT (P7.2)
```

Like most business objects, this object is not normalized. Although technically you do not need to add data types to an object during analysis (since analysis is implementation independent), we will assume a Natural implementation and include the data types.

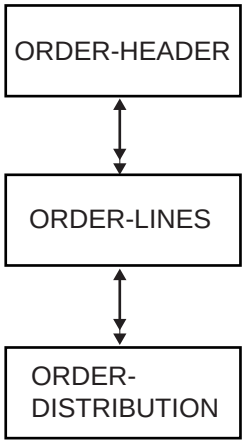
The limitations on an object's data types are determined by Natural and not by any DBMS. All Natural data types are allowed, including date data types and three-dimensional arrays.

You can use tools, such as Predict Case and Natural Architect Workstation, to assist in this analysis process.

Normalize Objects During Design

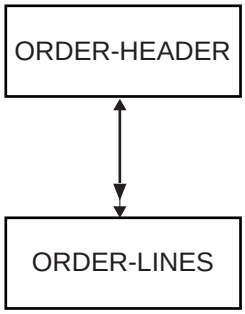
During the design phase, you normalize (break down) object data into the individual computer files that make up the object. How you do this depends both on the DBMS(s) used to implement the objects, as well as the degree to which you desire normalization. One of the benefits of the object approach is that you can choose a normalization strategy and change it later (if the application does not perform adequately, for example).

For the order object, the following fully-normalized structure is one possibility:



Normalized Structure

Another possibility for normalization (for Adabas) is:



Normalized Structure for Adabas

In the example for Adabas, you can implement the distribution concept as a periodic group on the ORDER-LINES entity.

The concept of normalization introduces new relationships called *intra-object relationships*. These relationships relate entities within an object and always behave in a cascading delete fashion. For example, when you delete an ORDER-HEADER, all corresponding ORDER-LINES are deleted as well.

Again, you can use both Predict Case and Natural Architect to assist in the design process.

Define All Entities and Relationships in Predict

After normalizing the object, you define the resulting entities in Predict. These entities include Adabas user views, VSAM views, DB2 tables, and RDB views. Since these views represent actual files, it is assumed that you have normalized the object during design and these files are the result. The entities can be:

- Created in Predict as a result of Predict Case schema generation.
- Created in Predict as a result of an upload from Natural Architect Workstation.
- Created in Predict as a result of a Predict Gateway upload from ADW, IEW, or Excelerator.
- Manually entered into Predict.

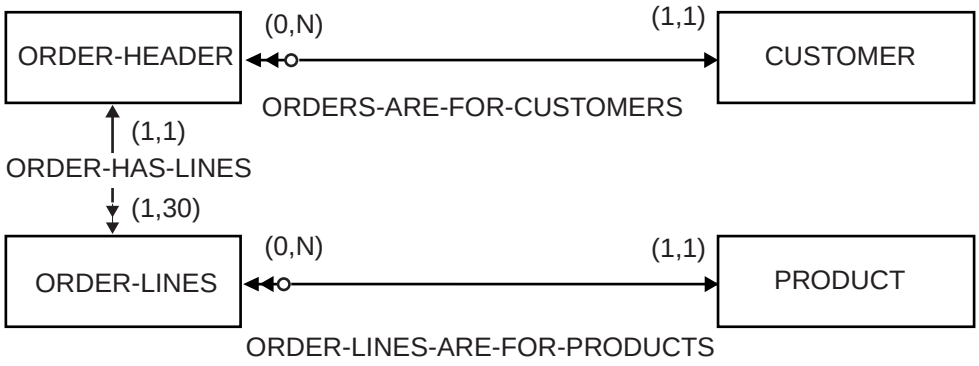
Add Object-Relationship Indicators

After defining the entities in Predict, you can identify which relationships Natural Construct will use. To do this, you can add relationships defined as type N (Natural Construct) within Predict.

You can further identify intra-object relationships by adding cascading delete and update constraints. To enforce referential integrity rules, you can identify inter-object relationships by specifying restricted delete and update constraints.

Note: For DB2 users, Natural Construct recognizes all type R relationships. However, since cascading deletes and restricted updates are not valid for DB2, intra-object relationships must be defined as type N relationships.

Consider the following normalized example:



Object Relationships

In the example above, the ORDER-HEADER and ORDER-LINES entities belong to the same object. Therefore, the ORDER-HAS-LINES relationship is an intra-object relationship and should be defined appropriately with restricted update and cascading delete constraints.

You define the intra-object relationship on the Predict Modify Relationship panel as follows:

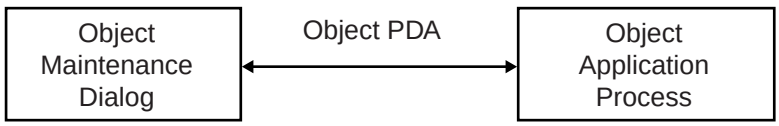
```
16:10:29            *****  P R E D I C T  4.2.1  *****             2001-12-24
                         - Modify File relation -
File relation ... NCST-ORDER-HAS-LINES            Modified 1998-08-24 at 11:37
Type ...........* N  NATURAL CONSTRUCT                  by DEVSS
Keys .. SAG-CST                                                    Zoom: N

Cardinality ..* 1  : CN
File 1                                              Minimum ... 1
  File ID ....* NCST-ORDER-HEADER                   Average ... 1.00
  Field ID ...* ORDER-NUMBER                        Maximum ... 1
File 2                                              Minimum ...
  File ID ....* NCST-ORDER-LINES                    Average ... 5.00
  Field ID ...* ORDER-LINE-KEY                      Maximum ... 30
Constraint attributes
  Update type .....* N  re-Number suffix
  Delete type .....* C  Cascade
  Constraint name ..
Usage ........*    (none)

Abstract     Zoom: N


EDIT:   Owner: N   Desc: N
```

Predict Modify File Relation Panel

The fact that the constraint type is C (Cascade) identifies NCST-ORDER-HAS-LINES as an intra-object relationship.

Create the Object Maintenance Subprogram and PDAs

To implement objects, Natural Construct passes a parameter data area (PDA) containing the object data between a subprogram that implements the object and any application process that uses the object:



Object Implementation

Natural Construct also passes a restricted PDA between a process and an object. This PDA contains information needed by the object maintenance subprogram from call to call. In an object that functions on Adabas files, for example, the ISNs (Internal Sequence Numbers) of records are kept to allow a faster form of updating. Any process that uses an object must pass the corresponding restricted PDA along with the data structure PDA. The process must not modify this PDA.

Because the object maintenance subprogram and PDAs work together, Natural Construct automatically generates the parameter data areas when you generate the subprogram.

The following example shows a generated object PDA structure.

Note: If required, you can regenerate the object PDA using the Object-Maint-PDA model and the restricted PDA using the Object-Maint-PDA-R model.

Example of a generated order PDA structure

```
I T L Name                            F Leng  Index/Init/EM/Name/Comment
- - - ---------------------------------- - ---- ------------------------------
    1 ORDER                                  /* Object Name
    2 ORDER-NUMBER                    N    6 /*
    2 ORDER-AMOUNT                    P 13.2 /*
    2 ORDER-DATE                      N    8 /*
    2 ORDER-CUSTOMER-NUMBER           N    5 /*
    2 ORDER-WAREHOUSE-ID              A    3 /*
    2 INVOICE-NUMBER                  N    6 /*
    2 ORDER-TIMESTAMP                 T      /*
    2 C#DELIVERY-INSTRUCTIONS         N    3 /* Counter Field
    2 DELIVERY-INSTRUCTIONS           A   60 (1:20)
  *
    2 C#NCST-ORDER-HAS-LINES          N    3 /* Counter field
    2 NCST-ORDER-HAS-LINES                   (1:30) /* NCST-ORDER-LINES
    3 LINE-NUMBER                     N    2 /*
    3 ORDER-PRODUCT-ID                A    6 /*
    3 LINE-DESCRIPTION                A   40 /*
    3 QUANTITY                        P    9 /*
    3 UNIT-COST                       P  7.2 /*
    3 TOTAL-COST                      P  9.2 /*
  *
    3 C#NCST-LINE-HAS-DISTRIBUTION    N    3 /* Counter field
    3 NCST-LINE-HAS-DISTRIBUTION             (1:10) /* NCST-ORDER-DISTRIBUTION
    4 DIST-LINE-NUMBER                N    2 /*
    4 DIST-NUMBER                     N    2 /*
    4 ACCOUNT                         A    9 /*
  R 4 ACCOUNT                                /* REDEF. BEGIN : ACCOUNT
    5 COST-CENTER                     A    2 /*
    5 ACCT                            A    4 /*
    5 PROJECT                         A    3 /*
    4 DIST-AMOUNT                     P  9.2 /*
  *
    1 ORDERPDA-ID                     N    6 /* Object identifier
  R 1 ORDERPDA-ID                            /* REDEF. BEGIN : ORDERPDA-ID
    2 STRUCTURE                              /* To allow MOVE BY NAME
    3 ORDER-NUMBER                    N    6 /*
```

Fields beginning with C# represent the number of occurrences of an element. C# variables are created for multiple-valued fields within a primary file or for multiple lines in a secondary file. (The same is true for secondary and tertiary files.)

➢ To generate the object maintenance subprogram:

1 Type “M” in the Function field on the Generation main menu.

2 Type “ORDSUBP” in the Module name field.

3 Type “Object-Maint-Subp” in the Model name field.

4 Press Enter.
The Standard Parameters panel is displayed.

5 Fill in the Standard Parameters panel as follows:

```
CUOBMA                  Object-Maint-Subp Subprogram                 CUOBMA0
Oct 28                      Standard Parameters                       1 of 2

 Module ..... ORDSUBP
 System ..... ORDER  _____________________________

 Title ..... Order Maintenance Subp.
 Description ..... This subprogram maintains the order object. All
                   updates to entities that make up an order must be
                   changed using this subprogram. _____________________
                   ____________________________________________________

 Message numbers .... _




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                                        right main
```

Standard Parameters Panel for the Object-Maint-Subp Model

6 Fill in the Additional Parameters panel as follows:

```
CUOBMB                 OBJECT-MAINT-SUBP Subprogram                 CUOBMB0
Oct 28                    Additional Parameters                      2 of 2

   Predict view ..... NCST-ORDER-HEADER_______________ *
   Primary key ..... ORDER-NUMBER_____________________ *
   Hold field ..... ORDER-TIMESTAMP___________________ *

   Object description ..... ____________________

                                      Generate   Source     Object
   Object PDA ..... ORDSMSA_ *          X
   Restricted PDA ..... ORDSMSR_ *      X
   Object name ..... ORDSMSA_____________________________

   Next action prefix ..... _
   Log file suffix ..... ________
   Trace relationships ..... _



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit                                  left  userX main
```

Additional Parameters Panel for the Object-Maint-Subp Model

You must specify a value in the Hold field, which logically holds the object subprogram records. Natural Construct captures the value of this field when an object is first displayed. Before updating the object, Natural Construct compares the captured value of the Hold field to the field value currently stored in the file. If these values differ (when someone has updated the object since it was first displayed, for example), Natural Construct displays an error message. If the update is allowed, Natural Construct also updates the Hold field to the new value (usually the current time).

Notice that the Generate field for the Object PDA and Restricted PDA is marked. This indicates that they do not currently exist. Natural Construct will generate the PDAs using the names shown.

7 Generate and stow the object maintenance subprogram.

The resulting object subprogram performs the following operations:

- Provides an implementation for the Add, Update, Delete, Get, Get Next Object, Existence Check, and Initialize Object methods. This includes an implementation of the mapping between the object definition and underlying files.
- Allows you to define additional methods via user exits for the Object-Maint-Subp model.
- Guarantees cascading delete constraints for all intra-object relationships.
- Guarantees normal referential integrity for all inter-object relationships.
- Requires you to specify cardinality for the relationships (an order must have between 1 and 30 lines, for example).
- Permits you to add additional semantics, such as which users have authority to perform which methods, via user exits and Predict verification rules.

  In addition to a data structure PDA and restricted PDA, Natural Construct also uses a common object PDA (CDAOBJ) and standard message-passing PDA (CDPDA-M) when invoking objects. These PDAs are constant for all objects.

Example of adding an order

```
DEFINE DATA LOCAL USING ORDERPDA    /* PDA generated
LOCAL USING ORDER PDR               /* Restricted PDA generated
LOCAL USING CDAOBJ
LOCAL USING CDPDA-M                 /* standard message PDA
END-DEFINE

/* Assign field into ORDERPDA
INPUT ORDER

/* Set derived message
ASSIGN CDAOBJ.#FUNCTION = 'ADD'

/* Send Message
CALLNAT 'ORDSUBP' ORDER ORDERPDA-ID ORDERPDR CDAOBJ MSG-INFO

/* Check response code
If MSG-INFO.##ERROR-FIELD NE ' ' THEN
.
.
.
END-IF
```

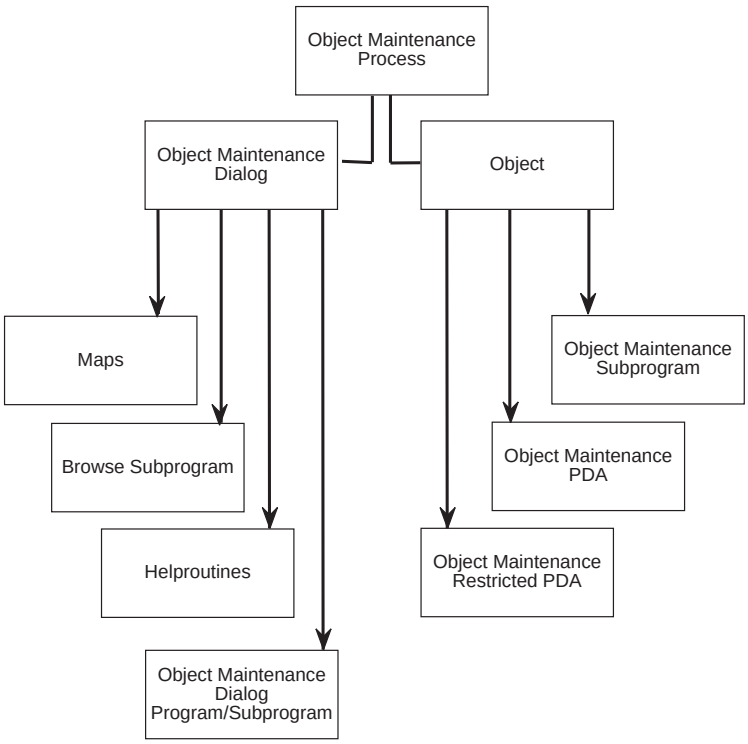
For more information about calling objects, see **Using Parameter Data Areas**, page 117.

Natural Construct and Process Design

Natural Construct does not inherently impose a design methodology for processes. It provides the flexibility for you to combine generated processes in any way you choose. Nonetheless, it is important to recognize a design methodology; if not the one implied by the supplied models, then one implied by your own models.

Object Maintenance Process

The following diagram illustrates a common object maintenance process:



Natural Construct Object Maintenance Process

Create the Object Components

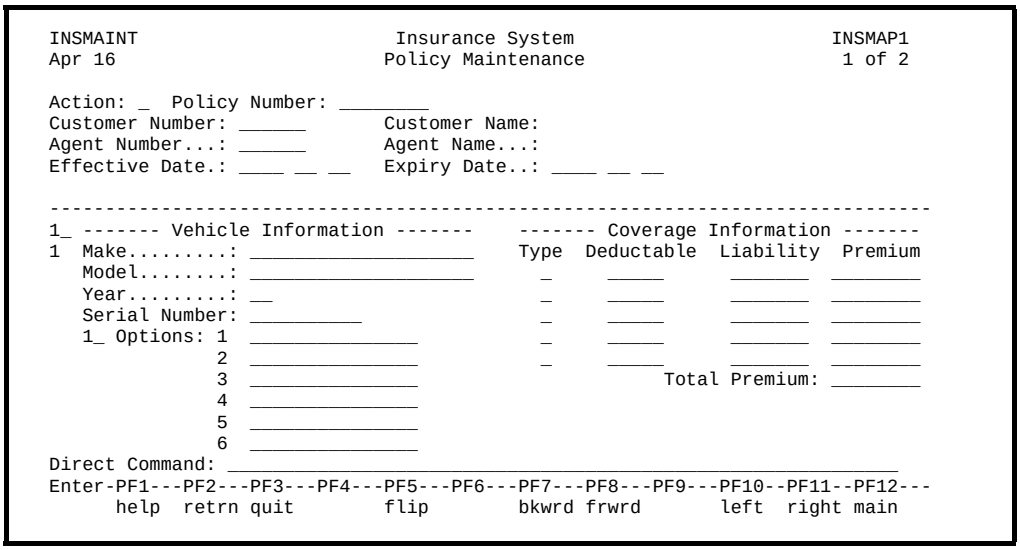
To create an object maintenance process, you first create the object components: the object maintenance subprogram, the object PDA, and the restricted PDA. Use the Object-Maint-Subp model to generate all these components. For information, see **Create the Object Maintenance Subprogram and PDAs, page 150**.

Create the Maps

To create the maps (panels) necessary to maintain the object, you must use the Map model (mainframe) or the Natural Map editor. A maintenance program can use multiple maps to maintain an object, so you must decide how many maps to use. Once generated, you can change the map's appearance (using the Map editor) to suit your needs.

Using the Map model to create your maps is particularly useful when the object contains arrays. The Map model generates all the correct index names and fields required to link the map to its calling program.

The following example shows a map to maintain policy objects:



Insurance System Policy Maintenance Map

Create the Object Maintenance Dialog Process

To create the object maintenance dialog process, you can use the Object-Maint-Dialog or Object-Maint-Dialog-Subp model. You can create this process as either a program or subprogram.

➢ To create the object maintenance dialog process:

1 Specify the coordinates for the scroll regions on each panel. Within each scroll region, users can press PF7 (bkwrd) and PF8 (frwrd) to scroll forward and backward through data. Natural Construct retrieves the scroll region coordinates from the map definition.

2 Activate PF10 (left) and PF11 (right) so users can scroll left and right through multiple maps defined for the process.

Create the Browse Process

To include the Browse action as one of the maintenance actions, you can create a browse subprogram using the Browse-Subp model. The name of this subprogram is one of the specification parameters for the object maintenance dialog program (Object-Maint-Dialog model) or subprogram (Object-Maint-Dialog-Subp model).

As a general rule, you can specify a framed window smaller than the physical screen for your browse subprogram. By doing this, the browse window appears to pop-up on the maintenance panel:

```
INSBROWS      Insurance System
Apr 16         Browse Policies         1 of 1

Policy  Customer Agent  Effective    Expiry
Number   Number  Number    Date       Date
-------- -------- ------ ---------- ---------
123141     22222   11111 1995-01-01 1996-01-0
1234       22222   11111 1995-01-01 1996-01-0
1241452    11111   22222 1995-01-01 1996-01-0
134131     22222   11111 1995-01-01 1996-01-0
142131     22222   11111 1995-01-01 1996-01-0
2          22222   22222 1995-01-01 1996-01-0
22         22222   22222 1995-01-01 1996-01-0
Policy Number: ________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7
      help  retrn quit        flip  place bkw
Position cursor or enter screen value to sele
```

Example of a Browse Window

To activate cursor sensitivity, you can define the PROCESS-SELECTED-RECORD user exit. Within this user exit, you can return the key of the selected record and indicate what action is performed on the returned key. You can return the record for display in the object maintenance process, or invoke another browse subprogram to further refine the details. For more information, see **Natural Construct Zoom-Down Query Process, page 161**.

Create the Helproutines

For any field that represents a foreign key (a field that determines its values from a table), you can generate a helproutine to provide a cursor-sensitive mechanism to select values. These active helproutines help users specify foreign key values. Each field that requires active help must have an HE='*helproutine name*' parameter.

To generate helproutines for each foreign field on your maintenance maps, use the Browse-Helpr model. The generated module name is placed on the appropriate map as a helproutine. To return cursor-selected values to the field, include the PROCESS-SELECTED-RECORD user exit in the generated helproutines.

The following example shows a helproutine window:

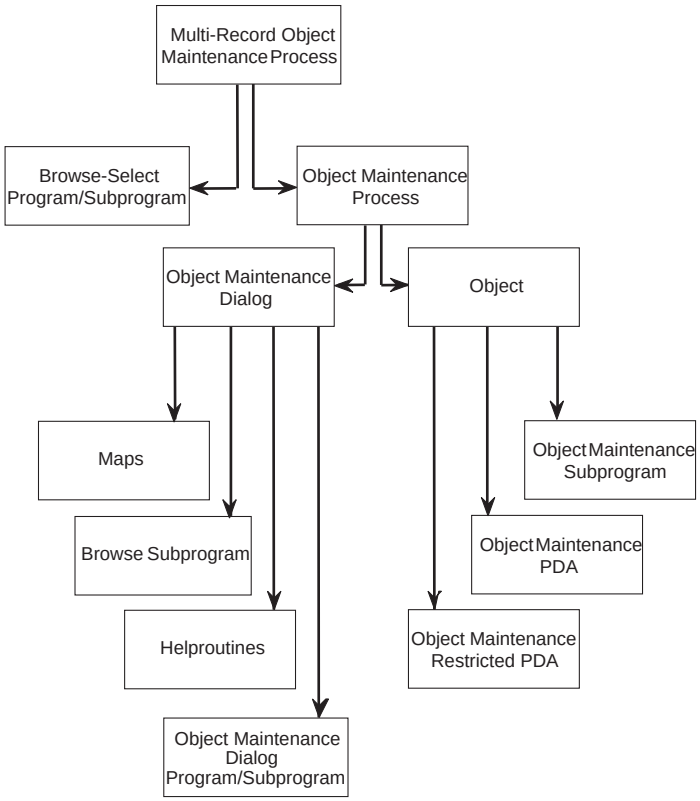
```
INSCUSH     Insurance System
Jan 27      Browse Customers      1 of 1

Customer
 Number          Business Name
-------- ------------------------------

10000    OAKES WELDING
10001    JOURNEYMEN FABRICATING
10002    LES RIVERS CUSTOM FABRICATING
10003    MONJEN STELL INC.
Customer Number: _____
Enter-PF1---PF2---PF3---PF4---PF5---PF6-
      help  retrn             flip  plac
Position cursor or enter screen value to
```

Example of a Helproutine Window

Multi-Record Object Maintenance Process

Within a multi-record object maintenance process, users can scroll through a series of records (typically the primary entity of an object) and specify actions to be performed against the objects. To specify an action, the user enters an action indicator (D for display, P for purge, etc.) next to the record on the panel.

The following diagram illustrates a multi-record object maintenance process:



Multi-Record Object Maintenance Process

To create a multi-record object maintenance process, first assemble the object maintenance process (for information, see **Object Maintenance Process, page 155**). Next, generate the selection panel using the Browse-Select model. This panel acts as the front-end component for the multi-record object maintenance process. It is the entry and exit point for the process, and the only program module.

The following example shows a selection panel:

```
INSSEL                          Insurance System
Nov 27                          Policy Selection                      1 of 1
Action Policy Number Customer Number Agent Number Effective Date Expiry Date
------ ------------- --------------- ------------ -------------- -----------
  _      123141           22222           11111      1995-01-01   1996-01-01
  _      1234             22222           11111      1995-01-01   1996-01-01
  _      1241452          11111           22222      1995-01-01   1996-01-01
  _      134131           22222           11111      1995-01-01   1996-01-01
  _      142131           22222           11111      1995-01-01   1996-01-01
  _      2                22222           22222      1995-01-01   1996-01-01
  _      22               22222           22222      1995-01-01   1996-01-01
  _      222              22222           33333      1995-01-01   1996-01-01
  _      332              22222           11111      1995-01-01   1996-01-01
  _      3333333          13451           11111      1995-01-01   1996-01-01
  _      44444444         22222           11111      1995-01-01   1996-01-01
  _      45               22222           11111      1995-01-01   1996-01-01
  _      50               22222           11111      1995-01-01   1996-01-01
Policy Number: ________
Direct Command: _____________________________________________________________
Add        Copy       Display    Modify     Purge      (PF5=flip)
```

Example of a Selection Panel

To generate the object maintenance dialog subprogram, use the Object-Maint-Dialog-Subp model. For more information, see **Create the Object Maintenance Dialog Process, page 156**. The object maintenance dialog subprogram implements the methods for actions performed and performs the specified action for each record selected.

In contrast to a standard maintenance process, the object maintenance dialog subprogram does not have a modifiable action. This subprogram can only act on the record and action passed to it. When multiple records are selected (possibly with different actions), you invoke the subprogram once for each selected record.

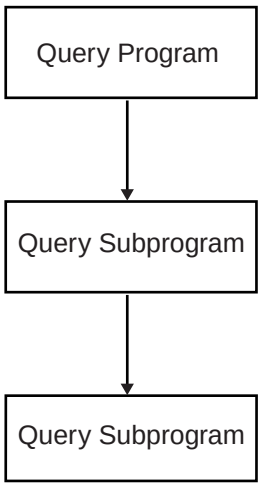
The following example shows an object maintenance dialog subprogram window that overlays the Policy Selection panel:

```
INSMTSUB                    Insurance System                    INSSLMP1
Apr 16                           Update                           1 of 2

Policy Number.: 123141__
Customer Number: 22222_       Customer Name: HEIDELMAN AND COMPANY
Agent Number...: 11111_       Agent Name...: BULK FOOD INTERNATIONAL
Effective Date.: 1995 1_ 1_   Expiry Date..: 1996 1_ 1_
------------------------------------------------------------------------------
1_ ------- Vehicle Information -------  ------- Coverage Information -----
1  Make.........: FORD_________________  Type  Deductable  Liability  Premi
   Model........: LTD_________________    _     _____      _______  ______
   Year.........: 95                      _     _____      _______  ______
   Serial Number: 1121212121              _     _____      _______  ______
   1_ Options: 1  P.W._____________       _     _____      _______  ______
               2  P.D._____________       _     _____      _______  ______
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF
      help  retrn quit        flip        bkwrd frwrd       left  right ma
Enter changes
```

Example of an Object Maintenance Dialog Subprogram Window

Natural Construct Zoom-Down Query Process

A Natural Construct query process probes the complex views of a number of interrelated entities. This query process has the following structure:



Query Process Structure

The following sections describe how to create zoom-down query process.

Create the Main Query Process

➢ To create the main query process:

1 Create the main program (using the Browse model) to browse the primary entity of the complex view.

2 Include the PROCESS-SELECTED-RECORD user exit. Within this user exit, call a subsequent query subprogram and pass it the key of the selected record.

The following example shows a query panel:

```
INSBROW                        Insurance System
 Apr 16                          Query Policies                          1 of 1
 Policy Number Customer Number Agent Number Effective Date Expiry Date
 ------------- --------------- ------------ -------------- -----------
   123141           32822           10191      1995-01-01   1996-01-01
   1234             76301           11980      1995-01-01   1996-01-01
   1241452          11291           22734      1995-01-01   1996-01-01
   134131           28072           91181      1995-01-01   1996-01-01
   142131           26913           52011      1995-01-01   1996-01-01
   2                26598           89932      1995-01-01   1996-01-01
   22               22549           25979      1995-01-01   1996-01-01
   227              59584           39323      1995-01-01   1996-01-01
   332              80297           81191      1995-01-01   1996-01-01
   3062351          13451           77611      1995-01-01   1996-01-01
   13764261         52702           10098      1995-01-01   1996-01-01
   45               21806           01615      1995-01-01   1996-01-01
   50               46282           19290      1995-01-01   1996-01-01
 Policy Number: ________ Show Customer Info _
 Direct Command: _____________________________________________________________
 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
       help  retrn quit        flip        bkwrd frwrd                   main
 Position cursor or enter screen value to select
```

Insurance System Query Panel

Create the Query Subprograms

Next, you create the query subprograms. To invoke the first query subprogram, a user moves the cursor to any record on the Query Policies panel and presses Enter:

```
INSBROW2                Insurance System
Apr 16                   Query Vehicles                    1 of 2

Vehicle                                                    Serial
Number          Make                Model          Year    Number
------- -------------------- -------------------- ---- ----------
   1    FORD                 LTD                   95  1121212121
   2    G.M.                 CORVETTE              65  A010120021
                     ****** End of Data ******




Vehicle Policy Number: 123141   Vehicle Number: __
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10-
      help  retrn quit        flip  place bkwrd frwrd       left
Position cursor or enter screen value to select
```

Example of a Query Subprogram Window

To create the query process, you can use the Browse-Subp model to generate a subprogram that browses a series of entities related to the primary entity. In this example, the subprogram lists the vehicles for the selected policy.

If the format and length of the passed #PDA-KEY parameter are not the same as the key being browsed, you must specify the unit and decimal positions on the Specific Parameters panel for the Browse-Subp model:

```
CUSCMF                    Browse-Subp Subprogram                    CUSCMF0
 Oct 28                     Specific Parameters                     4 of 5

     Parameter Override
     Note: Format and Length defaults to RPT_CTRL_KEY key
     Field Name ......... #PDA-KEY
     Natural format ..... A _____ *









 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
 right help  retrn quit                                    left  right main
```

Specific Parameters Panel for the Browse-Subp Model

For the above example, we specified “A” in the Format field and “8” in the length field. This indicates that the passed key format is A8.

To restrict the query to entities that match the primary entity (vehicles within the selected policy), you can assign the variable minimum/maximum key values appropriately on the Restriction Parameters panel for the Browse-Subp model:

```
CUSCMG                    Browse-Subp Subprogram                    CUSCMG0
Oct 28                    Restriction Parameters                     5 of 5

  Prefix Options
  Restrict browse with prefix .. X

  Number of characters (bytes) . 8__
  Number of components ......... __

  Protect prefix ............... X
  Suppress prefix .............. _

  Field Name ..... ____________________________________________


  Prefix Helproutine Specifications
     Component       Helproutine    Parameter Name
        1             ________ *    ____________________________
        2             ________ *    ____________________________


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit                                    left  userX main
```

Restriction Parameters Panel for the Browse-Subp Model

In the above example, the query on VEHICLES (which has a primary key of POLICY-NUMBER and VEHICLE-NUMBER) is restricted to those lines that begin with the policy number matching the record selected on the primary scroll.

You can create as many query subprograms as the data model allows, each time zooming down for greater detail.

You can also develop several parallel zoom-down paths. The path a user takes depends on what PF-keys the user presses or what selections the user makes. As with any query subprogram, you can invoke additional query subprograms from within the PROCESS-SELECTED-RECORD user exit.

GENERAL MODEL SPECIFICATIONS

This chapter describes general specifications for creating modules using the supplied models. It describes the Standard Parameters panel, which is similar for most program models, as well as how to set up a password file.

The following topics are covered:

Introduction

The Standard Parameters panel is similar for all program models; it is the first and sometimes the only specification panel. For information about the additional specification panels for a specific model, see the chapter in which that model is described.

Standard Parameters Panel

The following example shows the Standard Parameters panel for the Browse model:

```
CUSCMA                          Browse Program                        CU--MA0
Nov 28                       Standard Parameters                       1 of 4

 Module ............. MYBROWSE
 System ............. DEMO_____________________________
 Global data area ... CDGDA___
 With block ......... _____________________________

 Title ............... Browse ..._________________
 Description ........ This program is used to browse the ..._____________________
                      ____________________________________________________________
                      ____________________________________________________________
                      ____________________________________________________________

 First header ....... _____________________________________________________
 Second header ...... _____________________________________________________

 Command ............ _
 Message numbers .... _
 Password ........... _

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                                            right main
```

Standard Parameters Panel for the Browse Model

Note: The Standard Parameters panel for other models may not contain all the fields shown.

The fields on the Standard Parameters panel are:

Field	Description
Module	Name of the module (by default, the name specified in the Module name field on the Generation main menu). This is a required field. The module name must be alphanumeric, a maximum of eight characters in length, and cannot contain blanks.

Field	Description (continued)
System	Name of the system (by default, the name of the current library). This is a required field. The system name must be alphanumeric, not exceed 32 characters in length, and does not have to be associated with a Natural library ID. (The combination of the module name and system name is used as a key to access help information for the generated module.)
Global data area	Name of the global data area (GDA) used by the generated module. By default, "CDGDA" is displayed. This is a required field. To allow inter-program communication, generated modules require a small number of global variables. The CDGDA global data area contains the global variables required to test a generated module. Before creating a new application, copy this GDA from the SYSTEM library and rename it to match your naming conventions. Then add any additional global variables your application may require.
With block	Name of the GDA block used by the generated module. This is an optional field. You need only specify the lowest level block name; the corresponding path name is determined automatically.
Note:	For more information about GDA blocks, see the Natural documentation.
Title	Title for the generated module. The title is used internally to identify modules for the List Generated Modules function on the Generation main menu. This is a required field.
Description	Description of what the generated module does. This is an optional field, used internally for documentation purposes.
First header	First header for the generated module. This header is centered at the top of the generated panel and intensified. This is a required field.
Second header	Second header for the generated module. This header is centered under the first header and intensified. This is an optional field.
Command	If this field is marked, the generated module supports direct command processing. If this field is blank, the generated module does not support direct commands.

Field	Description (continued)
Message numbers	If this field is marked, the generated module uses message numbers rather than message text for all REINPUT and INPUT messages.
Note:	Use the same technique consistently throughout your application, since passing messages between modules using different techniques will not always produce the desired results.
Password	If this field is marked, the generated module performs password checking. If this field is blank, the generated module does not perform password checking. To include password checking, set up a password file. For information, see **Setting Up a Password File**, page 171.

Setting Up a Password File

You can specify password checking for many of the generated modules. Natural Construct builds the mechanism for password checking into your module by including the CCPASSW copycode member. Within this copycode, the CDPASSW subprogram is invoked and passed the module and system names.

To include password checking, you must set up a password file. The file is keyed on the module name used to catalog the module and the system name used to generate the module.

The password file can be a view of any file with Natural-Construct-Password as the data definition module name. The view must contain the following fields:

Field	Format
PASSWORD-KEY	A40 (32-character system name, plus an 8-character module name)
PASSWORD	A8 (8-character password)

When a user attempts to invoke the module, the CDPASSW subprogram reads the password file. If the module/system name combination exists in the file and does not have a password, the user can invoke the module. If the module/system name combination exists and has a password, the user must enter the correct password before the module is invoked. If a user enters five incorrect passwords, execution is aborted.

If you specify password checking, modify the CDPASSW subprogram to include a valid password view and any final processing you want to perform, and then catalog the modified subprogram. For more specific password checking, you can modify the CCPASSW copycode member (to call a different subprogram) or modify the CDPASSW subprogram (to refine your security standards).

BATCH MODEL

This chapter describes the Batch model. The Batch model generates programs that normally produce large volumes of output, which the programs route to a printer or terminal.

The following topics are covered:

Introduction

A batch program uses a primary key value to read the contents of a primary file in logical order. The primary key value can be a descriptor, superdescriptor, or subdescriptor, and does not have to be unique.

These programs differ from online programs as they usually read or update large volumes of data and produce multiple reports. For multiple files, you can use the Batch model (which supports as many as two secondary files and two tertiary files).

After selecting the files to use in your batch program, you can select additional related files by indicating which Predict relationships to process. File views returned from Predict can be used to create a report layout. Data read from the program can be written to four different reports, each of which may have a different format.

To view the specifications for a sample batch program, refer to the NCOREP program in the Natural Construct demo system.

Parameters for the Batch Model

The Batch model has five specification panels. Each panel and its parameters are described in the following sections.

Standard Parameters Panel

The following example shows the first specification panel for the Batch model:

```
CUBAMA                          Batch Program                          CUBAMA0
Sep 05                       Standard Parameters                        1 of 5

 Module ..............  BATCHJOB
 System ..............  DEMO_____________________________
 Global data area ...  ________
 With block .........  _____________________________

 Title ...............  Batch ..._________________
 Description ........  This program ..._____________________________________
                       ____________________________________________________
                       ____________________________________________________
                       ____________________________________________________




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                                          right main
```

Standard Parameters Panel for the Batch Model

This panel is similar for all models. For a description of the Standard Parameters panel, see **General Model Specifications**, page 167.

Report Heading Parameters Panel

The following example shows the second specification panel for the Batch model:

```
CUBAMB                          Batch Program                          CUBAMB0
Sep 05                   Report Heading Parameters                      2 of 5

                    Report Headings                     Printer    On Line
   ------------------------------------------------- ---------- ----------
 1 _________________________________________________     _          _
 2 _________________________________________________     _          _
 3 _________________________________________________     _          _
 4 _________________________________________________     _          _
 5 _________________________________________________     _          _
 6 _________________________________________________     _          _
 7 _________________________________________________     _          _
 8 _________________________________________________     _          _
   Note: To display heading on all reports, use Printer = *

            Printer Name             LS   PS   ZP   IS   ES   Form or Map
   ------------------------------- ---- ---- ---- ---- ---- --------------
 0 Terminal screen                 80_  23_   X    _    _     _________ *
 1 _______________________________ 133  60_   X    _    _     _________ *
 2 _______________________________ 133  60_   X    _    _     _________ *
 3 _______________________________ 133  60_   X    _    _     _________ *
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit  test                                  left  right main
```

Report Heading Parameters Panel for the Batch Model

This panel is divided into two sections. Use the upper section to enter report headings and indicate where they will appear on the report. Use the lower section to set up general formats for each report.

The fields on this panel are:

Field	Description
Report Headings	Headings for up to eight reports.
Printer	Printer file number for the corresponding report (see the Printer Name field description). If you type an asterisk (*) in the Printer field for a heading, the heading is displayed on all reports.
On Line	Line on which the heading is displayed. Type “1” for a first heading, “2” for a second heading, etc.

Field	Description (continued)
Printer Name	Where the output of the report is directed. The printer file numbers specify the following destinations: • (0) Output to terminal screen • (1) Output to first report • (2) Output to second report • (3) Output to third report Type the report name in the appropriate field. The name is used to set up a Natural DEFINE PRINTER statement. If this field is left blank, the name defaults to the printer file number (0, 1, 2, or 3).
LS	Line size. The number of characters that fit on one line on the report (default values are displayed).
PS	Page size. The number of lines per page (default values are displayed).
ZP	Zero printing. If this field is marked (by default), fields containing zeroes are printed on the report. If this field is not marked, zeroes are suppressed.
IS	Identical suppress. If this field is marked, identical entries on consecutive lines will not be printed on the report.
ES	Empty line suppression. If this field is marked, empty lines will not be printed on the report.
Form or Map	Name of the form or map layout used to build the report headings.

Testing the Appearance of Headings on the Report

A “test” key (PF4) is available on this panel, in addition to the standard PF-keys. After entering the required parameters, press PF4 (test) to preview how the headings will appear on your report.

Program Structure Panel

The following example shows the third specification panel, the Batch Program Structure panel:

```
CUBAMC                          Batch Program                          CUBAMC0
Sep 05                       Program Structure                          3 of 5


  Additional inputs .. _  INPUT USING MAP ________ *
  Multiple inputs .... _  REPEAT
  Input range ........ _    INPUT USING MAP ________ *
                           IF termination string . ___ THEN
                             STOP
                           END-IF
                           READ
                             .
                             .
                             .
  Perform ET ......... _     IF hold count ......... ___ THEN
                               END TRANSACTION
                             END IF
                           END-READ
                         END-REPEAT
                         END

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                       left  right main
```

Program Structure Panel for the Batch Model

Use this panel to specify the basic structure of the batch program. Four different program blocks are available:

- INPUT statement for initial parameters
- REPEAT loop
- INPUT statement that prompts for a start and end range of primary key values within the REPEAT loop
- END TRANSACTION statement for updates

The fields on this panel are:

Field	Description
Additional inputs	If this field is marked, parameters are input at the beginning of the program (before any files are read). Specify the parameters on the Additional Parameters panel.
INPUT USING MAP	Name of the external map the program uses to input the parameters.

Field	Description (continued)
Multiple inputs	If this field is marked, a REPEAT loop reads the primary file for multiple ranges of records. By default, you can enter a single range of primary key values at the beginning of the program (after any additional input parameters). If you mark this field, primary key ranges will be continually input until the specified termination string is reached in the input stream.
Input range	If this field is marked, the generated batch program reads only a portion of the file each time the REPEAT loop executes.
INPUT USING MAP	Name of the external map the program uses to input the parameters. To specify a map name, use the #INPUT1.*primary key* variable to indicate the beginning of the range and the #INPUT2.*primary key* variable to indicate the end of the range (#INPUT1.ORDER-NUMBER, for example). **Note:** The primary key is the key of the primary file used in the batch program.
IF termination string	Termination string indicating when the program terminates. If you specified a map name in the INPUT USING MAP field (see above), you can use the #TERMINATED-STRING variable (modifiable, format A3) to input the termination string value.
Perform ET	Mark this field to build a batch update program. An ET (END TRANSACTION) statement will be performed periodically to release records and update the database.
IF hold count	If you marked the Perform ET field, specify the number of primary records to process before issuing an ET statement.

Primary File Parameters Panel

The following example shows the fourth specification panel, the Primary File Parameters panel:

```
CUBAMD                          Batch Program                          CUBAMD10
Sep 05                    Primary File Parameters                        4 of 5

  Primary view ............. _____________________________________ *
    Natural (DDM) .......... _____________________________________
  Primary key .............. _____________________________________ *


  Predict Relationships
  Select all ............... _
  Select specific ..........
                            1 _____________________________________ *
                            2 _____________________________________ *
                            3 _____________________________________ *
                            4 _____________________________________ *




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        sec1                          left  right main
```

Primary File Parameters Panel for the Batch Model

Use this panel to specify the primary file and indicate which Predict relationships the batch program will use. Press PF5 (sec1) to define secondary and tertiary files. For more information, see **Accessing Secondary and Tertiary Files**, page 183.

The fields on this panel are:

Field	Description
Primary view	View name driving the batch program. The specified view must be defined in Predict. The primary file type can be Adabas, DB2, VSAM, or sequential.
Natural (DDM)	Name of the data definition module (DDM) for the primary file. All fields in the primary file must be in the specified DDM. This field defaults to the name of the primary file, so you only need to specify a DDM if it is different from the primary file name.

Field	Description (continued)
Primary key	Key used to read the file in logical (rather than physical) sequence and to control scrolling. The key must be defined as a descriptor, superdescriptor, or subdescriptor in the Predict file definition. (If the specified key does not exist in the corresponding Predict file, a message indicates that the Key field was not found in the file.)
Note:	The primary key name is not applicable when the primary file is a sequential work file.
Predict Relationships	
Select all	If this field is marked, the generated program performs file lookups (joins) on all files related to the primary file in Predict, mark this field. In each relationship, the cardinality of the primary file must be N or CN, while the cardinality of the related file must be 1 or C. The update constraint type must be R (restricted update), and the delete constraint type can be blank or R. Only type N (Natural Construct) relationships are processed. In the generated batch program, the relationship name is used as the view name. Specify the related file name when selecting fields in the WRITE-FIELDS user exit. To avoid having to make manual changes after generating your user exit, change the related file name to the relationship name before generation. The generated fields will then have the correct prefix.
Note:	For DB2 users, type R (referential constraint) relationships are also processed.
Select specific	Names of up to four Predict relationships for the specified primary file.

Updating Secondary/Tertiary Keys Defined as Unique

Natural Construct improves the handing of the secondary and tertiary key values for an SQL file, allowing users to rearrange and update these values without difficulty. For example, assume you have secondary key values “Adams” and “Baker” and you want to exchange their values (where Adams becomes Baker and Baker becomes Adams). Previous versions of Natural Construct first deleted the original record (Adams) and then tried to store the new record (Baker) — however Baker already existed as the original record for the second key, resulting in a conflict between the two keys.

Natural Construct does not automatically delete the original record. Instead, it reviews all existing keys for a possible match with the new key. For example, assume that Adams is the existing key for a record you want to change to Baker. Natural Construct reviews all existing key values for a possible match with Baker. If a match is found, Natural Construct updates the existing Baker record. If a match is not found, Natural Construct creates a new Baker record. This process is repeated for all secondary and tertiary key values. All original key values that were renamed are deleted.

For example, if secondary key values are modified as follows:

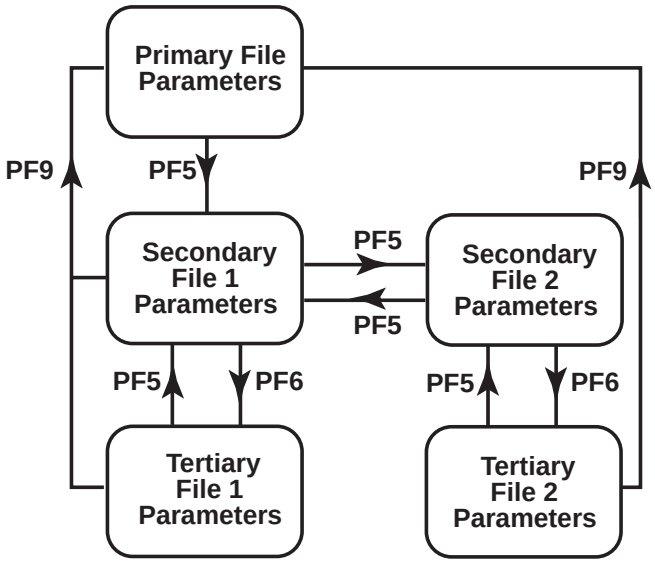
Original	New
Adams	Baker
Baker	Adams
Clark	Smith

Natural Construct performs the following database functions:

- Updates the original Baker with new Baker
- Updates the original Adams with new Adams
- Stores the new Smith
- Deletes the original Clark

Accessing Secondary and Tertiary Files

On the Primary File Parameters panel, you can press PF5 (sec1) to display the Define Secondary File 1 panel. Using this panel, you can define the secondary file and use PF-keys to access a second secondary file, a first tertiary file, or the primary file. The following diagram illustrates how to navigate between the panels:



Accessing Secondary and Tertiary Files

Accessing the Secondary File 1

You can specify a secondary file, which will be logically coupled (joined) to the primary file in your batch program. The secondary file must be normally related to the primary file with cardinality 1:N.

To define a secondary file, press PF5 (sec1) on the Primary File Parameters panel. The Secondary File 1 Parameters panel is displayed:

```
CUBAMD                          Batch Program                          CUBAMD20
Sep 05                  Secondary File 1 Parameters                      4 of 5

  Secondary view ........... _____________________________ *
    Natural (DDM) .......... _____________________________
  Secondary key ............ _____________________________ *
  Related by field ......... WARE_SUPER_____________________ *

  Predict Relationships
  Select all ............... _
  Select specific .........
                           1 _____________________________ *
                           2 _____________________________ *
                           3 _____________________________ *
                           4 _____________________________ *




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        sec2  ter1              prime left  right main
```

Secondary File 1 Parameters Panel for the Batch Model

The following table describes the fields on this panel:

Field	Description
Secondary view	Name of the secondary view, which must be defined in Predict. The secondary file type can be Adabas, DB2, VSAM, or sequential.
Natural (DDM)	Name of the data definition module (DDM) for the secondary file. All fields in the secondary file must be in this DDM. This field defaults to the name of the secondary file, so you only need to enter a DDM if it is different from the secondary file name.
Secondary key	Name of the scrolling key used to read the secondary file in logical rather than physical sequence. This key must be defined as a descriptor, superdescriptor, or subdescriptor in the Predict file definition.

Field	Description (continued)
Note:	The secondary key is not applicable when the secondary file is a sequential work file.
Related by field	Name of the primary key used to couple the primary and secondary files.
Predict Relationships	
Select all	If this field is marked, the generated program performs file lookups (joins) on all files related to the secondary file in Predict. In each relationship, the cardinality of the secondary file must be N or CN. The cardinality of the related file must be 1 or C. The update constraint type must be R (restricted update). The delete constraint type can be blank or R. Only type N (Natural Construct) relationships are processed. Within the generated batch program, the relationship name is used as the view name. Specify the related file name when selecting fields in the WRITE-FIELDS user exits. To avoid having to make manual changes after generating the user exit, change the related file name to the corresponding relationship name before generation. The generated fields will then have the correct prefix value.
Note:	For DB2 users, type R (referential constraint) relationships are also processed.
Select specific	Names of up to four Predict relationships for the specified secondary file.

Accessing the Secondary File 2

To define another secondary file, press PF5 (sec2) on the Secondary File 1 Parameters panel. The Secondary File 2 Parameters panel is displayed:

```
CUBAMD                          Batch Program                          CUBAMD20
Sep 05                  Secondary File 2 Parameters                      4 of 5

  Secondary view ........... _____________________________ *
    Natural (DDM) .......... _____________________________
  Secondary key ............ _____________________________ *
  Related by field ......... WARE_SUPER_____________________ *

  Predict Relationships
  Select all ............... _
  Select specific ..........
                          1 _____________________________ *
                          2 _____________________________ *
                          3 _____________________________ *
                          4 _____________________________ *




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        sec1  ter2             prime left  right main
```

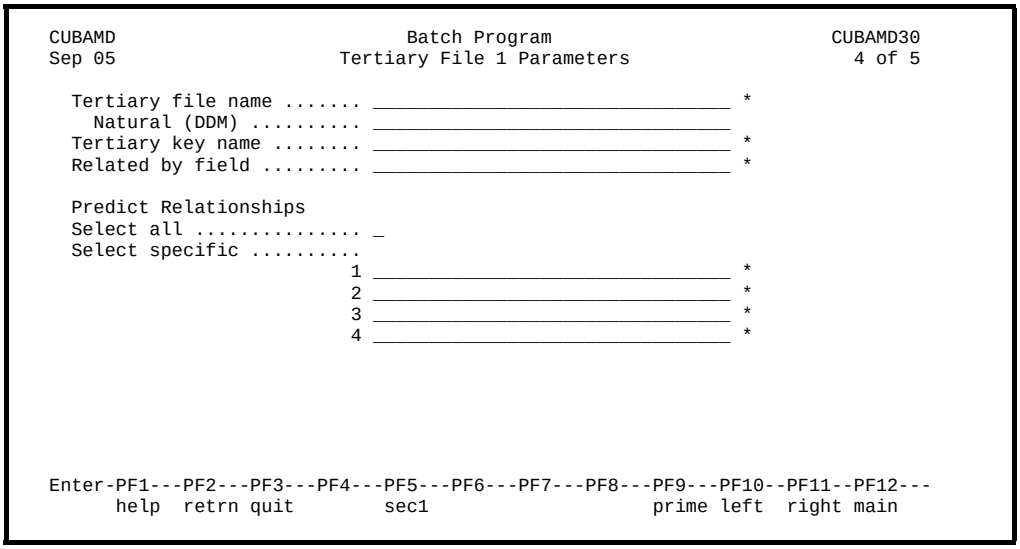
Secondary File 2 Parameters Panel for the Batch Model

Using this panel, you can relate another secondary file to the primary file. The fields on this panel are similar to those on the Secondary File 1 Parameters panel.

- To return to the Secondary File 1 Parameters panel, press PF5 (sec1).
- To return to the Primary File Parameters panel, press PF9 (prime).
- To define a tertiary file for the second secondary file, press PF6 (ter2). The Tertiary File 2 Parameters panel is displayed. This panel is similar to the Tertiary File 1 Parameters panel (described in the following section).

Accessing the Tertiary File 1

To define a tertiary file, which will be logically coupled (joined) to the first secondary file, press PF6 (ter1) on the Secondary File 1 Parameters panel. The Tertiary File 1 Parameters panel is displayed:



Tertiary File 1 Parameters Panel for the Batch Model

This panel is similar to the Secondary File 1 Parameters panel.

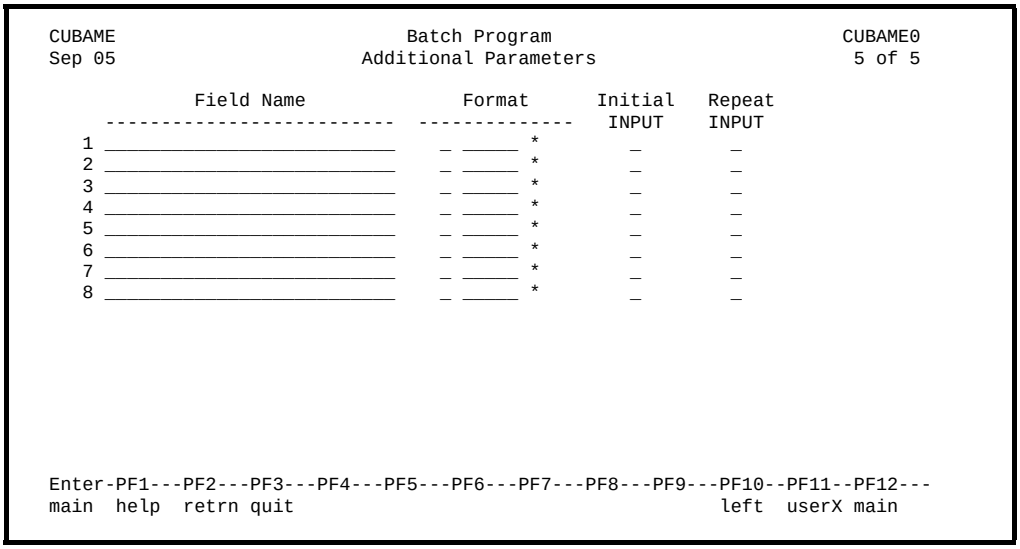
- To return to the Define Secondary 1 File panel, press PF5 (sec1).
- To return to the Primary File Parameters panel, press PF9 (prime).
- To access the Tertiary File 2 Parameters panel, display the Secondary File 2 Parameters panel and press PF6 (ter2).

Returning to the Primary File

To return to the Primary File Parameters panel from the secondary or tertiary file parameters panels, press PF9 (prime).

Additional Parameters Panel

The following example shows the fifth (and last) specification panel for the Batch model, the Additional Parameters panel:



Additional Parameters Panel for the Batch Model

Use this panel to specify the names of up to eight fields to input into the batch program, in addition to the range of key values for reading the primary file. These field names can be input either once at the beginning of the batch program or each time a new key range is specified. You can specify any field; the field does not have to be in the Predict file definition.

The fields on this panel are:

Field	Description
Field Name	Name(s) of the additional input field(s).
Format	Natural format and length. In the first portion of this field, type the code for the format of the corresponding field. In the last portion of this field, type the field length. To display a list of valid formats, press the help PF-key or enter “?” in the first-character position of this field.

Field	Description (continued)
Initial INPUT	If the Additional inputs field on the Program Structure panel is marked, this field is available for modification. To apply the corresponding field name to the entire file, mark this field.
Repeat INPUT	If the Multiple inputs field on the Program Structure panel is marked, this field is available for modification. To apply the field name to the specified input range, mark this field.

User Exits for the Batch Model

The following examples show the User Exits panel for the Batch model:

```
CSGSAMPL                    Natural Construct                       CSGSM1
Sep 05                       Batch User Exits                       1 of 1

           User Exit                Exists    Sample   Required Conditional
   ------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                           Subprogram
 _  BUILD-REPORT-LOCAL-VARS                   Example
 _  LOCAL-DATA                               Example
 _  START-OF-PROGRAM
 _  REPORT0-AT-TOP-OF-PAGE                                            X
 _  REPORT1-AT-TOP-OF-PAGE                                            X
 _  REPORT2-AT-TOP-OF-PAGE                                            X
 _  REPORT3-AT-TOP-OF-PAGE                                            X
 _  BEFORE-INITIAL-INPUT                      Example                 X
 _  AFTER-INITIAL-INPUT                       Example                 X
 _  BEFORE-RANGE-INPUT                        Example                 X
 _  AFTER-RANGE-INPUT                         Example                 X
 _  BEFORE-READ                              Example
 _  PRIME-WRITE-FIELDS                       Subprogram
 _  SEC1-WRITE-FIELDS                        Subprogram              X
 _  TER1-WRITE-FIELDS                        Subprogram              X
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
frwrd help  retrn quit                   frwrd bkwrd
```

User Exits Panel for the Batch Model — Page 1

```
CSGSAMPL                    Natural Construct                       CSGSM1
Sep 05                       Batch User Exits                       1 of 1

           User Exit                Exists    Sample   Required Conditional
   ------------------------------- -------- ---------- -------- ------------
 _  SEC2-WRITE-FIELDS                        Subprogram              X
 _  TER2-WRITE-FIELDS                        Subprogram              X
 _  AFTER-SECONDARY-FILE-PROCESSING           Example
 _  END-OF-PROGRAM
 _  PRIME-WRITE-HEADINGS                      Example                 X
 _  SEC1-WRITE-HEADINGS                       Example                 X
 _  SEC2-WRITE-HEADINGS                       Example                 X
 _  TER1-WRITE-HEADINGS                       Example                 X
 _  TER2-WRITE-HEADINGS                       Example                 X
 _  ERROR-ROUTINE                            Example                 X
 _  MISCELLANEOUS-SUBROUTINES                 Example




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                   frwrd bkwrd
```

User Exits Panel for the Batch Model — Page 2

For information about these user exits, see **User Exits for the Generation Models**, page 321. For information about the User Exit editor, see **User Exit Editor**, page 96.

BROWSE MODELS

This chapter describes the Browse models: Browse, Browse-Helpr, and Browse-Subp. These models generate modules that read a file in logical order and display record values on the screen.

The following topics are covered:

Introduction

You can display information from related files by including look-up commands within user exits. To reference database fields that are not in the primary file, define the additional views in the LOCAL-DATA user exit and write the look-up statements in the WRITE-FIELDS user exit. For information about these user exits, see **User Exits for the Generation Models**, page 321.

By default, all generated browse modules read files in ascending sequence (for example, from 1 to 999999 or from A to Z). If desired, you can generate browse modules that read a file in descending sequence (for example, from 999999 to 1 or Z to A) or modules that read a file in either order and allow the user to decide. To include this functionality, select and define the START-OF-PROGRAM user exit. For information, see **START-OF-PROGRAM**, page 369.

All Natural Construct-generated browse panels also support wildcard selection. For information, see **Wildcard Selection**, page 89.

Modules generated by the Browse models are useful to an application as:

- stand-alone programs invoked from a menu (Browse model)
- active helproutines that enable the user to select a field value from a list of valid values (Browse-Helpr model)
- subprograms that can be called from another program, such as a maintenance program (Browse-Subp model).

Browse vs. Browse-Select Models

The Browse, Browse-Helpr, and Browse-Subp models are similar to the Browse-Select, Browse-Select-Helpr, and Browse-Select-Subp models. (For more information, see **Browse-Select Models**, page 213.) When deciding which model to use, consider that the Browse models are easier to build if you do not need the added features of the Browse-Select models. Additional features of the Browse-Select models are:

- users can select multiple records on the same panel
- users can apply actions to the selected records

Comparison of Browse Model Types

The following table indicates when to use each model type. Following the table are examples of modules generated by each Browse model type.

Model Name	Description
Browse	Used when the generated browse is a stand-alone browse program (usually invoked by a FETCH statement in a menu program).
Browse-Helpr	Used when the generated browse is invoked as an active helproutine (to list the valid values for a field, for example). The helproutine always includes the primary key value as a parameter. You can also pass an additional parameter to the helproutine. If the helproutine is to return the selected value, include the PROCESS-SELECTED-RECORD user exit. A browse helproutine accepts and returns the value of the help field in the #PDA-KEY variable (the key value from which to begin scrolling and, optionally, the returned value).
Browse-Subp	Used when the generated browse is invoked as a sub-function of another program. For example, you can use a browse subprogram to perform the Browse action for a maintenance program, in which case, the maintenance program issues the browse without disturbing the current state and display of the panel. If the subprogram performs the Browse action for a maintenance program, include the PROCESS-SELECTED-RECORD user exit. The browse subprogram accepts the following parameters: • #PDA-KEY Key value from which scrolling begins and, optionally, the returned value. • CDSELPDA External parameter data area (PDA) containing the action values the browse subprogram returns to the calling object. • CDPDA-D External PDA containing standard parameters for dialogs. • CDPDA-M External PDA containing standard parameters for exchanging message information. • CDPDA-P External PDA containing global information that should be shared with the browse subprogram.

Tip: If a browse helproutine or subprogram contains complex display logic, it may be easier to first write the application as a browse program, test it, and then regenerate it as a browse helproutine or subprogram.

Browse Model

The following example shows a browse panel generated using the Browse model:

```
NCCSCUST                ***** CUSTOMER SUBSYSTEM *****
Aug 03                        - QUERY CUSTOMER -                        10:09 AM

Customer                                                        Warehouse
 Number          Business Name              Phone Number        Id
------      ----------------------         -------------       ---
 11111      BULK FOOD INTERNATIONAL        514-214-5667        111
 12121      ARTHUR AND ARTHUR LIMITED      416-732-8884        632
 13451      MIDLETON FOOD DEALERS          416-485-7854        222
 20202      MUTUAL LIFE OF CANADA          705-234-4326        632
 22222      HEIDELMAN AND COMPANY          705-234-4326        711
 22266      ROBERT ET FILS                 518-452-1114        711
 32222      HEIDELMAN AND COMPANY          705-234-4326        333
 32859      CHINATOWN FRESH                703-888-7854        111
 32885      TERRY & COMPANY                519-888-7854        999
 33333      JOHN & SON                     519-824-7854        111
 36321      CHARLES AND BREDEN FOOD        416-555-3333        422
 42226      OVEN BAKERY                    519-637-6651        222
 44444      BULK FOOD INTERNATIONAL        519-824-7854        111
Customer Number: _____  *Optional Data ->     _
Direct Command: ____________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                   bkwrd frwrd print             main
```

Sample Output from the Browse Model

To reposition the browse to a customer record that is not currently displayed, users can enter the customer number in the Customer Number field. If the key is an alphanumeric or numeric value, users can limit the range of records displayed by including wildcard characters (*, >, or <). In the above example, the key is numeric.

Note: To include wildcard support, you must select the Wildcard option before generating the program. For information, see **Wildcard Selection**, page 89.

Browse models also support a print PF-key (PF9), which routes reports to a printer. Users can press this key and specify wildcard characters to print an online report based on a range of records. Hardcopy support is optional.

To view the specifications for the Browse model example, refer to the NCCSCUST program in the Natural Construct demo system.

Browse-Helpr Model

The following example shows a browse helproutine window generated using the Browse-Helpr model:

```
NCTHWHSE ***** WAREHOUSE HELP *****
Sep 01                                10:19 AM

Warehouse          Description
---------  ----------------------------

111        TORONTO CENTRAL WAREHOUSE
112        FRESCO FREEZER LTD
113        SOUTHERN DISTRIBUTORS LIMITED
222        MONTREAL EASTERN WAREHOUSE
422        SMITH AND SMITH STORAGE INC.
544        SELNIK STORAGE
632        FRESCO FREEZER LTD
638        WATERLOO WAREHOUSING LTD.
Warehouse Id: ___
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7
      help  retrn             flip        bkw
Position cursor or ENTER screen value to sele
```

Sample Output from the Browse Helproutine Model

The browse helproutine window is displayed when the user presses PF1 (help) or enters a question mark (?) in the Warehouse ID field.

To view the specifications for the Browse-Helpr model example, refer to the NCTH-WHSE helproutine in the Natural Construct demo system.

Browse-Subp Model

The following example shows a browse window generated using the Browse-Subp model:

```
NCTFWHSB  ***** TABLE SUBSYSTEM *****
Aug 17        - WAREHOUSE BROWSE -   02:19 PM

Warehouse          Description
--------- ------------------------------

111       TORONTO CENTRAL WAREHOUSE
113       SOUTHERN DISTRIBUTORS LIMITED
222       MONTREAL EASTERN WAREHOUSE
422       SMITH AND SMITH STORAGE INC.
544       SELNIK STORAGE
632       FRESCO FREEZER LTD
638       WATERLOO WAREHOUSING LTD.
680       STOKES AND BABAGE LIMITED
Warehouse Id: ___
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7--
      help  retrn quit        flip        bkwrd
Position cursor or ENTER screen value to select
```

Sample Output from the Browse-Subp Model

This browse window is displayed when the user types the Browse code in the Action field and presses Enter.

To view the specifications for the Browse-Subp model example, refer to the NCTF-WHSB subprogram in the Natural Construct demo system.

Parameters for the Browse Models

Specification panels for Browse models are almost identical, since the generated code performs a similar function. The only exception is an additional specification panel for the Browse-Helpr and Browse-Subp models, called the Specific Parameters panel. The following sections describe the specification panels for all Browse model types, as well as the Specific Parameters panel for the Browse-Helpr and Browse-Subp models.

Standard Parameters Panel

The following example shows the first specification panel for Browse models, the Standard Parameters panel:

```
CUSCMA                          Browse Program                        CU--MA0
Sep 05                       Standard Parameters                       1 of 4

 Module ............. BRSE____
 System ............. DEMO_____________________________
 Global data area ... CDGDA___
 With block ......... _____________________________

 Title ............. Browse ..._______________
 Description ........ This program is used to browse the ..._____________________
                      ____________________________________________________________
                      ____________________________________________________________
                      ____________________________________________________________

 First header ....... _____________________________________________
 Second header ...... ____________________________________________________

 Command ............ _
 Message numbers .... _
 Password ........... _

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                                          right main
```

Standard Parameters Panel for Browse Models

This panel is similar for all models. For a description of the Standard Parameters panel, see **General Model Specifications**, page 167.

Note: An important difference between the Browse-Subp and the Browse model is that the Global data area field on the Standard Parameters panel is normally blank for the Browse-Subp. It should not contain the name of the standard application GDA (derived from CDGDA). You need only specify a GDA name for the Browse-Subp model if the browse subprogram does not use the standard application GDA.

Additional Parameters Panel

The following example shows the second specification panel for Browse models, the Additional Parameters panel:

```
CUSCMB                          Browse Program                          CUSCMB0
Sep 05                     Additional Parameters                         2 of 4

  Predict view ............. _____________________________________ *
    Natural (DDM) .......... _____________________________________
    Program view ........... _____________________________________
  Primary key .............. _____________________________________ *

  Horizontal panels ........ 1_
  Backward scroll pages .... 10

  Input using map .......... ________ *
  Reserved input lines ..... _
  Minimum key value ........ ____________________________________________________
  Maximum key value ........ ____________________________________________________
  Single prompt ............ _
  Multiple prompts ......... _

  Wildcard support ......... _        Export data support ...... _
  Hardcopy support ......... _        Window support ........... _

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        windw                         left  right main
```

Additional Parameters Panel for Browse Models

On this panel, you can specify additional parameters for your browse program. The fields on this panel are:

Field	Description
Predict view	Name of the Predict view used by the browse program (must be defined in Predict).

Natural (DDM)	Name of the data definition module (DDM) corresponding to the primary file. If this field remains blank, the DDM name defaults to the primary file name. The Predict definition of the primary file determines which fields are included in the DEFINE DATA section of your generated code. The format of the generated code in the DEFINE DATA section has the following structure:
	`1 Primary-file-name VIEW OF Data-definition-module`
	`  2 fields pulled from Predict of Primary-file-name`

Field	Description (continued)
Program view	View name for the primary file in the browse program. This view must be defined in the LOCAL-DATA user exit or a local data area (LDA). If this field is blank, a view is generated containing all fields in the Predict view. The MAX.OCCURS value in Predict determines how many occurrences of MU/PE fields are included on the panel.
Primary key	Name of the primary key by which scrolling takes place. This key must be defined as a descriptor, superdescriptor, or subdescriptor in the Predict file definition. Keys containing MUs (multiple-valued fields) and PEs (periodic groups) are supported. If this key does not exist in the corresponding Predict file, a message is displayed. **Note:** For DB2 users, add the combination of fields as a superdescriptor in Predict if you want to use more than one field to determine the sort sequence of the records being browsed.
Horizontal panels	Number of panels the generated program contains. The default is 1 panel. To change the number of panels, type a new number over the one displayed.
Backward scroll pages	Maximum number of pages the generated program can scroll. The default is 10, which indicates that users can scroll forward and backward within a 10-page range. If they scroll forward 11 pages, page 1 is forced out of the range and they cannot scroll back to it. **Note:** A Natural Construct-generated browse program does not allow backward scrolling over data that has not been previously scrolled through in a forward direction.

Field	Description (continued)
Input using map	Name of the layout map for the browse program. The map is included as part of the END OF PAGE processing to input the values that control scrolling. If you specify more than nine horizontal panels, the map name cannot exceed six bytes. Include the #SCR-CV control variable in the map definition. Also specify the #KEY-CV control variable for all input fields that are part of the browse key. Define the input fields used to reposition the browse key, as well as any additional input fields, within the #INPUT structure in the GDA for the browse program. If a map name is specified, it should be a short map that is displayed at the bottom of the panel. The CDLAYSC1 layout map is supplied for Browse models. For more information, see **Using Layout Maps**, page 120. If more than one horizontal panel is required, you can use a different input map for each panel. If you include an asterisk (*) in the map name (for example, MYMAP*), the asterisk is replaced by the panel number in the generated program (for example, MYMAP1, MYMAP2, MYMAP3). If you do not specify a layout map, Natural Construct places the input fields sequentially at the bottom of the panel. **Note:** For a list of the variables you can use, see **Variables You Can Use with a Browse Model Map**, page 202.
Reserved input lines	Number of lines reserved for input prompts (typically 1, 2, or 3). **Note:** This field only applies to external maps.
Minimum key value	Starting value for the browse. The combination of the minimum and maximum keys creates a logical window within the file. To establish a logical start and end of the file, specify minimum and maximum key values. The program will not browse before or after these values. The minimum key value must be a constant. The specified constant is placed into a variable called #MIN-KEY-VALUE, which you can override in the START-OF-PROGRAM exit.
Maximum key value	Ending value for the browse. The maximum key value must be a constant and greater than or equal to the minimum key value. The specified constant is placed into a variable called #MAX-KEY-VALUE, which you can override in the START-OF-PROGRAM exit.

Field	Description (continued)
Note:	You can set the minimum and maximum values as variables within user exit code. For example, if the first three characters of personnel ID represent the department code, you can restrict the browse to a specific department based on where the browse was called from or who was calling it. To do this, use the START-OF-PROGRAM user exit to look up and retrieve the current user's department code (assuming it is stored) and then use this information to populate a variable that overrides the #MIN-KEY-VALUE and #MAX-KEY-VALUE values (created when constants are populated through the specifications). If Smith belongs to department 555, for example, you can populate the minimum value with 555 and the maximum value with 55599999 to retrieve all data for department 555.
Single prompt	If this field is marked, a single prompt is displayed for all fields (for example, Date:____ __ __). This option applies if the key is a superdescriptor or redefined in Predict.
Multiple prompts	If this field is marked, one prompt is displayed for each field (for example, Year:____ Month:__ Day:__). This option applies if the key is a superdescriptor or redefined in Predict.
Wildcard support	If this field is marked, wildcard processing is enabled in the generated program and numeric key values are input into an alphanumeric field. This allows the user to enter *, >, or <.
Hardcopy support	If this field is marked, the hardcopy facility is enabled in the generated program.
Export data support	If this field is marked, records are exported to a work file in addition to, or instead of, the screen. You can then use the work file in other environments and on other platforms (for example, in a PC spreadsheet application). To write data from the generated report to a work file, select the EXPORT-DATA user exit and define the parameters to export to the work file. The work file number and delimiter character (used to delimit fields on the report) can be customized for your site.

Field	Description (continued)
Note:	If you mark this field and do not select and define the EXPORT-DATA user exit, a default WRITE WORK FILE statement is generated that includes all fields in the view.
Window support	If this field is marked, the module's output is displayed in a window, rather than a panel. By default, the window size is adjusted to its content. The window is placed on the screen so that the field from which the user invoked it is visible.

Variables You Can Use with a Browse Model Map

If you specified the Input using map field on the Additional Parameters panel, you can use the following variables with the map:

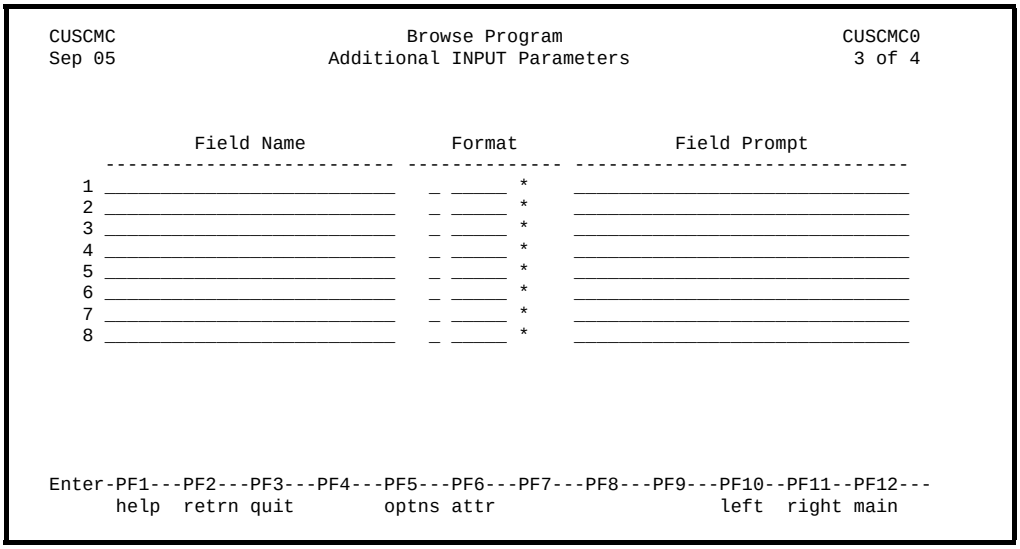
Variable	Format	Attributes	Description
#HPARM	A65	Output/ Nondisplay	Contains the key for Natural Construct's passive help file for the current program (supplied in CDDIALDA). Place this variable on the map and pass it to the CD-HELPR helproutine.
#DIRECT-COMMAND	A60	Modifiable	Indicates support for direct command processing (supplied in CDGETDCA).

Changing the Default Window Settings

To set the size of the Browse window, press PF5 to display the Define Windows Parameters window. For a description of this window, see **Changing the Default Window Settings for Generated Modules**, page 86.

Additional INPUT Parameters Panel

The following example shows the third specification panel for Browse models, the Additional INPUT Parameters panel:



Additional INPUT Parameters Panel for Browse Models

In addition to the key field, you can include up to eight input fields in a browse program. These fields are displayed at the bottom of the generated panel and allow the user to display more information. For example, you can create an additional input field called "Detail" (format L) to display additional record details. Users can mark the Detail field to display the details.

Note: Fields specified on the Additional INPUT Parameters panel do not have to be in the Predict file definition.

The fields on this panel are:

Field	Description
Field Name	Name(s) of the additional input field(s).
Format	Natural format and length of the input field (for example, A8). This includes the number of positions after the decimal point in numeric input fields.

Field	Description (continued)
Field Prompt	Field prompt displayed on the generated browse panel. Enclose words you want to display intensified within angle (<>) brackets (or whatever character is set for intensify). To change attribute characters, see **Changing the Default Attribute Parameters**, page 205. If you do not specify a field prompt, Natural Construct creates a default prompt using the name of the input field. The field is displayed at the bottom of the generated panel.

Specifying Options for Additional Input Fields

To specify options for additional input fields, press PF5 (optns) to display the Optional Parameters window:

```
CUSCMCA                    Natural Construct                    CUSCMCA0
Sep 05                   Optional Parameters                      1 of 1

                                                                 Prompt
          Session Parameters          Array     Panel    New Line   OFF
    ------------------------------ ---------- -------- ---------- --------
  1 ______________________________   __  __      __         _        _
  2 ______________________________   __  __      __         _        _
  3 ______________________________   __  __      __         _        _
  4 ______________________________   __  __      __         _        _
  5 ______________________________   __  __      __         _        _
  6 ______________________________   __  __      __         _        _
  7 ______________________________   __  __      __         _        _
  8 ______________________________   __  __      __         _        _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                              mai
```

Optional Parameters Window for Browse Models

The fields in this window are:

Field	Description
Session Parameters	One or more session parameters for the additional input field, such as Attribute Definition (AD) or Edit Mask (EM). For example: `AD=I SG=ON EM='>'X HE='HELPR'`
Array	Range of occurrences of an array variable to place on the panel. If you specify a range with different first and last values, a single prompt precedes all elements in the range. For multiple prompts, specify each occurrence separately.

Field	Description (continued)
Panel	Number of the panel on which the input prompt is displayed. If a panel number is not specified, the prompt is displayed on all panels.
New Line	If this field is marked, the input field for the prompt begins on a new line.
Prompt OFF	If this field is marked, the default prompt is suppressed.

Changing the Default Attribute Parameters

To change the dynamic attributes for your browse program, press PF6 (attr) to display the Dynamic Attribute Parameters window. For a description of this window, see **Changing the Default Attribute Parameters for Generated Modules**, page 87.

Specific Parameters Panel for the Browse-Helpr Model

The following example shows the fourth specification panel for the Browse-Helpr model, the Specific Parameters panel:

```
CUSCME                      Browse-Helpr Program                    CUSCME0
Sep 05                      Specific Parameters                      4 of 5

   Parameter Override
   Note: Format and Length defaults to PERSONNEL-ID key
   Field name ......... #PDA-KEY
   Natural format ..... A 8.0__ *

   Component Parameters
   Field name ......... ____________________________
   Natural format ..... _ _____ *
   Array .............. __ 1 __ 2 __ 3









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                                     left  right main
```

Specific Parameters Panel for the Browse-Helpr Model

On this panel, you can override the format and/or length of the passed parameter or pass an additional parameter to the helproutine. The key for the browse may differ from the key for the calling program. If the key differs, indicate the format and length of the passed key on this panel. Also indicate the name of any additional helproutine parameter, as well as its format and length.

Note: Browse helproutines normally include the PROCESS-SELECTED-RECORD user exit and assign the value of the #SELECTED-KEY variable to the #PDA-KEY value that is returned.

Use the top portion of this panel to specify the format and length of the help field (if it is different from that of the primary browse key).

Use the bottom portion of this panel to specify an additional parameter. If no parameter is specified, the generated helproutine has only one parameter (#PDA-KEY), containing the contents of the input field to which the helproutine is attached. If the helproutine changes the value of #PDA-KEY, the changed value is displayed in the input field when the helproutine returns control to the INPUT statement.

In some cases, it is preferable to pass an additional parameter to the helproutine. For example, if the helproutine selects an order number, you may want to restrict the selection to orders for a certain customer. Specify the additional parameter on the Specific Parameters panel. The parameter is passed to the helproutine as the first parameter, and #PDA-KEY is passed as the second parameter. For information about passing additional parameters, see **Using Parameter Data Areas**, page 117.

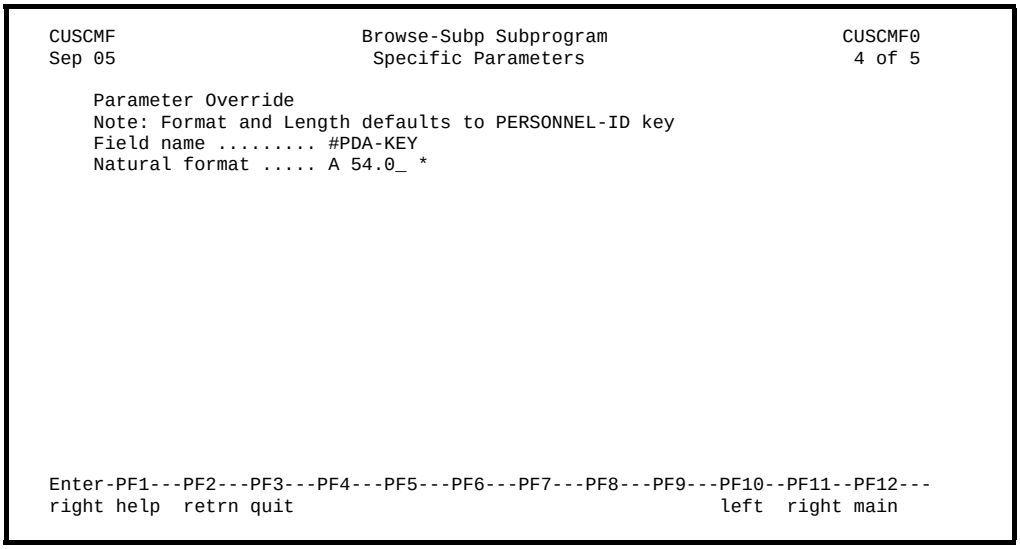
If the helproutine uses an additional parameter, a value must be supplied to the helproutine when it is invoked. To do this, supply a value, enter HE='*routine*',*parameter* in the INPUT statement, where *routine* is the helproutine name and *parameter* is the value passed to the additional parameter. Or enter the helproutine name and parameter in the HE field on the Extended Field Editing panel in the Map editor. The value passed must have the same format and length as the additional parameter.

The fields on the Specific Parameters panel are:

Field	Description
Field name	Name of the primary key. By default, #PDA-KEY is displayed.
Natural format	Natural length and format of the passed field (if it is different from that of the key being browsed). This format becomes the format for the #PDA-KEY field.
Field name	Name of the additional parameter.
Natural format	Natural length and format of the additional parameter.
Note:	Any valid combination of format, length, and decimal positions under Natural is allowed.
Array	Array dimensions. To declare the field an array, enter the array dimensions in the 1, 2, and 3 fields.

Specific Parameters Panel for the Browse-Subp Model

The following example shows the fourth specification panel for the Browse-Subp model, the Specific Parameters panel:



Specific Parameters Panel for the Browse-Subp Model

As with the browse helproutine key, the primary key for a browse subprogram can be different from that of the calling program. If the key differs, specify the format and length of the passed key on this panel.

By default, the Browse-Subp model generates the following parameters and data areas:

Parameter	Description
#PDA-KEY	On entry, this parameter supplies the value to begin browsing from. On exit, this parameter must be assigned the key of the record selected. (The parameter is normally assigned the value of the #SELECTED-KEY variable in the PROCESS-SELECTED-RECORD user exit.) The format and length of this parameter defaults to the format and length of the key being browsed. You can change the format and length on the Specific Parameters panel.
CDSELPDA	By default, this parameter data area contains the function specified in the browse subprogram. The function is returned to the calling program. (You can extend this data area as required.)

Parameter	Description (continued)
CDPDA-D CDPDA-M CDPDA-P	See **Using Parameter Data Areas**, page 117.
	Note: You can specify additional parameters in the PARAMETER-DATA user exit.

The fields on the Specific Parameters panel are:

Field	Description
Field name	Name of the primary key. By default, #PDA-KEY is displayed.
Natural format	Natural length and format of the passed field (if it is different from that of the key being browsed). This format becomes the format for the #PDA-KEY field.

Restriction Parameters Panel

The following example shows the fourth specification panel for the Browse model (and the fifth specification panel for the Browse-Helpr and Browse-Subp models), the Restriction Parameters panel:

```
CUSCMG                          Browse Program                          CUSCMG0
Sep 05                     Restriction Parameters                        4 of 4

  Prefix Options
  Restrict browse with prefix .. _

  Number of characters (bytes) . ___
  Number of components ......... __

  Protect prefix ............... _
  Suppress prefix .............. _

  Field name ................ ________________________________________________


  Prefix Helproutine Specifications
     Component       Helproutine    Parameter Name
        1             ________ *    ________________________________
        2             ________ *    ________________________________


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit                                       left  userX main
```

Restriction Parameters Panel for Browse Models

You can limit the program to only browse records prefixed by a global variable. If the prefix is a department code, for example, you can restrict the browse to only those orders prefixed by a department code. You can set the value of the code as a function of a user ID and store the value in the global data area.

For a browse helproutine or subprogram, you can limit the browse by passing the prefix portion of the key. To display only the lines for a particular order, for example, you can pass the Order Number (N6) to the subprogram and enter “N6” in the Natural Format field on the Specific Parameters panel. On the Restriction Parameters panel, mark the Restrict browse with prefix field and enter “6” in the Number of characters field. By default, #PDA-KEY is the field name.

Note: For DL/1 users, if you are creating a browse program for an IMS file that is not at level 1, you must specify a prefix that includes the key for all segments above the current segment.

The fields on the Restriction Parameters panel are:

Field	Description
Restrict browse with prefix	If this field is marked, the browse is limited to values for which the primary key is prefixed by or equal to the specified value.
Number of characters (bytes)	Number of bytes (of the primary key) to use as the prefix.
Number of components	When using a compound key, the number of key components used as a prefix.
Protect prefix	If this field is marked, the prefix portion of the primary key for the input field is protected. The prefix is displayed but cannot be changed.
Suppress prefix	If this field is marked, the prefix portion of the primary key is not displayed.
Note:	To use the Protect prefix or Suppress prefix option, the primary key must be a superdescriptor, a compound IMS key, or redefined in Predict.

Field	Description (continued)
Field name	Name of the field containing the prefix value. When you generate a browse helproutine or subprogram, the prefix portion of the key is assumed to be equal to #PDA-KEY (the value of the key passed to the helproutine or subprogram). To override this default, enter a variable name in this field. Define the variable in the LOCAL-DATA user exit and assign a value to the field in the ASSIGN-PREFIX-VALUE user exit. The assigned value (instead of #PDA-KEY) will then be used as the value for the prefix portion of the key.
Component	Number of the corresponding component ("1" indicates the first component; "2" indicates the second, etc.).
Helproutine	Name of the helproutine for the prefix. To attach a helproutine to the prefix of the primary key, enter the name of the helproutine in this field. You can specify a helproutine for each component.
Parameter Name	Parameter for the helproutine for each component. Define the help parameters in the LOCAL-DATA user exit.
Note:	The Prefix Helproutine Specifications fields are protected when entering parameters for a browse helproutine.

User Exits for the Browse Models

The user exits panels for the Browse, Browse-Helpr, and Browse-Subp models are identical and function in the same way. The following examples show these panels:

```
CSGSAMPL                        Browse Program                          CSGSM0
Sep 05                            User Exits                            1 of 1

               User Exits             Exists    Sample   Required Conditional
    ------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                             Subprogram
 _  PARAMETER-DATA                              Example                X
 _  HE-PARAMETER-INDEXES                        Example                X
 _  BUILD-REPORT-LOCAL-VARS
 _  LOCAL-DATA                                 Subprogram
 _  START-OF-PROGRAM                            Example
 _  SELECT-STATEMENT                           Subprogram              X
 _  AFTER-READ                                  Example
 _  REJECT-AFTER-MAX-KEY-CHECK
 _  EXPORT-DATA                                 Subprogram              X
 _  WRITE-FIELDS                               Subprogram
 _  TOP-OF-PAGE                                 Example
 _  BEFORE-INPUT                                Example
 _  BEFORE-STANDARD-KEY-CHECK                   Example
 _  AFTER-INPUT                                 Example
 _  HARDCOPY-EDITS                              Example                X
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
frwrd help  retrn quit                    bkwrd frwrd
```

Browse User Exits Panel

```
CSGSAMPL                        Browse Program                          CSGSM0
Sep 05                            User Exits                            1 of 1

               User Exits             Exists    Sample   Required Conditional
    ------------------------------- -------- ---------- -------- ------------
 _  HARDCOPY-TERMINATING-PROCESS                Example                X
 _  PROCESS-SELECTED-RECORD                    Subprogram              X
 _  END-OF-PROGRAM                              Example
 _  ASSIGN-PREFIX-VALUE                        Subprogram              X
 _  SET-PF-KEYS                                 Example
 _  MISCELLANEOUS-SUBROUTINES                   Example









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                    bkwrd frwrd
```

Browse User Exits Panel

For information about these user exits, see **User Exits for the Generation Models**, page 321. For information about the User Exit editor, see **User Exit Editor**, page 96.

BROWSE-SELECT MODELS

This chapter describes the Browse-Select models: Browse-Select, Browse-Select-Helpr, and Browse-Select-Subp. These models are similar to the Browse models, but provide more options. A generated browse program performs one pre-determined set of commands on a selected record (such as delete a record or modify a record); a generated browse-select program allows the user to specify which set of commands are performed.

The following topics are covered:

Introduction

Like the Browse models, Browse-Select models generate modules that read files in logical order and display record values on the screen. Users can scroll backward, forward, left, and right when using generated Browse-Select modules.

You can also display information from related files by including look-up commands within the user exits. To reference database fields that are not in the primary file, you must define the additional views in the LOCAL-DATA user exit and write the look-up statements in the WRITE-FIELDS user exit. For information about these user exits, see **User Exits for the Generation Models**, page 321.

By default, all generated browse-select modules read files in ascending sequence (for example, from 1 to 999999 or from A to Z). If desired, you can generate browse-select modules that read a file in descending sequence (for example, from 999999 to 1 or Z to A) or modules that read a file in either order and allow the user to decide. To include this functionality, select and define the START-OF-PROGRAM user exit. For information, see **START-OF-PROGRAM**, page 369.

All Natural Construct-generated browse-select panels also support wildcard selection. For information, see **Wildcard Selection**, page 89.

The Browse-Select models support an optional action/selection column. Use a Browse-Select model instead of a Browse model when you want the user to select a record for subsequent processing or perform actions on selected records.

Modules generated using the Browse-Select models are useful to an application as:

- stand-alone programs invoked from a menu (Browse-Select model)
- active helproutines that enable the user to select a field value from a list of valid values (Browse-Select-Helpr model)
- subprograms that can be called from another program, such as a maintenance program (Browse-Select-Subp model)

Browse vs. Browse-Select Models

The Browse-Select, Browse-Select-Helpr, and Browse-Select-Subp models are similar to the Browse, Browse-Helpr, and Browse-Subp models. (For more information, see **Browse Models**, page 191.) When deciding which model to use, consider that the Browse models are easier to build if you do not need the added functionality of the Browse-Select models. Additional features of the Browse-Select models are:

- users can select multiple records on the same panel
- users can apply actions to the selected records

Comparison of Browse-Select Model Types

The following table indicates when you can use each model type. Following the table are examples of modules generated using each Browse-Select model type.

Model Name	Description
Browse-Select	Used when the generated browse-select is a stand-alone program (usually invoked by a FETCH statement in a menu program).
Browse-Select-Helpr	Used when the generated browse-select is invoked as an active helproutine (for example, to list the valid values for a field). The helproutine always includes the primary key value as a parameter. You can also pass an additional parameter to the helproutine. If the helproutine is to return the selected value, include the PROCESS-SELECTED-RECORD user exit. A browse-select helproutine accepts and returns the value of the help field in the #PDA-KEY variable (the key value from which to begin scrolling and, optionally, the returned value).

Model Name	Description (continued)
Browse-Select-Subp	Used when the generated browse-select is invoked as a sub-function of another program. For example, you can use a browse-select subprogram to perform the Browse action for a maintenance program, in which case, the maintenance program invokes the subprogram without disturbing the current state of the panel. If the browse-select subprogram performs the Browse action for a maintenance program, include the PROCESS-SELECTED-RECORD user exit. The browse-select subprogram accepts the following parameters: • #PDA-KEY Key value from which scrolling begins and, optionally, the returned value. • CDSELPDA External parameter data area (PDA) containing the action values the browse-select subprogram returns to the calling object. • CDPDA-D External PDA containing standard parameters for dialogs. • CDPDA-M External PDA containing standard parameters for exchanging message information. • CDPDA-P External PDA containing global information that should be shared with the browse-select subprogram.

Tip: If a browse-select helproutine or subprogram contains complex display logic, it may be easier to first write the application as a browse-select program, test it, and then regenerate it as a browse-select helproutine or subprogram.

Browse-Select Model

The following example shows a browse-select panel generated using the Browse-Select model:

```
NCOSELX                  ***** ORDER SUBSYSTEM *****                  NCOSELM1
Dec 21                       - ORDER SELECTION -                       1:20 AM

        Order   Customer  Warehouse  Invoice    Order        Order
Action  Number   Number      ID      Number     Date         Amount
------  ------  --------  ---------  -------  --------  ----------------
   __      111   10003       112         555  99/03/25           6135.00
   __      121       2       544      111111  99/05/03          12745.00
   __      122   22266       544         999  99/08/12          10880.00
   __      777   10001       113      111111  95/02/23           1500.00
   __      991   33333       544      111111  98/12/17           1500.00
   __     1111   10003       422       11111  99/02/19           1500.00
   __     1121   10003       112         555  01/11/20           6135.00
   __     1234   11111       113      111111  93/07/09           1500.00
   __     9121       1       544      111111  99/03/25           3100.00
   __    15001   10007       544       10005  99/09/30           2400.00
   __   111112   11111       111      123131  99/03/19           2160.00
   __   111119       1       134      190001  93/03/30           8055.00
   __   200004   10001       113      105901  93/03/10           7573.00
Order Number: ______ Customer Details: _
Direct Command: ____________________________________________________________
 Copy       DEtail     DIsplay    Modify     Purge      (PF5=flip)
```

Sample Output from the Browse-Select Model

To reposition the list to a record that is not currently displayed, users can enter the order number in the Order Number field. If the key is either an alphanumeric or numeric value, users can limit the range of records displayed by including wildcard characters (*, >, <). In addition, users can select a record for additional processing by specifying an action code in the Action field.

Note: To include wildcard support, you must select the Wildcard option before generating the browse-select program. For information, see **Wildcard Selection**, page 89.

Browse-Select models also support a print PF-key (PF9), which routes reports to a printer. Users can press this key and specify wildcard characters to print an online report based on a range of records. Hardcopy support is optional.

To view the specifications for the Browse-Select model example, refer to the NCOSELX program in the Natural Construct demo system.

Browse-Select-Helpr Model

The following example displays a typical browse-select helproutine window generated using the Browse-Select-Helpr model:

```
NCOHCUST  ***** CUSTOMER NUMBER HELP *****
             - Customer Maintenance -     01:59 PM

       Customer
Action  Number          Business Name
------ -------- ------------------------------

  _     11111   QUAKER OATS
  _     22222   KENT VETERINARY CLINIC
  _     22266   ROBERT ET FILS
  _     32859   CHINATOWN FRESH
  _     76339   FROZEN FOOD FANTASIA
  _     87774   BRANT GROCERY LIMITED
           ****** End of Data ******

Customer Number: _____
Add        Copy       Display    Modify
Select     (PF5=flip)
```

Sample Output from the Browse-Select-Helpr Model

This window is displayed when the user presses PF1 (help) on the Maintain Order Entries panel or enters a question mark (?) in the Customer Number field. The actions available in this window are Add, Copy, Display, Modify, and Select.

To see the specifications for the Browse-Select-Helpr model example, refer to the NCOHCUST helproutine in the Natural Construct demo system.

Browse-Select-Subp Model

The Browse-Select-Subp model is similar to the Browse-Select and the Browse-Select-Helpr model except it generates a subprogram rather than a program or helproutine.

Parameters for the Browse-Select Models

Specifications for the Browse-Select model types are almost identical, since the generated code performs a similar function. The only exception is an additional specification panel for the Browse-Select-Helpr and Browse-Select-Subp models, called the Specific Parameters panel. The following sections describe the specification panels for all Browse-Select model types, as well as the Specific Parameters panel for the Browse-Select-Helpr and Browse-Select-Subp models.

Standard Parameters Panel

The following example shows the first specification panel for Browse-Select models, the Standard Parameters panel:

```
CUSCMA                      Browse-Select Program                      CU--MA0
Oct 29                       Standard Parameters                        1 of 5

 Module ............. BRSE-SEL
 System ............. DEMO_____________________________
 Global data area ... CDGDA___
 With block ......... _____________________________

 Title ............... Browse Select Example____
 Description ........ This program is used to browse the files._______________
                      ____________________________________________________________
                      ____________________________________________________________
                      ____________________________________________________________

 First header ....... Browse-Select Demo_____________________________
 Second header ...... ____________________________________________________

 Command ............ _
 Message numbers .... _
 Password ........... _

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                                          right main
```

Standard Parameters Panel for Browse-Select Models

This panel is similar for all models. For a description of the Standard Parameters panel, see **General Model Specifications**, page 167.

Note: An important difference between the Browse-Select-Subp and the Browse-Select model is that the Global data area field on the Standard Parameters panel is normally blank for the Browse-Select-Subp model. It should not contain the name of the standard application global data area (derived from CDGDA). You need only specify a GDA name if the browse-select subprogram does not use the standard application GDA.

Additional Parameters Panel

The following example shows the second specification panel for Browse-Select models, the Additional Parameters panel:

```
CUSLMB                    Browse-Select Program                    CUSCMB0
Oct 29                    Additional Parameters                     2 of 5

  Predict view ............. _____________________________________ *
    Natural (DDM) .......... _____________________________________
    Program view ........... _____________________________________
  Primary key .............. _____________________________________ *

  Horizontal panels ........ 1_
  Backward scroll pages .... 10

  Input using map .......... ________ *
  Reserved input lines ..... _
  Minimum key value ........ ____________________________________________
  Maximum key value ........ ____________________________________________
  Single prompt ............ _
  Multiple prompts ......... _

  Wildcard support ......... _        Export data support ...... _
  Hardcopy support ......... _        Window support ........... _

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        windw                        left  right main
```

Additional Parameters Panel for Browse-Select Models

Use this panel to specify additional parameters for your browse-select program. The fields on this panel are:

Field	Description
Predict view	Name of the Predict view used by the browse-select program (must be defined in Predict).

Natural (DDM)	Name of the data definition module (DDM) that corresponds to the primary file. If this field is blank, the DDM name defaults to the primary file name.
	
	The Predict definition of the primary file determines which fields are included in the DEFINE DATA section of the generated code. The format of the generated code in the DEFINE DATA section has the following structure:
	
	`1 Primary-file-name VIEW OF Data-definition-module`
	`  2 fields pulled from Predict of Primary-file-name`

Field	Description (continued)
Program view	View name for the primary file in the browse-select program. This view must be defined in the LOCAL-DATA user exit or a local data area (LDA). If this field is blank, a view is generated containing all fields in the Predict view for the file specified in the Primary File Name field. The MAX.OCCURS value in Predict determines how many occurrences of MU/PE fields are included.
Primary key	Name of the primary key that facilitates scrolling. This key must be defined as a descriptor, superdescriptor, or subdescriptor in the Predict file definition. Keys containing MUs (multiple-valued fields) and PEs (periodic groups) are supported. If the key does not exist in the corresponding Predict file, a message is displayed.
Note:	For DB2 users, if you want to use more than one field to determine the sort sequence of the records being browsed, add the combination of fields as a superdescriptor in Predict.
Horizontal panels	Number of panels the generated program contains. The default is 1 panel. To change the number of panels, type a new number over the one displayed.
Backward scroll pages	Maximum number of pages the generated program can scroll. The default is 10, which indicates that users can scroll forward and backward within a 10-page range. If they scroll forward 11 pages, page 1 is forced out of the range and they cannot scroll back to it.
Note:	A Natural Construct-generated browse-select program does not allow backward scrolling over data that has not previously been scrolled in a forward direction.
Input using map	Name of the layout map for the browse-select program. The map is part of the END OF PAGE processing to input values to control scrolling. If you specify more than nine horizontal panels, the map name cannot exceed six bytes. Include the #SCR-CV control variable in the map definition. Also specify the #KEY-CV control variable for all input fields that are part of the browse key. Define the input fields used to reposition the browse key, as well as any additional input fields, in the #INPUT structure in the GDA for the browse-select program. To support an action/selection column, include the column on the map as an array called #ACTIONS. Attach the #ACTION-CV control variable to #ACTIONS. To display a list of available actions on the generated panel, include the CDDIALDA.#KD-LINE1 and CDDIALDA.#KD-LINE2 variables on the map.

Field	Description (continued)
	Use the CDLAYSL1 layout map to create a map with an action column or the CDLAYSL2 layout map if the program does not support an action column. For information, see **Using Layout Maps**, page 120. If more than one horizontal panel is required, use a different input map for each panel. If an asterisk (*) is included in the map name (for example, MYMAP*), it is replaced by the panel number in the generated program (for example, MYMAP1, MYMAP2, and MYMAP3). If you do not specify a map name, Natural Construct places the input fields sequentially at the bottom of the panel.
Note:	For a list of the variables you can use, see **Variables You Can Use with a Browse-Select Model Map**, page 224.
Reserved input lines	Number of lines reserved for input prompts (typically, 1, 2, or 3).
Note:	This field only applies to external maps.
Minimum key value	Starting value for the browse. The combination of the minimum and maximum keys creates a logical window within the file. To establish a logical start and end of the file, specify minimum and maximum key values. The program will not browse before or after these values. The minimum key value must be a constant. The specified constant is placed into a variable called #MIN-KEY-VALUE, which you can override in the START-OF-PROGRAM exit.
Maximum key value	Ending value for the browse. The maximum key value must be a constant and greater than or equal to the minimum key value. The specified constant is placed into the #MAX-KEY-VALUE variable, which you can override in the START-OF-PROGRAM exit.
Note:	You can set the minimum and maximum values as variables within user exit code. For example, if the first three characters of personnel ID represent the department code, you can restrict the browse to a specific department based on where the browse was called from or who was calling it. To do this, use the START-OF-PROGRAM user exit to look up and retrieve the current user's department code (assuming it is stored) and then use this information to populate a variable that overrides the #MIN-KEY-VALUE and #MAX-KEY-VALUE values (created when constants are populated through the specifications). If Smith belongs to department 555, for example, you can populate the minimum value with 555 and the maximum value with 55599999 to retrieve all data for department 555.

Field	Description (continued)
Single prompt	If this field is marked, a single prompt is displayed for all fields (for example, Date:____ __ __). This option only applies if the key is a superdescriptor or redefined in Predict.
Multiple prompts	If this field is marked, a prompt is displayed for each field (for example, Year:____ Month:__ Day:__). This option applies if the key is a superdescriptor or redefined in Predict.
Wildcard support	If this field is marked, wildcard processing is enabled and numeric key values are input into an alphanumeric field. This allows the user to enter "*", ">", or "<".
Hardcopy support	If this field is marked, the hardcopy facility is enabled in the generated program.
Export data support	If this field is marked, records are exported to a work file in addition to, or instead of, the screen. You can then use the work file in other environments and on other platforms (for example, in a PC spreadsheet application). To write data from the generated report to a work file, select the EXPORT-DATA user exit and define the parameters to export to the work file. The work file number and delimiter character (used to delimit fields on the report) can be customized for your site. **Note:** If you mark this field and do not select and define the EXPORT-DATA user exit, a default WRITE WORK FILE statement is generated that includes all fields in the view.
Window support	If this field is marked, the module's output is displayed in a window, rather than a panel. By default, the window size is adjusted to its content. The window is placed on the screen so that the field from which the user invoked it is visible.

Variables You Can Use with a Browse-Select Model Map

If you specified the Input using map field on the Additional Parameters panel, you can use the following variables with the map:

Variable	Format	Attributes	Description
#PROGRAM	A8	Output	Contains the name of the program that invoked the map (supplied in CDDIALDA).
#HEADER1	A60	Output	First heading for the module.
#HEADER2	A58	Output	Second heading for the module.
#LEFT-PROMPT	A9	Output	Indicates the number of panels to the right of the current panel (multiple-panel programs). If the current panel is the rightmost panel, this variable contains the current date.
#RIGHT-PROMPT	A9	Output	Indicates the number of panels to the left of the current panel (multiple-panel programs). If the current panel is the leftmost panel, this variable contains the current time.
#HPARM	A65	Output/ Nondisplay	Contains the key for Natural Construct's passive help file for the current program (supplied in CDDIALDA). Place this variable on the map and pass it to the CD-HELPR helproutine.
#DIRECT-COMMAND	A60	Modifiable	Indicates support for direct command processing (supplied in CDGETDCA).
#KD-LINE1/ #KD-LINE2	A79	Output	Displays the names of the actions available to the user or alternate PF-key display formats (supplied in CDDIALDA). Place these variables at the bottom of map(s) above the standard PF-key lines.
#ACTION(*)	A1	Modifiable	Inputs the actions specified for each record.
#ACTION-CV(*)	C	Control Variable	Control variable attached to #ACTION(*).

Changing the Default Window Settings

To change the default settings for a browse-select window, press PF5 to display the Define Windows Parameters window. For a description of this window, see **Changing the Default Window Settings for Generated Modules**, page 86.

Additional INPUT Parameters Panel

The following example shows the third specification panel for Browse-Select models, the Additional INPUT Parameters panel:

```
CUSCMC                      Browse-Select Program                      CUSCMC0
Sep 06                  Additional INPUT Parameters                     3 of 5


            Field Name              Format              Field Prompt
    -------------------------- -------------- ------------------------------
  1 ________________________    _ _____ *     ______________________________
  2 ________________________    _ _____ *     ______________________________
  3 ________________________    _ _____ *     ______________________________
  4 ________________________    _ _____ *     ______________________________
  5 ________________________    _ _____ *     ______________________________
  6 ________________________    _ _____ *     ______________________________
  7 ________________________    _ _____ *     ______________________________
  8 ________________________    _ _____ *     ______________________________




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        optns attr                    left  right main
```

Additional INPUT Parameters Panel for Browse-Select Models

In addition to the key field, you can include up to eight input fields in a browse-select program. These fields are displayed at the bottom of the generated panel and allow the user to display more information. For example, you can create an additional input field called “Detail” (format L) to display additional record details. Users can mark the Detail field to display the details.

Note: Fields specified on the Additional INPUT Parameters panel do not have to be in the Predict file definition.

The fields on this panel are:

Field	Description
Field Name	Name(s) of the additional input field(s).
Format	Natural format and unit length of the input field (for example, A8). This includes the number of positions after the decimal point in numeric input fields.
Field Prompt	Field prompt displayed on the generated browse-select panel. Enclose words you want to display intensified within angle (<>) brackets (or whatever character is set for intensify). To change attribute characters, see **Changing the Default Attribute Parameters**, page 227. If you do not specify a field prompt, Natural Construct creates a default prompt using the name of the input field. The field is displayed at the bottom of the generated panel.

Specifying Options for Additional Input Fields

To specify options for additional input fields, press PF5 (optns) to display the Optional Parameters window:

```
CUSCMCA                      Natural Construct                      CUSCMCA0
Sep 06                     Optional Parameters                        1 of 1

                                                                     Prompt
           Session Parameters          Array     Panel    New Line    OFF
     ------------------------------ ---------- -------- ---------- --------
   1 ______________________________   __  __      __         _         _
   2 ______________________________   __  __      __         _         _
   3 ______________________________   __  __      __         _         _
   4 ______________________________   __  __      __         _         _
   5 ______________________________   __  __      __         _         _
   6 ______________________________   __  __      __         _         _
   7 ______________________________   __  __      __         _         _
   8 ______________________________   __  __      __         _         _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai
```

Optional Parameters Window for Browse-Select Models

The fields in this window are:

Field	Description
Session Parameters	One or more session parameters for the additional input field, such as Attribute Definition (AD) or Edit Mask (EM). For example: `AD=I SG=ON EM='>'X HE='HELPR'`
Array	Range of occurrences of an array variable to place on the panel. If you specify a range with different first and last values, a single prompt precedes all elements in the range. (For multiple prompts, specify each occurrence separately.)
Panel	Number of the panel on which the input prompt is displayed. If a panel number is not specified, the prompt is displayed on all panels.
New Line	If this field is marked, the input field for the prompt begins on a new line.
Prompt OFF	If this field is marked, the default prompt is suppressed.

Changing the Default Attribute Parameters

To change the dynamic attributes for your browse-select program, press PF6 (attr) to display the Dynamic Attributes Parameters window. For a description of this window, see **Changing the Default Attribute Parameters for Generated Modules**, page 87.

#Action Parameters Panel

The following example shows the fourth specification panel for Browse-Select models:

```
CUSLMD                     BROWSE-SELECT Program                     CUSLMD0
Nov 13                       #ACTION Parameters                       4 of 5

Format ..................... _ __ *    Total action lines ........... __
Starting line ............... __        Multiple screen lines ........ _
Starting column ............. 3_        Action Add as PF-key ......... _

Actions Supported
 _ Add        _ Browse     _ Clear      _ Display    _ Modify     _ Next
 _ Purge      _ Copy       _ Recall     _ Replace    _ Select     _ Detail
 _ Former

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        deflt                        left  right main
```

#Action Parameters Panel for Browse-Select Models

Note: For DB2 users, the Former action is disabled on this panel.

Use this panel to define the characteristics of the action/selection field. The fields on this panel are:

Field	Description
Format	Natural length and format of the action/selection field.
Total action lines	Total number of action lines displayed on the generated panel. This number corresponds to the maximum number of database records.
Starting line	Line number on which the action column begins. This number corresponds to the first line containing database information (following the panel and field headings).
Multiple screen lines	If this field is marked, each record requires more than one line. For example: Address: Number-Street City, Province Country, Postal Code

Field	Description (continued)
Starting column	Number of the column in which the action/selection fields are displayed (when not using an external map). This number determines the placement of the action/selection entries.
Action Add as PF-key	If this field is marked, the PF4 key is enabled for the Add action.
Actions Supported	Actions enabled for the generated panel.

Note: In programs generated using the Browse-Select models, the Action field applies to the first row in each record. For information on using user exit code to change this default, see **Multi-line Browse-Select Programs**, page 393.

Renaming Actions

➢ To rename the actions displayed on the #Action Parameters panel:

1 Change the action names in CDACTL.

2 Recompile the CDACTCST module.

Note: CDACTL connects the action names displayed on the #Action Parameters panel to the user-defined names in CDACT.

Adding Records

When generating a browse-select panel, you must decide how users will add records to a file: by entering "A" in the Action field or by pressing an Add PF-key. The following sections describe each of these methods.

Method 1: Entering "A" in the Action Column

To add a record, the user can enter "A" in the action column beside an existing record. This method is only effective when adding records to existing files with one or more records. First, it is impossible to add the first record because no selection column is available when the file is empty. Second, it is not very intuitive to specify the Add action beside an existing record when the record is not used as part of the add processing.

Method 2: Pressing the Add PF-key

To add a record, the user can press an Add PF-key. This method solves both of the problems mentioned in Method 1. Users can add a record to an empty file and have their cursors positioned anywhere on the screen when pressing the Add PF-key. To include this method, you must mark the Add action as PF-key field on the #Action Parameters panel.

Populating Fields with Default Parameters

To populate fields on the # Action Parameters panel with the default parameters, press PF5 (deflt). This function differs, depending on whether you are using a predefined map. For more information, see the following sections.

When Using a Predefined Map

If you are using a predefined map that has an action/selection column, press PF5 (deflt) to automatically enter the format and length of the action/selection column fields, as well as the number of action lines and the screen line number of the first action. The map must contain the #ACTIONS array.

When Not Using a Predefined Map

If you are not using a predefined map, press PF5 (deflt) to display the Build Screen Layout window:

```
CUSLMD                     BROWSE-SELECT Program
Nov 13                      Build Screen Layout                    1 of 1

  Field heading lines ..... 1
  Underline headings ...... Y
  Blank lines after headings ... 1
  Input key lines ............. 1

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
           retrn
```

Build Screen Layout Window for Browse-Select Models

The fields in this window are:

Field	Description
Field heading lines	Number of field heading lines. The default is one line.
Underline headings	Underline option for field headings. If “Y” is displayed (by default), field headings are underlined. If “N” is displayed, field headings are not underlined.

Field	Description (continued)
Blank lines after headings	Number of blank lines between the field headings and the data region. The default is one line.
Input key lines	Number of lines reserved at the bottom of the screen for input keys and additional fields. The default is one for input keys, plus the number of lines for additional input fields that begin on new lines. Do not include the Direct Command line in the calculation of this value.

After defining the layout, press Enter to see how the generated panel will look:

```
Header 1
Header 2

Column heading 1
--------------------

Action line 1
Action line 2
Action line 3
Action line 4
Action line 5
Action line 6
Action line 7
Action line 8
Action line 9
Action line 10
Action line 11
Action line 12
Action line 13
Input line 1
Direct command
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12--
      test  test  test  test  test  test  test  test  test  test  test  test
```

Test Window for Browse-Select Panel

Press Enter again to display the #Action Parameters panel with the required parameters filled in:

```
CUSLMD                     BROWSE-SELECT Program                     CUSLMD0
Nov 13                       #ACTION Parameters                       4 of 5

Format ..................... _ __ *    Total action lines ........... 14
Starting line ................ 7_        Multiple screen lines ........ _
Starting column .............. 3_        Action Add as PF-key ......... _

Actions Supported
 _ Add        _ Browse     _ Clear      _ Display    _ Modify     _ Next
 _ Purge      _ Copy       _ Recall     _ Replace    _ Select     _ Detail
 _ Former









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        deflt                        left  right main
Window tested
```

#Action Parameters Panel With Parameters Filled In

Specific Parameters Panel for the Browse-Select-Helpr Model

The following example shows the fifth specification panel for the Browse-Select-Helpr model, the Specific Parameters panel:

```
CUSCME               Browse-Select-Helpr Helproutine               CUSCME0
Sep 06                      Specific Parameters                     5 of 6

   Parameter Override
   Note: Format and Length defaults to CUSTOMER-NUMBER key
   Field name ......... #PDA-KEY
   Natural format ..... _ _____ *

   Component Parameters
   Field name ......... ____________________________
   Natural format ..... _ _____ *
   Array .............. __ 1 __ 2 __ 3









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit        optns                        left  right main
```

Specific Parameters Panel for the Browse-Select-Helpr Model

Use this panel to override the format and/or length of the passed parameter or pass an additional parameter to the helproutine. The key for the browse-select may differ from the key for the calling program. If the key differs, indicate the format and length of the passed key on this panel. Also indicate the name of any additional helproutine parameter, as well as its format and length.

Note: Browse helproutines normally include the PROCESS-SELECTED-RECORD user exit and assign the value of the #SELECTED-KEY variable to the #PDA-KEY value that is returned.

Use the top portion of this panel to specify the format and length of the help field (if it is different from that of the primary browse key).

Use the bottom portion of this panel to specify additional parameters. If no additional parameter is specified, the generated helproutine only has one parameter (#PDA-KEY), which contains the contents of the input field to which the helproutine is attached. If the helproutine changes the value of #PDA-KEY, the altered value is displayed in the input field when the helproutine returns control to the INPUT statement.

In some cases, it is beneficial to pass an additional parameter to the helproutine. If the helproutine selects an order number, for example, you may want to restrict the selection to orders for a certain customer by specifying the additional parameter on the Specific Parameters panel. This parameter is passed to the helproutine as the first parameter and #PDA-KEY is passed as the second. For information on passing additional parameters, see **Using Parameter Data Areas**, page 117.

If the helproutine uses an additional parameter, a value must be supplied to the helproutine when it is invoked. To do this, enter HE='*routine*',*parameter* in the INPUT statement, where *routine* is the helproutine name and *parameter* is the value passed to the additional parameter. Or you can enter the helproutine name and parameter in the HE field on the Extended Field Editing panel in the Map editor. The value passed must have the same format and length as the additional parameter.

The fields on the Specific Parameters panel are:

Field	Description
Parameter Override	
Field name	Name of the primary key. By default, #PDA-KEY is displayed.
Natural format	Natural format and length of the passed field (if it is different from that of the key being browsed). This format becomes the format for the #PDA-KEY field.

Field	Description (continued)
Component Parameters	
Field name	Name of the additional parameter.
Natural format	Natural format and length of the additional parameter.
Note:	Any valid combination of format, length, and decimal positions under Natural is allowed.
Array	Array dimensions. To declare the field an array, enter the array dimensions in the 1, 2, and 3 fields.

Specific Parameters Panel for the Browse-Select-Subp Model

The following example shows the fifth specification panel for the Browse-Select-Subp model, the Specific Parameters panel:

```
CUSCMF                 Browse-Select-Subp Subprogram                CUSCMF0
Sep 06                      Specific Parameters                      5 of 6

    Parameter Override
    Note: Format and Length defaults to CUSTOMER-NUMBER key
    Field name ......... #PDA-KEY
    Natural format ..... _ _____ *









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit        optns                        left  right main
```

Specific Parameters Panel for the Browse-Select-Subp Model

As with the browse-select helproutine key, the browse-select subprogram key can be different from that of the calling program. If the key differs, specify the format and length of the passed key on this panel.

By default, the Browse-Select-Subp model generates the following parameters and data areas:

Parameter	Description
#PDA-KEY	On entry, this parameter supplies the value to begin browsing from. On exit, this parameter must be assigned the key of the record selected. (The parameter is normally assigned the value of the #SELECTED-KEY variable in the PROCESS-SELECTED-RECORD user exit.) The format and length of this parameter defaults to the format and length of the key being browsed. However, you can change the format and length on the Specific Parameters panel.
CDSELPDA	By default, this parameter data area contains the function specified in the browse-select subprogram. The function is returned to the calling program. (You can extend this data area as required.)
CDPDA-D CDPDA-M CDPDA-P	See **Using Parameter Data Areas**, page 117.
Note:	You can specify additional parameters in the PARAMETER-DATA user exit.

The fields on the Specific Parameters panel for the Browse-Select-Subp model are:

Field	Description
Field name	Name of the primary key. By default, #PDA-KEY is displayed.
Natural format	Natural format and length of the passed field (if it is different from that of the key being browsed). This format becomes the format for the #PDA-KEY field.

Restriction Parameters Panel

The following example shows the fifth specification panel for the Browse-Select model (and the sixth specification panel for the Browse-Select-Helpr and Browse-Select-Subp models), the Restriction Parameters panel:

```
CUSCMG                     Browse-Select Program                     CUSCMG0
Sep 06                     Restriction Parameters                     5 of 5

  Prefix Options
  Restrict browse with prefix .. _

  Number of characters (bytes) . ___
  Number of components ......... __

  Protect prefix ............... _
  Suppress prefix .............. _

  Field name ............... ________________________________________________


  Prefix Helproutine Specifications
     Component       Helproutine    Parameter Name
        1             ________ *    ____________________________________
        2             ________ *    ____________________________________


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit        optns                        left  userX main
```

Restriction Parameters Panel for Browse-Select Models

For a browse-select program, you can limit the browse to records prefixed by a global variable. If the prefix is a department code, for example, you can restrict the browse to only those orders prefixed by a department code. You can set the value of the code as a function of a user ID and store the value in the global data area.

For a browse-select helproutine or subprogram, you can limit the browse by passing the prefix portion of the key. To display only the lines for a particular order, for example, you can pass the Order Number (N6) to the subprogram and enter “N6” in the Natural Format field on the Specific Parameters panel. On the Restriction Parameters panel, mark the Restrict browse with prefix field and enter “6” in the Number of characters field. By default, #PDA-KEY is the field name.

Note: For DL/1 users, when creating a browse-select program for an IMS file that is not at level 1, specify a prefix that includes the key for all segments above the current segment.

The fields on the Restriction Parameters panel are:

Field	Description
Restrict browse with prefix	If this field is marked, the browse is limited to values for which the primary key is prefixed by or equal to the specified value.
Number of characters (bytes)	Number of bytes (of the primary key) to use as the prefix.
Number of components	When using a compound key, the number of key components used as the prefix.
Protect prefix	If this field is marked, the prefix portion of the primary key for the input field is protected. The prefix is displayed but cannot be changed.
Suppress prefix	If this field is marked, the prefix portion of the primary key is not displayed.
Note:	To use the Protect prefix or Suppress prefix option, the primary key must be a superdescriptor, a compound IMS key, or redefined in Predict.
Field name	Name of the field containing the prefix value. When you generate a browse-select helproutine or subprogram, the prefix portion of the key is assumed to be equal to #PDA-KEY (the value of the key passed to the helproutine or subprogram). To override this default, enter a variable name in this field. Define the variable in the LOCAL-DATA user exit and assign a value to the field in the ASSIGN-PREFIX-VALUE user exit. The assigned value (instead of #PDA-KEY) will then be used as the value for the prefix portion of the key.
Component	Number of the corresponding component (“1” indicates the first component; “2” indicates the second, etc.).
Helproutine	Name of the helproutine for the prefix. To attach a helproutine to the prefix of the primary key, enter the name of the helproutine in this field. You can specify a helproutine for each component.
Parameter Name	Parameter for the helproutine for each component. You can define the help parameters in the LOCAL-DATA user exit.
Note:	The Prefix Helproutine Specifications fields are protected when entering parameters for a browse-select helproutine.

User Exits for the Browse-Select Models

The user exits panels for the Browse-Select, Browse-Select-Helpr, and Browse-Select-Subp models are identical and function in the same way. These panels are:

```
CSGSAMPL                   Browse-Select Program                      CSGSM0
Sep 06                           User Exits                           1 of 1

              User Exits             Exists    Sample   Required Conditional
   ------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                           Subprogram
 _  PARAMETER-DATA                            Example                X
 _  HE-PARAMETER-INDEXES                      Example                X
 _  BUILD-REPORT-LOCAL-VARS
 _  LOCAL-DATA                               Subprogram
 _  START-OF-PROGRAM                          Example
 _  SELECT-STATEMENT                         Subprogram              X
 _  AFTER-READ
 _  REJECT-AFTER-MAX-KEY-CHECK                                       X
 _  EXPORT-DATA                             Subprogram              X
 _  WRITE-FIELDS                            Subprogram
 _  TOP-OF-PAGE                                                      X
 _  BEFORE-INPUT                             Example
 _  BEFORE-STANDARD-KEY-CHECK                Example
 _  AFTER-INPUT                              Example
 _  HARDCOPY-EDITS                           Example                 X
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
frwrd help  retrn quit        optns       bkwrd frwrd
```

Browse-Select User Exits Panel

```
CSGSAMPL                   Browse-Select Program                      CSGSM0
Sep 06                           User Exits                           1 of 1

              User Exits             Exists    Sample   Required Conditional
   ------------------------------- -------- ---------- -------- ------------
 _  HARDCOPY-TERMINATING-PROCESS              Example                X
 _  END-OF-PROGRAM                            Example
 _  ASSIGN-PREFIX-VALUE                      Subprogram              X
 _  SET-PF-KEYS                              Example
 _  PROCESS-SELECTED-RECORD                   Example                X
 _  MISCELLANEOUS-SUBROUTINES                 Example
 _
 _
 _
 _
 _
 _
 _
 _
 _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        optns       bkwrd frwrd
```

Browse-Select User Exits Panel

For information about these user exits, see **User Exits for the Generation Models**, page 321. For information about the User Exit editor, see **User Exit Editor**, page 96.

DRIVER MODEL

This chapter describes the Driver model, which generates a module that executes a helproutine or subprogram for testing purposes.

The following topics are covered:

Introduction

The Driver model generates an INPUT statement — you supply the parameters to execute the helproutine or subprogram. The model also generates headings and PF-key names according to the value of *Language.

Note: If Natural Construct does not find SYSERR text for the specified value of *Language at generation time, it uses the English text.

The following example shows a driver program generated using the Driver model:

```
+-------------------------------------------------------------------------------+
| NCDRIVER                    Natural Construct                                 |
| Oc +-------------------------------------------------------------------+  1 |
|    | NCOSELH            ***** ORDER SUBSYSTEM *****                     |    |
| He | Oct 30                  - ORDER SELECTION -                1:15 PM |    |
| En |                                                                    | F1 |
|    | Action Order Number   Order Amount   Order Date                    |    |
|    | ------ ------------ ---------------- ----------                    |    |
+--- |   _          111             1100.00                               | ---+
     |   _          123             1500.00  19940210                     |
     |   _          222             3100.00                               |
     |   _          777             1500.00  19950223                     |
     |   _         1234             1500.00  19930709                     |
     |   _       111112            12210.00  19930318                     |
     | Order Number: ______ Detail _                                      |
     |  Copy       Display    Modify     Purge      (PF5=flip)           |
     |                                                                    |
     |                                                                    |
     |                                                                    |
     +--------------------------------------------------------------------+


NEXT ncdriver                                                  LIB=SAG
```

Sample Output from the Driver Model

To view the specifications for this example, see **Example of Generating a Driver Program**, page 245.

Note: You cannot regenerate modules that were generating using the Driver model.

Handling of X-Arrays

Because X-arrays must be materialized before they can be used in an INPUT statement, the Driver model materializes all X-arrays to one dimension. If other dimensions are required, manual changes must be made.

Parameters for the Driver Model

The Driver model has one specification panel, the Standard Parameters panel. This panel is described in the following section.

Standard Parameters Panel

The following example shows the Standard Parameters panel for the Driver model:

```
CUDRMA                          DRIVER Program                          CUDRMA0
Dec 11                       Standard Parameters                         1 of 1

  Module ............. NCDRIVER
  System ............. SYSCSTDE_____________________________

  Title .............. Driver Program___________
  Description ........ Driver program for ..._____________________________
                       ________________________________________________________
                       ________________________________________________________
                       ________________________________________________________

                                       Source      Object
  Called Module ...... ________ *




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit                                              userX main
```

Standard Parameters Panel for the Driver Model

This panel contains fields common to all models. For a description of the common fields, see **General Model Specifications**, page 167. The Standard Parameters panel for the Driver model also contains the following fields:

Field	Description
Called module	Name of the helproutine or subprogram you want to test using this driver program.
Source	Library containing source code for the specified helproutine or subprogram. When you enter the module name, Natural Construct searches for the module's source code and displays the name of the first library in which it is found. For the driver program to work, the specified module must be in the current library or its steplib chain.

Field	Description (continued)
Object	Library containing object code for the specified helproutine or subprogram. When you enter the module name, Natural Construct searches for the module's object code and displays the name of the first library in which it is found.

User Exits for the Driver Model

The following panel displays the user exits for the Driver model:

```
CSGSAMPL                      NATURAL CONSTRUCT                        CSGSM0
Dec 11                            User Exits                           1 of 1

               User Exit              Exists    Sample   Required Conditional
    -------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                             Subprogram
 _  LOCAL-DATA                                  Example
 _  START-OF-PROGRAM
 _  ADDITIONAL-INITIALIZATIONS                  Example
 _  END-OF-PROGRAM




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                   bkwrd frwrd
```

Driver Model User Exits Panel

For information about these user exits, see **User Exits for the Generation Models**, page 321. For information about the User Exit editor, see **User Exit Editor**, page 96.

Executing a Driver Program

To test a helproutine or subprogram, execute the driver program and specify the required parameters in the Driver Program window.

- If the driver program is calling a helproutine, see **Example of Generating a Driver Program**, page 245.

Note: If the driver program is calling a helproutine with multiple parameters, specify the parameters in the Helproutine Passed Parameters window. After entering the parameters, the Driver Program window is displayed.

- If the driver program is calling a subprogram, enter the parameters required for execution (one prompt is displayed for each parameter) in the Driver Program window. For example, the following window is displayed when the driver program calls the NCOSELS subprogram:

```
TES                          Natural Construct
Dec 11                         Driver Program                        1 of 1

Subprogram ......... NCOSELS

Input Parameters ... SELECTED-FUNCTION ___________ ##COMMAND
_____________________________________________________________ ##MAIN ________
##QUIT _________ ##MSG
_____________________________________________________________________________
##MSG-NR ____0 ##MSG-DATA ____________________________________
_____________________ ____________________________________ ##RETURN-
_ ##ERROR-FIELD ____________________________ ##ERROR-FIELD-INDEX1 ___0
##ERROR-FIELD-INDEX2 ___0 ##ERROR-FIELD-INDEX3 ___0 ##USER _________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help        quit                   bkwrd frwrd       right left
```

Driver Program Window for the NCOSELS Subprogram

The driver program executes the helproutine or subprogram using the specified parameters. For an example of generating and executing a driver program, see the following section.

Example of Generating a Driver Program

➢ To generate the sample driver program shown in **Introduction**, page 240:

1 Type the following on the Generation main menu:
   – "M" in the Function field.
   – "NCDRIVER" in the Module name field.
   – "Driver" in the Model field.

2 Press Enter.
The Standard Parameters panel is displayed.

3 Type "NCOSELH" in the Module name field.

4 Press Enter>
Natural Construct supplies the names of the libraries containing the source and object code for the NCOSELH helproutine.

5 Press Enter again>
You are returned to the Generation main menu.

6 Enter "G" in the Function field.
A confirmation message is displayed, indicating that the driver program has been generated successfully.

7 Enter "T" in the Function field.
The Driver Program window is displayed:

```
NCDRIVER                   Natural Construct
Dec 11                       Driver Program                        1 of 1

Helproutine ........ NCOSELH  #PDA-KEY ______0
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help        quit                   bkwrd frwrd       right left
```

Driver Program Window for the NCOSELH Helproutine

8 Enter “?” in the #PDA-KEY input field.
The NCOSELH helproutine window is displayed:

```
NCOSELH             ***** ORDER SUBSYSTEM *****
Dec 11                  - ORDER SELECTION -                3:33 PM

Action Order Number   Order Amount   Order Date
------ ------------ ---------------- ----------
  __          111             2717.00  19991018
  __          122            10880.00  19990812
  __          123            17760.00  19990105
  __          777             1650.00  19950223
  __          991             2500.00  19981217
  __         1111             1500.00  19990219
Order Number: ______ Detail _
 Copy       Display    Modify     Purge      (PF5=flip)
```

NCOSELH Helproutine Window

You can now test the helproutine.

9 Press PF2 to return to the Driver Program window.

EXTENDABLE-INPUT MODEL

This chapter describes the Extendable-Input model, which generates a subprogram that displays and/or updates extended information for a field on a panel.

The following topics are covered:

Parameters for the Extendable-Input Model

The Extendable-Input model has two specification panels: Standard Parameters and Additional Parameters. These panels are described in the following sections.

Standard Parameters Panel

The following example shows the first specification panel for the Extendable-Input model, the Standard Parameters panel:

```
CUETMA                  Extendable-Input Subprogram                  CUETMA0
Sep 10                      Standard Parameters                       1 of 2

 Module ............. EXTEND__
 System ............. CST341S_____________________________

 Title ............. Extendable Input ..._____
 Description ........ This subprogram displays ......_____________________
                      ____________________________________________________
                      ____________________________________________________
                      ____________________________________________________

 Message numbers .... _




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                                          right main
```

Standard Parameters Panel for the Extendable-Input Model

This panel is similar for all models. For a description of the Standard Parameters panel, see **General Model Specifications**, page 167.

Additional Parameters Panel

The following example shows the second specification panel for the Extendable-Input model, the Additional Parameters panel:

```
CUETMB                  Extendable-Input Subprogram                  CUETMB0
Oct 29                     Additional Parameters                      2 of 2

  Data area ............... ________ *
  Predict view ............ ____________________________ *

  Input using map ......... ________ *
  Field name .............. ____________________________ *
  Scroll lines per screen .. ___

  Window support ........... _









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        windw                        left  userX main
```

Additional Parameters Panel for the Extendable-Input Model

The fields on this panel are:

Field	Description
Data area	Name of the data area used by the extendable-input subprogram.
Predict view	Name of the Predict file (view) used by the subprogram.
Input using map	Name of the map used by the subprogram.
Field name	Name of the field for which the extendable-input panel is displayed.
Scroll lines per screen	Maximum number of scrollable lines per screen on the generated panel.
Window support	If this field is marked, the output from the generated subprogram is displayed in a window instead of on a panel. To change the window defaults, press PF5 (windw). For more information, see **Changing the Default Window Settings for Generated Modules**, page 86.

User Exits for the Extendable-Input Model

The following panel displays the user exits for the Extendable-Input model:

```
CSGSAMPL                    Natural Construct                        CSGSM0
May 15                Extendable-Input User Exits                    1 of 1

            User Exit                 Exists    Sample   Required Conditional
    ------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                            Subprogram
 _  PARAMETER-DATA
 _  LOCAL-DATA
 _  START-OF-PROGRAM
 _  BEFORE-INPUT
 _  UPDATE-EDITS                              Subprogram
 _  END-OF-PROGRAM
 _
 _
 _
 _
 _
 _
 _
 _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                   frwrd bkwrd
```

Extendable-Input User Exits Panel

For information about these user exits, see **User Exits for the Generation Models**, page 321. For information about the User Exit editor, see **User Exit Editor**, page 96.

MAINT MODEL

This chapter describes the Maint model, which generates a program that maintains a file using a unique key and, optionally, a related secondary file. The Maint model generates the code necessary to scroll through the MU/PE fields of a primary file or the records of a secondary file.

The following topics are covered:

Introduction

Natural Construct provides two alternatives to generate a maintenance process:

1 Use the Object models: Object-Maint-Dialog, Object-Maint-Dialog-Subp, and Object-Maint-Subp (which also generates the object PDA and restricted PDA). These models provide all the functionality needed for application development at the production level. The separation of dialog and data makes future changes to the maintenance process easier to implement.

2 Use the Maint model. This model creates fast prototypes or simple maintenance processes that are temporary. The process is faster to implement because you need only create one or two Natural objects: the maintenance program generated by the Maint model and, optionally, a map. Conversely, because dialog and data are combined, this maintenance process is not as easy to modify as the process generated by the Object models.

The differences between the two models include:

Object Models	Maint Model
Maintain up to four levels of files: primary, secondary, tertiary, and quaternary.	Maintains one or two levels of files: primary and secondary.
Support multiple scrolling regions.	Supports one scrolling region.
Support a link between scrolling regions on multiple panels.	Does not support a link between scrolling regions on multiple panels.
Provide automatic cursor repositioning after an error.	Does not provide automatic cursor repositioning after an error. (This functionality is available in user exits.)

The following example shows a maintenance program generated using the Maint model:

```
NCTFRPRD                 ***** TABLE SUBSYSTEM *****                 NCTFPRD1
Oct 30                     - PRODUCT MAINTENANCE -                   10:45 AM

*Action (A,B,C,D,M,N,P,R): _ Product ID: 256733

Description  : OATS AND BARLEY CEREAL_____________________________
Reorder Point: 22_____
Unit Cost    : 20.00_____

Manufacturer's Address   Street     : 45 CRESENT STREET___
                         City       : EDMONTON____________
                         Province   : ALBERTA_____________
                         Postal Code: P9E 8E4









Direct Command: _____________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
confm help  retrn quit        flip  pref                              main
Next PRODUCT ID displayed
```

Sample Output from the Maint Model

Note: Notice that PF5 (flip) and PF6 (pref) are available. For a description of these special function keys, see **Defining PF-Keys for Generated Applications**, page 122.

By default, a maintenance program generated using the Maint model prompts users to press the Enter key to confirm a Purge action. If you specify a confirmation key other than Enter, the program will force confirmation of Add, Modify, and Purge actions. For a description of how to change the confirmation key, see **Confirmation Key Setup**, page 132.

To view the specifications for the Maint model example, refer to the NCTFRPRD program in the Natural Construct demo system.

Parameters for the Maint Model

There are three specification panels for the Maint model: Standard Parameters, Specific Parameters, and Secondary File Parameters. These panels are described in the following sections.

Standard Parameters Panel

The following example shows the first specification panel for the Maint model, the Standard Parameters panel:

```
CUFMMA                          Maint Program                          CU--MA0
Sep 10                       Standard Parameters                        1 of 3

 Module ............. MNT_____
 System ............. CST341S_____________________________
 Global data area ... CDGDA___ *
 With block ......... _____________________________

 Title ............... Maintain..._______________
 Description ........ This program is used to maintain the...._________________
                      ____________________________________________________________
                      ____________________________________________________________
                      ____________________________________________________________

 First header ....... _____________________________________________
 Second header ...... _____________________________________________________

 Command ............ _
 Message numbers .... _
 Password ........... _

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                                          right main
```

Standard Parameters Panel for the Maint Model

This panel is similar for all models. For a description of the Standard Parameters panel, see **General Model Specifications**, page 167.

Additional Parameters Panel

The following example shows the second specification panel for the Maint model, the Additional Parameters panel:

```
CUFMMB                          MAINT Program                          CUFMMB0
Nov 13                      Additional Parameters                       2 of 3

 Predict view ............. _____________________________________ *
   Natural (DDM) .......... _____________________________________
 Primary key .............. _____________________________________ *
 Record description ....... ________________
 Log file name ............ _____________________________________ *
   Natural (DDM) .......... _____________________________________
 Input using map .......... ________ *
 Minimum key value ........ ____________________________________________________
 Maximum key value ........ ____________________________________________________
 Single prompt ............ _
 Multiple prompts ......... _
 Mark cursor field ........ ____________________________________________________

 #ACTION field length ..... 1  Add ...... X  Browse ... ________ *
                               Clear .... X  Display .. X
                               Modify ... X  Next ..... X
                               Purge .... X  Recall ... _  Former ... _
 Push-button support ...... _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                                      left  right main
```

Additional Parameters Panel for the Maint Model

The fields on this panel are:

Field	Description
Predict view	Name of the Predict view. A file definition for the specified file must exist in Predict.
Natural (DDM)	Name of the primary file (the DDM). All fields in the primary file must be in the specified DDM. (Since this field defaults to the primary file name, only enter a DDM name if it is different from the primary file name.)
Primary key	Name of the key in Predict for the primary file. This key becomes the primary key for accessing the Predict view for the maintenance program. It may be a descriptor, superdescriptor, or subdescriptor. If the specified key does not exist in the corresponding file, a message is displayed indicating that the key field was not found.
Note:	Maintenance programs generated by Natural Construct are only intended for use with views having a unique primary key. However, the primary key does not need to be defined with the UQ option.

Field	Description (continued)
Note:	You can use a combination of fields to define the primary key for a table. To do this, add a superdescriptor in Predict that defines the combination. Specify this superdescriptor as the primary key name.
Note:	You do not need to mark the primary key as a descriptor, as long as the field is a valid UDF redefinition of a key field from the master segment.
Record description	Description used in messages. By default, "Record" is displayed and messages are displayed in the form: "Record not found" or "Record displayed."
Log file name	Name of the view used to perform update logging. All fields in the primary file (with matching fields in the log file) are written to the log records. Other fields, such as LOG-DATE and LOG-USER, must also appear in the log view. For information, see **Other Log Fields**, page 620.
Natural (DDM)	Name of the DDM (data definition module) used for logging. If the log file DDM is different from the update log file, enter the name of the DDM used for logging. The update log file must be a view of the log file DDM.
Input using map	Name of an external map for the input panel. To use an external map, enter a map name in this field. To help you create maintenance maps, Natural Construct supplies the CDLAYFM1 and CDLAYFM2 layout maps. If the generated program does not maintain MU/PE fields or a secondary file, use CDLAYFM1. If the generated program does maintain MU/PE fields or a secondary file, use CDLAYFM2. This layout map contains all the variables needed to control scrolling. The #LINE-CV variable on the CDLAYFM2 map is only applicable to a secondary file. If you do not specify a map name, Natural Construct generates an INPUT statement and designs the panel with all prompts and fields aligned on the left. MU/PE fields are not included. The fields are in the same order as the fields in the Predict field definition.
Note:	If you are generating a program that maintains both a primary and secondary file, use an external map or a primary file containing MU/PE fields (for information, see **Secondary File Parameters Panel**, page 261).
Note:	For a list of variables you can use, see **Variables You Can Use with a Maint Model Map**, page 258.

Field	Description (continued)
Minimum key value	Minimum key for the range of records. To limit the range of records the program can access, enter a starting (minimum) value.
Maximum key value	Maximum key for the range of records. To limit the range of records the program can access, enter an ending (maximum) value. The minimum and maximum values form a "window" in which the program looks. Users cannot access keys outside this range.
Note:	If the key is defined as numeric, the minimum and maximum key values must be numeric and the specified maximum key must be greater than or equal to the specified minimum key. If the key values are variable, you can assign them in the START-OF-PROGRAM user exit.
Single prompt	If this field is marked, a single prompt is displayed for all fields (for example, Date:____ __ __). This option applies if the key is a superdescriptor or redefined in Predict.
Multiple prompts	If this field is marked, one prompt is displayed for each field (for example, Year:____ Month:__ Day:__). This option applies if the key is a superdescriptor or redefined in Predict. For more information, see **Redefined Fields**, page 610.
Note:	If the key is redefined and you want to have prompting for some redefined fields — but no prompting for other fields — you can use a map to display the prompts the way you want them to appear.
Mark cursor field	Name of the field marked by default when an error occurs. To avoid ambiguity, fully qualify the field name with a structure name.
Action Parameters	Action parameters. By default, all actions except Recall and Former are marked. (Recall is not a default action because values on the secondary file are not recalled.) To select an action, mark the corresponding field. At least one action must be selected.
#ACTION field length	Length (1, 2, or 3) of the field used for #ACTION names. For example, to use "DI" for Display, type "2" in this field.
Add	Adds records to the file.
Clear	Clears specified record values from the panel.
Modify	Modifies specified records.

Field	Description (continued)
Purge	Purges specified records from the file.
Browse	Browses records in the file. To include the Browse action, specify a subprogram name in this field. (Use the Browse-Subp or Browse-Select-Subp model to generate the browse subprogram.)
Display	Displays specified records.
Next	Displays the next record in the file.
Recall	Recalls the values for the last record cleared from the panel following a Display, Modify, or Purge action.
Former	Displays the contents of the record having the next lower primary key value from the current key value. If no lower value exists, the Start of Data reached message is displayed.
Note:	To add user-defined actions, define the SELECT-ADDITIONAL-ACTIONS and ADDITIONAL-ACTIONS-PROCESSING user exits. For more information, see **Add an Action**, page 134.
Push-button support	If this field is marked, actions are presented as cursor-sensitive push buttons. Users can press the Tab key to move from action to action. For more information about implementing actions as push buttons, see the description of the #KD-LINES(*) variable in **Variables You Can Use with a Maint Model Map**, page 258.

Variables You Can Use with a Maint Model Map

You can use the following variables with a Maint model layout map:

Variable	Format	Definition	Description
#PROGRAM	A8	Output	Contains the name of the program that invoked the map.
#HEADER1	A60	Output	Contains the first heading for the maintenance program.
#HEADER2	A58	Output	Contains the second heading for the maintenance program.

Variable	Format	Definition	Description (continued)
#LEFT-PROMPT	A9	Output	Indicates the number of panels to the left of the current panel. If the current panel is the leftmost panel, this variable contains the date.
#RIGHT-PROMPT	A9	Output	Indicates the number of panels to the right of the current panel. If the current panel is the rightmost panel, this variable contains the time.
#HPARM	A65	Output/ Nondisplay	Contains the key to the passive help file for the current program (system name concatenated with program name). You can place this variable on the map and pass it to the CD-HELPR helproutine.
#VAL-ACT	A18	Output	Contains the list of available actions.
#VAL-ACT2	A79	Output	Contains a second list of available actions (to accommodate long lists).
#ACTION	A1	Modifiable	Contains the action applied to the current record.
#DIRECT-COMMAND	A60	Modifiable	Indicates support for direct command processing in the generated module.
#KD-LINE1/ #KD-LINE2/ #KD-LINES(*)	A79	Output	Contain the names of actions available to the user (Add, Modify, Purge, etc.) or alternate PF-key display formats. Place these variables (supplied in CDDIALDA) either at the top of the map or at the bottom of the map above the standard PF-key lines. To use the push button feature, place these variables on the map(s) with the #KD-LINES-CV control variable and the '00V(NP'02 dynamic attribute. To display the push buttons in red, use the '00VRE(NP'02 dynamic attribute. **Note:** The CDLAYFM3 layout map supports push buttons.

Variable	Format	Definition	Description (continued)
#BKWRD-LAB1 #BKWRD-LAB2 #BKWRD-LAB3 #BKWRD-LAB4 #FRWRD-LAB1 #FRWRD-LAB2 #FRWRD-LAB3 #FRWRD-LAB4	A2	Output	Control backward/forward scrolling (supplied in CDKEYLDA). Place these variables on the map(s) next to the fields where you want to enable push buttons for backward and forward scrolling.
		Note:	Include reverse video display among the push button attributes.
#LIN	P3	Output	Contains a single dimension array to display the current index value beside a scroll region (sequential numbers starting from 1). The occurrences of this array match the highest upper bounds value specified for any scroll region.
#TOP-LINE	P3	Modifiable	Indicates the index of the first displayed occurrence of the fields in the scroll region. You can change the value of this field to reposition the scroll region to the specified occurrence.
#PANEL	N7		Determines which panel is displayed when PF10 (left) or PF11 (right) is pressed. For programs containing more than one panel, subtract 1 from #PANEL for PF10 and add 1 to #PANEL for PF11.
#LINE-CV(*)	C		Control variable used for secondary file fields.

Indexing Fields on a Map

The #LINE variable indexes fields that are scrolled using PF7 and PF8 (the standard forward and backward keys). The upper bounds value for MU/PE fields should match the MAX-OCCURS value for the field as defined in Predict.

Secondary file fields are added to the map as an array. The upper bounds value for these fields should match the value specified in the Scrollable records field on the Secondary File Parameters panel.

The occurrence value specified for all scrolling fields should match the value in the Screen records field on the Secondary File Parameters panel.

If a map is created in the Map editor, you can use the V command to select fields from the database views of the file. If you do so, you must temporarily alter the name of the secondary file fields so you can define them as arrays.

All secondary file fields must be defined with the #LINE-CV(*) control variable. This variable allows the program to only update modified secondary file records.

Secondary File Parameters Panel

The following example shows the third specification panel, the Secondary File Parameters panel:

```
CUFMMC                          Maint Program                          CUFMMC0
Sep 10                   Secondary File Parameters                      3 of 3

  Horizontal panels ........ 1
  Scrollable records ....... ___
  Scroll lines per screen .. __
  Input using map .......... ________ *

  Secondary File Parameters
  Secondary view ........... ____________________________________ *
    Natural (DDM) .......... ____________________________________
  Secondary key ............ ____________________________________ *

  Related Key Parameters
  Related keys must match .. _
  Primary key as prefix .... _

  Line Number as Suffix                Redefine or Super. as Suffix
  Natural format ........... _ __ *    Force uniqueness ......... _
  Remove empty lines ....... _         Allow duplicates ......... _
  Save empty lines ......... _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit                                      left  userX main
```

Secondary File Parameters Panel for the Maint Model

On this panel, specify secondary parameters if the program maintains two files, periodic groups (PEs), multiple-valued fields (MUs), or uses more than one panel of input data.

The fields on this panel are:

Field	Description
Horizontal panels	Number of panels required to specify all data in the view. By default, "1" is displayed. (The view may involve either one or two files.) The #PANEL value ranges from 1 to the number specified in this field. This field is used in conjunction with the Input using map field.
Note:	If you specify more than one panel, PF10 (left) and PF11 (right) are activated in the generated program to allow the user to move from panel to panel.
Scrollable records	If you specify a secondary file, enter the maximum number of secondary file records that can be read or saved. If scrolling MU/PE fields in the primary file is supported, this value represents the highest value that may be scrolled. (This value does not affect the number of MU/PE occurrences obtained; the MAX-OCCURS value in Predict determines this number.)
Scroll lines per screen	Number of MU/PE elements or secondary file records displayed on the panel at one time.
Input using map	If you specified a map in the Input using map field on the Additional Parameters panel, the map name is displayed. The field is repeated on this panel in case you want to change the map name. When you specify multiple panels, Natural Construct scans the specified map names for the asterisk (*) character and replaces it with the current panel number. For example, if you specify a TSTMAP* map name and "2" in the Total maintenance panels field, TSTMAP1 and TSTMAP2 are used. The left/right scrolling mechanism (generated into the program) allows windowing between these two maps, starting at TSTMAP1.
Note:	To specify any option on the Secondary File Parameters panel, specify a map name.
Secondary view	Name of the Predict view that is coupled with the primary file for the maintenance program. A file definition for the file must exist in Predict.
Natural (DDM)	Name of the DDM (data definition module) for the secondary file. All fields in the secondary file must be in this DDM. If you do not specify a secondary file DDM, this field defaults to the value in the Secondary view name field.

Field	Description (continued)
Secondary key	Key in the secondary file that is related to the key in the primary file. The key can be a descriptor, superdescriptor, or subdescriptor.
Related keys must match	If this field is marked, the secondary file key must be identical to the primary file key.
Primary key as prefix	If this field is marked, the key of the primary file is a prefix of the secondary file key (secondary file records are always displayed in the secondary key order). If the primary file key is a prefix of the secondary file key, specify how the relationship between the two files is established by using either the Line Number as Suffix options or the Redefine or Super as Suffix options. The biggest difference between the two options is that for the Line Number as Suffix options, you do not need to enter the suffix value for the secondary key. Because the suffix value is a line number, the suffix is determined during each update session within the generated program. For all four options, the secondary file key can be either a descriptor or a superdescriptor. **Note:** The secondary key suffix may consist of one or more distinct fields that determine the sort sequence of the secondary file records. If this is the case, define a superdescriptor in Predict containing the fields that relate the secondary file to the primary file, followed by the fields that determine the sort order of the secondary records.
Natural format	Natural format and length of the line number (N 4, I 2, for example). The generated program assumes the secondary file key is made up of the primary file key value, plus a line number. The value of the suffix can be displayed on the panel, but cannot be modified.
Remove empty lines	If this field is marked, the suffix components of the secondary file keys are renumbered (starting at 1) after an occurrence of the view is saved.
Save empty lines	If this field is marked, the suffix components of the secondary file keys are not renumbered after an occurrence of the view is saved.

Field	Description (continued)
Force uniqueness	If the secondary key suffix is more than a line number, mark this field to use the redefinition of the secondary key field in Predict to determine the suffix. The secondary key suffixes for each secondary file record should be modifiable when displayed on the map. The generated program also ensures a unique secondary file key. If the secondary key is a superdescriptor, place the trailing fields (beyond the primary key length) on the map.
Allow duplicates	If this field is marked, the generated program does not check the secondary key for duplicates.

User Exits for the Maint Model

The following examples show the Maint model User Exits panel:

```
CSGSAMPL                    Natural Construct                        CSGSM0
May 30                       Maint User Exits                        1 of 1

           User Exit                Exists    Sample   Required Conditional
   ------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                           Subprogram
 _  EXTENDED-RI-VIEWS
 _  LOCAL-DATA
 _  START-OF-PROGRAM                          Example
 _  BEFORE-INPUT                             Example
 _  REINPUT-SCREEN                           Subprogram              X
 _  AFTER-INPUT                              Example
 _  AFTER-SCREEN-CLEAR                       Example
 _  END-OF-PROGRAM
 _  AFTER-READ                               Example                 X
 _  SELECT-ADDITIONAL-ACTIONS                Example
 _  SET-PF-KEYS                              Example
 _  ADDITIONAL-ACTIONS-PROCESSING            Example
 _  BEFORE-ET                                Example                 X
 _  UPDATE-EDITS                             Subprogram
 _  EXTENDED-RI-CHECKS                       Subprogram
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
frwrd help  retrn quit                   bkwrd frwrd
```

Maint Model User Exits Panel

```
CSGSAMPL                    Natural Construct                        CSGSM0
May 30                       Maint User Exits                        1 of 1

           User Exit                Exists    Sample   Required Conditional
   ------------------------------- -------- ---------- -------- ------------
 _  MISCELLANEOUS-SUBROUTINES                Example
 _
 _
 _
 _
 _
 _
 _
 _
 _
 _
 _
 _
 _
 _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                   bkwrd frwrd
```

Maint Model User Exits Panel

For information about these user exits, see **User Exits for the Generation Models**, page 321. For information about the User Exit editor, see **User Exit Editor**, page 96.

MAP MODEL

This chapter describes the Map model, which facilitates the design and generation of maps and does not require the Natural map editor.

The following topics are covered:

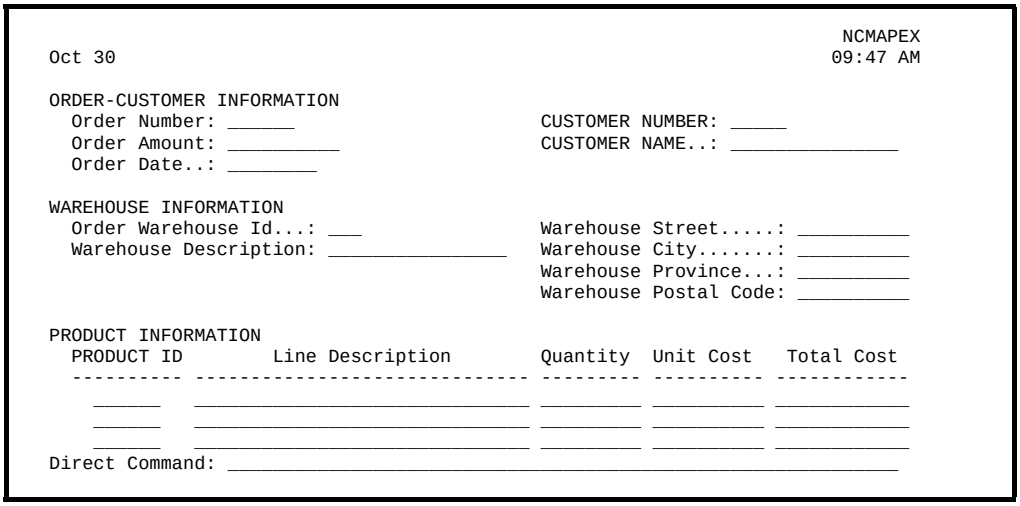
Introduction

Using the Map model, you can:

- Arrange sectors and fields, view the results, and easily make changes
- Maintain a consistent user interface throughout your application
- Modify field prompts and select fields to be included on the map

Note: The Map model also supports Predict automatic processing rules. For information, see **Verification Rules**, page 614.

The following example shows a map generated with the Map model:



Sample Output from the Map Model

To view the specifications for this example, refer to the NCMAPEX map in the Natural Construct demo system.

Designing a Map

This section explains, in general terms, how to design a map. For descriptions of the Standard Parameters and Field Layout Parameters panels, see **Standard Parameters Panel**, page 272, and **Field Layout Parameters Panel**, page 277.

Defining Sectors

Using the Map model, you can design a map by dividing it into sectors and then placing fields in the sectors. Sectors allow users to organize logically-related fields into distinct groups so they can easily perceive the relationships between fields on the generated panel.

When dividing a map into sectors, specify the number of sectors and their positions relative to each other. The Standard Parameters panel contains the Activate Sectors table, where you mark the relative locations of the sectors to be included. For example, marking the field at the intersection of column 1 and line 1 defines a sector in the upper lefthand area of the available map space. The sector is identified by its co-ordinates (11) on the Active Sectors table. Marking the intersection of line 1 and column 2 places a sector at the same height but to the right of sector 11.

The following example shows three defined sectors:

```
CUMPMA                          Map Model                          CUMPMA0
Jun 13                     Standard Parameters                      1 of 3

 Module      ........ TEST____  Map layout ......... ________ *

             Activate Sectors                 Active Sectors
                  1   2   3   4   5   6      ------------------------
             1    X   X   _   _   _   _       11  12
             2    _   _   _   _   _   _
             3    _   _   _   _   _   _
             4    _   _   _   _   _   _
             5    _   _   _   _   _   _
             6    X   _   _   _   _   _       61
                                              ------------------------
```

Section of Standard Parameters Panel Showing Active Sectors

The Active Sectors table shows the sectors that have been activated.

A sector's size and precise location are determined by the number and size of the fields placed in it. Continuing with the preceding example, sector 61 is not located at the bottom of the map, but rather under the sectors assigned to line 1.

The following example shows the locations of sectors 11, 12, and 61 when fields have been assigned to them:

```
L001   ---10---+----20---+----30---+----40---+----50---+----60---+----70---+---

   SECTOR 11::XXXXXXXXXX   SECTOR 12::XXXXXXXXXX

   SECTOR 61::XXXXXXXXXX









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn                        bkwrd frwrd       left  right
```

Test Map Showing Sector Locations

Each map can contain up to 36 sectors. However, only the activated sectors are displayed on the generated panel.

Scroll Sectors

By adding a scroll sector, you can make economical use of the screen space. A scroll sector displays a segment of a file and allows users to scroll through the records.

The Map model supports horizontal and vertical scroll sectors. For information, see **Field Layout Parameters Panel**, page 277.

Assigning Fields to a Sector

When you assign fields to a sector on a map, you can either:

- Define your own fields

  or

- Select fields from Predict views, local data areas (LDAs), or parameter data areas (PDAs)

➢ To assign fields to a sector:

1 Specify the views, LDAs, and/or PDAs to be used.

2 Select all or some fields from the views and/or data areas.

3 Assign the fields to active sectors.

4 Test the map.

5 Optionally, reorder the fields and sectors on the map.

The following sections describe each of the specification panels for the Map model and how they relate to these steps.

Parameters for the Map Model

The Map model has two specification panels: Standard Parameters and Field Layout. These panels are described in the following sections.

Standard Parameters Panel

The following example shows the first specification panel for the Map model:

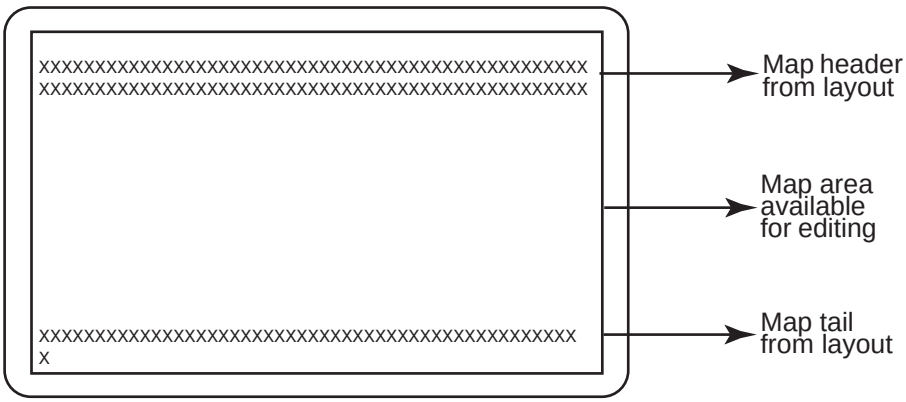
```
CUMPMA                            Map Model                            CUMPMA0
Sep 10                      Standard Parameters                         1 of 2

 Module ..... ________  Map layout ......... ________ *

            Activate Sectors                 Active Sectors
                  1   2   3   4   5   6      ------------------------
            1     _   _   _   _   _   _       11  12  13  14  15  16
            2     _   _   _   _   _   _       21  22  23  24  25  26
            3     _   _   _   _   _   _       31  32  33  34  35  36
            4     _   _   _   _   _   _       41  42  43  44  45  46
            5     _   _   _   _   _   _       51  52  53  54  55  56
            6     _   _   _   _   _   _       61  62  63  64  65  66
                                               ------------------------
 Qualifier  Predict Views or Data Areas      User Views or Data Areas
      1     ____________________________________ ____________________________________
      2     ____________________________________ ____________________________________
      3     ____________________________________ ____________________________________
      4     ____________________________________ ____________________________________
      5     ____________________________________ ____________________________________
      6     ____________________________________ ____________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        optns space             repos left  right main
```

Standard Parameters Panel for the Map Model

On this panel, you can:

- Activate the sectors you want to use on the map
- Specify a view, LDA, or PDA from which map parameters are obtained
- Display a window to change the optional map settings (PF5)
- Display a window to specify the amount of blank space between columns and lines (PF6)
- Display a window to reposition sectors on the map (PF9)

The fields on this panel are:

Field	Description
Module	Name of the map being created. The name must be alphanumeric, cannot exceed eight characters in length, and cannot contain blanks.
Map layout	Name of the map layout used to initialize the map (use the CDLAYMP1 supplied layout map with the Map/Object-Maint-Dialog model combination). All fields in the specified layout are embedded on the generated map and all basic map settings override those on the map profile. The following example shows the available area in a map with an embedded layout:  The two lines at the top of the layout display the map header. The line at the bottom forms the tail of the map. The area between the map header and tail is the available map area to display information.
Activate Sectors	Sectors you want to activate (make active). Only active sectors are displayed on the generated map. Sectors are identified by their co-ordinates on the Active Sectors table. For example, if you mark the intersection of line 1 and column 3, “13” is displayed on the Active Sectors table.
Active Sectors	Active sectors. After marking the sectors you want to make active (the Activate Sectors fields), the active sectors are displayed for reference.

Field	Description (continued)
Qualifier Predict Views or Data Areas	Up to six different qualifiers from which to obtain field information. The qualifier is either a view, a local data area (LDA), a parameter data area (PDA), or a structure. Natural Construct uses this qualifier as the field prefix.
User Views or Data Areas	Name of a user view or data area. To use a view name other than the Predict view name, specify the name on the appropriate line.

Setting Map Options

To set map options, press PF5 (optns) on the Standard Parameters panel. The Optional Parameters window is displayed:

```
CUMPMAB                 Natural Construct                 CUMPMAB0
Jun 18                 Optional Parameters                   1 of 1

  Profile ............ SYSPROF_    Standard keys ...... Y
  Page size .......... 23_         Line size .......... 80_
  ISA standard ....... X           SAA standard ....... _
  Intensify inputs ... X           Intensify .......... _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF
     help  retrn quit
```

Optional Parameters Window

The fields in this window are:

Field	Description
Profile	Name of the profile containing the basic settings for the generated map. By default, SYSPROF is displayed.
Note:	Values specified in the Page size, Line size, or Standard keys fields on the map profile override the values specified on the map layout. All basic map settings specified for the map layout override the values specified on the map profile.
Standard keys	Indicates whether lines are reserved for the function keys. If "Y" is displayed, the last two lines on the map are reserved for the function keys. If "N" is displayed, all lines are used for the generated map and the function keys are not displayed.
Note:	If the layout specified in the Map layout field has standard keys, this field must be "Y".

Field	Description (continued)
Page size	Number of lines used on the generated map. The default is 23.
Line size	Number of columns used on the generated map. The default is 80.
ISA standard	If this field is marked (default), map prompts follow ISA (Integrated Systems Architecture) format. For example: `Prompt...........: _`

SAA standard	If this field is marked, map prompts follow SAA (Systems Application Architecture) format. For example:
	`Prompt . . . . .   _ (for a modifiable field)`
	`Prompt . . . . :   _ (for a display-only field)`

Intensify inputs	If this field is marked (default), map input fields are displayed as intensified.
Intensify	If this field is marked, map field prompts are displayed as intensified.

Defining Sector Headings and Spacing Options

To define sector headings or set spacing options for columns and lines, press PF6 (space) on the Standard Parameters panel. The Spacing Parameters window is displayed:

```
CUMPMAA                  Natural Construct                  CUMPMAA0
Jun 18                   Spacing Parameters                   1 of 1

   Sector Column Spacing             Active Sectors
     1   2   3   4   5   6          ------------------------
    1   2_  2_  2_  2_  2_  2_          11  12
    2   2_  2_  2_  2_  2_  2_          21  22
    3   2_  2_  2_  2_  2_  2_          31  32
    4   2_  2_  2_  2_  2_  2_          41  42
    5   2_  2_  2_  2_  2_  2_
    6   2_  2_  2_  2_  2_  2_
                                      ------------------------
   Sector Line Spacing & Headings
  1   1_  ____________________________________________________
  2   1_  ____________________________________________________
  3   1_  ____________________________________________________
  4   1_  ____________________________________________________
  5   1_  ____________________________________________________
  6   1_  ____________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF
      help  retrn quit
```

Spacing Parameters Window

The fields in this window are:

Field	Description
Sector Column Spacing	Number of empty spaces displayed to the left of the corresponding sector. The default is 2.
Active Sectors	Active sectors on the map.
Sector Line Spacing & Headings	Line numbers of the active sectors, the number of blank lines before the sector heading or first field (if it does not have a heading), and the headings for each sector. (Sector headings begin on the left side of the generated map.)

Repositioning Sectors on the Map

To change the positions of the sectors on the map, press PF9 (repos) on the Standard Parameters panel. The Reposition Parameters window is displayed:

```
CUMPMAC                 Natural Construct                 CUMPMAC0
Jun 18                Reposition Parameters                1 of 1

     Reposition Activated Sectors     Active Sectors
           1   2   3   4   5   6    ------------------------
       1  1   1   2                     11  12
       2  2   1   2                     21  22
       3  3   1   2                     31  32
       4  4   1   2                     41  42
          5
          6
                                       ------------------------
 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF
       help  retrn quit                                 reord
```

Reposition Parameters Window

The fields in this window are:

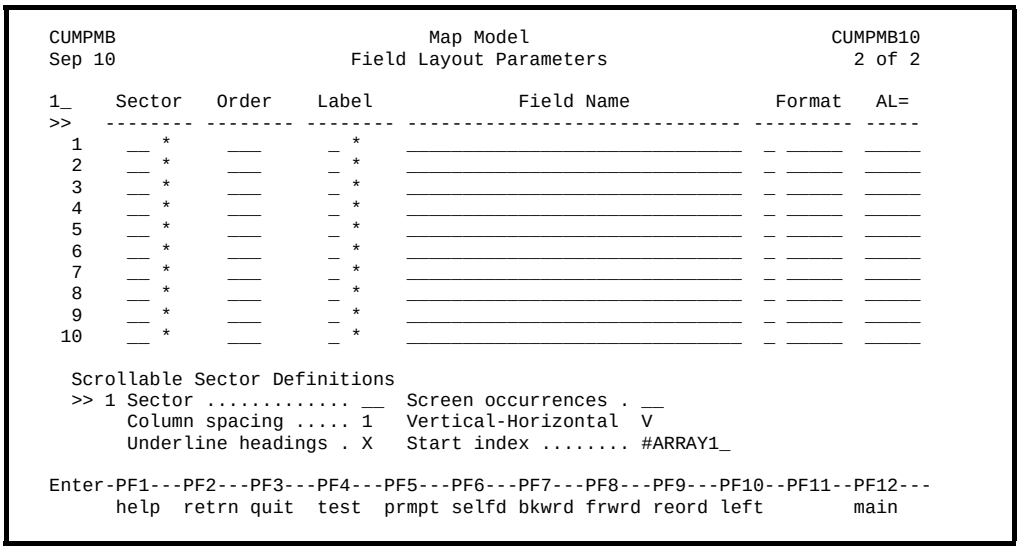
Field	Description
Reposition Activated Sectors	To move a sector (and its contents) to another location, enter its current identifier in the new position. The changes are displayed on the Active Sectors table. To test the map, see **Testing the Map**, page 280; to renumber the sectors, see the following section.
Active Sectors	Displays the sectors that are currently active.

Reordering Sectors after Repositioning

To reorder sectors after repositioning the sectors, press PF9 (reord). The changes are displayed in the Sector fields on the Field Layout Parameters panel, as well as in the Reposition Parameters window.

Field Layout Parameters Panel

The following example shows the second specification panel for the Map model, the Field Layout Parameters panel:



Field Layout Parameters Panel for the Map Model

On this panel, you can design the map layout by either:

- Adding field definitions manually

  or

- Selecting fields from the view or data areas specified on the Standard Parameters panel

  You can also add prompt names, define scroll sectors, reorder the sequence of fields, and test the map.

The fields on this panel are:

Field	Description
1_	Number of the first field specification line (10 lines are displayed at one time). You can specify up to 40 fields for a map, one field per line. To scroll through the lines, press PF7 (bkwrd), PF8 (frwrd), or enter a new line number in this field. For example, to display the eleventh field specification line, enter “11” in this field.
Sector	Number of the sector in which the corresponding field is placed. (If no sector is specified, the field is not displayed on the map.)
Order	Position of the field within the sector. If no value is specified for the Order field, Natural Construct calculates a default value at generation or test time. You can change the order in which fields are displayed in the sector by typing new numbers in these fields. The changes are displayed on the map when you test or generate it. To reorder the lines to reflect the new sequence, press PF9 (reord). For more information, see **Reordering Fields after Repositioning**, page 283.
Label	If the field has a qualifier (specified on the Standard Parameters panel), specify the line number of the qualifier (a number from 1 to 6). If the qualifier is a Predict view or data area, you can press PF6 (selfd) to select fields using the Data Parameters window. For a description of this window, see **Selecting Fields for the Map**, page 282.
Field Name	Identifier (name) of the field.
Format	Natural format and length of the field.
AL=	Override length of a field selected from a view or data area. To override the length of a selected field, specify the new length in this field.
Scrollable Sector Definitions	This section of the panel defines array information for scroll sectors on the map. You can define up to four scroll sectors for a map (in the example, the current sector is >>1). To define scroll sectors, specify array information in the following fields for each scroll sector:
Sector	Number of the sector to designate as a scroll sector.

Field	Description (continued)
Screen occurrences	Maximum number of field occurrences to be displayed on the map at one time. This number corresponds to the number of scrollable lines for a vertical scroll sector or the number of columns for a horizontal scroll sector.
Column spacing	Number of spaces between vertical arrays in a vertical scroll sector or between horizontal arrays in a horizontal scroll sector. The default is 1.
Vertical-Horizontal	Vertical or horizontal direction indicator for the scroll sector. To display a vertical indicator, type “V” in this field. To display a horizontal indicator, type “H”. A scroll sector is displayed on your map as follows:

```
Horizontal-array1-prompt: _____ _____ _____
Horizontal-array2.......: ___   ___   ___

Vertical array                  Vertical array
Header 1                          Header 2
________                       ------------------------------

_ _ _ _                           _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _                           _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _                           _ _ _ _ _ _ _ _ _ _ _ _
```

Field	Description (continued)
Underline headings	This field is marked by default, indicating that array headings are underlined in a vertical scroll sector. These fields have no effect on horizontal scroll sectors.
Start index	Starting index for array entries in the scroll sector. By default, #ARRAY1 is displayed. Use the default or specify a fixed number in this field. Programs using this map must assign a value to the starting index variable.
Warning:	Natural Construct-generated maintenance programs expect to use the default variables. Take caution when changing this field.

Testing the Map

To test the map, press PF4 (test) on the Field Layout Parameters panel. The Test Map panel is displayed:

```
L001   ---10---+----20---+----30---+----40---+----50---+----60---+----70---+---

   CUSTOMER INFORMATION
   Business Name::XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   Contact......::XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   Phone Number ::9999999999

   ADDRESS INFORMATION
   Street.....::XXXXXXXXXXXXXXXXXXXXXXXXX
   City.......::XXXXXXXXXXXXXXXXXXX
   Province...::XXXXXXXXXXXXXXXXXXX
   Postal Code::XXXXXX

   CREDIT INFORMATION
   Credit Rating......::XXX      Credit Limit::99999999999.99
   Discount Percentage::999.99




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn                         bkwrd frwrd       left  right
```

Test Map Panel

Press PF2 (retrn) to return to the Field Layout Parameters panel.

Adding Prompts and Text to the Map

To add the prompts and text to be displayed on the map, press PF5 (prmpt) on the Field Layout Parameters panel. The following fields are displayed:

```
CUMPMB                          Map Model                          CUMPMB20
Sep 10                   Field Layout Parameters                     2 of 2

1_   Field Prompt or Text               Field Name            OCC   SG  AD=O
>>   ------------------------- ------------------------------ ----- ---- ----
 _____________________________ ______________________________ _____   _    _
 _____________________________ ______________________________ _____   _    _
 _____________________________ ______________________________ _____   _    _
 _____________________________ ______________________________ _____   _    _
 _____________________________ ______________________________ _____   _    _
 _____________________________ ______________________________ _____   _    _
 _____________________________ ______________________________ _____   _    _
 _____________________________ ______________________________ _____   _    _
 _____________________________ ______________________________ _____   _    _
 _____________________________ ______________________________ _____   _    _

  Scrollable Sector Definitions
  >> 1 Sector ..... __  Screen occurrences . __
       Column spacing ..... 1   Vertical-Horizontal  V
       Underline headings . X   Start index ........ #ARRAY1_

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit  test  field selfd bkwrd frwrd reord left        main
```

Field Layout Parameters Panel with Field Prompt Fields

The fields on this panel are:

Field	Description
Field Prompt or Text	Text displayed on the map. If the field name is a vertical array, you can display the field heading on up to three lines by indicating each break with the slash (/) character. (To use the slash character as text for array headings, enclose it within double quotes.)
Field Name	Identifier (name) of the field.
OCC	Maximum number of occurrences for an array.
SG	If this field is marked, the sign option is included for the field (SG=ON). If this field is blank, the sign option is off (SG=OFF).
AD=O	If this field is marked, the field is an output field (for display only). If this field is blank, the field is an input field.

Selecting Fields for the Map

To select up to 40 fields for a map from Predict views, LDAs, or PDAs, press PF6 (selfd) on the Field Layout Parameters panel. The Data Parameters window is displayed:

```
CSUSELFV                  Natural Construct                  CSUSELF0
Sep 10                     Data Parameters                     1 of 1

  Label      Type      Predict Views or Data Areas    Select  All
 -------- ---------- -------------------------------- ------ ------
    1                 ________________________________   _      _
    2                 ________________________________   _      _
    3                 ________________________________   _      _
    4                 ________________________________   _      _
    5                 ________________________________   _      _
    6                 ________________________________   _      _
 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11
       help  retrn
```

Data Parameters Window for the Map Model

The fields in this window are:

Field	Description
Label	Qualifier number for the view or data area (assigned on the Standard Parameters panel).
Type	Object type of the entry specified in the Predict Views or Data Areas field.
Predict Views or Data Areas	Name of a Predict view, LDA, PDA, or structure. (You can only select fields from Predict views and data areas.)
Select	To select fields from the view or data area, mark this field and press Enter. A window is displayed, listing the fields in the specified view or data area. Mark the fields to be included on the map. After closing the window, the field names, formats, and lengths are displayed on the Field Layout Parameters panel.
All	To select all fields in the view or data area, mark the All field. Multiple-valued fields (MUs) within a periodic group (PE) are not included when selecting a view. Constants and arrays with a rank greater than 1 are not included when selecting a data area.
Note:	If there is more than one structure in a data area, only the first structure is selected when you mark this field. Previously-selected fields are not re-selected.

Reordering Fields after Repositioning

After changing the order of fields in a sector, or changing the sector in which a field occurs, press PF9 (reord) to reorder the fields to reflect their new positions.

The following example shows a new position specified for the Contact field in sector 11 and new order numbers specified for fields in the sector.

```
CUMPMB                          Map Model                          CUMPMB10
Jun 18                    Field Layout Parameters                    1 of 1

1_    Sector   Order   Label             Field Name              Format   AL=
>>   -------- -------- -------- ------------------------------ --------- -----
       11 *     1__      1 *    CUSTOMER-NUMBER_________________ N __5.0  _____
       11 *     3__      1 *    BUSINESS-NAME___________________ A _30.0  _____
       21 *     1__      1 *    PHONE-NUMBER____________________ N _10.0  _____
       21 *     2__      1 *    M-STREET________________________ A _25.0  _____
       21 *     3__      1 *    M-CITY__________________________ A _20.0  _____
       21 *     4__      1 *    M-PROVINCE______________________ A _20.0  _____
       21 *     5__      1 *    M-POSTAL-CODE___________________ A __6.0  _____
       11 *     2__      1 *    CONTACT_________________________ A _30.0  _____
          *     ___      _ *    ________________________________ _ _____  _____
          *     ___      _ *    ________________________________ _ _____  _____

  Scrollable Sector Definitions
  >> 1 Sector ............. __  Screen occurrences . __
       Column spacing ..... 1   Vertical-Horizontal  V
       Underline headings . X   Start index ........ #ARRAY1_

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit  test  prmpt selfd bkwrd frwrd reord left        main
```

Field Layout Parameters Panel, Specified New Position and Order Numbers

After testing the map and pressing PF9 (reord) to reorder the fields, the new position and order numbers are displayed on the Field Layout Parameters panel:

```
CUMPMB                            Map Model                            CUMPMB10
Jun 18                      Field Layout Parameters                      1 of 1

1_    Sector   Order   Label             Field Name              Format   AL=
>>   -------- -------- -------- ---------------------------- --------- -----
       11 *     10_      1 *    CUSTOMER-NUMBER_____________  N __5.0  _____
       11 *     20_      1 *    CONTACT_____________________  A _30.0  _____
       11 *     30_      1 *    BUSINESS-NAME_______________  A _30.0  _____
       21 *     10_      1 *    PHONE-NUMBER________________  N _10.0  _____
       21 *     20_      1 *    M-STREET____________________  A _25.0  _____
       21 *     30_      1 *    M-CITY______________________  A _20.0  _____
       21 *     40_      1 *    M-PROVINCE__________________  A _20.0  _____
       21 *     50_      1 *    M-POSTAL-CODE_______________  A __6.0  _____
       __ *     ___      _ *    ____________________________  _ _____  _____
       __ *     ___      _ *    ____________________________  _ _____  _____

  Scrollable Sector Definitions
  >> 1 Sector ............. __  Screen occurrences . __
       Column spacing ..... 1   Vertical-Horizontal  V
       Underline headings . X   Start index ........ #ARRAY1_

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit  test  prmpt selfd bkwrd frwrd reord left        main
Reordered successfully
```

Field Layout Parameters Panel, Reordered Fields in the Sector

MENU MODEL

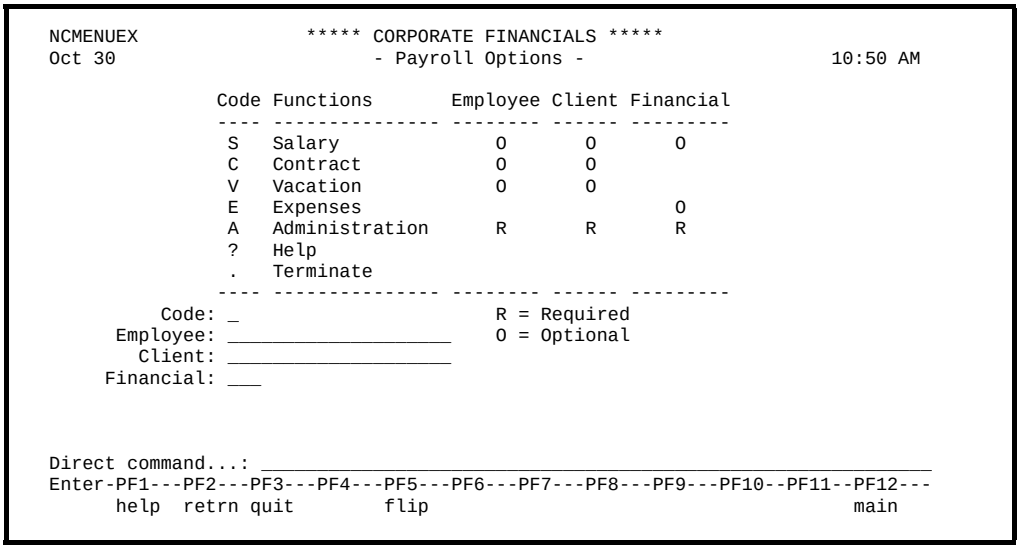
This chapter describes the Menu model, which generates a program that presents users with several choices in the form of a menu. The user enters a code for one of the choices to invoke a predefined function. You can also include additional fields on a menu, which may or may not require input.

Note: Although a map layout is not required for a menu program, it can give a consistent, tailored appearance to applications.

The following topics are covered:

Introduction

The following example shows a menu generated using the Menu model:



Sample Output From the Menu Model

To see the generated code for the Menu model example, refer to the NCMENUEX program in the Natural Construct demo system.

Parameters for the Menu Model

The Menu model has two specification panels: Standard Parameters and Additional Parameters. In addition, the Optional Parameters window allows you to add parameters with links to their functions. These panels are described in the following sections.

Standard Parameters Panel

The following example shows the first specification panel for the Menu model, the Standard Parameters panel:

```
CUMNMA                          Menu Program                          CU--MA0
Sep 10                      Standard Parameters                        1 of 2

 Module ............. ________
 System ............. CST341S_____________________________
 Global data area ... CDGDA___ *
 With block ......... _____________________________________

 Title ............... Menu ..._____________________
 Description ........ This menu ..._____________________________________________
                      _____________________________________________________________
                      _____________________________________________________________
                      _____________________________________________________________

 First header ....... _____________________________________________________
 Second header ...... _____________________________________________________

 Command ............ _
 Message numbers .... _
 Password ........... _

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                                          right main
```

Standard Parameters Panel for the Menu Model

This panel is similar for all models. For a description of the Standard Parameters panel, see **General Model Specifications**, page 167.

Additional Parameters Panel

The following example shows the second specification panel for the Menu model, the Additional Parameters panel:

```
CUMNMB                          Menu Program                          CUMNMB0
Jun 14                     Additional Parameters                       2 of 2

  Map layout ......... ________ *

     Code  Functions_____________________________________  Program Name
    ------ ---------------------------------------------- --------------
      S_    Salary_____________________________________   ________ *
      C_    Contract___________________________________   ________ *
      V_    Vacation___________________________________   ________ *
      E_    Expenses___________________________________   ________ *
      A_    Administration_____________________________   ________ *
      __   _____________________________________________   ________ *
      __   _____________________________________________   ________ *
      __   _____________________________________________   ________ *
      __   _____________________________________________   ________ *
      __   _____________________________________________   ________ *
      __   _____________________________________________   ________ *
      __   _____________________________________________   ________ *
      __   _____________________________________________   ________ *


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        optns attr                   left  userX main
```

Additional Parameters Panel for the Menu Model

The fields on this panel are:

Field	Description
Map layout	Name of the external map layout used for the menu program (use the CDLAYMN1 supplied layout map for the Menu model). If no map name is specified, Natural Construct builds the input panel using the information specified on this panel. For more information, see **Using Layout Maps**, page 120.
Note:	For a list of the variables you can use, see **Variables You Can Use with a Menu Model Map**, page 291.
Code	1 or 2 character codes the users must enter to invoke each menu function. This is a required field. Natural Construct aligns the function code and input fields under the function code column on the generated menu.
Note:	Because the first code specified in a direct command is compared internally to the current menu codes and then the main menu codes, the codes on all sub-menus should be different from codes on the main menu.

Field	Description (continued)
Functions	Function code descriptions displayed on the generated menu. If you are not using a layout map, you must provide a description for all codes specified in the Code field. The descriptions can be up to 45 characters in length.
Note:	To highlight the Function names, use the begin and end intensify characters (< and >). To change the default characters, press PF6 (attr) to display the Dynamic Attribute Maintenance window. For a description of this window, see **Changing the Default Attribute Parameters**, page 205.
Program Name	Names of the programs to FETCH if entering a menu code invokes a program. If the menu function does not invoke a program, this field must be blank.
Note:	Specify any alternate processing for menu codes in the SPECIAL-CODE-PROCESSING user exit. For information, see **SPECIAL-CODE-PROCESSING**, page 368.

Adding Optional Parameters

To add optional parameters to your menu, press PF5 (optns) on the Additional Parameters panel. The Optional Parameters window is displayed:

```
CUMNMBA                 Natural Construct                 CUMNMBA0
Oct 28                 Optional Parameters                  1 of 1

          Prompt        Name        Natural format
      ------------ ------------ --------------------
    1   Employee___   #CUST___       A 20.0_ *
    2   Client_____   #CLIENT_       A 20.0_ *
    3   Financial__   #FINANCE       N 3.0__ *
    4   ___________   ________       _ _____ *

    Link menu functions                                  1  2  3
    >> 1   Salary.....................................  0  0  0
       2   Contract...................................  0  0  _
       3   Vacation...................................  0  0  _
       4   Expenses...................................  _  _  0
       5   Administration.............................  R  R  R
       6  ...........................................  _  _  _
       7  ...........................................  _  _  _
       8  ...........................................  _  _  _
 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF
       help  retrn quit                  bkwrd frwrd
```

Optional Parameters Window for the Menu Model

Use this window to link up to four additional parameters to their associated menu functions. (The majority of the menus will not use this feature.)

The fields in this window are:

Field	Description
Prompt	Prompts displayed on the menu for any additional parameters.
Name	Name of a Natural internal variable to associate with the prompt. This variable checks the Required/Optional field to ensure that valid data is entered.
Natural format	Natural format and length of the specified variables. Use the single-character alphabetical abbreviations to represent the format and include the numeric length for the prompts (for example, N5). For a description of valid Natural formats and lengths, press PF1 for help or consult the Natural documentation.
Link menu functions	Names of the menu functions to be linked to the specified parameters. This link provides a cross reference between the additional parameters and menu functions. For each additional parameter and menu function, specify “R” (required) or “O” (optional).
Required/Optional	Codes (R for required or O for optional) that indicate whether users must specify the corresponding parameter to use the menu function. You can leave this field blank, providing there is at least one Required or Optional link to a menu function for each additional parameter. • If you do not specify a Required or Optional value for a Link menu functions field, the parameter will not be linked to the menu function. • If you specify a link, the linked non-global parameter is stacked (FETCHed) whenever the function is requested.

Variables You Can Use with a Menu Model Map

The following table describes the variables you can use with a map:

Variable	Format	Definition	Contents
#PROGRAM	A8	Output	Name of the program that invoked the map.
#HEADER1	A60	Output	First heading for the menu program.
#HEADER2	A58	Output	Second heading for the menu program.
#HPARM	A65	Output/ Nondisplay (AL=1)	Key to Natural Construct's passive help file for the current program (system name concatenated with program name). Place this variable on the map and pass it to the CD-HELPR helproutine.
#DIRECT-COMMAND	A60	Modifiable	Support for direct command processing.
#FUNCTION-HEADING	A45	Display	Heading for the list functions.
#CODE-IN-LIST(*)	A2/1:12	Display	List of valid codes.
#FUNCTION(*)	A45/1:12	Display	List of functions invoked from this menu.
#CODE	A2	Modifiable	Selected action code.

Note: If you added optional parameters, use the same names as used in the Prompt and Name fields in the Optional Parameters window. Add both the Prompt text and the valid Natural name as input fields on the map.

User Exits for the Menu Model

The following example shows the user exits for the Menu model:

```
CSGSAMPL                          Menu Program                          CSGSM0
Sep 10                             User Exits                           1 of 1

               User Exits             Exists    Sample   Required Conditional
    ------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                             Subprogram
 _  LOCAL-DATA
 _  START-OF-PROGRAM
 _  BEFORE-INPUT
 _  AFTER-INPUT
 _  BEFORE-PROCESSING-MENU-CODES
 _  SPECIAL-CODE-PROCESSING                     Example                X
 _  END-OF-PROGRAM
 _  SET-PF-KEYS


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                   bkwrd frwrd
```

Menu Program User Exits Panel

For information about these user exits, see **User Exits for the Generation Models**, page 321. For information about the User Exit editor, see **User Exit Editor**, page 96.

OBJECT-BROWSE MODELS

This chapter describes the Object-Browse series of models. These models provide easy access to database tables in a transparent and flexible way, regardless of where data resides and whether data is used to create a mainframe screen, GUI dialog, hardcopy report, Web page, spreadsheet, business service. The models are also ideal if access to the data is required for validation purposes.

The information in this chapter is available in HTML format from the Natural Business Services Documentation Overview.

OBJECT-GENERIC-SUBP MODEL

This chapter describes the Object-Generic-Subp model, which generates a business service. The information in this chapter is available in HTML format from the Natural Business Services Documentation Overview.

OBJECT-MAINT MODELS

This chapter describes how to use the Object-Maint series of models to generate the modules required for an object maintenance process. The information in this chapter is available in HTML format from the Natural Business Services Documentation Overview.

QUIT MODEL

This chapter describes the Quit model, which generates a quit program. A quit program releases resources used by an application. It displays a confirmation window that overlays the host panel and gives users the option of quitting an application entirely or resuming where they left off.

The following topics are covered:

Introduction

The Quit model generates a termination program invoked when a user:

- Enters a period (.) in the Function field
- Presses the quit PF-key
- Presses the retrn PF-key on a main menu

Note: PF3 (quit) and PF2 (retrn) are the default quit and return PF-keys for Natural Construct-generated modules.

The following example shows a quit confirmation window generated using the Quit model:

```
CS-CONFQ               Natural Construct
Sep 11          Confirm Application Termination                  1 of 1

Press PF3 to quit, press PF2 to resume Natural Construct
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10
           retrn quit
```

Output from Quit Model Example

To view the specifications for this example, refer to the NCQUIT program in the Natural Construct demo system.

The name of the quit program is assigned to the #DIALOG-INFO.##QUIT global variable in a Natural Construct-generated startup program.

Tip: Instead of generating your own quit program, you can use the CD-QUIT default quit program supplied with Natural Construct. If you use the default program, enter “CD-QUIT” in the Quit program field on the Standard Parameters panel for the Startup model. For more information, see **Startup Model**, page 315.

Parameters for the Quit Model

The Quit model has one specification panel, the Standard Parameters. This panel is described in the following section.

Standard Parameters Panel

```
CUQTMA                          Quit Program                          CUQTMA0
Sep 11                     Standard Parameters                         1 of 1

 Module ............. LGLQUIT_
 System ............. CST341S_____________________________
 Global data area ... DEMO____ *
 With block ......... _____________________________________

 Title ............... Terminate ..._____________
 Description ........ This is the termination program for CST341S ..._________
                      ____________________________________________________________
                      ____________________________________________________________
                      ____________________________________________________________


 Input using map .... ________ *
 Application ........ Application name ..._____

 Message numbers .... _
 Terminate Natural .. _

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit                                            userX main
```

Standard Parameters Panel for the Quit Model

This panel is similar for all models. The fields in the upper portion of this panel are described in **General Model Specifications**, page 167. The following table describes the fields in the lower portion of this panel:

Field	Description
Input using map	Name of the map used by the quit program. If you do not specify a map name, Natural Construct builds the INPUT statement.
Application	Application name used in the confirmation message. By default, “Application name” is displayed and the confirmation message is, “Press PF3 again to quit entirely out of Demo Application”.
Note:	PF3 is the default quit PF-key for Natural Construct-generated modules. You may have a different quit PF-key.

Field	Description (continued)
Message numbers	If this field is marked, the quit program uses message numbers rather than message text for all REINPUT and INPUT messages.
Note:	Regardless of whether you use message numbers or text, use the same technique consistently throughout your application since passing messages between modules using different techniques will not always produce the desired results.
Terminate Natural	If this field is marked, the quit program issues a Natural terminate command. If this field is blank, the generated quit program restores default Natural error trapping, sets the window size to the physical panel size, releases the Natural stack, backs out all outstanding database updates, and issues a Natural STOP command.

All information specified on the Standard Parameters panel is displayed in the banner (the summary information at the beginning of the program) for the generated quit program.

User Exits for the Quit Model

The following example shows the User Exits panel for the Quit model

```
CSGSAMPL                        Quit Program                          CSGSM0
Sep 11                           User Exits                           1 of 1
                                                                            C
              User Exits             Exists    Sample   Required Conditional
    ------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                             Subprogram
 _  LOCAL-DATA
 _  BEFORE-CONFIRMATION
 _  AFTER-INPUT
 _  BEFORE-RESUMING-PROCESSING
 _  END-OF-PROGRAM
 _
 _
 _
 _
 _
 _
 _
 _
 _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                   bkwrd frwrd
```

User Exits Panel for the Quit Model

For information about these user exits, see **User Exits for the Generation Models**, page 321. For information about the User Exit editor, see **User Exit Editor**, page 96.

REMOTE-PROCEDURE-CALL MODEL

This chapter describes the Remote-Procedure-Call model. This model generates a sub-program (client stub) that defines the parameters to be passed to a remote server subprogram.

The following topics are covered:

Introduction

The Remote-Procedure-Call model's functionality complements the Natural-supplied Remote Procedure Call facility. This model simplifies the process of generating a client stub, since it copies the parameters of the corresponding server subprogram. You can choose to use either the Remote-Procedure-Call model or the Natural RPC facility.

Note: For information about the functionality of the Remote-Procedure-Call model, refer to the Natural Installation documentation.

The remote subprogram must have the same name and parameters as the corresponding server subprogram. Since Natural cannot support two different files with the same name in one library, make one of the following preparations before generating the subprogram:

- After creating the server subprogram, log onto another library that defines the original library as a steplib and create the remote subprogram there.
- After creating the server subprogram, move the server subprogram to a library that is a steplib of the original library. Then create the remote subprogram in the original library.
- Create the remote subprogram in the same library as the server subprogram, but with a different name. Afterwards, move the remote subprogram and rename it.

Parameters for the Remote-Procedure-Call Model

The Remote-Procedure-Call model has one specification panel: Standard Parameters. This panel is described in the following section.

Standard Parameters Panel

```
CUSGMA                 REMOTE-PROCEDURE-CALL Subprogram              CUSGMA0
Sep 30                       Standard Parameters                      1 of 1

  Module ............. ________
  System ............. SYSCSTDE_____________________________

  Title .............. RPC ..._____________________
  Description ........ RPC subprogram for ..._____________________________________
                       ________________________________________________________
                       ________________________________________________________
                       ________________________________________________________

                                     Source      Object
  Subprogram ......... ________ *




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit                                            userX main
```

Standard Parameters Panel for the Remote-Procedure-Call Model

This panel is similar to the Standard Parameters panel for most models, except it has three additional fields. For a description of the common fields, see **General Model Specifications**, page 167. The additional fields on this panel are:

Field	Description
Subprogram	Name of the server subprogram. By default, the name of the module is displayed in this field because the two subprograms must have the same name and parameters.
Source	Name of the first library containing source code for the specified subprogram. After you enter the server subprogram name, Natural Construct searches the steplib chain for the subprogram's source code and displays the name of the first library in which it is found.

Field	Description (continued)
Object	Name of the first library containing object code for the specified subprogram. After you enter the server subprogram name, Natural Construct searches the steplib chain for the subprogram's object code and displays the name of the first library in which it is found.

User Exits for the Remote-Procedure-Call Model

The following example shows the user exits for the Remote-Procedure-Call model.

```
CSGSAMPL              REMOTE-PROCEDURE-CALL Subprogram               CSGSM0
Oct 01                          User Exits                           1 of 1

              User Exits             Exists    Sample   Required Conditional
   -------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                             Subprogram
 _  LOCAL-DATA                                  Example
 _  START-OF-PROGRAM
 _  ADDITIONAL-INITIALIZATIONS                  Example
 _  END-OF-PROGRAM
 _
 _
 _
 _
 _
 _
 _
 _
 _
 _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                   bkwrd frwrd
```

Remote-Procedure-Call User Exits

For information about these user exits, see **User Exits for the Generation Models**, page 321. For information about the User Exit editor, see **User Exit Editor**, page 96.

SHELL MODEL

This chapter describes the Shell model, which generates a template for a program, subprogram, helproutine, or subroutine.

The following topics are covered:

Introduction

The Shell model creates a DEFINE DATA ... END-DEFINE framework containing definitions for the global data area (GDA), parameter data area (PDA), local data areas (LDAs), or views specified on the Standard Parameters panel. You can use this time-saving model to generate startup modules for your applications.

The Shell model does not support the following functions:

- Process user exits
- Read or save Natural Construct specifications

Note: As the Shell model does not read or save specifications, you cannot regenerate modules generated with this model.

Shell Model Example

The following example shows code generated by the Shell model:

```
>                                    > +  Program     NCSHELL   Lib CST451S
Top    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7
  0010 ************************************************************************
  0020 * Program  : NCSHELL
  0030 * System   : DEMO
  0040 * Title    : Shell Example
  0050 * Generated: May 22,04 at 04:39 PM
  0060 * Function : This module is used to ...
  0070 *           |
  0080 *           |
  0090 *           |
  0100 *           |
  0110 *           |
  0120 *           |
  0130 * Mod Date   Mod By    Description of Changes
  0140 * MMM DD YY  ________ ____________________________________________
  0150 * ____________________________________________________
  0160 ************************************************************************
  0170 *
  0180 DEFINE DATA
  0190   GLOBAL USING NCGDA
  0200   LOCAL
  0210   /*
  0220   /* Local Variables
  0230   /*
  0240   /* Views
```

```
0250   01 NCST-CUSTOMER VIEW OF NCST-CUSTOMER
0260     02 CUSTOMER-NUMBER
0270     02 BUSINESS-NAME
0280     02 PHONE-NUMBER
0290     02 MAILING-ADDRESS
0300       03 M-STREET
0310       03 M-CITY
0320       03 M-PROVINCE
0330       03 M-POSTAL-CODE
0340     02 SHIPPING-ADDRESS
0350       03 S-STREET
0360       03 S-CITY
0370       03 S-PROVINCE
0380       03 S-POSTAL-CODE
0390     02 CONTACT
0400     02 CREDIT-RATING
0410     02 CREDIT-LIMIT
0420     02 DISCOUNT-PERCENTAGE
0430     02 CUSTOMER-WAREHOUSE-ID
0440     02 CUSTOMER-TIMESTAMP
0450   01 NCST-WAREHOUSE VIEW OF NCST-WAREHOUSE
0460     02 WAREHOUSE-ID
0470     02 WAREHOUSE-DESCRIPTION
0480     02 WAREHOUSE-ADDRESS
0490       03 WAREHOUSE-STREET
0500       03 WAREHOUSE-CITY
0510       03 WAREHOUSE-PROVINCE
0520       03 WAREHOUSE-POSTAL-CODE
0530 END-DEFINE
0540 *
0550 END
```

Output from the Shell Model Example

Add your code immediately following the END-DEFINE statement.

For a description of the parameters specified for this example, see **Example of a Shell Program**, page 314.

Parameters for the Shell Model

The Shell model has one specification panel, the Standard Parameters panel. This panel is described in the following section.

Standard Parameters Panel

```
CUSHMA                          Shell Program                          CUSHMA0
Sep 07                      Standard Parameters                         1 of 1

  Module ............. LGLSHELL
  Module type ........ P
  System ............. CST341S_____________________________ *

  Title .............. Function____________________
  Description ........ This module is used to ..._____________________________
                       ____________________________________________________________
                       ____________________________________________________________

  Global data area ... _  USING  ________
  Parameter data area  _  USING  ________ *
  Local data area .... _  USING  ________ ________ ________ ________ *

  Views ........  1    ____________________________________ *
                 2    ____________________________________ *
                 3    ____________________________________ *
                 4    ____________________________________ *
                 5    ____________________________________ *
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit                                                  main
```

Standard Parameters Panel for the Shell Model

The fields on this panel are:

Field	Description
Module	Name of the module (by default, the name specified on the Generation main menu). This is a required field. The module name must be alphanumeric, cannot exceed 8 characters in length, and cannot contain blanks.
Module type	Code for the type of module for which you are creating the shell program. Valid codes are: • P (Program) • N (Subprogram) • H (Helproutine) • S (Subroutine)

Field	Description (continued)
System	Name of the system (by default, the name of the current library). This is a required field. The system name must be alphanumeric, cannot exceed 32 characters in length, and does not have to be associated with a Natural library ID. (The combination of the module name and system name is used as a key to access help information.)
Title	Title for the shell program. The title is used internally to identify modules for the List Generated Modules function on the Generation main menu. This is a required field.
Description	Description of the shell program. This description is used internally for documentation purposes.
Global data area	If this field is marked, the generated module uses the global data area specified in the corresponding USING field.
Parameter data area	If this field is marked, the generated module uses the inline parameter data area specified in the USING field.
Note:	If the program type is P (Program) or S (Subroutine), you cannot specify parameter data.
Local data area	If this field is marked, the generated module uses the inline local data or external local data areas specified in the USING fields. You can specify up to four LDA names.
Views	Names of the Predict views used by the generated program.

Example of a Shell Program

To create the shell program example (shown in **Introduction**, page 310), fill in the Standard Parameters panel as follows:

```
CUSHMA                          Shell Program                          CUSHMA0
Sep 07                       Standard Parameters                        1 of 1

  Module ............. NCSHELL
  Module type ........ P
  System ............. DEMO   _____________________________ *

  Title .............. Shell Example_____________
  Description ........ This module is used to illustrate the features of______
                       the Shell model._____________________________________
                       ____________________________________________________

  Global data area ... x  USING  NCGDA___
  Parameter data area  _  USING  ________ *
  Local data area .... _  USING  ________ ________ ________ ________ *

  Views ........  1    NCST-CUSTOMER_____________________ *
                 2    NCST-WAREHOUSE____________________ *
                 3    _____________________________________ *
                 4    _____________________________________ *
                 5    _____________________________________ *
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit                                                  main
```

Standard Parameters Panel for the Shell Model Example

Tip: Shell programs are excellent candidates for generation from the Natural editor using the .g command. For more information, see **Statement Models**, page 395.

STARTUP MODEL

This chapter describes the Startup model, which generates startup programs for applications. These programs (often named Menu) initialize global variables and invoke the main menu program.

The following topics are covered:

Parameters for the Startup Model

The Startup model has one specification panel: Standard Parameters. This panel is described in the following section.

Standard Parameters Panel

```
CUSTMA                        Startup Program                        CUSTMA0
Sep 11                     Standard Parameters                        1 of 1

 Module ............. MENU____
 System ............. DEMO_____________________________
 Global data area ... CDGDA___ *
 With block ......... _____________________________

 Title ............... DEMO Startup_____________
 Description ........ This is the main startup program for DEMO._____________
                      ____________________________________________________________
                      ____________________________________________________________
                      ____________________________________________________________

 Specific Parameters
 Main menu program ............ ________ *
 Quit program ................. ________ *
 Natural Command Processor .... ________ *
 Error transaction processing . _


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit                                          userX main
```

Standard Parameters Panel for the Startup Model

The Standard Parameters panel is similar for all models. The fields in the upper portion of the panel are described in **General Model Specifications**, page 167. The fields in the lower portion of the panel (Specific Parameters) are:

Field	Description
Main menu program	Name of the program invoked by the startup program. This is usually the first panel displayed when a user issues the Natural MENU command.
Quit program	Name of the program invoked when a user ends a session by pressing PF3 (quit) or entering a “.” on the main menu.
Note:	If no special cleanup is required when the program terminates, you can use the CD-QUIT program supplied with Natural Construct.

Field	Description (continued)
Natural Command Processor	Name of a Natural command processor that processes commands entered on the direct command line. The specified command processor must have been created using the Natural SYSNCP systems utility. For more information, see **Using a Natural Command Processor**, page 114, or the Natural Utilities documentation.
Error transaction processing	If this field is marked, Natural Construct's default error transaction program (CDERRTA) is assigned to the *ERROR-TA system variable. For more information about *ERROR-TA, see the Natural documentation. For more information about recovering from errors, see **ERROR-INFO Structure**, page 111.

User Exits for the Startup Model

The following example shows the User Exits panel for the Startup model:

```
CSGSAMPL                       Startup Program                          CSGSM0
Sep 11                            User Exits                            1 of 1

              User Exits             Exists    Sample   Required Conditional
   ------------------------------- -------- ---------- -------- ------------
 _ CHANGE-HISTORY                             Subprogram
 _ LOCAL-DATA
 _ BEFORE-FETCH
 _ FINAL-PROCESSING                            Example
 _
 _
 _
 _
 _
 _
 _
 _
 _
 _
 _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                   bkwrd frwrd
```

User Exits Panel for the Startup Model

For information about these user exits, see **User Exits for the Generation Models**, page 321. For information about the User Exit editor, see **User Exit Editor**, page 96.

Example of a Startup Program

To generate the sample startup program (which has no visible output), fill in the Standard Parameters panel as follows:

```
CUSTMA                          Startup Program                          CUSTMA0
Sep 11                       Standard Parameters                          1 of 1

 Module ............. MENU____
 System ............. DEMO_____________________________
 Global data area ... CDGDA___ *
 With block ......... _____________________________

 Title ............... DEMO Startup_____________
 Description ........ This is the main startup program for DEMO._____________
                      ____________________________________________________________
                      ____________________________________________________________
                      ____________________________________________________________

 Specific Parameters
 Main menu program ............ CUBOGROW *
 Quit program ................. CD-QUIT_ *
 Natural Command Processor .... ________ *
 Error transaction processing . _


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit                                             userX main
```

Standard Parameters Panel for the Startup Model Example

TRANSFORM-BROWSE MODEL

This chapter describes the Transform-Browse model. This model transforms an existing browse module (a browse subprogram generated by the Browse model) into object-browse modules (modules generated by the Object-Browse series of models). The information in this chapter is available in HTML format from the Natural Business Services Documentation Overview.

USER EXITS FOR THE GENERATION MODELS

This chapter describes what a user exit is, how to invoke the User Exit editor, and how to define a user exit. It also describes the user exits available for generation models supplied with Natural Construct and Construct Spectrum.

The following topics are covered:

Introduction

The following Natural Construct models support user exits:

- Batch
- Browse, Browse-Helpr, and Browse-Subp
- Browse-Select, Browse-Select-Helpr, and Browse-Select-Subp
- Driver
- Extendable-Input
- Maint
- Menu
- Object-Browse-Subp and Object-Browse-Select-Subp
- Object-Generic-Subp
- Object-Maint-Subp, Object-Maint-Dialog, and Object-Maint-Dialog-Subp
- Quit
- Remote-Procedure-Call
- Startup

Note: Statement models do not support user exit processing.

For information about the user exits for the JCL models, see **Required Setup for the JCL Models**, page 571.

About User Exits

User exits are positions within a Natural Construct-generated module where you can insert customized or specialized processing. Changes to the user exit code are always preserved upon subsequent regeneration of the module.

Natural Construct models provide a wide variety of user exits. You can select from a list of available exits by entering SAMPLE at the > prompt in the User Exit editor. For more information, see **User Exits Panel**, page 323.

The exits vary depending on the type of module you are generating. Some exits contain sample code or subprograms, while others generate the DEFINE EXIT and END-EXIT lines only — you provide the actual code. You can modify any user exit code generated into the edit buffer.

Note: If you require code to be inserted in the generated module where no user exit currently exists, have your Natural Construct administrator recommend a suitable exit or add a new exit to the model.

Invoking the User Exit Editor

You can invoke the User Exit editor in two ways: from the Generation main menu using the Invoke User Exit Editor function or from the last specification panel for a model that supports user exits by pressing the userX PF-key (PF11, by default).

To invoke the User Exit editor from the Generation main menu, see **Invoke User Exit Editor Function**, page 68.

To invoke the User Exit editor from the model specification panels, press PF11 (userX) on the last specification panel for a model:

- If user exits are defined for the module, the existing user exit code is displayed in the User Exit editor. To select additional exits, enter SAMPLE at the > prompt to display the User Exits panel.
- If no user exits are defined for the module, the User Exits panel for the corresponding model is displayed.

User Exits Panel

➢ To invoke the User Exits panel from the User Exit editor:

1 Type “SAMPLE” at the > prompt in the User Exit editor.

2 Press Enter.
A selection panel of available user exits for the specified model is displayed. The available user exits vary, depending on what type of module being generated.

Note: The SAMPLE command is performed automatically when you invoke the User Exit editor and no user exits have been defined for a module.

The following example shows the first User Exits panel for the Browse model:

```
CSGSAMPL                        Browse Program                          CSGSM0
Sep 05                            User Exits                            1 of 1

               User Exits             Exists    Sample   Required Conditional
    ------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                             Subprogram
 _  PARAMETER-DATA                              Example                X
 _  HE-PARAMETER-INDEXES                        Example                X
 _  BUILD-REPORT-LOCAL-VARS
 _  LOCAL-DATA                                 Subprogram
 _  START-OF-PROGRAM                            Example
 _  SELECT-STATEMENT                           Subprogram              X
 _  AFTER-READ                                  Example
 _  REJECT-AFTER-MAX-KEY-CHECK
 _  EXPORT-DATA                                 Subprogram              X
 _  WRITE-FIELDS                               Subprogram
 _  TOP-OF-PAGE                                 Example
 _  BEFORE-INPUT                                Example
 _  BEFORE-STANDARD-KEY-CHECK                   Example
 _  AFTER-INPUT                                 Example
 _  HARDCOPY-EDITS                              Example                 X
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
frwrd help  retrn quit                    bkwrd frwrd
```

User Exits for the Browse Model — Panel 1

3 Press Enter or PF8 (frwrd) to display the remaining user exits:

```
CSGSAMPL                        Browse Program                          CSGSM0
Sep 05                            User Exits                            1 of 1

               User Exits             Exists    Sample   Required Conditional
    ------------------------------- -------- ---------- -------- ------------
 _  HARDCOPY-TERMINATING-PROCESS                Example                X
 _  PROCESS-SELECTED-RECORD                    Subprogram              X
 _  END-OF-PROGRAM                              Example
 _  ASSIGN-PREFIX-VALUE                        Subprogram              X
 _  SET-PF-KEYS                                 Example
 _  MISCELLANEOUS-SUBROUTINES                   Example
```

User Exits for the Browse Model — Panel 2

The fields on the User Exits panel are:

Field	Description
User Exits	Names of user exits available for this model. If a user exit is required for a model and is not conditional (its existence is not based on condition codes in the code frames), it is marked by default.
Exists	If the corresponding user exit is defined (exists), X is displayed. If the user exit does not exist, this field is blank.

Field	Description (continued)
Sample	If the corresponding user exit contains the DEFINE EXIT _ END-EXIT lines only, this field is blank. If the user exit contains a subprogram, Subprogram is displayed. If the user exit contains sample code, Example is displayed.
Required	If the corresponding user exit must be specified, X is displayed. If the user exit is optional, this field is blank.
Conditional	If the corresponding user exit is conditional (its existence is based on condition codes in the code frames), X is displayed. If the user exit is not conditional, this field is blank.

➢ To select a user exit displayed on the User Exits panel:

1 Mark the input field to the left of the user exits you want.

2 Press Enter.
The selected user exits are displayed in the User Exit editor.

You can define user exits directly within the User Exit editor, without using the SAMPLE command. You can also change the code generated using the SAMPLE command as desired. All user exit code is preserved when a module is regenerated. Fully qualify all references to database fields from within the user exits with the file name.

Defining User Exits

The code specified within a user exit depends on the type of module being generated and the type of user exit you are using. However, all user exits have the following format:

```
DEFINE EXIT user-exit-name

  user exit code

END-EXIT user-exit-name
```

Note: Do not insert comments or Natural code on the DEFINE EXIT and END-EXIT lines.

Note: If multiple user exits are generated with the same name, Natural Construct will merge them into a single user exit within the generated module.

Supplied User Exits

The following sections describe the user exits available for the supplied models. The user exits are listed in alphabetical order. For many exits, one or more examples are also included.

ADD-ACTION-PROCESSING

The code in this exit specifies additional processing performed for the Add action (in addition to the standard Natural Construct-generated processing).

ADD-COLUMNS

The code in this exit adds columns for a Visual Basic browse object. Use this user exit for modules generated by the VB-Browse-Object and VB-Browse-Local-Data-Object models.

Example of code in the ADD-COLUMNS user exit

```
DEFINE EXIT ADD-COLUMNS
 '
 ' AddColumn Name, Heading,Show by Default, justify, BDT, FormatLen
 AddColumn "Col1-name", "Head", True, CF_LEFT_JUSTIFIED, BDT_VALUE,
"A20"
 AddColumn "Col2-name", "Heading", True, CF_RIGHT_JUSTIFIED, "", "N3"
 AddColumn "Col3-name", "Heading",True, CF_CENTER, "", "D"
 END-EXIT
```

ADDITIONAL-ACTIONS-PROCESSING

The code in this exit processes any additional actions that were defined within the SELECT-ADDITIONAL-ACTIONS user exit. For more information, see **SELECT-ADDITIONAL-ACTIONS**, page 365.

ADDITIONAL-INITIALIZATIONS

This user exit generates the framework for any additional initializations to be performed in the INITIALIZATIONS subroutine.

Example of code in the ADDITIONAL-INITIALIZATIONS user exit

```
** SAG DEFINE EXIT ADDITIONAL-INITIALIZATIONS
* Assign parameters for help routine CD-HELPR
MOVE  'CU'  TO #MAJOR-COMPONENT
MOVE *PROGRAM   TO  #MINOR-COMPONENT
**SAG END-EXIT
END-SUBROUTINE /* INITIALIZATIONS
```

ADDITIONAL-TRANSLATE-MAP

The code in this exit defines the processing performed to translate field headings and prompts on the generated map.

Note: If the generated module supports cursor translation, you must define this exit.

ADDITIONAL-TRANSLATE-TEXT

The code in this exit defines the processing performed to translate field headings and prompts in the local data area.

Note: If the generated module supports cursor translation, you must define this exit.

ADDITIONAL-TRANSLATIONS

This exit generates the framework for translation parameters that are not defined in the translation local data areas (LDAs) for the generated module.

Natural Construct translates prompts and headings whenever they are called within a module. This exit allows you to specify additional translations at the appropriate position within the module, depending on the operation mode (Modify mode, Generate mode, Batch mode, for example).

ADJUST-OBJECT-ID-IN-MSG

The code in this exit overrides the default name that refers to an object in messages (concatenation of object description and ID).

AFTER-BROWSE-BY-FOREIGN-KEY

The code in this exit defines any special processing performed after a Visual Basic maintenance object browses for a foreign key. Use this exit with the VB-Maint-Object model.

AFTER-BROWSE-BY-OBJECT-KEY

The code in this exit defines any special processing performed after a Visual Basic maintenance object browses for the object key. Use this exit with the VB-Maint-Object model.

AFTER-BROWSE-CALLNAT

The code in this exit defines any special processing performed after calling the browse subprogram. Use this exit with the Object-Maint-Dialog model.

Tip: If you encounter an “unclaimed user exit” error message when generating a dialog that uses this user exit, ensure that you specified a browse subprogram name on the Additional Parameters panel for the Object-Maint-Dialog model.

AFTER-BROWSE-OBJECT

The code in this exit defines the processing performed after calling the browse object.

AFTER-BT-PROCESSING

The code in this exit defines the processing performed after the transaction has been backed out.

AFTER-CALLNAT-SUBPROGRAMS

The code in this exit defines any additional processing performed after the Natural subprograms are called.

AFTER-CALL-OBJECT

The code in this exit defines any special processing performed after regaining control from the target subprogram. Use this exit with the Subprogram-Proxy model.

Tip: If it is necessary to call the object multiple times, use the BEFORE-CALL-OBJECT exit to code an opening loop statement. Code the close loop statement in this user exit.

AFTER-CALL-TO-MAINT-OBJECT

The code in this exit defines the processing performed after the maintenance object is called by an object browse select subprogram. It is executed once for each row in the PDA.

Note: This user exit is only available if the object browse select subprogram accesses an object maintenance subprogram.

AFTER-COMPRESS-OUTPUT

The code in this exit defines any special processing performed after data in the data structures for the target subprogram is mapped to the data string returned to the server. For example, you can use this exit to modify the outgoing data stream prior to returning its contents to the Spectrum dispatch service. Use this exit with the Subprogram-Proxy model.

AFTER-ET-PROCESSING

The code in this exit defines any special processing performed after issuing an END TRANSACTION (ET) statement.

AFTER-EXPAND-INPUT

The code in this exit defines any special processing performed after the data string received from the server is mapped to the data structures used to call the target subprogram. Use this user exit for modules generated by the Subprogram-Proxy model. Also use this exit to modify the contents of local data structures needed by the target subprogram prior to doing a CALLNAT.

AFTER-GET

The code in this exit is executed after a new object is retrieved from the database. Use this exit to manipulate information in the object PDA, prior to returning the PDA to the calling module.

AFTER-GET-EDITS

The code in this exit converts or computes data after a GET operation moves the data from an entity record to an object. This exit must contain a series of subroutines named A-*entity-name*, where *entity-name* is a valid entity within the object. After an entity is retrieved from the database, the A-*entity-name* subroutine is performed once for each occurrence of the entity.

Since the values in the record buffer are copied to the object PDA using the MOVE BY NAME statement after the A-*entity-name* subroutine is performed, data manipulation in the subroutine should be made against the record buffer of the corresponding entity. Only manipulate directly against an object if the relevant fields have different names in the object PDA and the file.

Object-Maint-Subp Model

The Object-Maint-Subp model generates the #L2, #L3, and #L4 index variables for accessing the occurrences of the second, third, and fourth level entities within the object PDA. These indices must not be modified by the user exit code. However, the user exit code can access the object PDA fields by referencing the appropriate index tuples corresponding to the second, third, or fourth level entities (#L2, or #L2,#L3, or #L2,#L3,#L4, for example).

If these subroutines encounter invalid data within the object, they should assign values to the variables in the CDPDA-M parameter data area to terminate the process.

Example of using the subroutines in the AFTER-GET-EDITS user exit

The following INVOICE object contains values from the INVOICE-HEADER entity:

```
0010 DEFINE-EXIT AFTER-GET-EDITS
0020   DEFINE SUBROUTINE A-INVOICE-HEADER
0030    MOVE EDITED INVOICE-HEADER.INVOICE-DATE TO #DATE-VARIABLE
0040       (EM=YYYYMMDD)
0050    MOVE EDITED #DATE-VARIABLE (EM=LLL' 'DD','YY) TO
0060      INVOICE.INVOICE-DATE-EXTERNAL
0070   END-SUBROUTINE /* A-INVOICE-HEADER
0080 END-EXIT AFTER-GET-EDITS
```

AFTER-GET-FOREIGN-KEY-DESC

The code in this exit defines any special processing performed after a Visual Basic maintenance object fetches the foreign key description. Use this user exit for modules generated by the VB-Maint-Object model.

AFTER-INIT

The code in this exit is executed immediately after the object PDA variables are reset. You can use this exit to assign default initial values to the object fields.

AFTER-INITIAL-INPUT

The code in this exit is executed after the initial INPUT statement is executed.

AFTER-INPUT

The code in this exit is executed immediately after each input panel is displayed and the standard keys and direct commands are processed (AT END OF PAGE section). You can use this exit to define validity edits for user-defined fields or to add non-standard PF-key processing to a module.

For example, when you add a non-standard PF-key, you should set the #SCROLLING variable to TRUE so the generated module does not trap the PF-key as invalid. After processing the non-standard key, include the PERFORM NEW-SCREEN code to return to the main panel (main INPUT statement) for the module.

Note: If you do not include the PERFORM NEW-SCREEN code and continue with normal execution after processing this exit, an Invalid PF-key error message is displayed.

Example of user exit code for the Browse models

```
0010 DEFINE EXIT AFTER-INPUT
0020 *
0030 * Processing to be performed immediately after the exit checks,
0040 * after input.
0050 IF NOT (#OPTION = ' ' OR = 'M' OR = 'S' OR = 'C') THEN
0060   REINPUT 'Valid options are "M", "S" or "C" or blank'
0070   MARK *#OPTION ALARM
0080 END-IF
0090 END-EXIT AFTER-INPUT
```

Example of user exit code for the Object-Maint-Dialog model

```
0010 DEFINE EXIT AFTER-INPUT
0020   /*
0030   /* Compute total for current product line
0040  COMPUTE ORDER.TOTAL-COST(#ARRAY1) = ORDER.QUANTITY(#ARRAY1) *
0050                                     ORDER.UNIT-COST(#ARRAY1)
0060 END-EXIT AFTER-INPUT
```

AFTER-LOOKUP-SUBROUTINES

The code in this exit defines all subroutines specified in the Perform subroutine field for the Object-Maint-Dialog model. Before each subroutine is executed, the #LOOKUP-STATUS variable is assigned the value Found, Not Found, or Null (depending on the result of the lookup).

If a subroutine is defined for an array field within the object PDA, you can access the current occurrence of the array using the following indexes:

Dimension of Array	Index
1	#I1
2	#I1, #I2
3	#I1, #I2, #I3

Example of user exit code for the Object-Maint-Dialog model

```
0010 DEFINE EXIT AFTER-LOOKUP-SUBROUTINES
0020 *
0030 ************************************************************
0040 DEFINE SUBROUTINE PROCESS-PRODUCT
0050 ************************************************************
0060 *
0070   DECIDE ON FIRST VALUE #LOOKUP-STATUS
0080    VALUE 'FOUND'
0090      ASSIGN ORDER.LINE-DESCRIPTION(#I1) =
0100             NCST-PRODUCT.PRODUCT-DESCRIPTION
0110      ASSIGN ORDER.UNIT-COST(#I1) =
0120             NCST-PRODUCT.PRODUCT-UNIT-COST
0130     COMPUTE ORDER.TOTAL-COST(#I1) = ORDER.QUANTITY(#I1) *
0140                             ORDER.UNIT-COST(#I1)
0150    VALUE 'NOT FOUND'
0160    COMPRESS 'Product' ORDER.ORDER-PRODUCT-ID(#I1) 'not found'
0170             TO MSG-INFO.##MSG
0180      RESET ORDER.LINE-DESCRIPTION(#I1) ORDER.QUANTITY(#I1)
0190            ORDER.UNIT-COST(#I1) ORDER.TOTAL-COST(#I1)
0200     VALUE 'NULL'
0210       RESET ORDER.NCST-ORDER-HAS-LINES(#I1)
0220     NONE IGNORE
0230   END-DECIDE
0240 END-SUBROUTINE
0250 END-EXIT AFTER-LOOKUP-SUBROUTINES
```

AFTER-OBJECT-CALL

The code in this exit is executed after the object subprogram is called. It allows extra processing upon returning from the object subprogram (for example, you can override the MSG-INFO.##ERROR-FIELD or MSG-INFO.##MSG variables).

AFTER-PROCESS-ACTIONS

The code in this exit defines the processing performed after actions are processed.

AFTER-RANGE-INPUT

The code in this exit is executed after the INPUT statement is processed for the starting and ending values for the primary file.

AFTER-READ

The code in this exit is executed immediately following the retrieval of a view occurrence (after a READ statement). You can use this exit to join multiple files and maintain a view containing more than one secondary file, for example.

Note: The REJECT-AFTER-MAX-KEY-CHECK user exit is similar to this exit, except it is generated after a minimum or maximum key value is rejected. For information, see **REJECT-AFTER-MAX-KEY-CHECK**, page 362.

AFTER-ROW-ASSIGNMENT

The code in this exit is executed immediately after the view contents have been copied to the output object (row) PDA. You can alter the contents of the PDA passed back to the caller by using #ROW-INDEX to locate an item in the PDA.

Example of user exit code for the Object-Browse-Subp model

```
0140 DEFINE EXIT AFTER-ROW-ASSIGNMENT
0150 **
0160 ** Compute the derived column ANNUAL-VACATION-DAYS for current row
0170 COMPUTE EMPROW.ANNUAL-VACATION-DAYS(#ROW-INDEX)=
0180   EMPROW.LEAVE-TAKEN(#ROW-INDEX)+EMPROW.LEAVE-DUE(#ROW-INDEX)
0190 END-EXIT
```

Note: Use the #ROW-INDEX variable to reference the last assigned occurrence of the object row PDA within the AFTER-ROW-ASSIGNMENT user exit.

AFTER-SCREEN-CLEAR

The code in this exit is executed immediately after the CLEAR subroutine is performed. You can use this exit to reset the derived fields assigned when the AFTER-READ or BEFORE-INPUT user exit subroutines are performed.

AFTER-SECONDARY-FILE-PROCESSING

The code in this exit is executed from within the primary file processing loop, after all secondary and tertiary files are processed.

ASSIGN-PREFIX-VALUE

The code in this exit assigns the value for a browse prefix.

BEFORE-BROWSE-CALLNAT

The code in this exit defines any special processing performed before calling the browse subprogram. Use this exit with the Object-Maint-Dialog model.

Tip: If you encounter an “unclaimed user exit” error message when generating a dialog that uses this user exit, ensure that you specified a browse subprogram name on the Additional Parameters panel for the Object-Maint-Dialog model.

BEFORE-BROWSE-OBJECT

The code in this exit defines the processing performed before calling the browse object.

BEFORE-BT-PROCESSING

The code in this exit defines the processing performed before the transaction is backed out.

BEFORE-CALLNAT-SUBPROGRAMS

The code in this exit defines any additional processing performed before the Natural subprograms are called.

BEFORE-CALL-OBJECT

The code in this exit defines any special processing performed before the target subprogram is called. Use this exit with the Subprogram-Proxy model to perform any functions necessary prior to invoking the target module.

Tip: If it is necessary to call the object multiple times, use this exit to code an opening loop statement. Code the close loop statement in the AFTER-CALL-OBJECT user exit. For information, see **AFTER-CALL-OBJECT**, page 329.

BEFORE-CALL-TO-MAINT-OBJECT

The code in this exit defines the processing performed before the maintenance object is called by an object browse select subprogram. It is executed once for each row in the PDA.

Note: This user exit is only available if the object browse select subprogram accesses an object maintenance subprogram.

BEFORE-CHECK-BUSINESS-ERROR

The code in this exit is executed before a business error is processed. A business error applies to all data in the object.

BEFORE-CHECK-ERROR

The code in this exit defines the processing performed when an error condition is encountered within a generated module. Because an error condition will bypass the END-OF-PROGRAM user exit, use this exit if processing is required before leaving the program when an error condition occurs.

Example of code in the BEFORE-CHECK-ERROR user exit

```
1320 **SAG DEFINE EXIT BEFORE-CHECK-ERROR
1330 *
1340 * Use this user exit for specific error checking
1350   IF CSASTD.RETURN-CODE = CSLRCODE.#INTERRUPT(*)
1360     ASSIGN CU--PDA.#PDA-PHASE = #SAVE-PHASE
1370   END-IF
1380  **SAG END-EXIT
```

BEFORE-CHECK-PFKEYS

The code in this exit defines any additional processing performed before the PF-keys are checked.

BEFORE-COMPRESS-OUTPUT

The code in this exit defines any special processing performed before data in the data structures for the target subprogram is mapped to the data string returned to the server. Use this exit with the Subprogram-Proxy model to modify the contents of the local data structures prior to their contents being assigned back into the data stream after the CALLNAT statement.

BEFORE-CONFIRMATION

The code in this exit defines the processing performed before invoking the INPUT statement to confirm termination of the program.

BEFORE-ET

The code in this exit is executed immediately prior to processing the END OF TRANSACTION (ET) statement.

Maint Model

The code in this exit is executed when #UPDATE-PERFORMED is True.

Object-Maint-Subp Model

The code in this exit is executed whether CDAOBJ2.#ET-IF-SUCCESSFUL is True or False. This allows one last opportunity to modify the value of #ET-IF-SUCCESSFUL and change whether data is committed to the database.

Note: If the BEFORE-ET and BEFORE-ET-PROCESSING user exits are both defined and CDAOBJ2.#ET-IF-SUCCESSFUL is False, only the code in the BEFORE-ET exit will be executed.

BEFORE-ET-PROCESSING

The code in this exit is executed immediately prior to processing the END OF TRANSACTION (ET) statement and only when CDAOBJ2.#ET-IF-SUCCESSFUL is True. You can use this exit to perform special processing, such as integrity checking, that cannot be done before issuing the ET statement.

BEFORE-EXPAND-INPUT

The code in this exit defines any special processing performed before the data string received from the server is mapped to the data structures used to call the target subprogram. Use this exit with the Subprogram-Proxy model.

BEFORE-FETCH

The code in this exit is executed before the main menu program is FETCHed.

BEFORE-INITIAL-INPUT

The code in this exit defines any additional processing performed before the INPUT statement is executed.

BEFORE-INPUT

The code in this exit is executed immediately before the INPUT statement is processed in the AT END OF PAGE section. You can use this exit to lookup a code table (to display a description, as well as a code value), to issue SET CONTROL statements, or to capture or default map variables prior to displaying each panel.

Example of user exit code for the Browse-Select-Subp model

```
0010 DEFINE EXIT BEFORE-INPUT
0020 *
0030 * Processing to be performed before the INPUT statement.
0040 * Change standard message to indicate selection can be done ONLY
0050 * by positioning the cursor (not entering key value since input is
0060 * protected).
0070   ASSIGN MSG-INFO.##MSG = 'Position cursor to select.'
0080 END-EXIT BEFORE-INPUT
```

Example of user exit code for the Menu model

```
0010 DEFINE EXIT BEFORE-INPUT
0020 *
0030 * Processing to be performed before each INPUT statement.
0040   SET CONTROL 'WB'   /* Restore window size to physical screen size.
0050 END-EXIT BEFORE-INPUT
```

Example of user exit code for the Object-Maint-Dialog model

```
0010 DEFINE EXIT BEFORE-INPUT
0020 *
0030 * If order lines were scrolled, set distributions array to 1
0040   IF #LAST-ARRAY1 NE #ARRAY1 THEN
0050     ASSIGN #ARRAY2 = #NEXT-ARRAY2 = #CURR-INDEX(#PANEL,2) = 1
0060   END-IF
0070   ASSIGN #LAST-ARRAY1 = #ARRAY1
0080   /*
0090   /* Update total for the order
0100   COMPUTE ORDER.ORDER-AMOUNT = 0 + ORDER.TOTAL-COST(*)
0110 END-EXIT BEFORE-INPUT
```

BEFORE-OBJECT-CALL

The code in this exit defines the processing performed before an object is called.

BEFORE-PROCESS-ACTIONS

The code in this exit defines the processing performed before actions are processed.

BEFORE-PROCESSING-MENU-CODES

The code in this exit is executed before a menu code value is processed.

BEFORE-RANGE-INPUT

The code in this exit is executed before the INPUT statement is processed for the starting and ending values for the primary file.

BEFORE-READ

The code in this exit is executed before the READ loop is initiated.

BEFORE-RESUMING-PROCESSING

The code in this exit is performed when a user cancels a quit (termination) request.

BEFORE-ROW-ASSIGNMENT

The code in this exit is executed immediately prior to copying the file view to the object (Row) PDA. To reject this record, execute an ESCAPE ROUTINE statement.

Example of user exit code for the Object-Browse-Subp model

```
0010 DEFINE EXIT BEFORE-ROW-ASSIGNMENT
0020 *
0030 * Don't return employee records if they have zero in the leave-taken
0040 * and leave due fields.
0050 IF EMPLOYEES.LEAVE-TAKEN = 0 AND EMPLOYEES.LEAVE-DUE = 0 THEN
0060   ESCAPE ROUTINE
0070 END-IF
0080 **
0090 ** Reclassify divorced and widowed people as single
0100 IF EMPLOYEES.MARITAL-STATUS = 'D' OR = 'W' THEN
0110   ASSIGN EMPLOYEES.MARITAL-STATUS = 'S'
0120 END-IF
0130 END-EXIT
```

Note: Avoid rejecting a significant percentage of records, or a large number of consecutive records, by executing the ESCAPE ROUTINE statement within the BEFORE-ROW-ASSIGNMENT user exit, due to the negative impact this will have on performance.

BEFORE-STANDARD-KEY-CHECK

The code in this user exit checks any additional PF-keys defined for the modify subprogram, or prepares for standard PF-key validations.

Example of code in the BEFORE-STANDARD-KEY-CHECK user exit

```
DEFINE EXIT BEFORE-STANDARD-KEY-CHECK
*
* Use this user exit to check additional PF-keys or prepare for the
* standard PF-key check.
END-EXIT BEFORE-STANDARD-KEY-CHECK
```

BROWSE-ACTION-PROCESSING

The code in this exit specifies additional processing performed for the Browse action (in addition to the standard Natural Construct-generated processing).

Tip: If you encounter an “unclaimed user exit” error message when generating a dialog that uses this user exit, ensure that you specified a browse subprogram name on the Additional Parameters panel for the Object-Maint-Dialog model.

BUILD-REPORT-LOCAL-VARS

This user exit contains control variables or indices (indexes). It is generated in conjunction with the WRITE-FIELDS user exit to write arrays (or MUs/PEs) to a report.

CHANGE-HISTORY

This user exit keeps a record of changes to the generated module. It generates comment lines indicating the date, the user ID of the user who created or modified the module, and a description of any change.

Example of code in the CHANGE-HISTORY user exit

```
DEFINE EXIT CHANGE-HISTORY
* Changed on Aug 27,08 by DEVPR for release ____
* >
* >
* >
END-EXIT CHANGE-HISTORY
```

CLEAR-ACTION-PROCESSING

The code in this exit specifies additional processing performed for the Clear action (in addition to the standard Natural Construct-generated processing).

CLIENT-VALIDATIONS

The code in this exit specifies client validations for modules generated using the VB-Maint-Object model. See the following page for an example using this user exit.

Example of code in the CLIENT-VALIDATIONS user exit

```
DEFINE EXIT CLIENT-VALIDATIONS
  Select Case fs.FieldName
      '
      ' Local validation code example:
      ' Case <data field name>
      '     If <error condition> Then
      '        Err.Raise Number:=csterrValueIsRequired, _
      '                  Source:=ERROR_SOURCE_VALIDATE, _
      '                  Description:=<error description>
      '     End If
  Case "CUSTOMER-NUMBER"
  Case "BUSINESS-NAME"
  Case "PHONE-NUMBER"
  Case "M-STREET"
  Case "M-CITY"
  Case "M-PROVINCE"
     '
' See Rule: NCST-PROVINCE
  Case "M-POSTAL-CODE"
  Case "S-STREET"
  Case "S-CITY"
  Case "S-PROVINCE"
      '
      ' See Rule: NCST-PROVINCE
   Case "S-POSTAL-CODE"
   Case "CONTACT"
   Case "CREDIT-RATING"
   Case "CREDIT-LIMIT"
     '
     ' See Rule: NCST-CREDIT-LIMIT
  Case "DISCOUNT-PERCENTAGE"
  Case "CUSTOMER-WAREHOUSE-ID"
  Case "CUSTOMER-TIMESTAMP"
  End Select
END-EXIT
```

Note: You can also add the code to the Validate function in the Visual Basic module and then upload the module to the server. The code is preserved during regeneration.

CONSTANTS-AND-TYPES

The code in this exit defines any special processing performed to declare constants and types for a Visual Basic maintenance object. Use this exit with the VB-Maint-Object model.

COPY-ACTION-PROCESSING

The code in this exit specifies additional processing performed for the Copy action (in addition to the standard Natural Construct-generated processing).

DEFAULT-TRANSACTION-STYLE

The code in this exit allows the programmer to override the default transaction style for an object browse select subprogram.

Note: This user exit is only available if the object browse select subprogram accesses an object maintenance subprogram.

DEFINE-CUSTOM-LOCAL-METHODS

The code in this exit defines customized local methods for a Visual Basic maintenance object. Use this exit with the VB-Maint-Object model.

DEFINE-REPORT-PRINTER

The code in this exit defines the name or path name for the printer to which reports are routed.

DEFINE-TRANSLATION-HEADERS

The code in this exit defines the panel headings used by the generated module to support cursor translation.

Note: If the generated module supports cursor translation, you must define this exit.

DELETE-EDITS

The code in this exit validates the contents of an object record before the record is deleted. This exit must contain a series of subroutines named D-*entity-name,* where *entity-name* is a valid entity within the object. The subroutine is performed once for each occurrence of the corresponding entity, immediately before the entity is deleted.

Because object values are copied to the corresponding file view using the MOVE BY NAME statement before the D subroutines are performed, data validation in the D subroutine should be made against the record buffer of the file view. Only refer directly to the object when the relevant fields have different names in the object PDA and the file.

Object-Maint-Subp Model

The Object-Maint-Subp model generates the #L2, #L3, and #L4 index variables to access the occurrences of the second, third, and fourth level entities within the object PDA. These indices must not be modified by the user exit code. However, the user exit code can access the object PDA fields by referencing the appropriate index tuples that correspond to the second, third, or fourth level entities (#L2, or #L2,#L3, or #L2,#L3,#L4). If subroutines encounter invalid data within an object, they should assign values to the variables in the CDPDA-M parameter data area to terminate the process.

Example of using the subroutines in the DELETE-EDITS user exit

```
0010 DEFINE-EXIT DELETE-EDITS
0020           DEFINE SUBROUTINE D-INS-POLICY
0030             IF INS-POLICY.EXPIRY-DATE GE *DATX THEN
0040              ASSIGN MSG-INFO.##MSG = 'Cannot delete active policy'
0050              ASSIGN MSG-INFO.##ERROR-FIELD = 'EXPIRY-DATE'
0060               ESCAPE ROUTINE
0070             END-IF
0080           END-SUBROUTINE
0090 END-EXIT DELETE-EDITS
```

Note: The subroutines in this exit should assign the name of the object PDA field in error to the MSG-INFO.##ERROR-FIELD variable and a message to the MSG-INFO.##MSG or MSG-INFO.##MSG-NR variable.

DISPLAY-ACTION-PROCESSING

The code in this exit specifies additional processing performed for the Display action (in addition to the standard Natural Construct-generated processing).

ERROR-MESSAGE-PDAS

Use this exit to change the error message and/or processing for an object maintenance subprogram based on the error returned.

END-BUSINESS-SERVICE

The code in this exit defines the processing performed before the business service passes back the *n* rows of data to the client.

END-OF-PROGRAM

The code in this exit is executed once before the module is terminated. You can use this exit for any cleanup required (such as assigning a termination message or resetting windows, for example) before exiting the module.

If an error condition occurs, this user exit will not be executed. Use the BEFORE-CHECK-ERROR user exit if processing is required before leaving the program.

Tip: You can assign the current key value to a global variable in this exit, so it is carried into other modules that use the same key.

Example of user exit code for the Object-Maint-Subp model

```
0010 DEFINE EXIT END-OF-PROGRAM
0020 FOR #I = 1 TO 3
0030 * Strip Ncst off of file name references in messages.
0040  IF MSG-INFO.##MSG-DATA(#I) = MASK('Ncst ') THEN
0050   RESET MSG-INFO.##MSG-DATA-CHAR(#I,1:4)
0060   MOVE LEFT MSG-INFO.##MSG-DATA(#I) TO MSG-INFO.##MSG-DATA(#I)
0070  END-IF
0080 END-FOR
0090 END-EXIT END-OF-PROGRAM
```

ERROR-ROUTINE

The code in this exit specifies processing performed when an error occurs during the execution of a generated batch program (the input end key is smaller than the input start key, for example).

EXPORT-COLUMN-HEADERS

The code in this exit defines the processing performed to export selected column headings to a PC file.

EXPORT-DATA

The code in this exit defines parameters for export to a work file. The export parameters can be the same or different from the browse parameters.

If you mark the Export data support field and do not select this exit, only the primary key data is available for export. If you mark the field and select this exit, a series of windows is displayed to select the fields for export.

Note: If you are using Entire Connection, you can export the work file directly to a PC text file, which you can then import into any PC spreadsheet.

EXPORT-DATA-FIELDS

The code in this exit defines the processing performed to export selected fields to a PC file.

Note: Natural Construct provides a user exit model you can use to generate (and regenerate) the layout of fields and column headers to be exported to an ASCII file on a PC. For information, see **Object-Browse Models**, page 293.

EXTENDED-RI-CHECKS

The code in this exit performs further validations after a referential integrity check. For example, after relationship checking verifies that a record exists for the customer number specified in the order header, you can check that the customer is flagged as active and is in good credit standing. To verify these concerns, this exit must contain a V-*relationship-name* subroutine, where *relationship-name* is a valid relationship used by an object to check for referential integrity.

This exit contains the V-*relationship-name* validation routine, which is executed during the pre-editing phase after the Predict automatic rules are checked and after the V1-*entity-name* subroutine for the current entity has been executed.

The V-*relationship-name* subroutine is invoked from the appropriate E-*entity-name* routine or, in the case of the root entity, from the EDIT-OBJECT subroutine.

Note: For more information, see **Object-Maint Models**, page 297.

EXTENDED-RI-VIEWS

This user exit contains the views required by the validation routines defined in the EXTENDED-RI-CHECKS user exit. It is generated automatically when you select the EXTENDED-RI-CHECKS user exit and specify fields to be included in the referential integrity check views.

EXTEND-SELECTION-TABLE

Use this exit to assign user-defined values to the selection table for the Browse-Select models.

FINAL-PROCESSING

The code in this exit defines the processing performed immediately before leaving the module.

FOREIGN-KEY-OVERRIDE

The code in this exit overrides the browse key for a foreign field on a Visual Basic maintenance dialog. Use this exit with the VB-Maint-Object model.

FORMER-ACTION-PROCESSING

The code in this exit specifies additional processing performed for the Former action (in addition to the standard Natural Construct-generated processing).

HARDCOPY-EDITS

The code in this exit is executed after the INPUT statement is executed for the hardcopy screen options. You can use this exit to specify edit checks on the values specified for hardcopy device, lines per page, and page size.

HARDCOPY-TERMINATING-PROCESS

The code in this exit is executed after selected records are printed. You can use this exit to specify any processing required when terminating the hardcopy function.

HE-PARAMETER-INDEXES

The code in this exit defines additional parameters to receive indices for a multi-dimensional key (passed parameter). For example, if you are generating a browse helproutine or subprogram that is attached to an array field and you need to know the occurrence for which help was requested, you can define the index values for each dimension within this exit.

Example of user exit code for the Browse-Select models

```
0010 DEFINE EXIT HE-PARAMETER-INDEXES
0020 01  #HE-FLD-INDEX1 (I2)     /* Index for first dimension
0030   02  #HE-FLD-INDEX2 (I2)   /* Index for second dimension
0040   02  #HE-FLD-INDEX3 (I2)   /* Index for third dimension
0050 END-EXIT HE-PARAMETER-INDEXES
```

INITIALIZE-SEARCH-KEYS

The code in this exit initializes search keys for a Visual Basic browse object. Use this exit with the VB-Browse-Object model.

Example of code in the INITIALIZE-SEARCH-KEYS user exit

```
DEFINE EXIT INITIALIZE-SEARCH-KEYS
   'With m_BrowseBase.SearchKeys.Item(<logicalkeyname>)
       ' .Item(<searchkeyname>)
       ' .FieldBDT = <BDT name here>
   'End With
END-EXIT
```

INPUT-KEY

The code in this exit defines the input key used by a generated module to INPUT selected fields.

Note: Natural Construct provides a user exit model you can use to generate (and regenerate) the input fields and prompts for inputting data. For information, see **Object-Browse Models**, page 293.

INSERT-ROWS

The code in this exit inserts rows for a Visual Basic browse or local data object. Use this exit with the VB-Browse-Object and VB-Browse-Local-Data-Object models.

Example of user exit code for the VB-Browse-Object model

```
DEFINE EXIT INSERT-ROWS
' AddData parameters are:
' 1)... n) The number of parameters depends on the number of columns
'   that have been specified in the InitColumns sub.
'   You must pass one parameter to AddData for each column specified
'   in the InitColumns sub.
'
'   Note: Only scalar columns are supported -- multi-dimensional
'         columns are not supported.
'       AddData <CUSTOMER-NUMBER>,<BUSINESS-NAME>,<PHONE-NUMBER>,<M-ST
'               <M-CITY>,<M-PROVINCE>,<M-POSTAL-CODE>,<S-STREET>, _
'               <S-CITY>,<S-PROVINCE>,<S-POSTAL-CODE>,<CONTACT>, _
'              <CREDIT-RATING>,<CREDIT-LIMIT>,<DISCOUNT-PERCENTAGE>,<
'                <CUSTOMER-TIMESTAMP>,<UNIQUE-ID>
END-EXIT
```

Example of user exit code for the VB-Browse-Local-Data-Object model

```
DEFINE EXIT INSERT-ROWS
       AddData "1", "<field1>", "<field2>", etc.
       AddData "2", "<field1>", "<field2>", etc.
 END-EXIT
```

INVOKE-CUSTOM-LOCAL-METHODS

The code in this exit invokes customized local methods for a Visual Basic maintenance object. Use this exit with the VB-Maint-Object model.

LOCAL-DATA

The code in this exit defines additional local variables that are used in conjunction with other user exits. If you use this exit with a Browse, Browse-Select, or Object-Browse model, a window is displayed in which you can define additional local data.

Note: Except for PDAs required by the object maintenance subprogram, never define Natural Construct-generated LDAs or PDAs in the LOCAL-DATA or PARAMETER-DATA user exit. If you are listing the LDAs required for the CALLNAT from the object maintenance dialog to the object maintenance subprogram in your comments, do not preface the comments with "USING" (for example, "**USING *xxx*pda").

Using the LOCAL-DATA User Exit Window

If you specify a program view name on a Browse/Browse-Select/Object-Browse model specification panel, you must define the view for the file in the LOCAL-DATA user exit.

When you select the LOCAL-DATA user exit, the following window is displayed:

```
CUSCSLDA              Natural Construct
Aug 29              LOCAL-DATA User Exit              1 of 1

Define View of Primary File ..  _
Define Local Using Data Area .  ________
```

LOCAL-DATA User Exit Window

To define the entire primary file view within a user exit, mark the Define View of Primary File field and press Enter. You can then edit the sample code and delete the fields you do not want.

If the view is defined in a local data area (LDA), enter the name of the LDA in the Define Local Using Data Area field.

Note: If you are using the LOCAL-DATA user exit with a model that is not a Browse, Browse-Select, or Object-Browse model, no window is displayed.

Example of user exit code for the LOCAL-DATA user exit

```
0010 DEFINE EXIT LOCAL-DATA
0020   LOCAL
0030   01 #CITY-PROVINCE(A50)
0040   01 NCST-CUSTOMER VIEW OF NCST-CUSTOMER
0050     02 CUSTOMER-NUMBER
0060     02 BUSINESS-NAME
0070     02 PHONE-NUMBER
0080     02 SHIPPING-ADDRESS
0090       03 S-STREET
0100       03 S-CITY
0110       03 S-PROVINCE
0120       03 S-POSTAL-CODE
0130     02 CONTACT
0140     02 CREDIT-RATING
0150     02 CREDIT-LIMIT
0160 END-EXIT LOCAL-DATA
```

Example of user exit code for the Browse-Select model

```
0010 DEFINE EXIT LOCAL-DATA
0020   01 NCSTDB2-CUSTOMER-PROGRAM-VIEW VIEW OF NCSTDB2-CUSTOMER
0030   02 CUSTOMER_NUMBER
0040   02 BUSINESS_NAME
0050   02 PHONE_NUMBER
0060   02 M_STREET
0070   02 M_CITY
0075   02 N@M_CITY
0080   02 REDEFINE N@M_CITY
0090       03 FILLER-90(A1)
0100       03 N#M_CITY(L)
0110   02 M_PROVINCE
0120   02 M_POSTAL CODE
0130   02 S_STREET
0140   02 S_CITY
0150   02 S_PROVINCE
0160   02 S_POSTAL CODE
0170   02 CONTACT
0180   02 PROVINCE
0190   02 M_CITY
0200   02 N@M_CITY
0210   02 REDEFINE N@M_CITY
0220       03 FILLER-90(A1)
0230       03 N#M_CITY(L)
0240   02 M_PROVINCE
0250   02 M_POSTAL CODE
0260   02 S_STREET
0270   02 S_CITY
0280   02 S_PROVINCE
0290   02 S_POSTAL CODE
0300   02 CONTACT
0310   02 CREDIT_RATING
0320   02 CREDIT_LIMIT
0330   02 DISCOUNT_PERCENTAG
0340   02 CUSTOMER_WAREHOUSE
0350   02 LOG_COUNTER
0360 END-EXIT LOCAL-DATA
```

MISCELLANEOUS-SUBROUTINES

The code in this exit defines any subroutines invoked within your user exit code. It is placed immediately before the END statement in the generated module.

Example of code in the MISCELLANEOUS-SUBROUTINES user exit

```
DEFINE EXIT MISCELLANEOUS-SUBROUTINES
**
************************************************************************
DEFINE SUBROUTINE some-subroutine
************************************************************************
**
  ESCAPE ROUTINE IMMEDIATE
END-SUBROUTINE /* some-subroutine
END-EXIT MISCELLANEOUS-SUBROUTINES
```

MODIFY-ACTION-PROCESSING

The code in this exit specifies additional processing performed for the Modify action (in addition to the standard Natural Construct-generated processing).

NEXT-ACTION-PROCESSING

The code in this exit specifies additional processing performed for the Next action (in addition to the standard Natural Construct-generated processing).

ON-ERROR-MSG-NR

Use this exit to code special handling for runtime errors generated by the subprogram proxy or any of the modules executed because of the target module call. The subprogram proxy automatically handles errors that occur due to invalid data passed from the client.

Note: This exit is placed in the generated code prior to any error handling done by the subprogram proxy. This allows you to override the automatic handling of format errors, as well as add additional error handling.

OVERRIDE-MAXIMUM

Use this exit to override the MIN-MAX values of MOVE BY NAME structures in secondary files. This exit is used by the Object-Maint-Subp model to allow developers to further restrict the rows returned from secondary files.

After generation, the object maintenance subprogram contains the following code:

```
#CURRENT-RELATIONSHIP := 'NCST-ORDER-HAS-LINES'
   MOVE ALL H'00' TO #NCST-ORDER-HAS-LINES.MIN-MAX
   PERFORM OVERRIDE-MINIMUM
   ASSIGN #FROM.NCST-ORDER-HAS-LINES = #NCST-ORDER-HAS-LINES
   MOVE ALL H'FF' TO #NCST-ORDER-HAS-LINES.MIN-MAX
   PERFORM OVERRIDE-MAXIMUM
   ASSIGN #THRU.NCST-ORDER-HAS-LINES = #NCST-ORDER-HAS-LINES
   READ NCST-ORDER-LINES BY ORDER-LINE KEY
                        FROM #FROM.NCST-ORDER-HAS-LINES
```

Use this exit to specify the maximum key value.

Example of code in the OVERRIDE-MAXIMUM user exit

```
DEFINE EXIT OVERRIDE-MAXIMUM
** You can override the MIN-MAX values of MOVE BY NAME structures
** of the secondary files in this USER-EXIT.
**
DECIDE ON FIRST VALUE OF #CURRENT-RELATIONSHIP
VALUE 'NCST-ORDER-HAS-LINES'
  MOVE ALL H'XX' TO #NCST-ORDER-HAS-LINES.MIN-MAX  /* example override
for maximum
VALUE NONE
  IGNORE
END-DECIDE
END-EXIT
```

OVERRIDE-MINIMUM

Use this exit to override the MIN-MAX values of MOVE BY NAME structures in secondary files. This exit is used by the Object-Maint-Subp model to allow developers to further restrict the rows returned from secondary files.

After generation, the object maintenance subprogram contains the following code:

```
#CURRENT-RELATIONSHIP := 'NCST-ORDER-HAS-LINES'
   MOVE ALL H'00' TO #NCST-ORDER-HAS-LINES.MIN-MAX
   PERFORM OVERRIDE-MINIMUM
   ASSIGN #FROM.NCST-ORDER-HAS-LINES = #NCST-ORDER-HAS-LINES
   MOVE ALL H'FF' TO #NCST-ORDER-HAS-LINES.MIN-MAX
   PERFORM OVERRIDE-MAXIMUM
   ASSIGN #THRU.NCST-ORDER-HAS-LINES = #NCST-ORDER-HAS-LINES
   READ NCST-ORDER-LINES BY ORDER-LINE KEY
                        FROM #FROM.NCST-ORDER-HAS-LINES
```

Use this exit to specify the minimum key value.

Example of code in the OVERRIDE-MINIMUM user exit

```
DEFINE EXIT OVERRIDE-MINIMUM
** You can override the MIN-MAX values of MOVE BY NAME structures
** of the secondary files in this USER-EXIT.
**
DECIDE ON FIRST VALUE OF #CURRENT-RELATIONSHIP
VALUE 'NCST-ORDER-HAS-LINES'
MOVE ALL H'BB' TO #NCST-ORDER-HAS-LINES.MIN-MAX /* example override
                                                   for minimum
VALUE NONE
  IGNORE
END-DECIDE
END-EXIT
```

PARAMETER-DATA

This user exit defines additional parameters used in conjunction with other programs.

Note: Except for PDAs required by the object maintenance subprogram, never define Natural Construct-generated LDAs or PDAs in the LOCAL-DATA or PARAMETER-DATA user exit. If you are listing the LDAs required for the CALLNAT from the object maintenance dialog to the object maintenance subprogram in your comments, do not preface the comments with "USING" (for example, "**USING *xxx*pda").

Note: The PARAMETER-DATA user exit must be specified for the Object-Generic-Subp model. For more information, see **Object-Generic-Subp Model**, page 295.

Example of code in the PARAMETER-DATA user exit

```
DEFINE EXIT PARAMETER-DATA
** PARAMETER USING PDAname
** PARAMETER
**  01 #Additional-parameter1
**  01 #Additional-parameter2
END-EXIT PARAMETER-DATA
```

PARAMETER-ROW

The code in this exit defines additional parameters for each row.

PRIME-WRITE-FIELDS

The code in this exit writes information to a specified report via WRITE, DISPLAY, or PRINT statements. When you mark the PRIME-WRITE-FIELDS user exit and press Enter, the Building User Exit PRIME-WRITE-FIELDS window is displayed.

Building User Exit PRIME-WRITE-FIELDS Window

```
CUBASRP                 Natural Construct                 CUBASRP0
Oct 28           PRIME-WRITE-FIELDS Build Report            1 of 1

   Prt              Printer Name            LS   PS   ZP   IS   ES
-------- -------------------------------- ---- ---- ---- ---- ----
    _ 0    Terminal screen                  80   23    X
    _ 1                                     133  60    X
    _ 2                                     133  60    X
    _ 3                                     133  60    X
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--
      help  retrn
```

Building User Exit PRIME-WRITE-FIELDS Window

This window lists the print file(s) specified on the model specification panel. Mark the print file you want to build the report and press Enter. A series of windows is displayed. For a description of these windows, see **WRITE-FIELDS**, page 379.

Example of user exit code for the Batch model

```
0010 DEFINE EXIT PRIME-WRITE-FIELDS
0020     WRITE(1) '='(131)
0030     WRITE(1)
0040       'Order Number:' NCST-ORDER-HEADER.ORDER-NUMBER
0050       'Date:' NCST-ORDER-HEADER.ORDER-DATE
0060       'Amount:' NCST-ORDER-HEADER.ORDER-AMOUNT
0070     SKIP(2) 1
0080     WRITE(2) '='(131)
0090     WRITE(2)
0100       'Order Number:' NCST-ORDER-HEADER.ORDER-NUMBER
0110       'Cust. Numb:' NCST-ORDER-HEADER.ORDER-CUSTOMER-NUMBER
0120       'Date:' NCST-ORDER-HEADER.ORDER-DATE
0130       'Amount:' NCST-ORDER-HEADER.ORDER-AMOUNT
0140 END-EXIT PRIME-WRITE-FIELDS
```

PRIME-WRITE-HEADINGS

The code in this exit writes column headings (specified for the primary file) to the specified report(s). If different reports use different primary data, generate this user exit for each report.

PRIVATE-INSTANCE-VARIABLES

The code in this exit defines any special processing performed to declare private instance variables for a Visual Basic maintenance object. Use this exit with the VB-Maint-Object model.

PRIVATE-PROCEDURES

The code in this exit defines private procedures for a Visual Basic maintenance object. Use this exit with the VB-Maint-Object model.

PROCESS-ERROR-MESSAGE

Use this exit to override the error routine for an object maintenance subprogram that uses message numbers. You can change the default error routine (CDUMSGU) to an error routine of your choice (for example, MYERROR).

PROCESS-SELECTED-RECORD

Browse Models

The code in this exit is performed when a record is selected (either by using the cursor or by entering a key value for the record while the record is displayed). You can use this exit to specify processing performed on the selected record.

The #SELECTED-KEY variable contains the key value for the selected record. The #SELECTED-ISN variable contains the ISNs (Internal Sequence Numbers) of the selected record (for Adabas files/user views). In non-Adabas files, the #SELECTED-UQ variable contains the value of the unique field for the selected record. The #SELECTED-UQ variable is generated only if a field is marked as unique in Predict. For more information, see **Natural DB2 and SQL/DS Users**, page 50.

Example of user exit code for the Browse-Subp model

```
0010 DEFINE EXIT PROCESS-SELECTED-RECORD
0020 **
0030 ** Processing to be performed after a record has been
0040 ** selected.
0050 IF *PF-KEY = 'ENTR' THEN
0060   ASSIGN #PDA-KEY = #SELECTED-KEY    /* Return selected value.
0070 ** Action field is used if this is a maintenance browse
0080 ** subprogram.
0090   ASSIGN CDSELPDA.SELECTED-FUNCTION = 'D'
0100 ** Display the selected record.
0110   ESCAPE BOTTOM(PROG.) IMMEDIATE
0120 END-IF
0130 END-EXIT PROCESS-SELECTED-RECORD
```

Browse-Select Models

The code in this exit is performed once for each record marked with a valid action. If any actions are selected, this user exit is required. Define the processing for a selected record based on the action code.

The #ACTION variable contains the action code entered by a user. The #SELECTED-KEY variable contains the key value for the selected record.

For Adabas files/user views, the #SELECTED-ISN variable contains the ISNs (Internal Sequence Numbers) for the selected record. For non-Adabas files, the #SELECTED-UQ variable contains the value of the unique field for the selected record. The #SELECTED-UQ variable is only generated if a field is marked as unique in Predict. For more information, see **Natural DB2 and SQL/DS Users**, page 50.

Example of user exit code for the Browse-Select model

```
0010 DEFINE EXIT PROCESS-SELECTED-RECORD
0020 * This user exit is invoked once for each record that is
0030 * marked with a valid action.
0040 * The following variables are set prior to invoking this
0050   CALLNAT 'NCOSELN'        /* CALLNAT the object subprogram to
0060            #SELECTED-KEY   /* process a record. The record's
0070            #ACTION         /* key is stored in #SELECTED-KEY.
0080            DIALOG-INFO     /* Apply action indicated in
0090            MSG-INFO        /* ACTION field.
0100            PASS
0110 END-EXIT PROCESS-SELECTED-RECORD
```

Example of user exit code for the Browse-Select-Helpr model

```
0010 DEFINE EXIT PROCESS-SELECTED-RECORD
0020 *
0030 * Processing to be performed after a record has been selected.
0040   IF #ACTION = #SELECT(*) THEN
0050     ASSIGN #PDA-KEY = #SELECTED-KEY
0060     CALLNAT 'MODCV' #SET-CV-TO-MODIFIED
0070 * Set passed CV variable to modified to activate lookup for
0080 * customer name in the ORDER maintenance program (calling
0090 * object)
0100     ESCAPE BOTTOM(PROG.) IMMEDIATE
0110   ELSE
0120     CALLNAT 'NCCMAINN'
0130            #SELECTED-KEY
0140            #ACTION
0150            DIALOG-INFO
0160            MSG-INFO
0170            PASS                        /* Defined in CDPDA-P
0180   END-IF
0190 END-EXIT PROCESS-SELECTED-RECORD
```

Example of user exit code for the Browse-Select-Subp model

```
0010 DEFINE EXIT PROCESS-SELECTED-RECORD
0020 *
0030 * Processing to be performed after a record has been selected.
0040 IF *PF-KEY = 'ENTR' THEN
0050   GET ORDER-DETAIL #SELECTED-ISN
0060   RESET MSG-INFO
0070   CALLNAT 'NCOSODET'
0080           ORDER-DETAIL.ORDER-NUMBER
0090           CDSELPDA.SELECTED-FUNCTION
0100           DIALOG-INFO
0110           MSG-INFO
0120           PASS
0130 END-IF
0140 END-EXIT PROCESS-SELECTED-RECORD
```

PUBLIC-METHODS

The code in this exit defines any special processing performed to declare Public methods for a Visual Basic maintenance object. Use this exit with the VB-Maint-Object model.

PUBLIC-PROPERTIES

The code in this exit defines any special processing performed to declare Public properties for a Visual Basic maintenance object. Use this exit with the VB-Maint-Object model.

PUBLIC-PROPERTY-PROCEDURES

The code in this exit defines any special processing performed to declare public property procedures for a Visual Basic maintenance object. Use this exit with the VB-Maint-Object model.

Example of code in the PUBLIC-PROPERTY-PROCEDURES user exit

```
DEFINE EXIT PUBLIC-PROPERTY-PROCEDURES
'
'================================================
' Public Property Procedures
'================================================
' Use this exit to declare Public Property Procedures.
'
END-EXIT
```

PURGE-ACTION-PROCESSING

The code in this exit specifies additional processing performed for the Purge action (in addition to the standard Natural Construct-generated processing).

READ-INPUT-CRITERIA

This exit is available for the Object-Browse-Subp model when a browse module has been transformed into object browse modules (by the Transform-Browse model) and the original browse module was generated with minimum and/or maximum values for key fields.

In general, this exit is not required but can be used to add additional keys to an object browse subprogram that was transformed from a browse module that had minimum and/or maximum key values.

Note: If minimum and/or maximum specifications are not required, we recommend that you do not include additional key fields in a transformed object browse subprogram because of the added complexity (it is easier to generate another object browse subprogram for the additional keys). However, if you want to add an additional key and take advantage of the minimum and/or maximum functionality, you can do this by supplying your own code or by using the sample code provided in this exit (by removing the comment indicators).

REINPUT-SCREEN

If #ERROR is set to TRUE, the code in this exit is executed automatically after the UPDATE-EDITS user exit is executed and the maps and scrolling lines are repositioned. For more information, see **UPDATE-EDITS**, page 373.

If the UPDATE-EDITS positioning technique is used, you can use the code in this exit to test the value of #ERROR-NR and issue the corresponding REINPUT (using the MARK field-name safely, since the positioning mechanism has guaranteed that the current map displays the field in error).

Object-Maint-Dialog Model

If an additional field is on a map that is not part of the object, include a REINPUT statement for edit checks on this field. The actual edit checks are included in another user exit, depending on the type of edit check being performed. (User exits that might include the actual edit checks include the AFTER-INPUT, AFTER-GET, and ACTION-PROCESSING user exits, such as ADD-ACTION-PROCESSING, BROWSE-ACTION-PROCESSING, etc.)

REJECT-AFTER-MAX-KEY-CHECK

The code in this exit defines processing performed after the minimum or maximum key value is rejected in a browse or browse-select program. It provides more efficient code when processing minimum and maximum key values.

Note: The AFTER-READ user exit is similar to this exit, except it is generated before the minimum or maximum key value is checked. For information, see **AFTER-READ**, page 334.

REPORT*n*-AT-TOP-OF-PAGE

The code in this exit performs additional processing when a new page of report *n* is started, where *n* is a number from 0 to 3.

REPORT-COLUMN-HEADERS

The code in this exit defines the column headings displayed on a printed report.

REPORT-DATA-FIELDS

The code in this exit defines the processing performed to route selected fields to a printer.

Note: Natural Construct provides a user exit model you can use to generate (and regenerate) the layout of fields and column headers to be routed to a printer. For information, see **Object-Browse Models**, page 293.

REPORT-HEADERS

The code in this exit defines the field headings displayed on a printed report.

ROW-STATE-INPUT-CONVERSION

The code in this exit defines the processing performed to convert the input for a row state into an internal method, which is required if user-defined row states are used.

Note: This user exit is only available if the object browse select subprogram accesses an object maintenance subprogram.

SCREEN-HEADERS

The code in this exit defines the panel headings displayed on a generated panel.

SEC1-WRITE-FIELDS

The code in this exit includes statements (WRITE, DISPLAY, or PRINT) in your batch program that write information to specified reports. The specification panels are the same as those for the WRITE-FIELDS user exit. For a description of these panels, see **WRITE-FIELDS**, page 379.

Note: Use this exit to define write fields contained in the primary file, first secondary file, or related files.

Example of user exit code for the Batch model

```
0010 DEFINE EXIT SEC1-WRITE-FIELDS
0020     WRITE(1)
0030       2X NCST-ORDER-LINES.ORDER-PRODUCT-ID
0040       3X NCST-ORDER-LINES.LINE-DESCRIPTION
0050       NCST-ORDER-LINES.QUANTITY
0060       NCST-ORDER-LINES.UNIT-COST
0070       NCST-ORDER-LINES.TOTAL-COST
0080 END-EXIT SEC1-WRITE-FIELDS
```

SEC1-WRITE-HEADINGS

The code in this exit writes column headings for secondary file 1 to the specified report(s). If different reports use different secondary data, generate this user exit for each report.

Example of user exit code for the Batch model

```
0010 DEFINE EXIT SEC1-WRITE-HEADINGS
0020  WRITE(1)
0030   'Product' (I) 1X 'Id' (I) 15X 'Description' (I) 16X
0040   'Quantity' (I) 2X 'Unit' (I) 1X 'Cost' (I) 3X 'Total' (I) 1X
0050       'Cost' (I)
0060     WRITE(1)
0070       '------' 1X '--------------------
0080       '----' 1X '------' 1X '-------'
0090 END-EXIT SEC1-WRITE-HEADINGS
```

SEC2-WRITE-FIELDS

The code in this exit defines statements (WRITE, DISPLAY, or PRINT) that write information to specified reports. The specification panels are the same as those for the WRITE-FIELDS user exit (for a description of these panels, see **WRITE-FIELDS**, page 379).

Note: Use this exit to define write fields contained in the primary file, first secondary file, or related files.

Example of user exit code for the Batch model

```
0010 DEFINE EXIT SEC2-WRITE-FIELDS
0020     WRITE(2)
0030       NCST-CUSTOMER.BUSINESS-NAME
0040       NCST-CUSTOMER.PHONE-NUMBER
0050       NCST-CUSTOMER.M-STREET
0060       NCST-CUSTOMER.M-CITY
0070       NCST-CUSTOMER.M-PROVINCE
0080         (AL=10)
0090       NCST-CUSTOMER.M-POSTAL-CODE
0100 END-EXIT SEC2-WRITE-FIELDS
```

SEC2-WRITE-HEADINGS

The code in this exit writes column headings for secondary file 2 to specified report(s). If different reports use different secondary data, generate this user exit for each report.

Example of user exit code for the Batch model

```
0010 DEFINE EXIT SEC2-WRITE-HEADINGS
0020  WRITE(2)
0030       34X 'Phone' (I) 63X 'Postal' (I)
0040  WRITE(2)
0050   8X 'Business' (I) 1X 'Name' (I) 13X 'Number' (I) 13X
0060   'Street' (I) 19X 'City' (I) 10X 'Province' (I) 3X 'Code' (I)
0070  WRITE(2)
0080   '————————————————' 1X '——————' 1X
0090   '————————————' 1X '——————————
0100   '——————' 1X '———'
0110 END-EXIT SEC2-WRITE-HEADINGS
```

SELECT-ADDITIONAL-ACTIONS

The code in this exit defines any additional action codes to appear as valid actions in the actions list. You must add the action code to the CDACTA parameter data area and to the #CODES table in CDACT to be valid. For more information, see **Add an Action**, page 134.

The code in this exit has the following format:

```
ASSIGN #valid-action(*) = 'X'
```

where *valid-action* is defined in CDACTA and CDACT.

This user exit works with the ADDITIONAL-ACTIONS-PROCESSING user exit. For information, see **ADDITIONAL-ACTIONS-PROCESSING**, page 326.

SELECT-STATEMENT (SQL only)

The code in this exit generates a SELECT statement for SQL-defined files and is used by the Browse and Object-Maint model types. Although the models generate a generic SELECT statement for all SQL files, this may not be the most efficient statement for a particular file. For example, a DB2-defined file may require a different SELECT statement syntax than an Oracle-defined file.

You can either let Natural Construct generate the generic SELECT statement as before (within the generated code), or generate it within the SELECT-STATEMENT user exit (where statement modifications can be made).

If the SELECT-STATEMENT user exit is defined, the model generates the END-SELECT statement and you define the beginning of the SELECT statement in the exit. For example:

```
0010 DEFINE EXIT SELECT-STATEMENT
0020     SELECT *
0030       INTO VIEW NCSTDB2-CUSTOMER
0040       FROM NCSTDB2-CUSTOMER
0050       WHERE (CUSTOMER_NUMBER >= #START.CUSTOMER_NUMBER)
0060       ORDER BY CUSTOMER_NUMBER
0070 END-EXIT
```

SELECT-STATEMENTS (SQL only)

This exit is similar to the SELECT-STATEMENT user exit, except it is used by the Object-Browse-Subp model. It defines a SELECT statement for SQL-defined files and is generated for each logical key (L1 to L6). If this exit is defined, and the file is relational (DB2, Oracle etc.), then the data access subroutines must be coded in user exits (instead of the generated code).

Note: If the data access subroutines are generated into user exits, they will not be maintained by Natural Construct.

For data to be returned, you cannot remove the PERFORM statements generated into the user exit code. For example:

```
L-1-ORDER_CUSTOMER_NUM.
SELECT *
       INTO VIEW NCSTDB2-ORDER_HEADER
            FROM NCSTDB2-ORDER_HEADER WHERE
   ((ORDER_CUSTOMER_NUM >=
     KEY-TABLE.ORDER_CUSTOMER_NUM(#LOW-KEY-VALUE) AND
     ORDER_CUSTOMER_NUM <=
     KEY-TABLE.ORDER_CUSTOMER_NUM(#HIGH-KEY-VALUE)))
       ORDER BY ORDER_CUSTOMER_NUM
       OPTIMIZE FOR 25 ROWS
  PERFORM R-S-ORDER_CUSTOMER_NUM
END-SELECT /* L-1-ORDER_CUSTOMER_NUM.
```

Note: If one key field uses this user exit, all key fields must use the exit.

Example of code in the SELECT-STATEMENTS user exit

```
**SAG DEFINE EXIT SELECT-STATEMENTS
************************************************************************
DEFINE SUBROUTINE R-CUSTOMER_NUMBER
************************************************************************
**
DECIDE FOR FIRST CONDITION

WHEN KEY-TABLE.CUSTOMER_NUMBER(#LOW-KEY-VALUE) = KEY-
TABLE.CUSTOMER_NUMBER(#HIGH-KEY-VALUE)
L-1-CUSTOMER_NUMBER.
SELECT *
INTO VIEW NCSTDB2-CUSTOMER
FROM NCSTDB2-CUSTOMER WHERE
CUSTOMER_NUMBER = KEY-TABLE.CUSTOMER_NUMBER(#LOW-KEY-VALUE)
ORDER BY BUSINESS-NAME
OPTIMIZE FOR 25 ROWS
PERFORM R-S-CUSTOMER_NUMBER
END-SELECT /* L-1-CUSTOMER_NUMBER.
WHEN NONE
L-2-CUSTOMER_NUMBER.
SELECT *
INTO VIEW NCSTDB2-CUSTOMER
FROM NCSTDB2-CUSTOMER WHERE
((CUSTOMER_NUMBER &gt;= KEY-TABLE.CUSTOMER_NUMBER(#LOW-KEY-VALUE) AND
CUSTOMER_NUMBER &lt;= KEY-TABLE.CUSTOMER_NUMBER(#HIGH-KEY-VALUE)))
ORDER BY CUSTOMER_NUMBER
OPTIMIZE FOR 1 ROWS
PERFORM R-S-CUSTOMER_NUMBER
END-SELECT /* L-2-CUSTOMER_NUMBER.
END-DECIDE
END-SUBROUTINE /* R-CUSTOMER_NUMBER
**SAG END-EXIT
```

SET-DATA-LENGTH

The code in this exit overrides the data length value used by the Spectrum dispatch service to determine the number and length of transmissions sent back to the client.

To set the data length, assign the desired value to CDAPROXY.DATA-LENGTH. If this value is not set in the user exit, it is automatically calculated based on the blocks of data to be returned.

SET-PF-KEYS

The code in this exit is executed before the PF-keys are set and allows the addition of non-standard PF-keys to a browse program. For more information, see **Adding a New PF-Key**, page 128. You define the additional PF-keys in the CDKEYLDA local data area.

Example of user exit code to add a non-standard PF-key

```
0010 DEFINE EXIT SET-PF-KEYS
0020 /*
0030 /* set non-standard PF-key
0040 RESET INITIAL CDKEYLDA.#YOUR-KEY
0050 END-EXIT SET-PF-KEYS
```

SET-RETURN-BLOCKS

The code in this exit defines which blocks to return the client. This user exit is placed before all generated block return designations. This allows you full control over which blocks are returned to the client, including overriding the generated block handling code. The code in this exit is part of a DECIDE FOR EVERY statement and defines any processing performed to reduce the number of blocks returned to the client. Use this exit with the Subprogram-Proxy model.

Note: Block optimization is handled automatically when generating a subprogram proxy for object browse and object maintenance subprograms.

Example of code in the SET-RETURN-BLOCKS user exit

```
DEFINE EXIT SET-RETURN-BLOCKS
**
** This exit is used to reduce the blocks that are returned to the
** client based on some application criteria. The code entered is
** part of a DECIDE FOR EVERY statement.
WHEN some logical condition
  RESET #PDA.#RB-block-name /* Don't return this block to the client
WHEN #SPC-TRUE /* This value can be used for unconditional assignments
  RESET #PDA.#RB-another-block
END-EXIT
```

SPECIAL-CODE-PROCESSING

The code in this exit defines the processing performed for each menu code (if entering a code on the generated menu does not FETCH a program).

START-OF-PROGRAM

Code in this exit is executed once at the beginning of the generated module after all standard initial values are assigned. You can use this exit to do any initial set-up required, such as initializing input values from globals, setting window or page sizes, or capturing security information for the restricted data area.

Tip: Use this user exit to set or reset variables or call routines that must be executed before the normal processing in a subprogram proxy.

Browse and Browse-Select Models

For the Browse and Browse-Select series of models, you can use this exit to indicate whether a browse module reads files in ascending (from 1 to 999999 or A to Z), descending (from 999999 to 1 or Z to A), or user-defined sequence. The user-defined sequence allows users to read files in either sequence as desired.

Note: If you do not include the START-OF-PROGRAM user exit, the default is ascending sequence and the Sequence field is not displayed on the generated browse panel.

For an example of using this exit, refer to the NCOSEL module in the Natural Construct demo system.

When you mark this user exit and press Enter, the following code is displayed:

```
Module ............. NCBROWSE
Title .............. Browse ...
>                                          > + ABS: _ X-Y: _ S    8 L    1
  All  ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
  0010 DEFINE EXIT START-OF-PROGRAM
  0020 *
  0030 * Processing to be performed once at the start of the program.
  0040 * Adjust the #SEQUENCE variable to get ascending, descending sequence.
  0050 * For example, uncomment these lines:
  0060 * MOVE 'D'    TO #SEQUENCE
  0070 * MOVE (AD=I) TO #SEQUENCE-CV
  0080 END-EXIT
```

START-OF-PROGRAM User Exit for a Browse Module

Modify this code as follows:

Sequence:	Action:
Ascending only	Do not include this user exit or modify the code. The Sequence field will not be displayed on the generated browse panel.
Descending only	Remove the comment indicator from line 0060: `MOVE 'D' TO #SEQUENCE`
Ascending by default, but can be changed by the user	Remove the comment indicator from line 0070: `MOVE (AD=I) TO #SEQUENCE-CV`
Note:	If you select this option, the Sequence field will be displayed on the generated browse panel.
Descending by default, but can be changed by the user	Remove the comment indicators from line 0060 and line 0070.

START-ROW-PROCESSING

The code in this exit defines the processing performed by an object browse select subprogram before the first row is processed.

Note: This user exit is only available if the object browse select subprogram accesses an object maintenance subprogram.

STATE-FOR-ABORTED-TRANSACTIONS

The code in this exit allows the user to change parameters in an object browse select subprogram when a business service transaction is aborted.

Note: This user exit is only available if the object browse select subprogram accesses an object maintenance subprogram.

TER1-WRITE-FIELDS

The code in this exit defines statements (WRITE, DISPLAY, or PRINT) that write information to specified reports. The specification panels are the same as those for the WRITE-FIELDS user exit. For a description of these panels, see **WRITE-FIELDS**, page 379.

Note: Define this exit if you are writing fields contained in the primary file, first secondary file, first tertiary file, or related files.

TER1-WRITE-HEADINGS

The code in this exit writes column headings (specified for tertiary file 1 to the specified report(s). If different reports use different tertiary data, generate this user exit for each report.

TER2-WRITE-FIELDS

The code in this exit defines statements (WRITE, DISPLAY, or PRINT) that write information to specified reports. The specification panels are the same as those for the WRITE-FIELDS user exit. For a description of these panels, see **WRITE-FIELDS**, page 379.

Note: Define this exit if you are writing fields contained in the primary file, second secondary file, second tertiary file, or related files.

TER2-WRITE-HEADINGS

The code in this exit writes column headings (specified for tertiary file 2 to the specified report(s). If different reports use different tertiary data, generate this user exit for each report.

TOP-OF-PAGE

The code in this exit is executed whenever a TOP OF PAGE condition occurs. For example, you can use the code in this exit to print panel and column headings.

TRANSLATE-COLUMN-HEADERS

The code in this exit defines the processing performed to allow cursor translation of the column headings. If the generated module supports cursor translation, you must define this exit.

TRANSLATE-INPUT-KEY

The code in this exit defines the input key used by the generated module to support cursor translation. If the generated module supports cursor translation, you must define this exit.

TRANSLATE-SCREEN-HEADERS

The code in this exit defines the processing performed to support cursor translation of the panel headings. If the generated module supports cursor translation, you must define this exit.

UNIQUE-TRANSACTION-STYLE

The code in this exit allows you to create a user-defined transaction type. The supplied transaction types are:

Transaction Style	Description
#AGGRESSIVE-ROW-OBJECT (default)	Server issues an End Transaction statement after each row. If there is a row failure, continues to process other rows.
#PASSIVE-ROW-OBJECT	Server issues an End Transaction statement after each row. If there is a row failure, stops processing remaining rows.
#BUSINESS-SERVICE-OBJECT	Server issues an End Transaction statement after *n* rows are processed.
#CLIENT-CONTROLLED-OBJECT	Server does not issue the End Transaction statement; the client code issues the ET statement.
#UNIQUE-OBJECT	You have full control over where End Transaction statements are issued.

UPDATE-EDITS

The code in this exit performs edit checks before adding, updating, or purging a record. The exit contains validation subroutines that execute edit checks at different points during the processing of an entity. You can create subroutines for each entity in an object.

The following table describes the validation subroutines available within this exit:

Subroutine	Description
V0-*entity-name*	Executed during the pre-editing phase before the Predict automatic rules are checked and the children of the current entity are processed.
V1-*entity-name*	Executed during the pre-editing phase after the Predict automatic rules are checked and before the children of the current entity are processed.
V2-*entity-name*	Executed during the post-editing phase after the Predict automatic rules are checked and all children of the current entity are processed.

Natural Construct assumes that primary fields are always on the current panel (on each map) and that secondary fields may require repositioning before displaying a message and marking the field in error.

Normally, a REINPUT statement displays an error message. However, when multiple input maps are used, an error may be detected when the field in error is not on the current panel. Issuing a REINPUT statement may cause the error message to be displayed for that field. To avoid this, assign a number to #ERROR-NR that represents the error, assign #PANEL to the desired map number (with 1 being the leftmost), assign #ERROR=TRUE, and perform the NEW-SCREEN subroutine. The generated module repositions the maps and invokes the REINPUT-SCREEN user exit (which can issue the error message).

Maint Model

If a maintenance program uses a secondary file or has multiple-valued fields (MUs) or periodic groups (PEs), you can reposition the scrolling area on a panel to a particular set of occurrences. To do this, you assign a value to #LINE before performing the NEW-SCREEN subroutine. In addition to repositioning the maps, the program repositions the scrolling area so the desired occurrence is displayed.

Example of user exit code for the Maint model

```
0010 DEFINE EXIT UPDATE-EDITS
0020 *
0030 IF #ACTION = #PURGE(*) THEN
0040   /*
0050   /* Perform purge edits.
0060   ESCAPE ROUTINE IMMEDIATE /* Skip Add/Modify edits
0070 END-IF
0080 *
0090 * Add/Modify edits
0100 DECIDE FOR FIRST CONDITION
0110   WHEN NCST-PRODUCT.PRODUCT-DESCRIPTION EQ ' '
0120     REINPUT 'Description is required'
0130       MARK *NCST-PRODUCT.PRODUCT-DESCRIPTION ALARM
0140   WHEN NCST-PRODUCT.PRODUCT-REORDER-POINT EQ 0
0150     REINPUT 'Reorder Point is required'
0160       MARK *NCST-PRODUCT.PRODUCT-REORDER-POINT ALARM
0170   WHEN NONE IGNORE
0180 END-DECIDE
0190 END-EXIT UPDATE-EDITS
```

Object-Maint Models

The code in this exit validates or manipulates the contents of an object. Before an entity is updated or stored, the subroutines in this exit are performed once, in the following order, for each occurrence of an entity:

1 The V0-*entity-name* subroutine is performed before the Predict rules associated with the fields of the current entity are processed. It retrieves data for validation. If an error occurs, this subroutine terminates the process immediately.

2 The V1-*entity-name* subroutine is performed after the Predict rules are processed. If no Predict rules are processed, you require either a V0-entity-name or V1-entity-name subroutine.

3 The V2-*entity-name* subroutine is performed after all child records are processed. This subroutine is required when child attributes affect the parent attributes. For example, each child record contributes an amount to the total value of the parent record (this contribution can be done in the V2-*child-entity-name* subroutine). After all child records are processed, the V2-*entity-name* subroutine of the current entity is invoked to validate the total value.

Because of the pre-order and post-order execution of the V0, V1, and V2 subroutines, the entities at the lowest level (entities without a child) do not need a V2 subroutine. They contribute to the parent through their V0 and V1 subroutines. At each node in the object hierarchy tree, the record buffers of the current entity and its parent (and ancestors) are available. Thus, the data validation or manipulation in the V0, V1, or V2 subroutine should be made against the record buffers for the corresponding views.

Prior to performing the V*n* subroutines, the object field values are copied to the corresponding file view using the MOVE BY NAME statement. Only refer directly to the object when the relevant fields have different names in the object PDA and file.

The Object-Maint-Subp model generates the #L2, #L3, and #L4 index variables to access the occurrences of the second, third, and fourth level entities within the object PDA. These indices must not be modified by the user exit code. However, user exit code can access the object PDA fields by referencing the appropriate index tuples corresponding to the second, third, or fourth level entities (for example, #L2, or #L2,#L3, or #L2,#L3,#L4).

Note: The subroutines in the UPDATE-EDITS user exit should assign the name of the object PDA field in error to the MSG-INFO.##ERROR-FIELD variable and a message to the MSG-INFO.##MSG or MSG-INFO.##MSG-NR variable (defined in the CDPDA-M parameter data area).

Example of using the subroutines in the UPDATE-EDITS user exit

Consider the Invoice object. It is created from INVOICE-HEADER, which has many INVOICE-GROUPs, each of which has many INVOICE-LINE records. The three files contain the respective fields: HEADER-AMOUNT, GROUP-AMOUNT, and LINE-AMOUNT. You can implement the amount accumulation from the bottom level to the top level using the subroutines on the following page.

```
0010 DEFINE-EXIT UPDATE-EDITS
0020   DEFINE SUBROUTINE V-INVOICE-HEADER
0030     IF INVOICE-HEADER.INVOICE-AUTHORIZED = 'N'
0040       ASSIGN MSG-INFO.##MSG = 'Invoice is not authorized'
0050       ASSIGN MSG-INFO.##ERROR-FIELD = 'INVOICE-AUTHORIZED'
0060       ESCAPE ROUTINE
0070     END-IF
0080     /*
0090     /* Reset the total amount before going to the invoice groups
0100     /* and lines.
0110     RESET INVOICE-HEADER.HEADER-AMOUNT
0120     /*
0130     /* Convert the displayed date of the object to the internal
0140    /* date to be stored in the file. Reference to the object field
0150    /* is required because the date field has different name on the
0160     /* object and the file
0170     MOVE EDITED INVOICE.INVOICE-DATE-EXTERNAL TO #DATE-VARIABLE
0180       (EM=LLL' 'DD','YY)
0190     MOVE EDITED #DATE-VARIABLE (EM=YYYYMMDD) TO
0200       INVOICE-HEADER.INVOICE-DATE
0210   END-SUBROUTINE /* V-INVOICE-HEADER
0220   /*
0230   DEFINE SUBROUTINE V2-INVOICE-HEADER
0240     /*
0250     /* Validate the total amount accumulated from the groups
0260     IF INVOICE-HEADER.HEADER-AMOUNT <= 0
0270       ASSIGN MSG-INFO.##MSG = 'Invalid total invoice amount::1:'
0280      ASSIGN MSG-INFO.##MSG-DATA(1) = INVOICE-HEADER.HEADER-AMOUNT
0290       ESCAPE ROUTINE
0300     END-IF
0310   END-SUBROUTINE /* V2-INVOICE-HEADER
0320   /*
0330   DEFINE SUBROUTINE V-INVOICE-GROUP
0340     /*
0350     /* Reset the group amount before going to the lines.
0360     RESET INVOICE-GROUP.GROUP-AMOUNT
0370   END-SUBROUTINE /* V-INVOICE-GROUP
0380   DEFINE SUBROUTINE V2-INVOICE-GROUP
0390     /*
0400     /* Validate the group amount accumulated from the lines
0410     IF INVOICE-GROUP.GROUP-AMOUNT <= 0
0420       ASSIGN MSG-INFO.##MSG = 'Invalid group amount::1:'
0430       ASSIGN MSG-INFO.##MSG-DATA(1) = INVOICE-GROUP.GROUP-AMOUNT
0440       ASSIGN MSG-INFO.##ERROR-FIELD = 'GROUP-AMOUNT'
0450       ESCAPE ROUTINE
0460     END-IF
0470     /*
0480     /* Accumulate the group amount to the total amount
0490   ADD INVOICE-GROUP.GROUP-AMOUNT TO INVOICE-HEADER.HEADER-AMOUNT
0500   END-SUBROUTINE /* V2-INVOICE-GROUP
0510   /*
0520   DEFINE SUBROUTINE V-INVOICE-LINE
0530     /*
0540     /* Accumulate the line amount to the group amount
0550     ADD INVOICE-LINE.LINE-AMOUNT TO
0560       INVOICE-GROUP.GROUP-AMOUNT
0570   END-SUBROUTINE /* V-INVOICE-LINE
0580 END-EXIT UPDATE-EDITS
```

Example of user exit code for the Object-Maint-Subp model

```
0130 DEFINE EXIT UPDATE-EDITS
0140 ************************************************************
0150 DEFINE SUBROUTINE V0-NCST-ORDER-HEADER
0160 ************************************************************
0170 *
0180   IF NCST-ORDER-HEADER.INVOICE-NUMBER = 0
0190     ASSIGN MSG-INFO.##MSG = 'Invoice number is required.'
0200     ASSIGN MSG-INFO.##ERROR-FIELD = 'INVOICE-NUMBER'
0210     ESCAPE ROUTINE
0220   END-IF
0230   /* Initialize order date when creating a new order
0240   IF CDAOBJ.#FUNCTION = 'STORE'
0250     /* Update date in file
0260     ASSIGN NCST-ORDER-HEADER.ORDER-DATE = *DATN
0270   END-IF
0280   /*
0290   /* Reset total order amount before going to the order lines
0300   /* to obtain their contribution.
0310   RESET NCST-ORDER-HEADER.ORDER-AMOUNT
0320 END-SUBROUTINE
0330 *
0340 ************************************************************
0350 DEFINE SUBROUTINE V2-NCST-ORDER-HEADER
0360 ************************************************************
0370 *
0380 * Validate total order amount accumulated from the order lines
0390   IF NCST-ORDER-HEADER.ORDER-AMOUNT LT 1000.0 THEN
0400     ASSIGN MSG-INFO.##MSG = 'Order Amount not less than 1000'
0410     ASSIGN MSG-INFO.##ERROR-FIELD = 'ORDER-AMOUNT'
0420   END-IF
0430 END-SUBROUTINE
0440 *
0450 ************************************************************
0460 DEFINE SUBROUTINE V0-NCST-ORDER-LINES
0470 ************************************************************
0480 *
0490 * If no product is ordered, terminate the process without
0500 * going to the lower level (order distribution).
0510   IF NCST-ORDER-LINES.QUANTITY LE 0
0520     ASSIGN MSG-INFO.##MSG = 'One product must be ordered'
0530     ASSIGN MSG-INFO.##ERROR-FIELD = 'QUANTITY'
0540     ESCAPE ROUTINE
0550   END-IF
0560 *
0570 * Update the order line amount
0580   COMPUTE NCST-ORDER-LINES.TOTAL-COST =
0590   NCST-ORDER-LINES.QUANTITY*NCST-ORDER-LINES.UNIT-COST
0600   /*
0610   /* Accumulate the line amount to the total order amount
0620 ADD NCST-ORDER-LINES.TOTAL-COST TO NCST-ORDER-HEADER.ORDER-AMOUNT
0630 END-SUBROUTINE
0640 END-EXIT UPDATE-EDITS
```

USER-DEFINED-FUNCTIONS

The code in this exit defines the processing performed for user-defined functions processed against an object. You can use this exit to specify user-defined action codes, for example. For more information, see **Add an Action**, page 134.

The format for this user exit is:

```
VALUE 'function-name'
processing
```

USER-DEFINED-METHODS

The code in this exit defines the methods available to the generated module.

USER-DEFINED-PENDING-STATE

The code in this exit allows the user to change parameters when a row transaction is successful, but has not been committed to the databases. This may be necessary if a row state requires a specific message before it is committed.

Note: This user exit is only available if the object browse select subprogram accesses an object maintenance subprogram.

USER-DEFINED-SUCCESSFUL-STATE

The code in this exit allows the user to change parameters when a row transaction is successful and will be committed to the database. This may be necessary if a row state requires a specific message when the transaction is successful.

Note: This user exit is only available if the object browse select subprogram accesses an object maintenance subprogram.

USER-TRACE-COMMANDS

Use this exit to expand the number of trace commands available in the proxy. By default, supplied trace commands return the size of the data blocks and their initialized values in the DATASIZES and INITIALIZE parameters.

Code in this user exit is placed ahead of all system-generated trace command code, which allows you to override the default trace functions.

WRITE-COLUMN-HEADERS

The code in this exit defines the column headings displayed on a generated panel.

WRITE-DATA-FIELDS

The code in this exit defines the processing performed to display selected fields on a generated panel.

Note: Natural Construct provides a user exit model you can use to generate (and re-generate) the layout of fields and column headers to be displayed on a terminal screen. For information, see **Object-Browse Models**, page 293.

WRITE-FIELDS

The code in this exit defines additional logic to write information based on the contents of the #PANEL variable. To build fields for display, Natural Construct defaults the display value to the specified browse key. This functionality is ideal for generating quick browse routines to check the records in a file. For example, you can specify a browse program using the customer name as the key field; by default, Natural Construct will generate a browse program that browses the Customer file by customer name. When you select this user exit, a series of windows is displayed. These windows are described in the following sections.

Data Parameters Window

```
CSUSELFV                  Natural Construct                  CSUSELF0
Aug 30                     Data Parameters                     1 of 1

    Label      Type      Predict Views or Data Areas    Select  All
   -------- ---------- ------------------------------- ------ ------
      1     View       NCST-CUSTOMER___________________   X      _
      2     Parameter  ORDERPDA________________________   X      _
      3                ________________________________   _      _
      4                ________________________________   _      _
      5                ________________________________   _      _
      6                ________________________________   _      _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11
      help  retrn
```

Data Parameters Window

The fields in this window are:

Field	Description
Label	Label number for the corresponding view or data area.
Type	Type of object in the corresponding Predict Views or Data Areas field.
Predict Views or Data Areas	Name of the view, local data area, parameter data area, or structure. (You may only use fields from Predict views and data areas.)
Select	Mark this field to display a selection window from which you can select the fields. (For a description of this window, see the following section.)
All	Mark this field to select all the fields from the corresponding view or data area. Multiple-valued fields (MUs) within a periodic group (PE) are not included when selecting from a view; constants and arrays with a rank greater than 1 are not included when selecting from a data area.
Note:	If there is more than one structure in a data area, only the first structure is included when the All field is marked. Existing fields are not re-selected.

Select Field Window — Selecting from a View

```
CSGSELFD                  Natural Construct                   CSGSELF0
Aug 30               Select NCST-CUSTOMER Field                 1 of 1

   Type  Level            Field Name            Format Length  Desc
 ------ ------ ------------------------------- ------ ------ ------
          1    _ CUSTOMER-NUMBER                  N      5.0    D
          1    _ BUSINESS-NAME                    A     30.0    D
          1    _ PHONE-NUMBER                     N     10.0
    GR    1    _ MAILING-ADDRESS
          2    _ M-STREET                         A     25.0
          2    _ M-CITY                           A     20.0
          2    _ M-PROVINCE                       A     20.0
          2    _ M-POSTAL-CODE                    A      6.0
    GR    1    _ SHIPPING-ADDRESS
          2    _ S-STREET                         A     25.0
          2    _ S-CITY                           A     20.0
          2    _ S-PROVINCE                       A     20.0
          2    _ S-POSTAL-CODE                    A      6.0
          1    _ CONTACT                          A     30.0
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--
      help  retrn
```

Select Field Window — Selecting from a View

This window displays the name of the specified view (the NCST-CUSTOMER view, in this example) as the second heading and lists the fields in that view. Mark fields you want to include. To display additional fields, press Enter. After scrolling through all the fields in the current view and selecting those you want to include in your report, the fields for the next view are displayed (if you selected another).

The fields in this window are:

Field	Description
Type	Field type (periodic group, multiple-valued, etc.) for the corresponding field.
Level	Level of the corresponding field.
Field Name	Names of the fields in the current view.
Format	Natural format of the corresponding field.
Length	Length of the corresponding field.
Desc	Descriptor indicator. If D is displayed, the corresponding field is a descriptor.

Select Field Window — Selecting from a Data Area

```
CSGSELVR              Natural Construct               CSGSELV0
Aug 30          Select Parameter ORDERPDA Field          1 of 1

 Type  Level             Field Name             Format Length
------ ------ --------------------------------- ------ ------
         1    _ ORDER
         2    _ ORDER-NUMBER                      N      6.0
         2    _ ORDER-AMOUNT                      P     13.2
         2    _ ORDER-DATE                        N      8.0
         2    _ ORDER-CUSTOMER-NUMBER             N      5.0
         2    _ ORDER-WAREHOUSE-ID                A      3.0
         2    _ INVOICE-NUMBER                    N      6.0
         2    _ ORDER-TIMESTAMP                   T
         2    _ C#DELIVERY-INSTRUCTIONS           N      3.0
         2    _ DELIVERY-INSTRUCTIONS             A     60.0
         2    _ C#NCST-ORDER-HAS-LINES            N      3.0
         2    _ NCST-ORDER-HAS-LINES
         3    _ LINE-NUMBER                       N      2.0
         3    _ ORDER-PRODUCT-ID                  A      6.0
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10
      help  retrn
```

Select Field Window — Selecting from a Data Area

This window displays the name of the specified data area (the ORDERPDA parameter data area, in this example) as the second heading and lists the fields in that data area. Mark the fields you want to include. To display additional fields, press Enter. After you scroll through all the fields in the current data area and select those you want to include in your report, the fields for the next data area are displayed (if you selected another).

The fields in the Select Field window for a data area are the same as those in the Select Field window for a view. For a description of this window, see **Select Field Window — Selecting from a View**, page 381.

Build Report Panel 1

After all the specified views and data areas are processed, the first Build Report panel is displayed:

```
CSUBLDRP                      Natural Construct                      CSUBLD10
Aug 30                   BROWSE Program Build Report                   1 of 2

  1_ Panel(*)  Order    Label    Field Name                    Format   AL=
  >> ------- -------- -------- ------------------------------ ---------- -----
  1    __      ___      1 *    CUSTOMER-NUMBER_____________   N 5.0__   _____
  2    __      ___      1 *    BUSINESS-NAME_______________   A 30.0_   _____
  3    __      ___      1 *    PHONE-NUMBER________________   N 10.0_   _____
  4    __      ___      2 *    ORDER-NUMBER________________   N 6.0__   _____
  5    __      ___      2 *    ORDER-DATE__________________   N 8.0__   _____
  6    __      ___      2 *    ORDER-WAREHOUSE-ID__________   A 3.0__   _____
  7    __      ___      _ *    ____________________________   _ _____   _____
  8    __      ___      _ *    ____________________________   _ _____   _____
  9    __      ___      _ *    ____________________________   _ _____   _____
  10   __      ___      _ *    ____________________________   _ _____   _____
  11   __      ___      _ *    ____________________________   _ _____   _____
  12   __      ___      _ *    ____________________________   _ _____   _____
  13   __      ___      _ *    ____________________________   _ _____   _____
  14   __      ___      _ *    ____________________________   _ _____   _____


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn deflt test  gen   selfd frwrd bkwrd reord       right proto
```

Build Report Panel 1

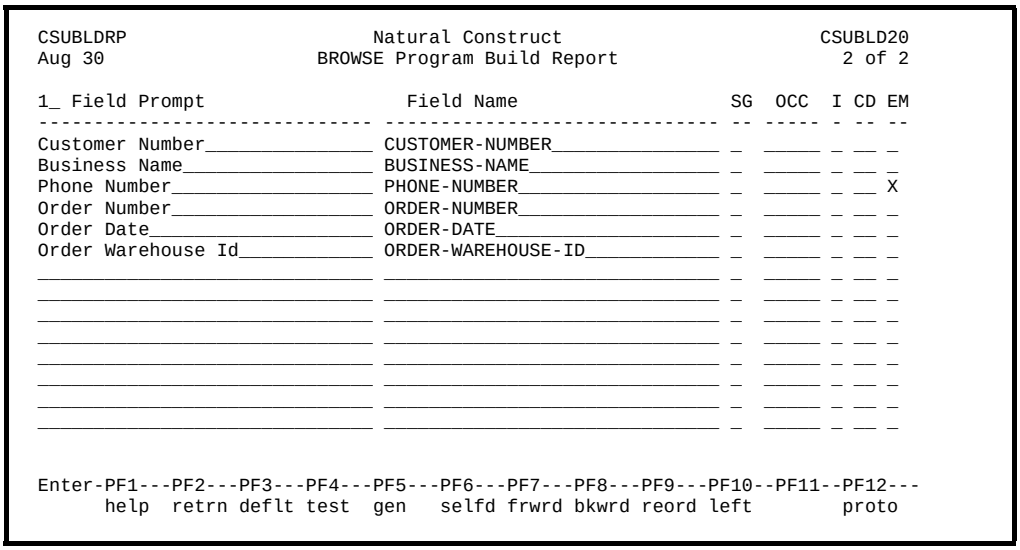
Define the appearance of the report on this panel. The fields on the first Build Report panel are:

Field	Description
1	Position indicator for the list of field names. To reposition the list, enter a new line number. The specified number and field are redisplayed at the top of the list.
Panel (*)	Number of the panel on which the corresponding field will be displayed. To display the field on all panels, type * (asterisk) in this field.
Order	Order of the corresponding field on the specified panel. We recommend you number the fields by 10 (10, 20, 30, etc.) instead of by 1 (1, 2, 3, etc.). This makes it easier to insert new fields between existing fields (11, 12, 13, for example) without renumbering all fields on a panel.
Note:	If you later change the field order on a panel, press the PF9 (reord) key to have Natural Construct reorder the fields.
Note:	You can delete the entire field specification line by entering a .D line command in this field.

Field	Description (continued)
Label	Number of the label (qualifier) for the view or data area. You can use up to six labels to build the report.
Field Name	Names of the fields selected for your report.
Format	Natural format and length for the corresponding field.
AL=	To override the default length of the corresponding field, type the new length in this field.

Build Report Panel 2

After entering the parameters on the first Build Report panel, press PF11 (right) to display the second Build Report panel:



Build Report Panel 2

This panel displays the default field prompts and other attributes for the selected fields.

The fields on the second Build Reports panel are:

Field	Description
Field Prompt	Field names displayed on the report. The default headings in Predict are displayed. You can change the field prompts by typing new names over the ones displayed.
Note:	When adding a user-defined field, or a Predict field without a heading, Natural Construct determines the field prompt by converting the field name to mixed case and substituting blanks for - (dash) or _ (underscore) characters.
Field Name	Names of the fields selected for your report.
SG	Sign option. Mark this field to have the generated report include the sign option for the corresponding field (SG=ON). If this field is blank, the sign is off (SG=OFF).
OCC	Number of occurrences of the corresponding field. If a fixed number is specified, that number is always displayed on the report (even if some occurrences are empty). To display a variable number of occurrences, enter C* or C#.
I	Mark this field to intensify the corresponding field (AD=I). If this field is blank, the field is displayed with default intensity (AD=D).
CD	Color definition for the corresponding field. (This option only applies when producing print file 0.)
EM	Mark this field to use the Predict edit mask for the corresponding field (EM=ON). If this field is blank, the edit mask is not used (EM=OFF) for the field.

Build Report Panels

The non-standard PF-keys on the Build Report panels are:

Key Number	Key Name	Function
PF3	deflt	Displays the Default Parameters window.
PF4	test	Displays a mock report layout to test the report layout and headings.
PF5	gen	Generates code into the edit buffer.
PF6	selfd	Displays the Data Parameters panel.
PF9	reord	Reorders the field positions (typically used after you add or delete fields).
PF12	proto	Invokes the prototyping function.

These PF-keys are described in the following sections. For more information about Natural Construct PF-keys, see **Natural Construct PF-Keys**, page 85.

Note: These PF-keys are Natural Construct defaults. The PF-keys for your organization may be different.

Specifying Default Parameters

To specify the statement type, size, and spacing of the report, press PF3 (deflt) on the second Build Report panel. The Default Parameters window is displayed:

```
CSUBLDA                 Natural Construct                CSUBLDA0
Aug 30                  Default Parameters                 1 of 1

          Generate statement type ..... _ WRITE
                                        X DISPLAY
          Leading blank columns ........ __
          Column spacing factor ........ 1
          Window\line width ............ ___
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--P
      help  retrn
```

Default Parameters Window

The fields in this window are:

Field	Description
Generate statement type	Type of Natural statement (WRITE or DISPLAY) used for the report. By default, DISPLAY is marked.
Leading blank columns	Number of blank columns preceding the report columns.
Column spacing factor	Number of spaces between fields on the report. The default is 1 space.
Window/line width	Number of columns used for the generated report. If you specified a number on a model specification panel, that number is displayed in this field. (Any number you specify in this field will override the number specified on the model specification panel.)

Testing the Report Layout

After marking all the fields and specifying their positions on the panel, press PF4 (test) on the second Build Report panel to display a mock report layout to see how the fields are displayed:

```
CSGBLDTE                        BROWSE Program
Aug 30                           Test Display                              1 of 1

Panel .. 01 ------------------------------------------------------------------------
Customer Number         Business Name          Phone Number Order Number
--------------- ------------------------------ ------------ ------------
     99999      XXXXXXXXXXXXXXXXXXXXXXXXXXXXX 999999999999    999999


Panel .. 02 ------------------------------------------------------------------------
Order Date Order Warehouse Id
---------- ------------------
 99999999         XXX









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
       test  test  test  test  test  test  test  test  test  left  right test
```

Build Report Panel — After Pressing PF4 (test)

Press Enter to return to the Build Report panel.

Generating Code into the Edit Buffer

Press PF5 (gen) on the second Build Report panel to generate and place the resulting code in the edit buffer.

Selecting Additional Views or Data Areas

Press PF6 (selfd) on the second Build Report panel to return to the Data Parameters window, where you can specify additional views or data areas. For a description of this window, see **Data Parameters Window**, page 380.

Reordering Fields on the Report

Press PF9 (reord) on the second Build Report panel to reorder the fields. This function allows you to insert new fields into your report without manually renumbering all of the field order numbers.

Invoking the Prototyping Function

Press PF12 (proto) on the second Build Report panel to invoke the prototyping function. Natural Construct assigns an order and panel number for each currently-displayed field:

```
CSUBLDRP                    Natural Construct                       CSUBLD10
Aug 30                  BROWSE Program Build Report                   1 of 2

  1_ Panel(*)  Order    Label    Field Name                    Format   AL=
  >> ------- -------- -------- ------------------------------ ---------- -----
  1    01      010      1 *    CUSTOMER-NUMBER_____________   N 5.0__   _____
  2    01      020      1 *    BUSINESS-NAME_______________   A 30.0_   _____
  3    01      030      1 *    PHONE-NUMBER________________   N 10.0_   _____
  4    01      040      2 *    ORDER-NUMBER________________   N 6.0__   _____
  5    02      010      2 *    ORDER-DATE__________________   N 8.0__   _____
  6    02      020      2 *    ORDER-WAREHOUSE-ID__________   A 3.0__   _____
  7    __      ___      _ *    ____________________________   _ _____   _____
  8    __      ___      _ *    ____________________________   _ _____   _____
  9    __      ___      _ *    ____________________________   _ _____   _____
  10   __      ___      _ *    ____________________________   _ _____   _____
  11   __      ___      _ *    ____________________________   _ _____   _____
  12   __      ___      _ *    ____________________________   _ _____   _____
  13   __      ___      _ *    ____________________________   _ _____   _____
  14   __      ___      _ *    ____________________________   _ _____   _____


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
       help  retrn deflt test  gen   selfd frwrd bkwrd reord      right  proto
```

Build Report Panel 1 — After Pressing PF12 (proto)

Browse Models

For a simple browse panel, provide a single WRITE, PRINT, or DISPLAY statement. The elements in the WRITE statement can include literals, database fields, and user-defined variables. If the primary key is a superdescriptor or subdescriptor, refer to it using the #KEY variable. (References to the primary key must be fully qualified with the file name.) To display multiple lines for one record, use separate WRITE statements.

For a multi-panel program, add additional logic to write information based on the contents of the #PANEL variable. For example, the code for three panels can be:

```
0010 DECIDE ON FIRST VALUE #PANEL
0020   VALUE 1
0030     WRITE VEHICLES.REG-NUM VEHICLES.MAKE VEHICLES.MODEL
0040   VALUE 2
0050     WRITE VEHICLES.REG-NUM VEHICLES.BODY-TYPE VEHICLES.COLOR
0060   VALUE 3
0070     WRITE VEHICLES.REG-NUM VEHICLES.OWNER-PERSONNEL-NUMBER
0080   NONE
0090     IGNORE
0100 END-DECIDE
```

Additional user fields can also control what information is displayed. If a detail flag is required, for example, the user exit code can be:

```
0010 DISPLAY EMPLOYEES.PERSONNEL-NUMBER EMPLOYEES.LAST-NAME
0020 IF #DETAIL THEN
0030   WRITE 10X 'Address:' EMPLOYEES.NUMBER STREET
0040   WRITE 20X EMPLOYEES.CITY STATE
0050 END-IF
```

Note: To reference database fields that are not in the primary file, define the additional views in the LOCAL-DATA user exit and write the look-up statements in this exit.

Note: The #ACTION variable is included as a non-display field on a browse-select report. It acts as a placeholder for the action/selection column, as well as provides a heading for the column. The column overlays this field on the INPUT NO ERASE statement generated into the program.

Example of user exit code for the Browse model

```
0010 DEFINE EXIT WRITE-FIELDS
0020     DISPLAY(0)
0030       'Cust/No' NCST-CUSTOMER.CUSTOMER-NUMBER (AD=I)
0040       '/Business Name' NCST-CUSTOMER.BUSINESS-NAME
0050       '/Phone No' NCST-CUSTOMER.PHONE-NUMBER
0060       'WHS/ID' NCST-CUSTOMER.CUSTOMER-WAREHOUSE-ID
0070       '/Contact' NCST-CUSTOMER.CONTACT
0080         (AL=25)
0090 *
0100 * Display additional data upon request.
0110 DECIDE ON FIRST VALUE #OPTION
0120   VALUE 'M' /* Display Mailing Address
0130     WRITE 06X 'Mailing Address:'(I) NCST-CUSTOMER.M-STREET
0140     WRITE 23X NCST-CUSTOMER.M-CITY
0150     WRITE 23X NCST-CUSTOMER.M-PROVINCE
0160     WRITE 23X NCST-CUSTOMER.M-POSTAL-CODE
0170     WRITE ' '
0180   VALUE 'S' /* Display Shipping Address
0190     WRITE 06X 'Shipping Address:'(I) NCST-CUSTOMER.S-STREET
0200     WRITE 24X NCST-CUSTOMER.S-CITY
0210     WRITE 24X NCST-CUSTOMER.S-PROVINCE
0220     WRITE 24X NCST-CUSTOMER.S-POSTAL-CODE
0230     WRITE ' '
0240   VALUE 'C'
0250    WRITE 06X 'Credit Rating..:'(I) NCST-CUSTOMER.CREDIT-RATING
0260     WRITE 06X 'Credit Limit...:'(I)
0270                NCST-CUSTOMER.CREDIT-LIMIT(AD=L SG=OFF)
0280     WRITE 06X 'Disc. Percentage:'(I)
0290                NCST-CUSTOMER.DISCOUNT-PERCENTAGE(EM=Z9.99'%')
0300     WRITE ' '
0310   ANY
0320     IF *LINE-COUNT > 15
0330       /*
0340       /* Trigger new page if there is not enough space for
0350       /* next record's additional data.
0360       NEWPAGE
0370     END-IF
0380   NONE IGNORE
0390 END-DECIDE
0400 END-EXIT WRITE-FIELDS
```

Example of user exit code for the Browse-Helpr/Browse-Subp model

```
0010 DEFINE EXIT WRITE-FIELDS
0020 DISPLAY 'Warehouse'(I) NCST-WAREHOUSE.WAREHOUSE-ID(AD=I)
0030         'Description'(I) NCST-WAREHOUSE.WAREHOUSE-DESCRIPT
0040 END-EXIT WRITE-FIELDS
```

Example of user exit code for the Browse-Select-Helpr model

```
0010 DEFINE EXIT WRITE-FIELDS
0020   DISPLAY '/Action'(I) #ACTION(AD=N)
0030 * Place-holder for action column
0040     'Customer/Number'(I) NCST-CUSTOMER.CUSTOMER-NUMBER(LC=' ')
0050     '/Business Name'(I) NCST-CUSTOMER.BUSINESS-NAME
0060 END-EXIT WRITE-FIELDS
```

Example of user exit code for the Browse-Select-Subp model

```
0010 DEFINE EXIT WRITE-FIELDS
0020     WRITE(0)
0030       3X NCST-ORDER-HEADER.ORDER-NUMBER
0040       6X NCST-ORDER-HEADER.ORDER-DATE
0050       6X NCST-ORDER-HEADER.ORDER-AMOUNT
0060 END-EXIT WRITE-FIELDS
```

Example of user exit code for the Browse-Select model

```
0010 DEFINE EXIT WRITE-FIELDS
0020 WRITE 9T NCST-ORDER-HEADER.ORDER-NUMBER (AD=I)
0030       18T NCST-ORDER-HEADER.ORDER-CUSTOMER-NUMBER
0040       30T NCST-ORDER-HEADER.ORDER-WAREHOUSE-ID
0050       39T NCST-ORDER-HEADER.INVOICE-NUMBER
0060       47T NCST-ORDER-HEADER.ORDER-DATE(EM=99'/'99'/'99)
0070       57T NCST-ORDER-HEADER.ORDER-AMOUNT
0080 IF DETAIL THEN
0090   FIND-CUST.
0100   FIND(1) NCST-CUSTOMER WITH CUSTOMER-NUMBER =
0110        NCST-ORDER-HEADER.ORDER-CUSTOMER-NUMBER
0120     DECIDE FOR FIRST CONDITION
0130       WHEN NCST-CUSTOMER.S-CITY NE ' ' AND
0140            NCST-CUSTOMER.S-PROVINCE NE ' '
0150        COMPRESS NCST-CUSTOMER.S-CITY ',' TO #CITY-PROVINCE
0160    LEAVING NO
0170         COMPRESS #CITY-PROVINCE NCST-CUSTOMER.S-PROVINCE
0180              TO #CITY-PROVINCE
0190       WHEN NCST-CUSTOMER.S-CITY NE ' '
0200         ASSIGN #CITY-PROVINCE = NCST-CUSTOMER.S-CITY
0210       WHEN NCST-CUSTOMER.S-PROVINCE NE ' '
0220         ASSIGN #CITY-PROVINCE = NCST-CUSTOMER.S-PROVINCE
0230       WHEN NONE
0240         ASSIGN #CITY-PROVINCE = '** Unknown Province **'
0250     END-DECIDE
0260     WRITE 18T 'Customer Name:'(I) NCST-CUSTOMER.BUSINESS-NAME
0270               'Contact:'(I) NCST-CUSTOMER.CONTACT
0280     WRITE 18T 'Phone        :'(I) NCST-CUSTOMER.PHONE-NUMBER
0290     WRITE 18T 'Address      :'(I) NCST-CUSTOMER.S-STREET
0300     PRINT 33T #CITY-PROVINCE
0310     WRITE 18T '              ' NCST-CUSTOMER.S-POSTAL-CODE
0320     /*
0330     /* Try to put record and its details on the same panel.
0340     IF *LINE-COUNT GE 14 THEN
0350       NEWPAGE
0360     ELSE
0370       SKIP 1
0380     END-IF
0390   END-FIND
0400   /*
0410   /* Try to put the record and its detail on the same panel.
0420   IF *NUMBER(FIND-CUST.) NE 1 AND *LINE-COUNT GE 14 THEN
0430     NEWPAGE
0440   END-IF
0450 END-IF
0460 **
0470 END-EXIT WRITE-FIELDS
```

Multi-line Browse-Select Programs

Programs generated using the Browse-Select models usually display one row per record on the screen, although these models allow many lines to be written for each record. By default, the Action field is only activated for the first row in each new record. The following example of a selection screen shows this default behavior:

```
NCCSCUS1             ***** MULTI-LINE BROWSE SELECT *****
Aug 26                 - SELECTION ON FIRST LINE ONLY -                9:52 AM

Action  Cust          Business Name          Phone Number
        Number
------ ------ ------------------------------ ------------

  __        1  Software AG       (CANADA)     519-622-0889
               Mailing Addr : 151 Savage Drive Cambridge Ontario N1T 1S6
               Shipping Addr: P.O. BOX 9 Cambridge Ontario N1T 1S6
  __        2  JOURNEYMEN FABRICATING         519-234-6422
               Mailing Addr : 230 LONGWOOD ST. Kitchener Ontario N3H 2S6
               Shipping Addr: 230 LONGWOOD ST. Kitchener Ontario N3H 2S6
  __      511  Software AG   US               519-624-5623
               Mailing Addr : 1 Bay Street TORONTO ONTARIO B4B 3B3
               Shipping Addr: 100 YOUNG STREET STN A TORONTO ONTARIO B4B 3B3
  __    10001  Journeymen Fabricating         519-234-6422
               Mailing Addr : RR3 Kitchener Ontario N3H 2S5
               Shipping Addr: RR3 Kitchener Ontario N3H 2S5
  __    10002  Les Rivers Custom Fabricating  519-623-6850
Customer Number: _____
Direct command...: ____________________________________________________________
 Copy       DEtail     DIsplay    Modify     Purge      (PF5=flip)
```

Multi-Line Browse-Select Program Example

To change the default, write user exit code to change the default values for the control variables used by the generated program.

Example of changing defaults for the Action field

The following example removes the Action field from the first record line and applies it to subsequent record lines only. Notice that the UPDATE-SELECTION-TABLE subroutine is performed after each line is written:

```
DEFINE EXIT START-OF-PROGRAM
  /*
  /* Override the default attributes for the first and subsequent
  /* display lines.
  ASSIGN #ATTR-LINE1 = (AD=NP) /* No selection allowed for first row
  ASSIGN #ATTR-REST = (AD=I)   /* Allow selection for subsequent rows
END-EXIT
DEFINE EXIT WRITE-FIELDS
  DISPLAY 'Action' #ACTION(AD=N)
          'Cust/Number' NCST-CUSTOMER.CUSTOMER-NUMBER
          NCST-CUSTOMER.BUSINESS-NAME
          NCST-CUSTOMER.PHONE-NUMBER
  PERFORM UPDATE-SELECTION-TABLE
  PRINT(AD=I) 14X 'Mailing Addr :' NCST-CUSTOMER.MAILING-ADDRESS
  PERFORM UPDATE-SELECTION-TABLE
  PRINT(AD=I) 14X 'Shipping Addr:' NCST-CUSTOMER.SHIPPING-ADDRESS
END-EXIT
```

To indicate the record line for which the action applies, use the #SELECTED-LINE variable in the PROCESS-SELECTED-RECORD user exit. For more information, see **PROCESS-SELECTED-RECORD**, page 358.

The following selection screen example shows the new default behavior:

```
NCCSCUS2             ***** MULTI-LINE BROWSE SELECT *****
Aug 26                - SELECTION ON DETAIL LINES ONLY -             11:22 AM

Action  Cust          Business Name          Phone Number
       Number
------ ------ ------------------------------ ------------

          1  Software AG       (CANADA)     519-622-0889
  __         Mailing Addr : 151 Savage Drive Cambridge Ontario N1T 1S6
  __         Shipping Addr: P.O. BOX 9 Cambridge Ontario N1T 1S6
          2  JOURNEYMEN FABRICATING         519-234-6422
  __         Mailing Addr : 230 LONGWOOD ST. Kitchener Ontario N3H 2S6
  __         Shipping Addr: 230 LONGWOOD ST. Kitchener Ontario N3H 2S6
        511  Software AG   US               519-624-5623
  __         Mailing Addr : 1 Bay Street TORONTO ONTARIO B4B 3B3
  __         Shipping Addr: 100 YOUNG STREET STN A TORONTO ONTARIO B4B 3B3
      10001  Journeymen Fabricating         519-234-6422
  __         Mailing Addr : RR3 Kitchener Ontario N3H 2S5
  __         Shipping Addr: RR3 Kitchener Ontario N3H 2S5
      10002  Les Rivers Custom Fabricating  519-623-6850
Customer Number: _____
Direct command...: _____________________________________________________________
 Copy       DEtail     DIsplay    Modify     Purge      (PF5=flip)
```

Multi-Line Browse-Select Program Example — After Changing Defaults

For additional examples of multi-line browse-select programs, refer to the Customer subsystem in the Natural Construct demo system.

STATEMENT MODELS

This chapter describes the models that generate individual Natural statements or functions (blocks of code). You can use the statement models as building blocks when creating user exit code or specialized programs that are not suited to an existing Natural Construct model.

The following topics are covered:

Introduction

Using statement models can greatly reduce the time required to write recurring code segments, as well as minimize testing requirements. Unlike other models, you can invoke statement models from some of the Natural Construct editors (Generated Code, User Exit, and Code Frame editors) to generate code as you need it. You can also use the Generation main menu to generate statements and functions into separate files.

Advantages of Using Statement Models

Using statement models to generate portions of your program code:

- Reduces your dependency on the Natural documentation to recall statement syntax.
- Reduces the number of keystrokes required to code Natural statements, since keywords are generated automatically.
- Produces statement code that:
  - has a consistent indentation scheme
  - programmatically performs tedious calculations
  - accesses your system files to retrieve Predict views, MSGGEN error message numbers, etc.

Invoking Statement Models

There are two ways to invoke statement models:

- Invoke the models from within the User Exit, Code Frame, or Generated Code editor and then combine these building blocks with generated or handwritten code.

Unix Note:
The .g line command is not supported by Unix third-party editors, such as vi and emacs.

- Invoke the models from the Generation main menu as you would any other model. Using this method, you can generate the statement code and save it in a separate file. You can then use the files as building blocks and insert them in your programs.

  The following sections explain how to invoke a statement model from an editor that supports the .g command. For information about invoking the models from the Generation main menu, see **Invoke User Exit Editor Function**, page 68.

Generating Default Code Structures

When you use a comma (,) after the statement model name, Natural Construct generates a default code structure. For example, you can enter the following on a line in the editor:

```
.g(decide-on,)
```

or (using the model alias):

```
.g(do,)
```

Natural Construct generates the following default structure:

```
0020 /*
0030 DECIDE ON FIRST #NUMBER
0040   VALUE 1
0050     ASSIGN ...
0060   VALUE 2
0070     ASSIGN ...
0080   ANY
0090     if any are true
0100   ALL
0110     if all are true
0120   NONE
0130     IGNORE
0140 END-DECIDE
```

At this point, you are still in the editor and can add your data and comments, as well as check the code for compile errors.

Note: If the generated defaults are not acceptable, you can omit the comma after the model name and enter the values in a specification window.

Many statement models use the most common statement syntax structure for the defaults; for example, the Add statement model generates two variables into a third variable. Other statement models take the opposite approach, where all possibilities are defaults; for example, the Set-Key statement model uses the Natural Construct Control record to generate all possibilities. You can then delete the values that are not applicable and/or edit the names as required.

When you use the comma (,) and specify data, Natural Construct stacks the data in the model specification window. If the stacked data passes all edits, then the code is generated directly into your editors using that data. This functionality is useful for statement models that require one or two key parameters, such as the View and Callnat models. For example, to generate the CUSTOMER view of the NCST-CUSTOMER file, you can enter:

```
.g(view,NCST-CONSTRUCT-CUSTOMER)
```

or

```
.g(view,NCST-CONSTRUCT-CUSTOMER,MY-CUSTOMER)
```

Invoking a Statement Model From an Editor

You can invoke a statement model from the Natural or Natural Construct editors. To specify the statement model name:

- Type the statement model name in full.
- Type just enough characters to uniquely identify the statement model. For example, you can specify .g(arb) to invoke the Arbitrary-Field-Processing statement model.
- Type the statement model's alias. For example, the alias of the Arbitrary-Field-Processing statement model is "AF". For a list of the statement models and their aliases, see **Supplied Statement Models**, page 400.

To invoke the statement model, use one of the following line commands:

Command	Result
.g	Displays the list of available statement models.
.g(*model name*,)	Generates default code for the specified statement model into the editor. If the statement model has required arguments, a window is displayed first so you can fill in the parameters.

Command	Result (continued)
.g(*model name*)	Displays the first model specification window for the specified statement model. If the specified model does not exist, this command displays a list of available models for selection.
.g(*model name,p1,p2,p3...*)	Generates default code into the editor. You can use this command when you know the model name and required parameters (and do not want to access the model selection or specification windows).

Note: Although you can invoke any statement model using the .g command, remember that the source buffer is cleared before program generation begins.

Supplied Statement Models

The following table lists the statement models supplied with Natural Construct, the two-character alias you can use to invoke each model, and the statement or function each model generates:

Model Name	Alias	Generates
Add	AD	ADD statement
Arbitrary-Field-Processing	AF	Statement for arbitrary view/data area processing function
At-Break	AB	AT BREAK statement
At-Top-of-Page	TP	AT TOP OF PAGE statement
Callnat	CN	CALLNAT statement
Close	CR or CW	CLOSE PRINTER/WORK FILE statement
Compress	CO	COMPRESS statement
Decide-For	DF	DECIDE FOR statement
Decide-On	DO	DECIDE ON statement
Define-Subroutine	DS	DEFINE SUBROUTINE statement
Define-Printer	DP	DEFINE PRINTER statement
Define-Variable	DA	Define Variable function
Define-Window	DW	DEFINE WINDOW statement
Divide	DV	DIVIDE statement
Escape	ES	ESCAPE statement
Examine	EX	EXAMINE statement
Examine-Translate	ER	EXAMINE TRANSLATE statement
Fetch	FE	FETCH statement
Find	FI	FIND statement
For	FO	FOR statement
Format	FM	FORMAT statement
Get	GE	GET statement

Model Name	Alias	Generates (continued)
Histogram	HI	HISTOGRAM statement
If	IF	IF statement
If-Is	II	IF statement with IS logical condition criterion
If-Mask-Scan	IM	IF MASK/SCAN statement
If-Selection	IS	IF SELECTION NOT UNIQUE statement
Include	IN	INCLUDE statement
Input	IP	INPUT statement
Move	MO	MOVE statement
Multiply	MU	MULTIPLY statement
Process-Command	PO	PROCESS COMMAND ACTION statement
Read	RD	READ statement
Read-Work-File	RW	READ WORK FILE statement
Reinput	RI	REINPUT statement
Repeat	RP	REPEAT statement
Separate	SP	SEPARATE statement
Sequence	SQ	SEQUENCE statement (repeats a string, incrementing a sequence value)
Set-Control	SC	SET CONTROL statement
Set-Key	SK	SET KEY statement
Set-Window	SW	SET WINDOW statement
Stack	SA	STACK statement
Store	SE	STORE statement
Substring	SS	Option that moves field to substring, moves substring to field
Subtract	SB	SUBTRACT statement

Model Name	Alias	Generates (continued)
User-Exit	UE	Text member modules containing field layout specifications for modules generated using the Object-Browse-Dialog model
View	VW	View from Predict
Write-Work-File	WF	WRITE WORK FILE statement

The following sections describe the functionality of each statement model, as well as the specification windows and options available. An example of each statement model is also provided.

Note: If the statement models installed at your site do not appear as they do here, they may have been modified by your Natural Construct administrator.

Add Statement Model

The Add statement model (AD) generates a Natural ADD statement, which adds two or more operands (values).

Generating the Default Code into the Editor

To generate the ADD statement, enter one of the following line commands:

```
.g(ADD,)
```

or

```
.g(AD,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 ADD #amount1
0040     #amount2
0050  TO #result
0060
```

You can then replace the variables (indicated by the "#" prefix) to complete the ADD statement.

Using the ADD Statement Window

The ADD Statement window is displayed when you invoke the Add statement model from the Generation main menu. You can also invoke the ADD Statement window from an editor by entering one of the following line commands:

```
.g(ADD)
```

or

```
.g(AD)
```

The ADD Statement window is displayed:

```
                          ADD Statement

ADD  ROUNDED _
     ____________________________________________________________
     ____________________________________________________________
     ____________________________________________________________
     ____________________________________________________________
     ____________________________________________________________

  TO ____________________________________________________________

  GIVING ________________________________________________________
```

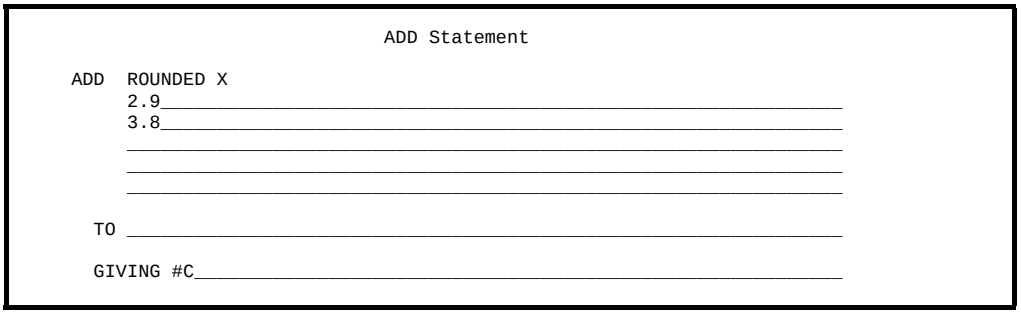
ADD Statement Window

The fields in this window are:

Field	Description
ADD ROUNDED	If this field is marked, the result of the Add operation is rounded. For example, if 43.6 is the result and the target field has no decimals, the rounded result is 44. This is an optional field.
	On the lines provided, specify the values that are added together. Specify each of the values (numeric constants or variables) on a separate line. This is a required field.
TO	The result of the addition of the above values are added to the variable specified in this field.
GIVING	If a variable is not specified in the TO field, you must specify one in this field. The result of the addition of the above values are stored in this field.
Note:	You must specify either a TO or a GIVING value, not both.

Add Statement Model Example

The following specifications generate an ADD statement:



ADD Statement Window Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 ADD ROUNDED 2.9
0040             3.8
0050 GIVING #C
```

Arbitrary-Field-Processing Statement Model

The Arbitrary-Field-Processing statement model (AF) generates code related to fields based on specified criteria. Depending on your selection criteria, this statement model generates code based on the name of a local or parameter data area (LDA or PDA) and/or a Predict view (see also **View Statement Model**, page 562).

Generating the Default Code into the Editor

To generate the Arbitrary-File-Processing function, enter one of the following line commands:

```
.g(ARBITRARY-FIELD-PROCESSING,)
```

or

```
.g(AF,)
```

The Select Predict View window is displayed to select a Predict view:

```
Position cursor or enter screen value to select
CPHFIB                Natural Construct                 CPHFIB0
Oct 29               Select Predict View                 1 of 1

Predict view                     File type
-------------------------------- --------------------------------
NCSTDB2-CUSTOMER                 DB2 Table
NCSTDB2-ORDER_DISTRIBUTION       DB2 Table
NCSTDB2-ORDER_HEADER             DB2 Table
NCSTDB2-ORDER_INSTRUCTIONS       DB2 Table
NCSTDB2-ORDER_LINES              DB2 Table
NCSTDB2-PRODUCT                  DB2 Table
NCSTDB2-WAREHOUSE                DB2 Table
NCSTVSAM-CUSTOMER                VSAM File
NCSTVSAM-ORDER-DISTRIBUTION      VSAM File
NCSTVSAM-ORDER-HEADER            VSAM File
Predict view ............. NCSTDB2_____________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10-
      help  retrn                         bkwrd frwrd
```

Select Predict File Window

For information on selecting a view from this window, see **Active Field-Level Help**, page 55.

After selecting a Predict file name and pressing Enter, the default code is displayed in the editor. The following example shows a portion of the code generated for the NCST-EMPLOYEES view for Adabas files:

```
0010 /* if PERSONNEL ID equals null assign default value
0020 IF NCST-EMPLOYEES.PERSONNEL-ID = ' '
0030   ASSIGN NCST-EMPLOYEES.PERSONNEL-ID = ...
0040 END-IF
0050 /*
0060 /* if First Name equals null assign default value
0070 IF NCST-EMPLOYEES.FIRST-NAME = ' '
0080   ASSIGN NCST-EMPLOYEES.FIRST-NAME = ...
0090 END-IF
0100 /*
0110 /* if Name equals null assign default value
0120 IF NCST-EMPLOYEES.NAME = ' '
0130   ASSIGN NCST-EMPLOYEES.NAME = ...
0140 END-IF
0150 /*
0160 /* if MARITAL STATUS equals null assign default value
0170 IF NCST-EMPLOYEES.MAR-STAT = ' '
0180   ASSIGN NCST-EMPLOYEES.MAR-STAT = ...
0190 END-IF
0200 /*
```

Using the ARBITRARY-FIELD-PROCESSING Function Window

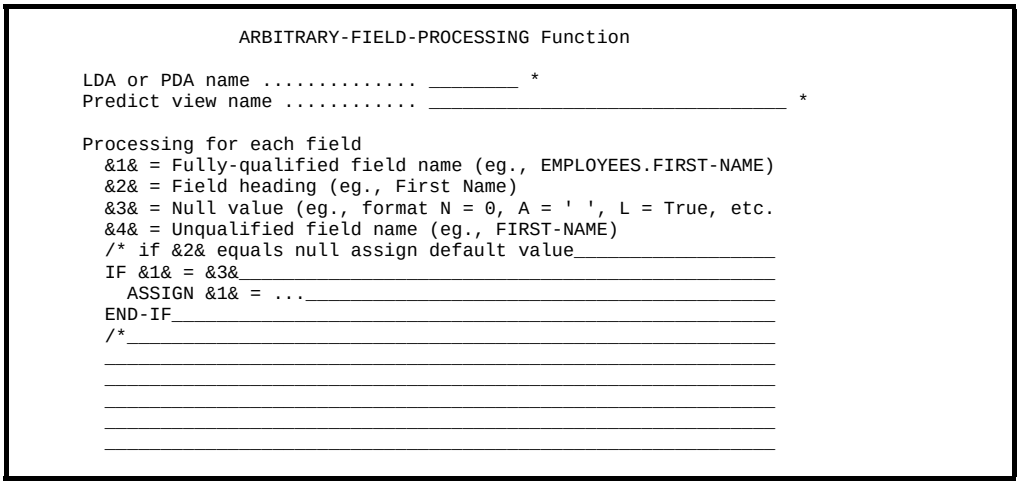
The ARBITRARY-FIELD-PROCESSING Function window is displayed when you invoke the Arbitrary-Field-Processing statement model from the Generation main menu. You can also invoke the window from an editor by entering one of the following line commands:

```
.g(ARBITRARY-FIELD-PROCESSING)
```

or

```
.g(AF)
```

The ARBITRARY-FIELD-PROCESSING Function window is displayed:



ARBITRARY-FIELD-PROCESSING Function Window

The fields in this window are:

Field	Description
LDA or PDA name	Name of a local or parameter data area from which to generate code. You must supply either an LDA or PDA name or a Predict view name (or both). To select an LDA or PDA, press PF1. After specifying an LDA/PDA, press PF5 (optns) to tailor the data area. For information, see **Tailoring the LDA or PDA**, page 409.
Predict view name	Name of a view in Predict. You must supply either an LDA or PDA name or a Predict view name (or both). To select a Predict view, press PF1. After specifying a view name, press PF5 (optns) to tailor the view. For information, see **Tailoring the Predict View**, page 410.
Processing for each field	Code generated for each field selected from the specified data area and/or view. Four substitution options are available. These options allow you to generate fields and field attributes into the specified code. You can accept the default values and/or make changes to the code. The substitution options are:
&1&	Substitutes the fully-qualified field name for the &1& variable. In a personnel file, for example, the field name can be EMPLOYEES.LEAVE-TAKEN(*).
&2&	Substitutes the specified field heading for the &2& variable. In a personnel file, for example, the field heading can be Personnel ID Number.
&3&	Substitutes a zero (numeric) or blank (alphanumeric) for the &3& variable.
&4&	Substitutes the unqualified field name for the &4& variable. In a personnel file, for example, the substituted field name can be LEAVE-TAKEN.

Tailoring the LDA or PDA

When you have supplied an LDA or PDA name in the appropriate field of the ARBITRARY-FIELD-PROCESSING Function window, press PF5 (optns) to tailor the LDA or PDA. The LDA or PDA name *modulename* window is displayed. The following example shows the window for an LDA called CDGBTOOL:

```
                    LDA or PDA Name CDGBTOOL

      Include Redefined Fields
      All redefinitions ............ X
      FILLER fields ................ X

      Optional Structure Limitations
      INPUT - INPUT-OUTPUT only .... _
      OUTPUT - INPUT-OUTPUT only ... _
      Specific structure ........... ________________________________

      Include Field Rank Selection
      Scalar fields ................ X
      One-dimensional arrays ....... X
      Two-dimensional arrays ....... X
      Three-dimensional arrays ..... X
```

LDA or PDA Name Window

The fields in the LDA or PDA window are:

Field	Description
Include Redefined Fields	
All redefinitions	To include all redefinitions of a field, mark this field.
FILLER fields	To include Filler fields when generating field definitions, mark this field.
Optional Structure Limitations	
INPUT/INPUT-OUTPUT only	To include only those fields with INPUT and INPUT-OUTPUT structures, mark this field.
OUTPUT/INPUT-OUTPUT only	To include only those fields with OUTPUT and INPUT-OUTPUT structures, mark this field.

Field	Description (continued)
Specific structure	Name of the structure containing the fields. To include only the fields from a specific structure, specify the name of the structure.
Include Field Rank Selection	
Scalar fields	To include all scalar fields, mark this field. If you do not want to include the fields, enter a blank in the field.
One-dimensional arrays	To include all one-dimensional arrays, mark this field.
Two-dimensional arrays	To include all two-dimensional arrays, mark this field.
Three-dimensional arrays	To include all three-dimensional arrays, mark this field.

Tailoring the Predict View

If you specified a Predict file name in the ARBITRARY-FIELD-PROCESSING Function window, press PF5 (optns) to tailor the Predict view. The Predict view name *modulename* window is displayed. For example:

```
                    Predict view name NCST-CUSTOMER

  Include Redefined Fields              Include Descriptors
  Base field ................... X        Super or subdescriptors ...... _
  FILLER fields ................ X        Hyperdescriptors ............. _
  All redefinitions ............ X
  First redefinition ........... _
  Maximum array size (1,2,3) ... 3

  Include PE Groups                     Include MU Fields
  Periodic group elements ..... X      Multiple-valued fields ...... X
  Periodic group field ......... _      Multiple-valued fields PE .... X

  Include DB2 Options
  Derived fields ............... _
  Null fields .................. _
```

Predict View Name Window

The fields in this window are:

Field	Description
Include Redefined Fields	
Base field	To include the base fields of redefined fields, mark this field.
FILLER field	To include Filler fields when generating field definitions, mark this field.
All redefinitions	To include all redefinitions of a field, mark this field.
First redefinition	To include only the first redefined field, mark this field.
Maximum array size (1,2,3)	To specify the maximum array size to be included, enter the size number in this field.
Include Descriptors	
Super or subdescriptors	To include super or subdescriptors, mark this field.
Hyperdescriptors	To include hyperdescriptors, mark this field.
Include PE Groups	
Periodic group elements	To include periodic group elements, mark this field.
Periodic group field	To include the periodic group field, mark this field.
Include MU Fields	
Multiple-valued fields	To include multiple-valued fields, mark this field.
Multiple-valued fields PE	To include multiple-valued fields within periodic groups, mark this field.
Include DB2 Options	
Derived fields	To include DB2 files and derived fields, mark this field.
Null fields	To include DB2 files and fields that support nulls, mark this field.

Arbitrary-Field-Processing Statement Model Example

The following specifications generate an Arbitrary-Field-Processing function:

```
                ARBITRARY-FIELD-PROCESSING Function

LDA or PDA name ..............  ________ *
Predict view name ............. CST-EMPLOYEES_____________________ *

Processing For Each Field
  &1& = Fully-qualified field name (eg., EMPLOYEES.FIRST-NAME)
  &2& = Field heading (eg., First Name)
  &3& = Null value (eg., format N = 0, A = ' ', L = True, etc.
  &4& = Unqualified field name (eg., FIRST-NAME)
  /* if &2& equals null assign default value_____________________
  IF &1& = &3&_______________________________________________________
    REINPUT '&2& is required'________________________________________
    MARK *&1& ALARM__________________________________________________
  END-IF_____________________________________________________________
  /*_____________________________________________________________
  _____________________________________________________________
  _____________________________________________________________
  _____________________________________________________________
  _____________________________________________________________
```

Arbitrary-Field-Processing Statement Model Example

```
                    Predict view name CST-EMPLOYEES

Include Redefined Fields                Include Descriptors
Base field ................... X       Super or subdescriptors ...... _
FILLER fields ................ X       Hyperdescriptors ............. _
All redefinitions ............ X
First redefinition ........... _
Maximum array size (1,2,3) ... 3

Include PE Groups                       Include MU Fields
 Periodic group elements ..... X      Multiple-valued fields ...... X
 Periodic group field ......... _     Multiple-valued fields PE .... X

Include DB2 Options
Derived fields ............... _
Null fields .................. _
```

Tailoring the Predict View Fields
for the Arbitrary-Field-Processing Model Example

Example of code generated from the sample specifications

```
0010 /* if PERSONNEL ID equals null assign default value
0020 IF CST-EMPLOYEES.PREFIX-PERSONNEL-ID = ' '
0030   REINPUT 'PERSONNEL ID is required'
0040   MARK *CST-EMPLOYEES.PREFIX-PERSONNEL-ID ALARM
0050 END-IF
0060 /*
0070 /* if First Name equals null assign default value
0080 IF CST-EMPLOYEES.PREFIX-FIRST-NAME = ' '
0090   REINPUT 'First Name is required'
0100   MARK *CST-EMPLOYEES.PREFIX-FIRST-NAME ALARM
0110 END-IF
0120 /*
0130 /* if Fld equals null assign default value
0140 IF CST-EMPLOYEES.PREFIX-FLD(*) = ' '
0150   REINPUT 'Fld is required'
0160   MARK *CST-EMPLOYEES.PREFIX-FLD(*) ALARM
0170 END-IF
0180 /*
0190 /* if Fld 1 equals null assign default value
```

At-Break Statement Model

The At-Break statement model (AB) generates a Natural AT BREAK statement, which causes a specified statement (or statements) to be executed whenever a control field value changes or processing ends. This statement is used in conjunction with automatic break processing and is available with the FIND, READ, HISTOGRAM, SORT, and READ WORK FILE statements.

Generating the Default Code into the Editor

To generate the AT BREAK statement, enter one of the following line commands:

```
.g(AT-BREAK,)
```

or

```
.g(AB,)
```

The following default code is generated:

```
0010 *
0020 *
0030 /*
0040 /*
0050      AT BREAK OF #field-name
0060         /*
0070         /* Perform .....
0080         /*
0090         /*
0100         /*
0110         /*
0120         /*
0130      END-BREAK
0140 *
0150 *
0160 *
```

You can then replace the variables to complete the AT-BREAK statement.

Using the AT-BREAK Statement Window

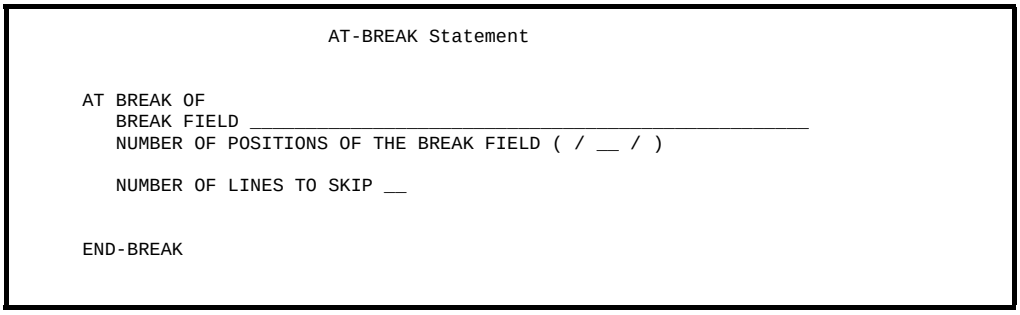
The AT-BREAK Statement window is displayed when you invoke the statement model from the Generation main menu. You can also invoke the AT-BREAK Statement window from an editor by entering either of the following line commands:

```
.g(AT-BREAK)
```

or

```
.g(AB)
```

The AT-BREAK Statement window is displayed:



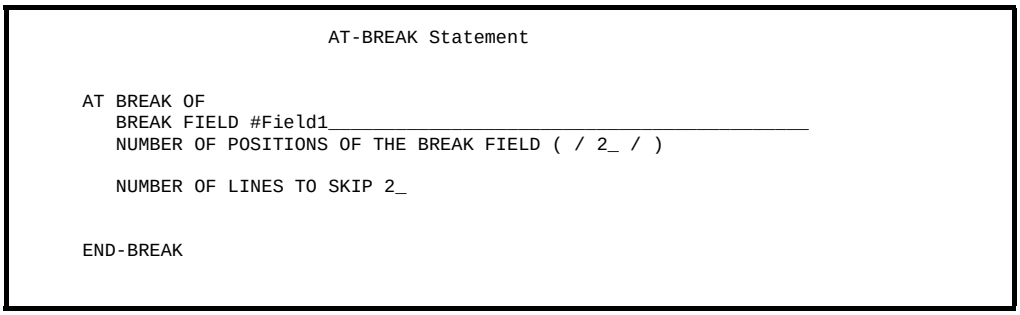
AT-BREAK Statement Window

The fields in this window are:

Field	Description
BREAK FIELD	Name of the break field, which is usually a database field. If a user-defined variable is used, it must be initialized prior to the evaluation of the automatic break processing.
NUMBER OF POSITIONS OF THE BREAK FIELD	Number of positions of the break field to check.
NUMBER OF LINES TO SKIP	Number of lines to skip after the break field.

At-Break Statement Model Example

The following specifications generate an AT BREAK statement:



At-Break Statement Model Example

Example of code generated from the sample specifications

```
0010 *
0020 *
0030 /*
0040 /*
0050      AT BREAK OF #FIELD1 /2/
0060         /*
0070         /* Perform .....
0080         /*
0090         /*
0100         /*
0110      END-BREAK
0120 *
```

At-Top-of-Page Statement Model

The At-Top-of-Page statement model (TP) generates a Natural AT TOP OF PAGE statement, which specifies the processing performed when a new page is started.

Generating the Default Code into the Editor

To generate the AT-TOP-OF-PAGE statement, enter one of the following line commands:

```
.g(AT-TOP-OF-PAGE,)
```

or

```
.g(TP,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 AT TOP OF PAGE
0040   WRITE NOTITLE *PROGRAM
0050     34T 'Heading line 1'
0060    72T 'Page' *PAGE-NUMBER (NL=3 AD=L SG=OFF)
0070      / *DATX(EM=&DATE-EM)
0080     34T 'Heading line 2'
0090     72T *TIMX(EM=&TIME-EM)
0100   SKIP 1
0110 END-TOPPAGE
0120
```

You can then replace the variables to complete the AT-TOP-OF-PAGE statement.

Using the AT-TOP-OF-PAGE Statement Window

The AT-TOP-OF-PAGE Statement window is displayed when you invoke the statement model from the Generation main menu. You can also invoke the AT-TOP-OF-PAGE Statement window from an editor by entering either of the following line commands:

```
.g(AT-TOP-OF-PAGE)
```

or

```
.g(TP)
```

The AT-TOP-OF-PAGE Statement window is displayed:

```
                        AT-TOP-OF-PAGE Statement

AT-TOP-OF-PAGE  Number ...  __
                Name .....  ________________________________
  WRITE NOTITLE
      *PROGRAM _
    Heading .. ____________________________________________________________
  *PAGE-NUMBER _
    /
         *DATX _
    Heading .. ____________________________________________________________
         *TIMX _
    /
    Heading .. ____________________________________________________________
  SKIP 1
END-TOPPAGE

Line size ...... 80_
Frame OFF ...... _
```

AT-TOP-OF-PAGE Statement Window

The fields in this window are:

Field	Description
AT-TOP-OF-PAGE Number	Numeric value (0 to 31) that identifies the report for which the AT-TOP-OF-PAGE statement applies.
Name	Logical name for the report (if it has a logical name).
Note:	If you do not specify a value in either the Number or Name field, the AT-TOP-OF-PAGE statement applies to the first report (Report 0).
WRITE NOTITLE	
*PROGRAM	To display the name of the program on the first line in the top left corner of the specified report, mark this field.
Heading	To display a heading on the first line of the report, type the heading in this field.
*PAGE-NUMBER	To display the report page number on the first line in the top right corner, mark this field.
*DATX	To display the date on the second line down from the top left corner, mark this field. By default, the date is displayed as a three-character month, a two-digit day, and a two-digit year (Jul 20, 93, for example).

Field	Description (continued)
Heading	To display a heading on the second line of the report, type the heading in this field.
*TIMX	To display the time on the second line down from the top right corner, mark this field. By default, the time is displayed as a two-digit hour, a two-digit minute, and AM or PM (11:23 AM, for example).
Heading	To display a heading in the third line of the report, type the heading in this field.
Line size	Number of characters per line on the report (the default is 80). This value is used for centering purposes only.
Frame OFF	By default, a frame (border) is displayed around the report window. If you do not want to display a frame, mark this field.

All headings are centered on the report.

Note: The location of the above fields in the generated TOP OF PAGE block may differ for your installation.

At-Top-of-Page Statement Model Example

The following specifications generate an AT TOP OF PAGE statement:

```
                        AT-TOP-OF-PAGE Statement

AT-TOP-OF-PAGE  Number ...  5_
                Name .....  _____________________________
  WRITE NOTITLE
      *PROGRAM _
    Heading .. Generating New Information_____________________________
  *PAGE-NUMBER _
    /
         *DATX _
    Heading .. Generating Generic Information_____________________________
         *TIMX _
    /
    Heading .. _____________________________________________________________
  SKIP 1
END-TOPPAGE

Line size ...... 80_
Frame OFF ...... _
```

At-Top-of-Page Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 AT TOP OF PAGE(5)
0040   WRITE(5) NOTITLE
0050     28T 'Generating New Information'
0060       /
0070     26T 'Generating Generic Information'
0080   SKIP(5) 1
0090 END-TOPPAGE
```

Callnat Statement Model

The Callnat statement model (CN) generates a Natural CALLNAT statement, which calls a Natural subprogram for execution.

Generating the Default Code into the Editor

If you know the subprogram you want to call in the CALLNAT statement, you can generate the code quickly by entering one of the following line commands in the editor:

`.g(CALLNAT,`*`subprogram`*`)`

or

`.g(CN,`*`subprogram`*`)`

where *subprogram* is the name of a valid subprogram.

The following default code is generated:

```
0010   CALLNAT 'CDUTRANS'
0020            #TRANSLATION-DATA(*)
0030            #DATA-AREA-NAME
0040            MSG-INFO
```

Note: The parameters must be defined in the DEFINE DATA section of your generated program.

Using the Callnat Statement Window

You can invoke the Callnat statement model from the Generation main menu, as you do any program statement model. You can also invoke the Callnat Statement window from an editor by entering one of the following line commands:

`.g(CALLNAT,)`

or

`.g(CN,)`

or

`.g(CALLNAT)`

or

`.g(CN)`

The Callnat Statement window is displayed:

Callnat Statement Window

The only field in this window is the Callnat field. In this field, type the name of the subprogram to invoke or press PF1 (help) to select from a list of subprograms. You must specify the correct name of an existing subprogram and the source code for the specified subprogram must exist in the current library or steplib.

Note: Natural Construct locates the required parameters in the CALLNAT statement and supplies them to your CALLNAT statement.

Callnat Statement Model Example

The following specifications generate a CALLNAT statement:

Callnat Statement Model Example

Example of code generated from the sample specifications

```
CALLNAT  'ORDERN'
  ORDER
  ORDERPDA-ID
  ORDERPDR
  CDAOBJ
  MSG-INFO
```

Close Statement Model

The Close statement model (CR or CW)) generates a Natural CLOSE statement for a printer or work file. You can use this statement within a program to close a printer/work file.

Generating the Default Code into the Editor

To generate a CLOSE statement, enter one of the following line commands:

```
.g(CLOSE,)
```

or

```
.g(CR,)
```

or

```
.g(CW,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 CLOSE PRINTER (1)
0040 CLOSE WORK FILE 01
```

Revise the code to complete the CLOSE statement.

Using the CLOSE Statement Window

The CLOSE Statement window is displayed when you invoke the Close statement model from the Generation main menu. You can also invoke the CLOSE Statement window from an editor by entering either of the following line commands:

```
.g(CLOSE)
```

or

```
.g(CR)
```

or

```
.g(CW)
```

The CLOSE Statement window is displayed:

```
                      CLOSE Statement

CLOSE PRINTER  Number ... __
               Name ..... ____________________

CLOSE WORK FILE  Number ... __
```

CLOSE Statement Window

Note: A printer file is closed automatically when a DEFINE PRINTER statement for the same printer is executed or command mode is reached. A work file is closed automatically when command mode is reached.

The fields in this window are:

Field	Description
CLOSE PRINTER Number	Number of the printer you want to close. Valid printer numbers are 0 to 31.
Name	Logical name of the printer you want to close.
Note:	Name must be the same as was defined for a printer in the DEFINE PRINTER statement.
CLOSE WORK FILE Number	Number of the work file (as defined in Natural) you want to close.

Close Statement Model Example

The following specifications generate a CLOSE statement:

```
                        CLOSE Statement

   CLOSE PRINTER  Number ... 3_
                  Name ..... _____________________________


   CLOSE WORK FILE  Number ... 7_
```

Close Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 CLOSE PRINTER (3)
0040 CLOSE WORK FILE 07
```

Compress Statement Model

The Compress statement model (CO) generates a Natural COMPRESS statement, which combines the contents of two or more operands into a single field.

Generating the Default Code into the Editor

To generate a COMPRESS statement, enter one of the following line commands:

```
.g(COMPRESS,)
```

or

```
.g(CO,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 COMPRESS source1
0040          source2
0050    TO target-field
0060      LEAVING NO SPACE
0070
```

Revise the code to complete the COMPRESS statement.

Using the COMPRESS Statement Window

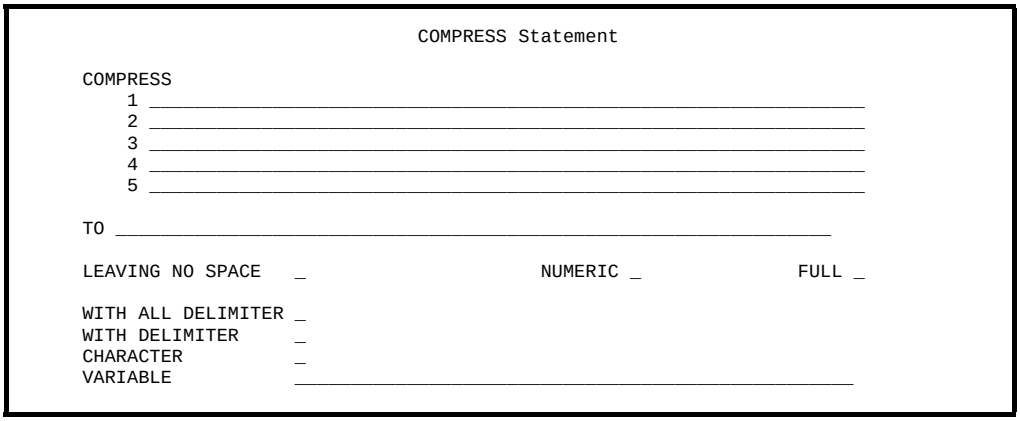
The COMPRESS Statement window is displayed when you invoke the Compress statement model from the Generation main menu. You can also invoke the COMPRESS Statement window by entering one of the following line commands:

```
.g(COMPRESS)
```

or

```
.g(CO)
```

The COMPRESS Statement window is displayed:



COMPRESS Statement Window

The fields in this window are:

Field	Description
COMPRESS 1,2,3,4,5	Names of the fields that are compressed into one new field (one field per line). To include text in the compressed field, enclose the text within single quotes on a separate line. This is a required field.
TO	Name of the compressed field (must be an alphanumeric variable; can be one of the source fields). This is a required field.
LEAVING NO SPACE	If you do not want spaces inserted between each source field in the compressed field, mark this field.
NUMERIC	This option determines how sign characters and decimal characters are to be handled: • Without NUMERIC, decimal points and signs in numeric source values are suppressed before the values are transferred. • With NUMERIC, decimal points and signs in numeric source values are also transferred to the target field.

Field	Description (continued)
FULL	Without FULL, leading zeros (in numeric fields) and trailing blanks (in alphanumeric fields) are removed from the source fields before the values are transferred. For a numeric source field containing all zeros, one “0” will be transferred. With FULL, the values of the source fields in their actual lengths - that is, including leading zeros and trailing blanks - will be transferred to the target field.
WITH ALL DELIMITER	Without ALL, a delimiter is placed in the target field only between values actually transferred. With ALL, a delimiter is also placed in the target field for each blank value that is not actually transferred. This means that the number of delimiters in the target field corresponds to the number of source fields minus 1.
WITH DELIMITER	If you want the default input delimiter character inserted between each source field, mark this field. For example, if the source fields are “one,” “two,” and “three,” and the input delimiter is a “/” (ID=/), the resulting field is “one/two/three.” (Use the GLOBALS command to see your current default input delimiter.)
CHARACTER	If you want a character inserted between each source field, specify the character in this field.
VARIABLE	If a variable contains the delimiter character you want to use, specify the variable name in this field.
Note:	One of the DELIMITER, CHARACTER, or VARIABLE fields must be specified.

Compress Statement Model Example

The following specifications generate a COMPRESS statement:

```
                    COMPRESS Statement

COMPRESS
    1 variable1_____________________________________________________________
    2 variable2_____________________________________________________________
    3 variable3_____________________________________________________________
    4 ______________________________________________________________________
    5 ______________________________________________________________________
TO varm_____________________________________________________________________

LEAVING NO SPACE   _                    NUMERIC _             FULL _

WITH ALL DELIMITER _
WITH DELIMITER     x
CHARACTER          _
VARIABLE           ________________________________________________
```

COMPRESS Statement Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 COMPRESS VARIABLE1
0040          VARIABLE2
0050          PARMATER1
0060    TO VARM
0070      WITH DELIMITER
0080
```

Decide-For Statement Model

The Decide-For statement model (DF) generates a DECIDE FOR statement, which decides what actions are performed and under what conditions. Each condition is referred to as a case.

Generating the Default Code into the Editor

To generate the DECIDE FOR statement, enter one of the following line commands:

```
.g(Decide-For,)
```

or

```
.g(DF,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 DECIDE FOR FIRST CONDITION
0040   WHEN #VARIABLE = ...
0050     ASSIGN ...
0060   WHEN #VARIABLE2 = ...
0070     ASSIGN ...
0080   WHEN ANY
0090     any condition is true
0100   WHEN ALL
0110     all conditions are true
0120   WHEN NONE
0130     IGNORE
0140 END-DECIDE
```

Revise the code to complete your DECIDE FOR statement.

Using the DECIDE-FOR Statement Window

The DECIDE-FOR Statement window is displayed when you invoke the Decide-For statement model from the Generation main menu. You can also invoke the DECIDE-FOR Statement window by entering one of the following line commands:

```
.g(DECIDE-FOR)
```

or

```
.g(DF)
```

The DECIDE-FOR Statement window is displayed:

```
                        DECIDE-FOR Statement

    DECIDE FOR FIRST X EVERY _
>> 1 WHEN #VARIABLE = ...___________________________________________
       ASSIGN ...___________________________________________________
       _____________________________________________________________
       _____________________________________________________________
       _____________________________________________________________

      WHEN ANY
        _____________________________________________________________
        _____________________________________________________________
      WHEN ALL
        _____________________________________________________________
        _____________________________________________________________
      WHEN NONE
        IGNORE_______________________________________________________
        _____________________________________________________________
    END-DECIDE
```

DECIDE-FOR Statement Window

The fields in this window are:

Field	Description
DECIDE FOR FIRST	To process the first true condition only, mark this field.
EVERY	To process every true condition, mark this field.
Note:	You must mark one of the above fields (FIRST or EVERY).
>> 1	Indicates the current case number (case 1, for example). If you have more than one case, press PF8 (frwrd) to specify the next case. Press PF7 (bkwrd) to display the previous case. You must specify at least one case; you can specify up to nine cases.
Note:	You can copy conditions/statements from one case to the next by pressing PF8 to advance to the next case and pressing PF5 (copy) to copy the conditions/statements from the previous case. You can then modify the copied information.
WHEN *logical condition* =	Logical condition(s) to be processed. On the blank lines, list the action(s) performed when the associated condition is true.
WHEN ANY	Action(s) performed when any of the conditions are true. This is an optional field.

Field	Description (continued)
WHEN ALL	Action(s) performed when all of the conditions are true. (You cannot use this field if you marked the FIRST field.) This is an optional field.
WHEN NONE	Action(s) performed when none of the conditions are true. By default, no action is performed when no conditions are true. This is a required field.

Decide-For Statement Model Example

The following specifications generate a DECIDE FOR statement:

```
                        DECIDE-FOR Statement

   DECIDE FOR FIRST X EVERY _
>> 1 WHEN #FUNCTION = 'A' AND #PARM = 'X'_____________________________
     PERFORM ROUTINE-A_____________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________
     _____________________________________________________________________

     WHEN ANY
     _____________________________________________________________________
     _____________________________________________________________________
     WHEN ALL
     _____________________________________________________________________
     _____________________________________________________________________
     WHEN NONE
     IGNORE_______________________________________________________________
     _____________________________________________________________________
   END-DECIDE
```

Decide-For Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 DECIDE FOR FIRST CONDITION
0040   WHEN #FUNCTION = 'A' AND #PARM = 'X'
0050     PERFORM ROUTINE-A
0060   WHEN NONE
0070     IGNORE
0080 END-DECIDE
```

Decide-On Statement Model

The Decide-On statement model (DO) generates a Natural DECIDE ON statement, which specifies multiple actions to be performed depending on the value(s) contained in a variable.

Generating the Default Code into the Editor

To generate a DECIDE ON statement, enter one of the following line commands:

```
.g(Decide-On,)
```

or

```
.g(DO,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 DECIDE ON FIRST #NUMBER
0040   VALUE 1
0050     ASSIGN ...
0060   VALUE 2
0070     ASSIGN ...
0080   ANY
0090     if any are true
0100   ALL
0110     if all are true
0120   NONE
0130     IGNORE
0140 END-DECIDE
0150
```

Revise the code to complete your DECIDE-ON statement.

Using the DECIDE-ON Statement Window

The DECIDE-ON Statement window is displayed when you invoke the Decide-On statement model from the Generation main menu. You can also invoke the DECIDE-ON Statement window from an editor by entering one of the following line commands:

```
.g(Decide-On)
```

or

```
.g(DO)
```

The DECIDE-ON Statement window is displayed:

```
                        DECIDE-ON Statement

 DECIDE ON FIRST X EVERY _  VALUE OF #NUMBER_____________________
>> 1 VALUE 1_____________________________________________________
       ASSIGN ..._____________________________________________
       _______________________________________________________
       _______________________________________________________
       _______________________________________________________

     ANY VALUE
       _____________________________________________________
       _____________________________________________________
     ALL VALUE
       _____________________________________________________
       _____________________________________________________
     NONE VALUE
       IGNORE_____________________________________________
       _____________________________________________________
 END-DECIDE
```

DECIDE-ON Statement Window

The fields in this window are:

Field	Description
DECIDE ON FIRST	To process only the first value, mark this field.
EVERY	To process every value, mark this field.
Note:	You must mark one of the above fields (FIRST or EVERY).
VALUE OF	Name of the field that is checked for the specified values. This is a required field.
>> 1	Indicates the current value number (value1, for example). If you have more than one value, press PF8 (frwrd) to specify the next case. Press PF7 (bkwrd) to display the previous value. You must specify at least one value; you can specify up to nine values.
Note:	You can copy values from one case to the next by pressing PF8 to advance to the next case and pressing PF5 to copy the values from the previous case. You can then modify the copied information.

Field	Description (continued)
VALUE	Value that is checked for, which is one of the following (text strings must be enclosed within single quotes): • one value • multiple values separated by commas • a range of values separated by a colon In the fields below, specify the action(s) performed for the specified value(s).
ANY VALUE	Statements executed when any of the values in the VALUE field are found. These statements are executed in addition to the statements specified in the VALUE field. This is an optional field.
ALL VALUE	Statements executed when all of the values in the VALUE field are found. These statements are executed in addition to the statements specified in the VALUE field. (This option cannot be used if the FIRST field was marked.) This is an optional field.
NONE VALUE	Statements executed when no values are found. By default, no statements are executed. This is a required field.

Decide-On Statement Model Example

The following specifications generate a DECIDE ON statement:

```
                        DECIDE-ON Statement

 DECIDE ON FIRST X EVERY _  VALUE OF *PF-KEY_____________________
>> 1 VALUE 'PF1'_____________________________________________________
       PERFORM ROUTINE-UPD_____________________________________________
       _________________________________________________________________
       _________________________________________________________________
       _________________________________________________________________

     ANY VALUE
       _________________________________________________________________
       _________________________________________________________________
     ALL VALUE
       _________________________________________________________________
       _________________________________________________________________
     NONE VALUE
       IGNORE__________________________________________________________
       _________________________________________________________________
 END-DECIDE
```

Decide-On Statement Model Example for Value 1

```
                        DECIDE-ON Statement

 DECIDE ON FIRST X EVERY _  VALUE OF *PF-KEY_____________________
>> 2 VALUE 'PF2'_____________________________________________________
     PERFORM ROUTINE-ADD_____________________________________________
     ________________________________________________________________
     ________________________________________________________________
     ________________________________________________________________

   ANY VALUE
     END TRANSACTION_____________________________________________
     WRITE 'Record has been modified'____________________________
   ALL VALUE
     ________________________________________________________________
     ________________________________________________________________
   NONE VALUE
     IGNORE__________________________________________________________
     ________________________________________________________________
 END-DECIDE
```

Decide-On Statement Model Example for Value 2

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 DECIDE ON FIRST *PF-KEY
0040   VALUE 'PF1'
0050     PERFORM ROUTINE-UPD
0060   VALUE 'PF2'
0070     PERFORM ROUTINE-ADD
0080   ANY
0090     END TRANSACTION
0100     WRITE 'Record has been modified'
0110   NONE
0120     IGNORE
0130 END-DECIDE
```

Define-Subroutine Statement Model

The Define-Subroutine statement model (DS) generates a Natural DEFINE SUBROUTINE statement, which defines a subroutine using a PERFORM statement.

Generating the Default Code into the Editor

To generate a DEFINE SUBROUTINE statement, enter one of the following line commands:

```
g(Define-Subroutine,)
```

or

```
g(DS,)
```

The following default code is generated:

```
0010 /* PERFORM subroutine-name
0020 *
0030 ************************************************************
0040 DEFINE SUBROUTINE subroutine-name
0050 ************************************************************
0060 *
0070 END-SUBROUTINE /* subroutine-name
```

Revise the code to complete the DEFINE SUBROUTINE statement.

Using the DEFINE-SUBROUTINE Statement Window

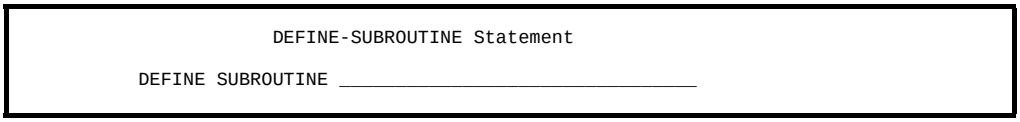
The DEFINE-SUBROUTINE Statement window is displayed when you invoke the Define-Subroutine statement model from the Generation main menu. You can also invoke the DEFINE-SUBROUTINE Statement window from an editor by entering one of the following line commands:

```
.g(Define-Subroutine)
```

or

```
.g(DS)
```

The DEFINE-SUBROUTINE Statement window is displayed:



DEFINE-SUBROUTINE Statement Window

Type the name of the subroutine in the DEFINE SUBROUTINE field. When you press Enter, the default code is generated into the editor with the specified name.

Define-Subroutine Statement Model Example

The following specifications generate a DEFINE SUBROUTINE statement:

```
                    DEFINE-SUBROUTINE Statement

        DEFINE SUBROUTINE CSTSUBRT_________________________
```

Define-Subroutine Statement Model Example

Example of code generated from the sample specifications

```
0010 /* PERFORM CSTSUBRT
0020 *
0030 ************************************************
0040 DEFINE SUBROUTINE CSTSUBRT
0050 ************************************************
0060 *
0070 END-SUBROUTINE /* CSTSUBRT
```

Define-Printer Statement Model

The Define-Printer statement model (DP) generates a DEFINE PRINTER statement.

Generating the Default Code into the Editor

To generate default code for a DEFINE PRINTER statement, enter one of the following line commands:

```
g(Define-Printer,)
```

or

```
g(DP,)
```

The default code is generated. To complete the DEFINE PRINTER statement, revise the code as desired.

Using the Define-Printer Window

The Define-Printer window is displayed when you invoke the Define-Printer statement model from the Generation main menu. You can also invoke the Define-Printer window from an editor by entering one of the following line commands:

```
.g(Define-Printer)
```

or

```
.g(DP)
```

The Define-Printer window is displayed:

```
DEFINE PRINTER ( ____________________________________ = __  )

    OUTPUT  ________
    PROFILE ________
    FORMS   ________
    NAME    ________
    DISP    ____
    CLASS   _
    COPIES  ___
    PRTY    __
```

Define Printer Window

For information on the parameters for the DEFINE PRINTER statement, refer to the Natural documentation.

Define-Variable Statement Model

The Define-Variable statement model (DA) generates the code required to declare and initialize a Natural data variable.

Generating the Default Code into the Editor

To generate a Define-Variable function, enter one of the following line commands:

```
g(Define-Variable,)
```

or

```
g(DA,)
```

The following default code is generated:

```
0010 /*
0020 01 #variable (A5)
0030        INIT <'initial-value'>
0040
```

Revise the code to complete the Define-Variable function.

Using the DEFINE-VARIABLE Statement Windows

The first DEFINE-VARIABLE Statement window is displayed when you invoke the Define-Variable statement model from the Generation main menu. You can also invoke the DEFINE-VARIABLE Statement window from an editor by entering one of the following line commands:

```
.g(Define-Variable)
```

or

```
.g(DA)
```

The first DEFINE-VARIABLE Statement window is displayed:

```
                    DEFINE-VARIABLE Statement 1 of 2

VARIABLE _____________________________  CONSTANT _

  FORMAT _ _____ *

   ARRAY         FROM INDEX                  TO INDEX                OCC
  ------ ---------------------------- ---------------------------- ---
     1   ____________________________ ____________________________ ___
     2   ____________________________ ____________________________ ___
     3   ____________________________ ____________________________ ___

  EM= _____________________________

  HD= _____________________________

  PM: _ N I C
```

DEFINE-VARIABLE Statement — Window 1

The fields in this window are:

Field	Description
VARIABLE	Name that is assigned to the variable. (All rules for Natural variable names apply.)
CONSTANT	To treat the variable as a named constant, mark this field. The assigned value is used each time the variable is referenced and cannot be modified during program execution.
FORMAT	Natural format and length of the variable field.
ARRAY	Dimensions of an array. You can specify up to three dimensions of an array for each variable.
FROM INDEX	Starting index for the dimension. (The starting index can be derived from the ending index and the number of occurrences.)
TO INDEX	Ending index for the dimension. (The ending index can be derived from the starting index and the number of occurrences.)
Note:	The FROM INDEX and TO INDEX values can be numeric integer constants or a previously-defined, named constant. For database arrays, the value can be a previously-defined, user-defined variable.
OCC	Number of occurrences for the dimension.
EM	Edit mask for the field (if one is required).

Field	Description (continued)
HD	Default header for the field. To display the header on multiple lines, use a "/" character to indicate where the header is divided.
PM	Code for the print mode of the field. The print mode indicates how the field is displayed on a panel. Valid print modes are: • N (non-unprintable) • I (inverse direction, used in Middle East countries) • C (an alternative character set)

After filling in the VARIABLE and FORMAT fields on the first DEFINE-VARIABLE Statement window, press PF11 (right) to display the second window:

```
                    DEFINE-VARIABLE Statement 2 of 2

INIT  _____________________________

INIT ALL  _____________________________

       ARRAY 1      ARRAY 2      ARRAY 3             VALUE
     FROM TO ALL  FROM TO ALL  FROM TO ALL  ------------------------------
    1 __  __  _    __  __  _    __  __  _   _____________________________
    2 __  __  _    __  __  _    __  __  _   _____________________________
    3 __  __  _    __  __  _    __  __  _   _____________________________
    4 __  __  _    __  __  _    __  __  _   _____________________________
    5 __  __  _    __  __  _    __  __  _   _____________________________
    6 __  __  _    __  __  _    __  __  _   _____________________________
    7 __  __  _    __  __  _    __  __  _   _____________________________
    8 __  __  _    __  __  _    __  __  _   _____________________________

FULL LENGTH  _
LENGTH ___
```

DEFINE-VARIABLE Statement — Window 2

Use this window to specify initial field values. The fields in this window are:

Field	Description
INIT	To initialize a single variable, specify the initial value in this field.
INIT ALL	To initialize all occurrences in all dimensions with the same value, specify the value in this field.

Field	Description (continued)
ARRAY 1,2,3 FROM TO ALL VALUE	To initialize each occurrence (entry by entry) for each dimension, do one of the following: • specify the FROM and TO occurrence numbers • mark ALL and specify the initial value in the VALUE field
FULL LENGTH	For alphanumeric arrays, a single character or string of characters can fill (entirely or partially) the array as an initial value. To fill the entire field with the initial value, mark this field.
LENGTH	To fill only the first *n* positions of the field with the initial value, specify the number of positions.

Define-Variable Statement Model Example

The following example shows the specifications to generate a DEFINE VARIABLE statement:

```
                    DEFINE-VARIABLE Statement 1 of 2

VARIABLE #COLOR_______________________________  CONSTANT _

  FORMAT A 5.0__ *

   ARRAY          FROM INDEX                      TO INDEX                  OCC
  ------ ------------------------------- ------------------------------- ---
     1   1_____________________________ 4_____________________________ 4__
     2   1_____________________________ 4_____________________________ 4__
     3    _____________________________ _____________________________ ___

  EM= X''X''X''X''X____________________

  HD= ___________________________________

  PM: _ N I C
```

Define-Variable Statement Model Example — Window 1

```
                    DEFINE-VARIABLE Statement 2 of 2

INIT  _____________________________

INIT ALL  _____________________________

       ARRAY 1      ARRAY 2      ARRAY 3              VALUE
     FROM TO ALL  FROM TO ALL  FROM TO ALL  ----------------------------
    1 __  __  X    1_  4_  _    __  __  _   'BLUE','GREEN','BLACK','WHITE'
    2 __  __  _    __  __  _    __  __  _   _____________________________
    3 __  __  _    __  __  _    __  __  _   _____________________________
    4 __  __  _    __  __  _    __  __  _   _____________________________
    5 __  __  _    __  __  _    __  __  _   _____________________________
    6 __  __  _    __  __  _    __  __  _   _____________________________
    7 __  __  _    __  __  _    __  __  _   _____________________________
    8 __  __  _    __  __  _    __  __  _   _____________________________

FULL LENGTH  _
LENGTH ___
```

Define Variable Statement Model Example — Window 2

Example of code generated from the sample specifications

```
0010 01 #COLOR (A5/1:4,1:4)
0020    INIT (V,1:4) <'BLUE','GREEN','BLACK','WHITE'>
0030    (EM=X''X''X''X''X)
```

Define-Window Statement Model

The Define-Window statement model (DW) generates a Natural DEFINE WINDOW statement. This statement defines the size, position, and attributes of a window.

Generating the Default Code into the Editor

To generate a DEFINE WINDOW statement, enter one of the following line commands:

```
.g(DEFINE-WINDOW,)
```

or

```
.g(DW,)
```

The following default code is generated:

```
0010 /*
0020 /* Define Window #window-name
0030 DEFINE WINDOW #window-name
0040               SIZE 13 * 40
0050               BASE TOP RIGHT
0060               CONTROL WINDOW
0070               FRAMED ON
0080               POSITION TEXT
```

Revise the code to complete the DEFINE WINDOW statement.

Using the DEFINE-WINDOW Statement Windows

The first of two DEFINE-WINDOW Statement windows is displayed when you invoke the Define-Window statement model from the Generation main menu. You can also invoke the DEFINE-WINDOW Statement windows from an editor by entering one of the following line commands:

```
.g(DEFINE-WINDOW)
```

or

```
.g(DW)
```

The first DEFINE-WINDOW statement window is displayed:

```
                    DEFINE-WINDOW Statement 1 of 2

DEFINE WINDOW ____________________________

  SIZE  AUTO _
        QUARTER _
        Height ....... ____________________
        Width ........ ____________________

  BASE  CURSOR _
        TOP  LEFT _  BOTTOM  LEFT _
            RIGHT _         RIGHT _
        Line ......... ____________________
        Column ....... ____________________

  REVERSED _ ( CD=BL _ GR _ NE _ PI _ RE _ TU _ YE _ )

  TITLE ____________________________________________________________
```

DEFINE-WINDOW Statement — Window 1

The fields in this window are:

Field	Description
DEFINE WINDOW	Name of the window. (All rules for Natural variable names apply.) This is a required field.
SIZE	You can have the window size set automatically or you can specify an exact size in the following fields:
AUTO	To set the window size according to the default when the program is executed, mark this field.
QUARTER	To make the window one quarter (25%) of the size of the physical screen, mark this field.
	To assign a specific window size that is different from the first two options, specify the height and width of the window in the following fields. The maximum window size is the size of a full screen.
Height	Number of lines (from top to bottom) the window spans. The minimum is 2 lines without a frame and 4 lines with one.
Width	Number of columns (from left to right) the window spans. The minimum is 10 columns without a frame and 12 columns with one.

Field	Description (continued)
BASE	Position of the window in relation to the full screen. This is an optional setting. To specify the position, mark one of the following options:
CURSOR	To position the top left corner of the window at the current cursor position, mark this field.
TOP LEFT	To position the window at the top left corner of the screen, mark this field.
TOP RIGHT	To position the window at the top right corner of the screen, mark this field.
BOTTOM LEFT	To position the window at the bottom left corner of the screen, mark this field.
BOTTOM RIGHT	To position the window at the bottom right corner of the screen, mark this field.
Line	To position the top left corner of the window at a particular line, specify the line number. (Line numbers must be specified as a numeric constant or variable.)
Column	To position the top left corner of the window at a particular column, specify the column number. (Column numbers must be specified as a numeric constant or variable.)
REVERSED	If the terminal supports reversed video and/or color, you can mark this field and specify the following color codes: • BL (blue) • GR (green) • NE (neutral/white) • PI (pink) • RE (red) • TU (turquoise) • YE (yellow)
TITLE	If the window has a frame, specify a title for the window in this field. The title can be a variable or a text constant, which must be enclosed within single quotes. This is an optional field.

After defining at least the window name, press PF11 (right) to display the second DEFINE-WINDOW statement window:

```
            DEFINE-WINDOW Statement 2 of 2

CONTROL  WINDOW _
         SCREEN _

FRAMED   ON _  CD=BL _ GR _ NE _ PI _ RE _ TU _ YE _
        OFF _
        POSITION
          SYMBOL _     TOP _  AUTO _  SHORT _   LEFT _
                    BOTTOM _                    RIGHT _

          TEXT _    MORE _     LEFT _
                               RIGHT _

          OFF _
```

DEFINE-WINDOW Statement — Window 2

The fields in this window are:

Field	Description
CONTROL	The PF-key lines, message line, and statistics line can be displayed in the window or on the full screen. To specify the location of the PF-key lines, message line, and statistics line, mark one of the following two options:
WINDOW	Display the lines inside the window.
SCREEN	Display the lines outside the window on the full screen.
Note:	If you do not specify a location, the lines are displayed according to the default for your Natural environment.
FRAMED	To place a frame around the window, specify one of the following options:
ON	Place a frame around the window.
OFF	Do not use a frame.
Note:	If you do not specify either ON or OFF, the window is displayed according to the Framed field default for your Natural environment.

Field	Description (continued)
CD=	If your terminal supports color, you can mark a color code: • BL (blue) • GR (green) • NE (neutral/white) • PI (pink) • RE (red) • TU (turquoise) • YE (yellow)
POSITION	To display the window position information within the frame of the window, mark one of the following options:
SYMBOL	Display the window position information in the form of symbols (More, <, -, +, >). You can also specify where and how the symbols are displayed by marking one or more of the following options:
TOP	Display the window position information at the top of the window.
BOTTOM	Display the window position information at the bottom of the window.
AUTO	Display the window position information in the form of words (Top, Bottom, and More) instead of symbols. Use this option if only the plus (+) or minus (-) symbols are displayed (the logical page is fully visible in the window horizontally).
SHORT	Suppress the display of the word “More” preceding the symbols.
LEFT	Display the window position information at the left of the window.
RIGHT	Display the window position information at the right of the window.
TEXT	Display the window position information in the form of text.
MORE	Display only the word “More”. To specify where the display is located within the window, mark one of the following options:
LEFT	Display the position information at the left of the window.
RIGHT	Display the position information at the right of the window.
OFF	Suppress the display of the window position information.

Define-Window Statement Model Example

The following specifications generate a DEFINE WINDOW statement:

```
                      DEFINE-WINDOW Statement 1 of 2

  DEFINE WINDOW #WINDOW1_____________________________

    SIZE  AUTO _
          QUARTER _
          Height ....... 13_____________________________
          Width ........ 40_____________________________

    BASE  CURSOR _
          TOP  LEFT _  BOTTOM  LEFT _
              RIGHT X         RIGHT _
          Line ......... _____________________
          Column ....... _____________________

    REVERSED _ ( CD=BL _ GR _ NE _ PI _ RE _ TU _ YE _ )

    TITLE ____________________________________________________________________
```

Define-Window Statement Model Example — Window 1

```
              DEFINE-WINDOW Statement 2 of 2

    CONTROL  WINDOW X
             SCREEN _

    FRAMED   ON X  CD=BL _ GR _ NE _ PI _ RE _ TU _ YE _
            OFF _
            POSITION
              SYMBOL X     TOP _  AUTO _  SHORT _   LEFT _
                         BOTTOM _                  RIGHT _

              TEXT _    MORE _     LEFT _
                                  RIGHT _

              OFF _
```

Define-Window Statement Model Example — Window 2

Example of code generated from the sample specifications

```
/*
/* Define Window #WINDOW1
DEFINE WINDOW #WINDOW1
              SIZE 13 * 40
              BASE TOP RIGHT
              CONTROL WINDOW
              FRAMED ON
              POSITION SYMBOL
```

Divide Statement Model

The Divide statement model (DV) generates a Natural DIVIDE statement, which divides two operands.

Generating the Default Code into the Editor

To generate a DIVIDE statement, enter one of the following line commands:

```
.g(DIVIDE,)
```

or

```
.g(DV,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 DIVIDE #divisor
0040   INTO #dividend
0050   GIVING #quotient
0060   REMAINDER #remainder
```

You can then replace the variables (indicated by the "#" prefix) to complete the DIVIDE statement.

Using the DIVIDE Statement Window

The DIVIDE Statement window is displayed when you invoke the Divide statement model from the Generation main menu. You can also invoke the DIVIDE Statement window from an editor by entering one of the following line commands:

```
.g(DIVIDE)
```

or

```
.g(DV)
```

The DIVIDE Statement window is displayed:

```
                          DIVIDE Statement

DIVIDE  ROUNDED _
   ____________________________________________________________

   INTO ____________________________________________________________

   GIVING ____________________________________________________________

   REMAINDER ____________________________________________________________
```

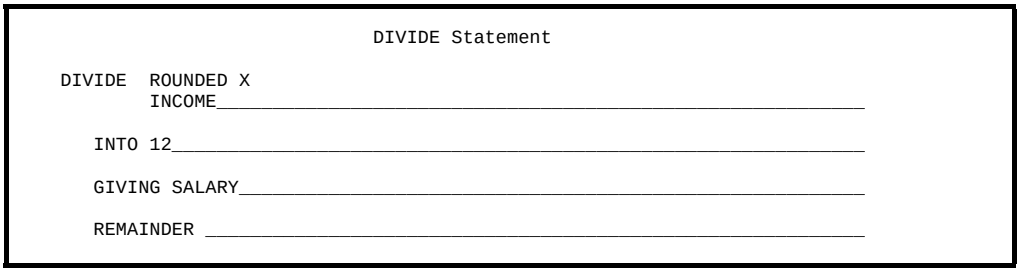
DIVIDE Statement Window

The fields in this window are:

Field	Description
DIVIDE ROUNDED	To round the result of the divide operation, mark this field. This field is optional. Specify the divisor on the line below. This is a required field.
INTO	Numeric constant or variable that is divided into. This is a required field.
GIVING	If you specify a variable name in this field, the value of the INTO field is not modified and the result of the division operation is stored in this variable. Otherwise, the result is stored in the INTO field. If the value in the INTO field is a numeric constant, this is a required field.
REMAINDER	Name of the field in which the remainder from the divide operation is placed. This is an optional field.
Note:	The ROUNDED and REMAINDER options cannot be used together.

DIVIDE Statement Model Example

The following specifications generate a DIVIDE statement:



Divide Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 DIVIDE ROUNDED 12
0040   INTO INCOME
0050   GIVING SALARY
```

Escape Statement Model

The Escape statement model (ES) generates a Natural ESCAPE statement, which interrupts the linear flow of the execution of a processing loop or a routine.

Generating the Default Code into the Editor

To generate an ESCAPE statement, enter one of the following line commands:

```
.g(ESCAPE,)
```

or

```
.g(ES,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 ESCAPE BOTTOM(label.) IMMEDIATE
```

Revise the code to complete the ESCAPE statement.

Using the ESCAPE Statement Window

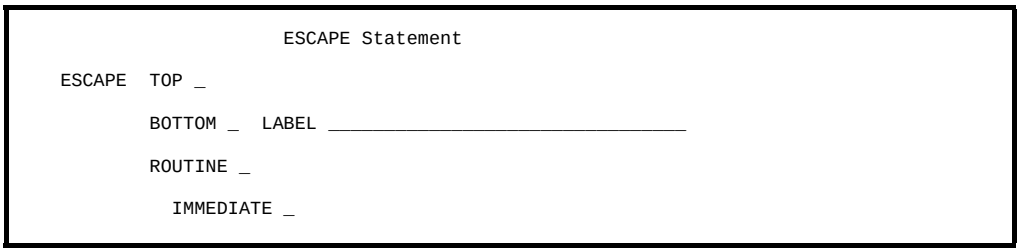
The ESCAPE Statement window is displayed when you invoke the Escape statement model from the Generation main menu. You can also invoke the ESCAPE Statement window from an editor by entering one of the following line commands:

```
.g(ESCAPE)
```

or

```
.g(ES)
```

The ESCAPE Statement window is displayed:



ESCAPE Statement Window

The fields in this window are:

Field	Description
ESCAPE	Select the ESCAPE destination by marking one of the following fields:
TOP	To continue processing at the top of the processing loop, mark this field.
BOTTOM	To continue processing with the first statement following the processing loop, mark this field.
LABEL	If you marked BOTTOM, you can specify an identifying name or label for the ESCAPE statement in this field. Bottom processing continues with the next statement following the processing loop identified by the LABEL number. This is an optional field.
ROUTINE	To have the current Natural routine (which may have been invoked via a PERFORM, CALLNAT, FETCH RETURN, or a main program) relinquish control, mark this field.
IMMEDIATE	If you do not want loop-end (final BREAK and END-OF-DATA) processing performed, mark this field and mark either the BOTTOM or ROUTINE field.

Escape Statement Model Example

The following specifications generate an ESCAPE statement:

```
                    ESCAPE Statement

ESCAPE  TOP _

        BOTTOM X  LABEL FIND-EMPLOYEES__________________

        ROUTINE _

          IMMEDIATE X
```

Escape Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 ESCAPE BOTTOM(FIND-EMPLOYEES.) IMMEDIATE
```

Examine Statement Model

The Examine statement model (EX) generates a Natural EXAMINE statement. This statement scans the contents of an alphanumeric field or a range of fields within an array for a character string. It replaces and/or counts the number of occurrences of the character string, as well as giving information about the result of the examine operation.

Generating the Default Code into the Editor

To generate an EXAMINE statement, enter one of the following line commands:

```
.g(EXAMINE,)
```

or

```
.g(EX,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 EXAMINE FULL examine-variable
0040   FOR text
0050   GIVING POSITION position
0060          LENGTH length
0070          INDEX index
```

Revise the code to complete the EXAMINE statement.

Using the EXAMINE Statement Window

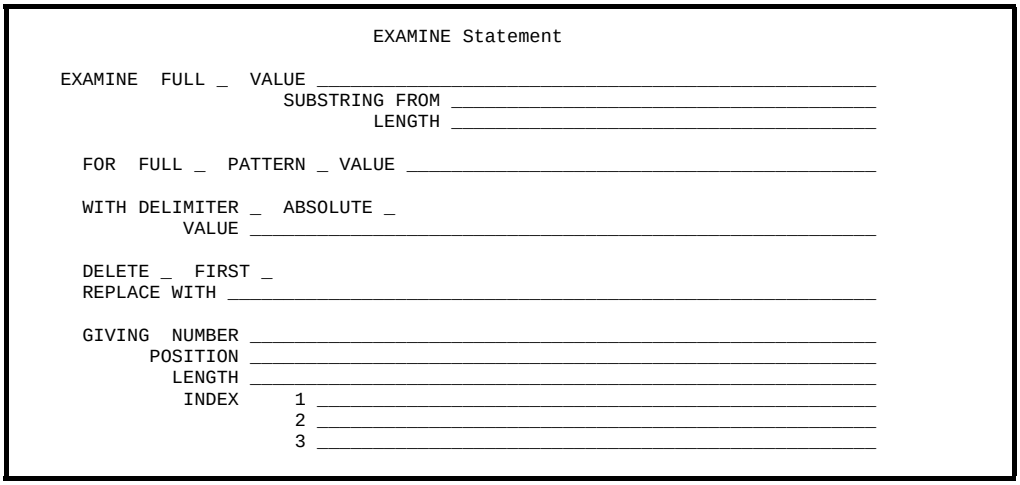
The EXAMINE Statement window is displayed when you invoke the Examine statement model from the Generation main menu. You can also invoke the EXAMINE Statement window from an editor by entering one of the following line commands:

```
.g(EXAMINE)
```

or

```
.g(EX)
```

The EXAMINE Statement window is displayed:



EXAMINE Statement Window

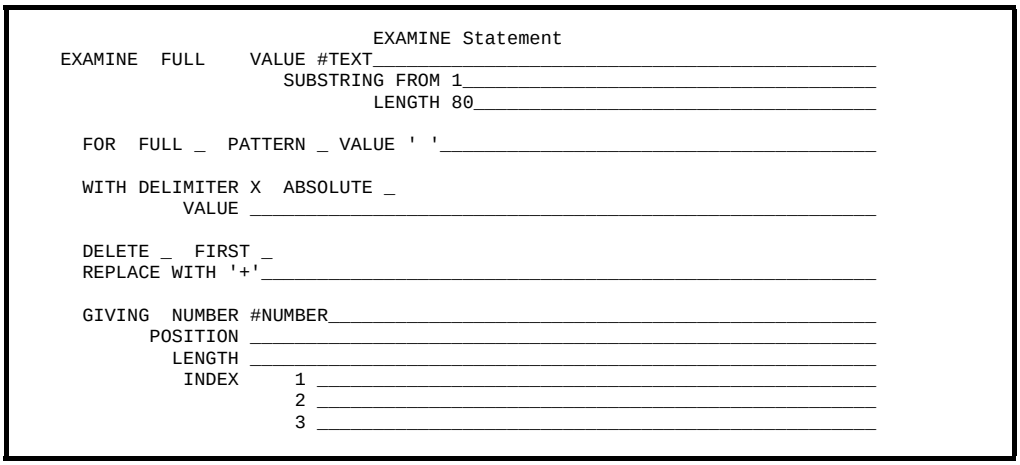
The fields in this window are:

Field	Description
EXAMINE FULL	To process the entire value, including trailing blanks, mark this field. This is an optional field.
VALUE	Name of the field to examine. This is a required field.
SUBSTRING FROM	Starting position of a substring (if you are using a substring).
LENGTH	Length of the substring. This is an optional field unless the starting position is specified.
FOR FULL	To process the entire EXAMINE statement, including trailing blanks, mark this field. This is an optional field.
PATTERN	If the value specified in the VALUE field is a pattern (a variable containing symbols for positions not examined), mark this field. For example, you can examine the field for any value that contains “IM” and “E”, regardless of how many characters are between the “IM” and “E”, by specifying “IM*E”. Valid results for this example are: Image, Imitate, and Imagine.
VALUE	Name of an alphanumeric variable or a character string enclosed within single quotes. This is a required field.

Field	Description (continued)
WITH DELIMITER	To scan for a value based on what delimiter characters surround it, mark this field and specify the following:
ABSOLUTE	To scan for the value, regardless of what characters surround it, mark this field.
VALUE	If you marked the ABSOLUTE field, specify the character(s) in this field. The character(s) can be a constant delimiter string enclosed within single quotes or an alphanumeric variable containing the delimiter characters.
DELETE	To delete all occurrences of the value, mark this field.
FIRST	To delete or replace only the first occurrence of the value, mark this field.
REPLACE WITH	To replace the value with another value, specify the new value in this field. If you marked the FIRST field, only the first occurrence of the value is replaced. Otherwise, all occurrences are replaced.
GIVING NUMBER	To return the number of occurrences of the value, specify a numeric variable in this field. (If you marked the FIRST field, the number does not exceed 1.)
POSITION	To return the byte position (of the first occurrence of the Examine for value within the field), specify a numeric variable in this field.
LENGTH	To return the final length of the field (after all delete/replace operations), specify a numeric variable in this field.
INDEX	If the examined field is an array, you can return the occurrence numbers (indexes) of the field by specifying the numeric variables as follows: • For a one-dimensional array, specify the name of the numeric variable on the first line in this field. • For a two-dimensional array, specify the names on the first two lines (one name per line). • For a three-dimensional array, specify the names on three lines (one name per line).

Examine Statement Model Example

The following specifications generate an EXAMINE statement:



Examine Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 EXAMINE SUBSTRING (#TEXT,1,80)
0040   FOR ' '
0050   WITH DELIMITER
0060   REPLACE '+'
0070   GIVING NUMBER #NUMBER
```

Examine-Translate Statement Model

The Examine-Translate statement model (ER) generates a Natural EXAMINE TRANSLATE statement. This statement translates the contents of a field into upper case, lower case, or other specified characters.

Generating the Default Code into the Editor

To generate an EXAMINE TRANSLATE statement, enter one of the following line commands:

```
.g(EXAMINE-TRANSLATE,)
```

or

```
.g(ER,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 EXAMINE translate field AND
0040          TRANSLATE INTO UPPER CASE
```

Revise the code to complete the EXAMINE TRANSLATE statement.

Using the EXAMINE TRANSLATE Function Window

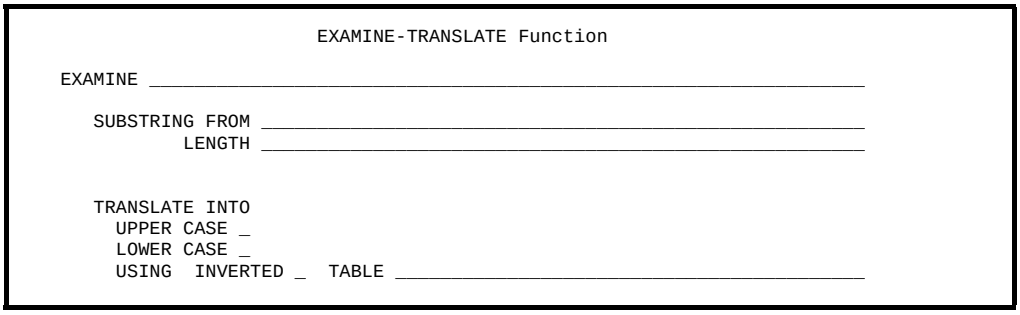
The EXAMINE TRANSLATE Function window is displayed when you invoke the Examine-Translate statement model from the Generation main menu. You can also invoke the EXAMINE-TRANSLATE Function window from an editor by entering one of the following line commands:

```
.g(EXAMINE-TRANSLATE)
```

or

```
.g(ER)
```

The EXAMINE-TRANSLATE Function window is displayed:



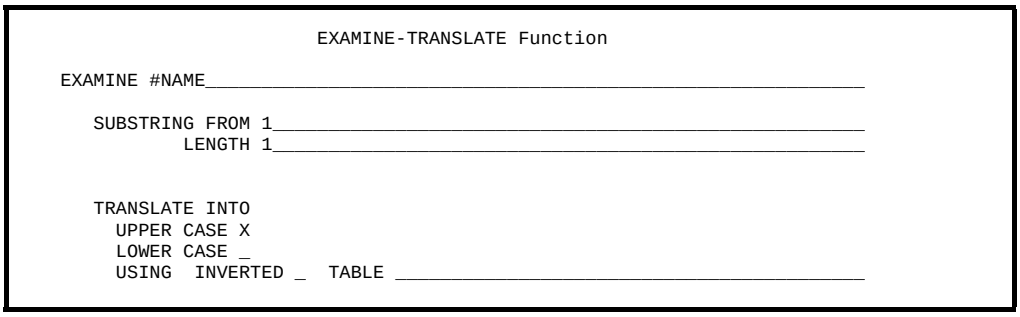
EXAMINE-TRANSLATE Function Window

The fields in this window are:

Field	Description
EXAMINE	Name of the field you want to translate. This is a required field.
SUBSTRING	The EXAMINE TRANSLATE statement supports the Natural Substring option. Normally, the entire contents of a field are translated. Using the Substring option, you can translate only a portion of a field.
FROM	To examine and translate only a portion of the field, specify the starting position of that portion. This is an optional field.
LENGTH	To examine and translate only a portion of the field, specify the length of the portion. This is an optional field.
TRANSLATE INTO	To translate the contents of the field, mark one of the following options:
UPPER CASE	Translate the contents of the field into upper case.
LOWER CASE	Translate the contents of the field into lower case.
USING INVERTED	Revert to the previous translation. (The previous translation must have been done using the same translation table.)
TABLE	Translate the contents of the field into other characters. Specify the name of the character translation table (must be A2 or B2 format). Translation is based on this table.

Examine-Translate Statement Model Example

The following specifications generate an EXAMINE TRANSLATE statement:



Examine-Translate Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 EXAMINE SUBSTRING (#NAME,1,1) AND
0040          TRANSLATE INTO UPPER CASE
```

Fetch Statement Model

The Fetch statement model (FE) generates a Natural FETCH statement, which loads and executes a Natural object program that has been written as a main program. The program must have been previously stored in the Natural system file with a CATALOG or STOW command.

Generating the Default Code into the Editor

To generate a FETCH statement, enter one of the following line commands:

```
.g(FETCH,)
```

or

```
.g(FE,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 FETCH program-name
```

Revise the code to complete the FETCH statement.

Using the FETCH Statement Window

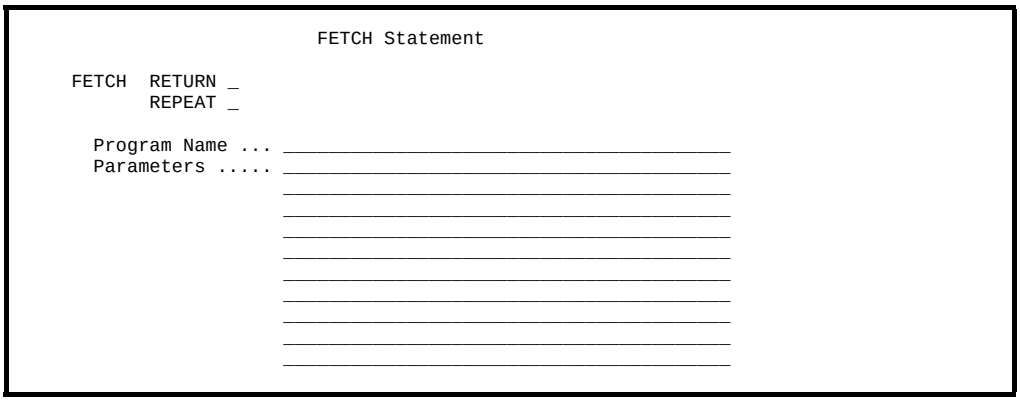
The FETCH Statement window is displayed when you invoke the Fetch statement model from the Generation main menu. You can also invoke the FETCH Statement window from an editor by entering one of the following line commands:

```
.g(FETCH)
```

or

```
.g(FE)
```

The FETCH Statement window is displayed:



FETCH Statement Window

The fields in this window are:

Field	Description
FETCH RETURN	To suspend the execution of the calling program (not terminate) and activate the FETCHed program as a higher level subprogram, mark this field. Control is returned to the calling program when an END or ESCAPE ROUTINE statement is encountered in the FETCHed program. Control is returned to the statement following the FETCH RETURN statement. This is an optional field.
REPEAT	To suppress the user-input prompt for each statement issued during the execution of the FETCHed program, mark this field. This is an optional field.

Field	Description (continued)
Program Name	Name of the program to FETCH. You can specify the name as an alphanumeric constant (enclosed within single quotes) or as the contents of an alphanumeric user-defined variable. This is a required field.
Parameters	Parameters passed to the FETCHed program. You can specify a maximum of 10 parameters. Enclose constant character strings within single quotes.

Fetch Statement Model Example

The following specifications generate a FETCH statement:

```
                      FETCH Statement

   FETCH  RETURN X
          REPEAT _

     Program Name ... #MAINPROG_____________________________________
     Parameters ..... #PARM1________________________________________
                      #PARM2________________________________________
                      _____________________________________________
                      _____________________________________________
                      _____________________________________________
                      _____________________________________________
                      _____________________________________________
                      _____________________________________________
                      _____________________________________________
                      _____________________________________________
```

Fetch Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 FETCH RETURN #MAINPROG
0040               #PARM1
0050               #PARM2
```

Find Statement Model

The Find statement model (FI) generates a Natural FIND statement. This statement selects a set of records from the database based on search criteria consisting of fields defined as descriptors (keys).

Generating the Default Code into the Editor

To generate a FIND statement, enter one of the following line commands:

```
.g(FIND,)
```

or

```
.g(FI,)
```

The following default code is generated:

```
0010 /*
0020 /* Find view
0030 FIND1.
0040      WITH descriptor = value
0050   /*
0060   /*
0070   /*
0080   /*
0090 END-FIND /* FIND1.
```

Revise the code to complete the FIND statement.

Using the FIND Statement Window

The FIND Statement window is displayed when you invoke the Find statement model from the Generation main menu. You can also invoke the FIND Statement window from an editor by entering one of the following line commands:

```
.g(FIND)
```

or

```
.g(FI)
```

The FIND Statement window is displayed:

```
                              FIND Statement

LABEL   ____________________________________
FIND ALL _  NUMBER _  LIMIT ____________________________________________________
  RECORDS IN FILE ____________________________________ *

  WITH  LIMIT ____________________________________________________________________
        ____________________________________________________________________
        ____________________________________________________________________

  WHERE ____________________________________________________________________
        ____________________________________________________________________

  IF NO RECORDS FOUND
       ____________________________________________________________________
       ____________________________________________________________________
  END-NOREC

  SKIP  __
END-FIND
```

FIND Statement Window

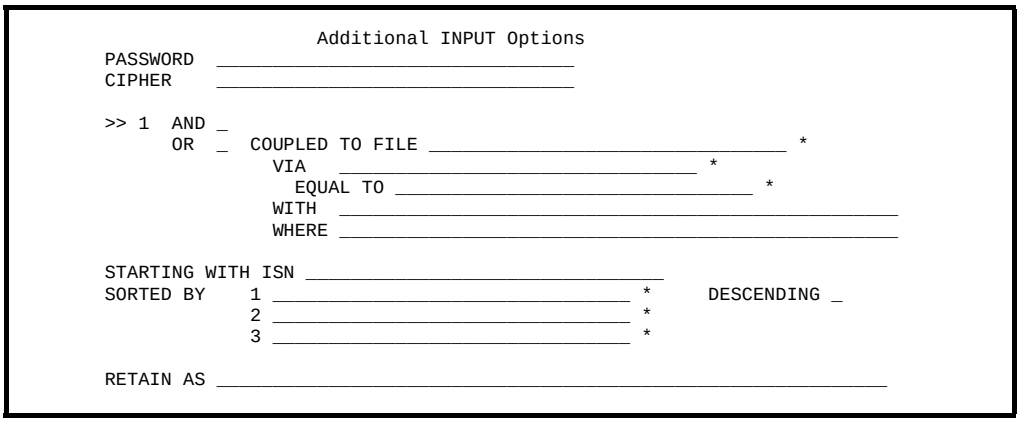
The fields in this window are:

Field	Description
LABEL	Label for the FIND statement. This is an optional field.
FIND ALL	To process all records in a selected set, mark this field.
NUMBER	To only determine the number of selected records, mark this field.
LIMIT	To limit the records selected, enter the number of records you want to select in this field.
Note:	You can only specify one of the FIND ALL, NUMBER, and LIMIT fields.
RECORDS IN FILE	Name (as defined in a global data area or local data area) of the view to find. This is a required field.
WITH LIMIT	The WITH LIMIT criteria the FIND statement uses to search for the records. The WITH LIMIT clause is evaluated by the DBMS. This is a required field.
WHERE	The WHERE criteria the FIND statement uses to search for the records. This criteria is evaluated after a record selected with the basic search criteria has been read, but before any processing is performed. The WHERE clause is evaluated by Natural. This is an optional field.

Field	Description (continued)
IF NO RECORDS FOUND	Specify any statement(s) to be executed if no records matching the selection criteria are found.
SKIP	Number of blank lines to reserve for additional statements within the FIND loop.

Setting Additional Input Options

To set more options for the FIND statement, press PF5 (optns) in the FIND Statement window. The Additional INPUT Options window is displayed:



Additional INPUT Options Window

Use this window to specify a search using the Natural coupling facility for Adabas files. This facility permits database descriptors from different files to be specified in the search criteria of a single FIND statement.

The same Adabas file may not be used in two different FIND COUPLED clauses within the same FIND statement. Database fields in a file specified within the COUPLED clause are not available for subsequent reference in the program unless another FIND or READ statement is issued separately against the coupled file.

The fields in this window are:

Field	Description
PASSWORD	To provide a password when retrieving data from an Adabas file that is password-protected, enter the password in this field.
CIPHER	To provide a cipher key when retrieving data from Adabas files that are enciphered, enter the key in this field.

Field	Description (continued)
>> 1	You can couple up to four files with the main file. To specify the next file, press PF8 (frwrd). Press PF7 (bkwrd) to display the previous file.
AND	To use the AND logical connector to couple the files, mark this field. You can use either AND or OR, but not both.
OR	To use the OR logical connector to couple the files, mark this field. You can use either OR or AND, but not both.
COUPLED TO FILE	Name of the file to be coupled with the main file.
VIA	To logically couple multiple Adabas files in a search query, specify a descriptor from the main file in this field and a descriptor from the coupled file in the EQUAL TO field. The main file and the coupled file need not be physically coupled in Adabas.
EQUAL TO	To logically couple multiple Adabas files in a search query, specify a descriptor from the main file in the VIA field and a descriptor from the coupled file in the EQUAL TO field. The main file and the coupled file need not be physically coupled in Adabas.
WITH	Basic search criteria for the records of the coupling file.
WHERE	Additional selection criterion that is evaluated after a record has been read but before any processing is performed.
STARTING WITH ISN	To retrieve records starting with an ISN (Adabas Internal sequence number), specify the number in this field.
SORTED BY 1,2,3	To sort the selected records based on the sequence of one to three descriptors, specify the descriptor names on the lines provided. The sorting sequence may be in either ascending or descending order. The descriptors used to control the sort sequence may be different from those used to select the records. You can specify a multiple-valued field (MU) with no index, but not a descriptor contained in a periodic group (PE).

Field	Description (continued)
DESCENDING	To sort the records in descending order, mark this field. Otherwise the records are sorted in ascending order.
RETAIN AS	To retain the result of an extensive search in large files for further processing, specify the set name in this field. The set name identifies the record set. It may be specified as an alphanumeric constant or as the content of an alphanumeric user-defined variable. Duplicate set names are not checked; consequently if a duplicate set name is specified, the new replaces the old set. The RETAIN AS and SORTED BY fields may not be used together.

Find Statement Model Example

The following specifications generate a FIND statement:

```
                              FIND Statement

LABEL   FIND-EMP_____________________________
FIND ALL X  NUMBER _  LIMIT _____________________________________________
   RECORDS IN FILE CST-EMPLOYEES_____________________ *

   WITH  LIMIT ____________________________________________________________
         CITY = 'FRANKFURT'_______________________________________________
         ________________________________________________________________

   WHERE __________________________________________________________________
         ________________________________________________________________

   IF NO RECORDS FOUND
         WRITE 'No employees found'_____________________________________
         ________________________________________________________________
   END-NOREC

   SKIP  __
 END-FIND
```

Find Statement Model Example

```
                        Additional INPUT Options

    PASSWORD  ____________________________________
    CIPHER    ____________________________________

    >> 1  AND X
          OR  _  COUPLED TO FILE VEHICLES_____________________________ *
                   VIA  ____________________________________ *
                     EQUAL TO ____________________________________ *
                   WITH  MAKE = 'VW'_____________________________________________
    >>             WHERE ____________________________________________________


    SORTED BY    1 ____________________________________ *     DESCENDING _
                 2 ____________________________________ *
                 3 ____________________________________ *

    RETAIN AS ____________________________________________________________________
```

Find Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /* Find CST-EMPLOYEES
0030 FIND-EMP.
0040 FIND CST-EMPLOYEES
0050      WITH CITY = 'FRANKFURT
0060      AND COUPLED VEHICLES
0070                  WITH MAKE = 'VW'
0080   IF NO RECORDS FOUND
0090     WRITE = 'NO EMPLOYEES FOUND'
0100
0110   END-NOREC
0120 END-FIND /* FIND-EMP.
```

For Statement Model

The For statement model (FO) generates a Natural FOR statement, which initiates a processing loop and controls the number of times the loop is processed.

Generating the Default Code into the Editor

To generate a FOR statement, enter one of the following line commands:

```
.g(FOR,)
```

or

```
.g(FO,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 FOR1.
0040 FOR #I = 1 TO #MAX-I
0050   statement1
0060   statement2
0070 END-FOR /* FOR1.
```

Revise the code to complete the FOR statement.

Using the FOR Statement Window

The FOR Statement window is displayed when you invoke the For statement model from the Generation main menu. You can also invoke the FOR Statement window from an editor by entering one of the following line commands:

```
.g(FOR)
```

or

```
.g(FO)
```

The FOR Statement window is displayed:

```
                              FOR Statement

LABEL _____________________________________
FOR #I_____________________________  EQ 1_____________________________________
    TO #MAX-I_____________________________ STEP _____________________________

   statement ..._____________________________________________________________
   ___________________________________________________________________________
   ___________________________________________________________________________
   ___________________________________________________________________________
   ___________________________________________________________________________

END-FOR
```

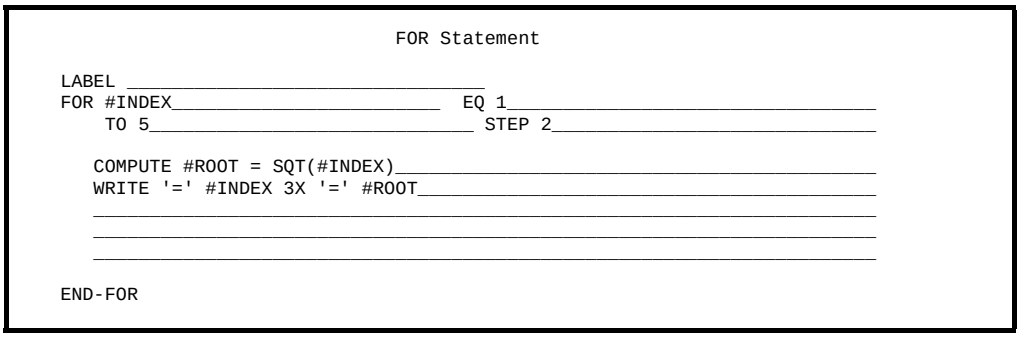
FOR Statement Window

The fields in this window are:

Field	Description
LABEL	Label for the FOR statement. This is an optional field.
FOR, EQ, TO	To control the number of times the loop is executed, specify a variable in the FOR field and a range of numeric values in the EQ and TO fields. These are required fields.
STEP	Value that is the increment of the value in the TO field (may be positive or negative). The default is 1.
statement	Statement(s) performed within the FOR statement loop.

For Statement Model Example

The following specifications generate a FOR statement:



For Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 FOR #INDEX = 1 TO 5 STEP 2
0040   COMPUTE #ROOT = SQT(#INDEX)
0050   WRITE '=' #INDEX 3X '=' #ROOT
0060 END-FOR
```

Format Statement Model

The Format statement model (FM) generates a Natural FORMAT statement, which specifies input and output parameter settings. At execution time, FORMAT statement settings override default settings set by a GLOBALS command, a SET GLOBALS statement, or by your Natural administrator. The settings remain in effect until the end of a program or until another FORMAT statement is encountered. (The settings may be overridden by parameters specified in a DISPLAY, INPUT, PRINT, WRITE, WRITE TITLE, or WRITE TRAILER statement.)

Generating the Default Code into the Editor

To generate a FORMAT statement, enter one of the following line commands:

```
.g(FORMAT,)
```

or

```
.g(FM,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 FORMAT AD=MILW HC=L IS=ON LS=133 PS=60 ZP=OFF
```

Revise the code to complete the FORMAT statement.

Using the FORMAT Statement Window

The FORMAT Statement window is displayed when you invoke the Format statement model from the Generation main menu. You can also invoke the FORMAT Statement window from an editor by entering one of the following line commands:

```
.g(FORMAT)
```

or

```
.g(FM)
```

The FORMAT Statement window is displayed:

```
                          FORMAT Statement

FORMAT  Number ... __
        Name ..... ________________________________

  Session Parameters ........... AD=M _ A _ O _ P _
                                 AD=B _ C _ I _ N _ U _ V _
                                 AD=L _ R _ Z _
                                 AD=T _ W _
                                 Filler ... _
                                 CD=BL _ GR _ NE _ PI _ RE _ TU _ YE _
                                 HC=C _ L _ R _
                                 LS= ___
                                 PS= ___
                                 UC= _
                                 IP= ON _ OFF _
                                 IS= ON _ OFF _
                                 KD= ON _ OFF _
                                 SG= ON _ OFF _
                                 ZP= ON _ OFF _
```

FORMAT Statement Window

The fields in this window are:

Field	Description
FORMAT Number	Number of the report to which the generated FORMAT statement applies. The default is Report 0.
Name	If your report has a name or a number, specify the name or number in this field. The name or number must have been assigned using a DEFINE PRINTER statement.
Session Parameters	
AD=M	To set the defaults for the input/output fields, mark one of the following options: • M (modifiable input fields) • A (input-only fields) • D (display-only fields) • P (protected, non-modifiable input fields; used in conjunction with a control variable)

Field	Description (continued)
AD=B	To set the default field representation for the report, mark one of the following options: • B (blinking) • C (cursive or italic) • I (intensified) • U (underlined) • V (reverse video)
AD=L	To set the default alignment option for the specified report, mark one of the following options: • L (left justified) • R (right justified) • Z (numeric values are displayed with leading zeros and right justified)
AD=T	To set the translation option, mark one of the following fields: • T (translate lower case characters to upper case) • W (translate upper case characters to lower case)
Filler	To use a filler character to replace blank spaces, specify the character (for input/output or output fields only).
CD=BL	To set the default color (for color terminals only), mark one of the following color definition options: • BL (blue) • GR (green) • NE (neutral/white) • PI (pink) • RE (red) • TU (turquoise) • YE (yellow)
HC=C	To set the column heading default for the specified report, mark one of the following centering options: • C (centered) • L (left justified) • R (right justified)
LS=	Number of characters per line (default is physical line size).
PS=	Number of lines per page (default is physical page size).
UC=	Character used for underlining.

Field	Description (continued)
IP=	To have input prompting text on, mark “ON”. The field name is generated as prompt text preceding the field in an INPUT statement (even if no text is specified preceding the INPUT/ OUTPUT statement). To have input prompting text off, mark “OFF”.
IS=	To suppress the display of a value that is identical to the previous value in the field, mark “ON”.
KD=	To display the names assigned to the PF-keys on the bottom two lines of the screen (with any output created by the INPUT, WRITE, DISPLAY, or PRINT statement), mark “ON” and reduce the logical page size (PS) by two.
SG=	To allocate a sign position for a numeric field, mark “ON”. To have numeric fields with negative values printed without a minus (-) sign, mark “OFF”.
ZP=	To suppress field values of all zeros, mark “ON”. To print one zero for each field value containing all zeros, mark “OFF”.

Format Statement Model Example

The following specifications generate a FORMAT statement:

```
                          FORMAT Statement

FORMAT  Number ... __
        Name ..... SRC_____________________________

  Session Parameters ........... AD=M _ A _ O _ P _
                                 AD=B _ C _ I _ N _ U _ V _
                                 AD=L _ R _ Z _
                                 AD=T _ W _
                                 Filler ... _
                                 CD=BL _ GR _ NE _ PI _ RE _ TU _ YE _
                                 HC=C _ L _ R _
                                 LS= ___
                                 PS= ___
                                 UC= _
                                 IP= ON _ OFF _
                                 IS= ON _ OFF _
                                 KD= ON _ OFF _
                                 SG= ON _ OFF _
                                 ZP= ON _ OFF _
```

Format Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 FORMAT(SRC) AD=MILW HC=L IS=ON LS=133 PS=60 ZP=OFF
```

Get Statement Model

The Get statement model (GE) generates a Natural GET statement, which reads a record with a given Adabas ISN (Internal Sequence Number) without initiating a processing loop.

Generating the Default Code into the Editor

To generate a GET statement, enter one of the following line commands:

```
.g(GET,)
```

or

```
.g(GE,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 GET IN FILE view-name
0040     RECORD adabas-isn
```

Revise the code to complete the GET statement.

Using the GET Statement Window

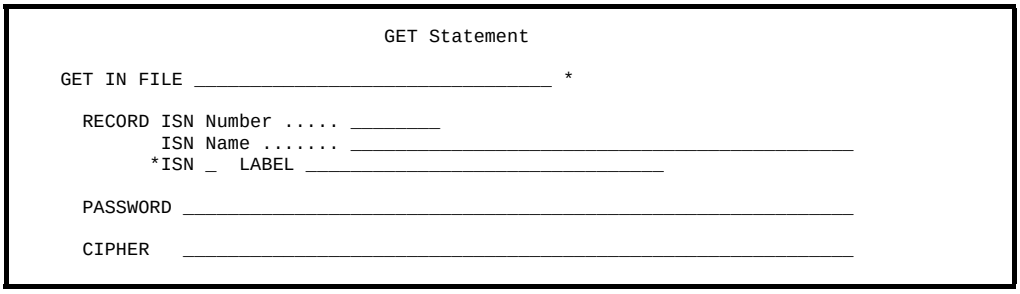
The GET Statement window is displayed when you invoke the Get statement model from the Generation main menu. You can also invoke the GET Statement window from an editor by entering one of the following line commands:

```
.g(GET)
```

or

```
.g(GE)
```

The GET Statement window is displayed:



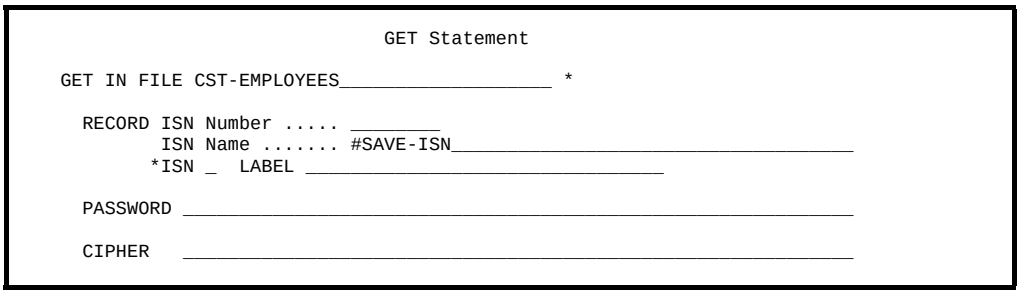
GET Statement Window

The fields in this window are:

Field	Description
GET IN FILE	Name of the Predict file to retrieve as defined in a global data area (GDA) or local data area (LDA).
RECORD ISN Number	If the ISN is a numeric constant, specify the number in this field.
ISN Name	If the ISN is a user-defined variable, specify the name of the variable in this field.
*ISN	If the ISN is a Natural system variable ISN, mark this field.
LABEL	If you mark the *ISN field, specify the label or reference name of the ISN in this field.
PASSWORD	Password for a password-protected Adabas file.
CIPHER	Cipher key for an enciphered Adabas file.

Get Statement Model Example

The following specifications generate a GET statement:



Get Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 GET IN FILE CST-EMPLOYEES
0040     RECORD #SAVE-ISN
```

Histogram Statement Model

The Histogram statement model (HI) generates a Natural HISTOGRAM statement, which reads the values of a database field that is defined as a descriptor, subdescriptor, or superdescriptor.

The values are read directly from the Adabas inverted lists (ISNs) and returned in ascending sequence. The HISTOGRAM statement initiates a processing loop, but does not access any database fields other than the one specified within the statement.

Generating the Default Code into the Editor

To generate a HISTOGRAM statement, enter one of the following line commands:

```
.g(HISTOGRAM,)
```

or

```
.g(HI,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 HIST.
0040 HISTOGRAM view
0050      FOR descriptor
0060      STARTING FROM start-value THRU end-value
0070   /*
0080   /*
0090   /*
0100   /*
0110 END-HISTOGRAM /* HIST.
```

Revise the code to complete the HISTOGRAM statement.

Using the HISTOGRAM Statement Window

The HISTOGRAM Statement window is displayed when you invoke the Histogram statement model from the Generation main menu. You can also invoke the HISTOGRAM Statement window from an editor by entering one of the following line commands:

```
.g(HISTOGRAM)
```

or

```
.g(HI)
```

The HISTOGRAM Statement window is displayed:

```
                              HISTOGRAM Statement
LABEL ____________________________________
HISTOGRAM  LIMIT ____________________________________________
     VALUE IN FILE ____________________________________ *
     DESCENDING    _
     VALUE FOR FIELD ____________________________________ *
       STARTING FROM ____________________________________________
                THRU ____________________________________________
     WHERE
       ____________________________________________________________________________
       ____________________________________________________________________________

     PASSWORD ____________________________________________

     SKIP __

END-HISTOGRAM
```

HISTOGRAM Statement Window

The fields in this window are:

Field	Description
LABEL	Name or label for the processing loop. This is an optional field.
HISTOGRAM LIMIT	Maximum number of records processed. This is an optional field.
VALUE IN FILE	Name of the view to retrieve, as defined in a global data area (GDA) or local data area (LDA). The view may only contain the descriptor field you want to histogram. This is a required field.
DESCENDING	To retrieve records in descending order, mark this field. Otherwise the records are sorted in ascending order.
VALUE FOR FIELD	Name of the descriptor, subdescriptor, or superdescriptor. This is a required field.
STARTING FROM	Starting value for processing. This is an optional field.
THRU	Ending value for processing. This is an optional field.
WHERE	Additional search criteria. These criteria are evaluated after a value is read and before any processing is performed. The specified descriptor, subdescriptor, or superdescriptor must be the same one referred to in the HISTOGRAM statement. This is an optional field.

Field	Description (continued)
PASSWORD	Password for a password-protected Adabas file.
SKIP	Number of blank lines to reserve for additional statements within the HISTOGRAM loop.

Histogram Statement Model Example

The following specifications generate a HISTOGRAM statement:

```
                        HISTOGRAM Statement

LABEL HIST-EMP_____________________________
HISTOGRAM  LIMIT ___________________________________________
     VALUE IN FILE CST-EMPLOYEES_____________________ *
     DESCENDING    _
     VALUE FOR FIELD CITY_____________________________ *
       STARTING FROM 'M'_____________________________________
                THRU _____________________________________
     WHERE
       ___________________________________________________________________________
       ___________________________________________________________________________

     PASSWORD ___________________________________________

     SKIP __

END-HISTOGRAM
```

Histogram Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 HIST-EMP.
0040 HISTOGRAM CST-EMPLOYEES
0050      FOR CITY
0060      STARTING FROM 'M'
0070 END-HISTOGRAM /* HIST-EMP.
```

If Statement Model

The If statement model (IF) generates a Natural IF statement, which controls the execution of a statement (or group of statements) based on a logical condition.

Generating the Default Code into the Editor

To generate an IF statement, enter the following line command:

```
.g(IF,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 IF some-condition THEN
0040   /*
0050   /*
0060   do this
0070   do this
0080 ELSE
0090   /*
0100   /*
0110   do that
0120   do that
0130 END-IF
```

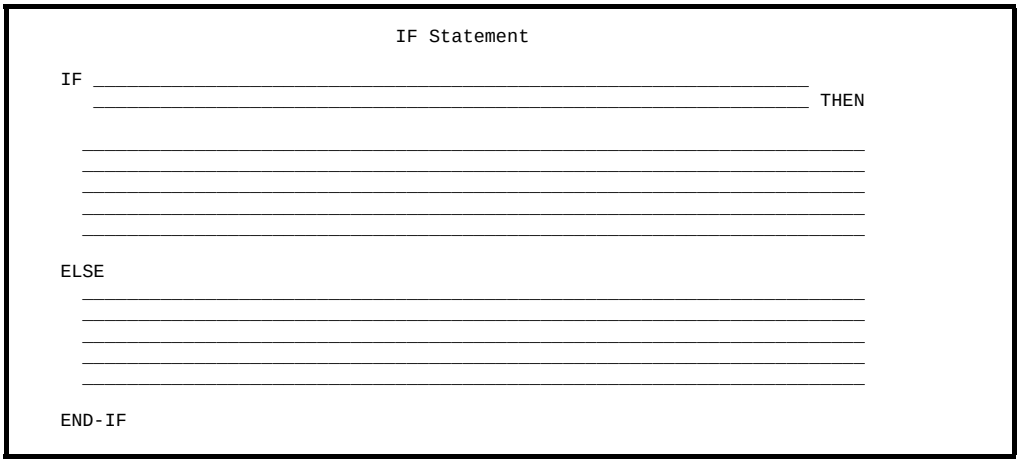
Revise the code to complete the IF statement.

Using the IF Statement Window

The IF Statement window is displayed when you invoke the If statement model from the Generation main menu. You can also invoke the IF Statement window from an editor by entering the following line command:

```
.g(IF)
```

The IF Statement window is displayed:



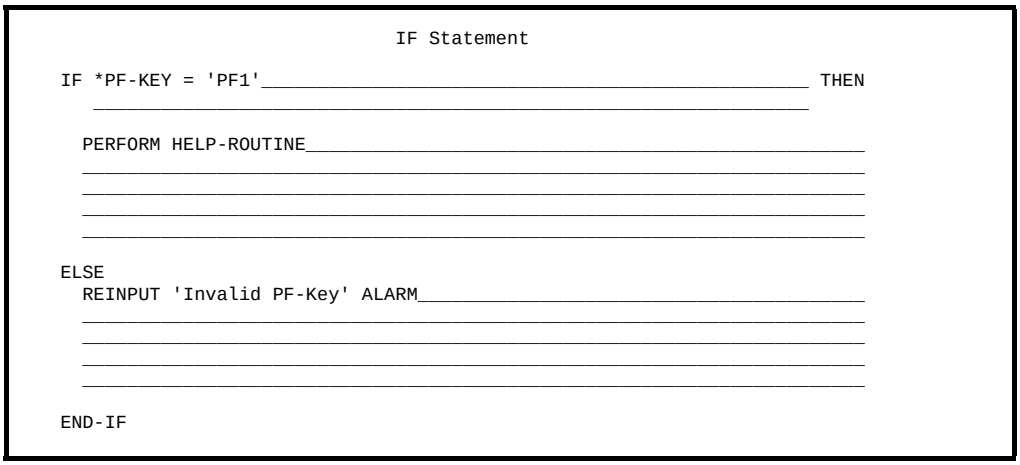
IF Statement Window

The fields in this window are:

Field	Description
IF	Logical condition that determines whether the statement(s) specified in the IF statement is executed.
THEN	Action performed if the logical condition is true.
ELSE	Action performed if the logical condition is not true. This is an optional field.

If Statement Model Example

The following specifications generate an IF statement:



If Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 IF *PF-KEY = 'PF1' THEN
0040   PERFORM HELP-ROUTINE
0050 ELSE
0060   REINPUT 'Invalid PF-Key' ALARM
0070 END-IF
```

If-Is Statement Model

Like the If statement model, the If-Is statement model (II) controls the execution of a statement (or group of statements) based on a logical condition. The If-Is statement model contains the Natural IS option, which checks a field for a specific format.

Generating the Default Code into the Editor

To generate an IF-IS function, enter one of the following line commands:

```
.g(IF-IS,)
```

or

```
.g(II,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 IF #field IS (N10) THEN
0040   do this
0050 ELSE
0060   do that
0070 END-IF
0080
```

Revise the code to complete the IF-IS function.

Using the IF-IS Statement Window

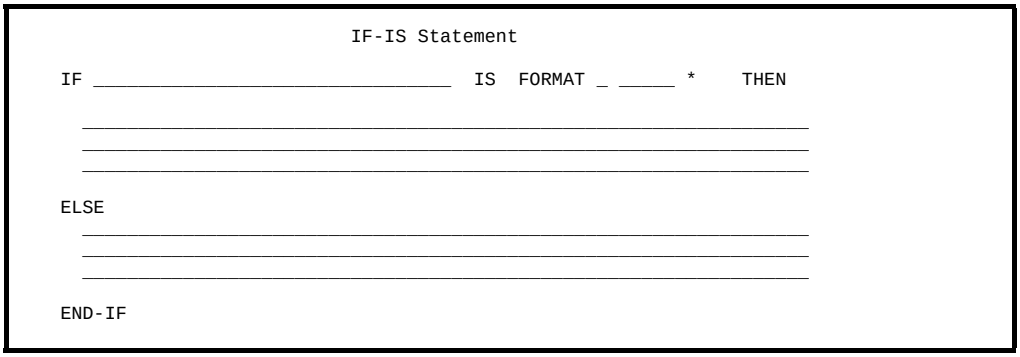
The IF-IS Statement window is displayed when you invoke the If-Is statement model from the Generation main menu. You can also invoke the IF-IS Statement window from an editor by entering one of the following line commands:

```
.g(IF-IS)
```

or

```
.g(II)
```

The IF-IS Statement window is displayed:



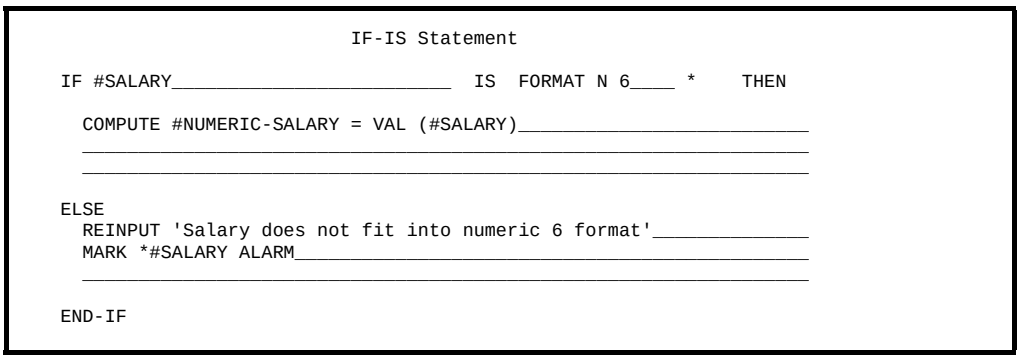
IF-IS Statement Window

The fields in this window are:

Field	Description
IF	Logical condition that determines whether the statement(s) specified in the IF statement is executed.
IS FORMAT	Length and format of the field specified in the IF field.
THEN	Action performed if the logical condition is true.
ELSE	Action performed if the logical condition is not true. This is an optional field.

If-Is Statement Model Example

The following specifications generate an IF statement with the IS option:



If-Is Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 IF #SALARY IS (N60) THEN
0040   COMPUTE #NUMERIC-SALARY = VAL (#SALARY)
0050 ELSE
0060   REINPUT 'Salary does not fit into numeric 6 format'
0070   MARK *#SALARY ALARM
0080 END-IF
```

If-Mask-Scan Statement Model

The If-Mask-Scan statement model (IM) generates a Natural IF MASK/SCAN statement, which allows you to scan portions of a field to check for a specific format (Mask), or scan a field for a specific value (Scan).

Generating the Default Code into the Editor

To generate an IF-MASK/SCAN statement, enter one of the following line commands:

```
.g(IF-MASK-SCAN,)
```

or

```
.g(IM,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 IF #text NE MASK ('AB'NNNN'-'YYMMDD) THEN
0040   do this
0050 ELSE
0060   do that
0070 END-IF
```

Revise the code to complete the IF-MASK/SCAN statement.

Using the IF-MASK-SCAN Statement Window

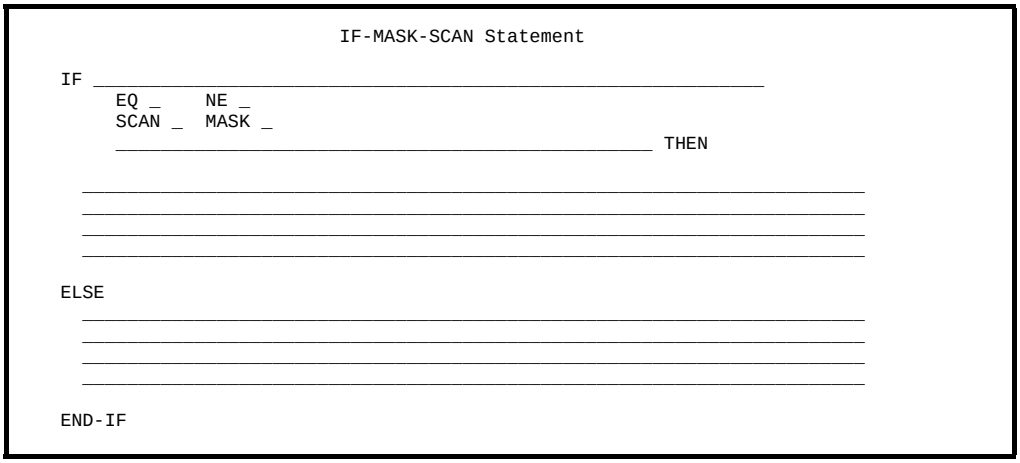
The IF-MASK-SCAN Statement window is displayed when you invoke the If-Mask-Scan statement model from the Generation main menu. You can also invoke the IF-MASK-SCAN Statement window from an editor by entering one of the following line commands:

```
.g(IF-MASK-SCAN)
```

or

```
.g(IM)
```

The IF-MASK-SCAN Statement window is displayed:



IF-MASK-SCAN Statement Window

The fields in this window are:

Field	Description
IF	Portion of a field checked for a specific format, or the name of the field scanned for a specific value. The field format may be alphanumeric, numeric, packed numeric, or binary. This is a required field.
EQ	Mark this field to apply a logical condition to the scan or the mask.
NE	Mark this field to apply a logical condition to the scan or the mask.
SCAN	To scan for a value, mark this field. For the Scan function, the variable specified in the IF field is scanned for the value specified in the field below the SCAN field. This is a required field for the Scan function only. The If-Mask-Scan statement model supports the Natural Variable Mask option. The value used for the Scan function can be an alphanumeric constant or database field, or a user-defined variable (trailing blanks are not included).

Field	Description (continued)
MASK	To scan portions of a field and check for a specific format, mark this field. If you mark MASK and press Enter, the Option MASK window is displayed to define the characteristics of the mask. You can also press PF5 (optns) when MASK is marked to invoke the window.
THEN	Action performed if the logical condition is true.
ELSE	Action performed if the logical condition is not true. This is an optional field.

Specifying Checking Criteria

Press PF5 (optns) in the IF-MASK-SCAN Statement window to specify checking criteria for the mask:

```
                              Option  MASK

                  ....+....1....+....2....+....3....+....4....+....5....+...
Mask characters  ____________________________________________________________
Mask string .... ____________________________________________________________

  L    Lower case (a-z)            DD   Valid day (00-31)
  U    Upper case (A-Z)            MM   Valid month (01-12)
  A    Alpha (a-z,A-Z)             YY   Year (00-99)
  N    Number (0-9)                YYYY Year (2522-2009)
  H    Hexadecimal(A-F,0-9)        C    Alpha, numeric, or blank
  P    Printable characters        .    Position not checked
  S    Special character           *    Positions not checked
  $    Blank character             Z    Half byte (left:A-F right:0-9)
  n... Positions checked for numeric value (0-n...)

  #..  Sequence of positions from   ______ TO ______
  X    Equivalent position in value ____________________________________

VARIABLE MASK ______________________________________________________________
```

Option MASK Window

The fields in this window are:

Field	Description
Mask characters	Code(s) to identify the characters and positions within the mask. A numeric ruler is displayed for your convenience. Each code must correspond to one position and each position may only be specified once. For example, if you specify a character in the first position of this field, you cannot specify a character in the first position of the Mask string field. Each code must also be listed in the table displayed in this window. Valid codes are: • L (lower case letter from a to z) • U (upper case letter from A to Z) • A (lower or upper case letter) • N (number from 0 to 9) • H (hexadecimal number from A to F or 0 to 9) • P (printable lower case letter, upper case letter, or number from 0 to 9) • S (special character, such as #, @, or *) • $ (blank character) • n...(one or more positions, which are checked for a numeric value in the range of 0 to n...) • #... (sequence of positions, which must be in numeric form. If you choose this option, specify the starting value in the “from” portion of this field and the ending value in the “TO” portion) • X (character string enclosed within single quotes or a variable in A, N, P, or B format. The mask is checked against the equivalent position in the specified string or variable. The length of the string or variable must be longer than the length of the mask) • DD (any valid day from 01 to 31. For example, you could not have 30 for February, or 29 if it was not a Leap year) • MM (any month from 01 to 12) • YY (any two-position year from 00 to 99) • YYYY (any four-position year from 0000 to 2099) • C (lower case letter, an upper case letter, a number from 0 to 9, or a blank) • .. (a single position, which is not checked) • * (multiple positions, which are not checked) • Z (a half-byte, which is from A to F on the left and from 0 to 9 on the right)

Field	Description (continued)
Mask string	Positions checked for a specific character. This field is used in conjunction with the Mask characters field. Each character specified must correspond to a position that is not specified in the Mask characters field. (If you specify a character in the first position of the Mask characters field, for example, you cannot specify a character in the first position of this field.)
VARIABLE MASK	Alphanumeric variable. The contents of this variable are used for the mask definition. Trailing blanks within the variable are ignored. If you did not specify a value in the Mask characters or Mask string fields, this is a required field.
Note:	You must define either the VARIABLE MASK definition field or the Mask characters and/or Mask string fields.

If-Mask-Scan Statement Model Example

The following specifications generate an IF MASK/SCAN statement:

```
                        IF-MASK-SCAN Statement

IF #TEXT_____________________________________________________________
     EQ _    NE X
     SCAN _  MASK X
     ____________________________________________________ THEN

  WRITE 'Invalid format'_____________________________________________
  ___________________________________________________________________
  ___________________________________________________________________
  ___________________________________________________________________

ELSE
  ___________________________________________________________________
  ___________________________________________________________________
  ___________________________________________________________________
  ___________________________________________________________________

END-IF
```

If-Mask-Scan Statement Model Example

```
                              Option  MASK

                 ....+....1....+....2....+....3....+....4....+....5....+...
Mask characters     NNNNN YYYYMMDD_____________________________________________
Mask string .... AB___________________________________________________________

  L    Lower case (a-z)            DD   Valid day (00-31)
  U    Upper case (A-Z)            MM   Valid month (01-12)
  A    Alpha (a-z,A-Z)             YY   Year (00-99)
  N    Number (0-9)                YYYY Year (2522-2009)
  H    Hexadecimal(A-F,0-9)        C    Alpha, numeric, or blank
  P    Printable characters        .    Position not checked
  S    Special character           *    Positions not checked
  $    Blank character             Z    Half byte (left:A-F right:0-9)
  n... Positions checked for numeric value (0-n...)

  #..  Sequence of positions from   ______ TO ______
  X    Equivalent position in value ____________________________________

VARIABLE MASK _________________________________________________________________
```

If-Mask-Scan Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 IF #TEXT NE MASK ('AB 'NNNNN' 'YYYYMMDD) THEN
0040   WRITE 'Invalid format'
0050 END-IF
```

If-Selection Statement Model

The If-Selection statement model (IS) generates a Natural IF SELECTION NOT UNIQUE statement. This statement verifies that, in a sequence of fields, only one field is marked. The THEN action is executed if either of the following is true:

- Each field specified in the IF SELECTION NOT UNIQUE statement has a null value.
- More than one field specified in the IF SELECTION NOT UNIQUE statement has a non-null value.

Generating the Default Code into the Editor

To generate an IF SELECTION NOT UNIQUE statement, enter one of the following line commands:

```
.g(IF-SELECTION,)
```

or

```
.g(IS,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 IF SELECTION NOT UNIQUE IN
0040    #field1
0050    #field2
0060    #field3 THEN
0070   do this
0080 ELSE
0090   do that
0100 END-IF
```

Revise the code to complete the IF SELECTION NOT UNIQUE statement.

Using the IF-SELECTION Statement Window

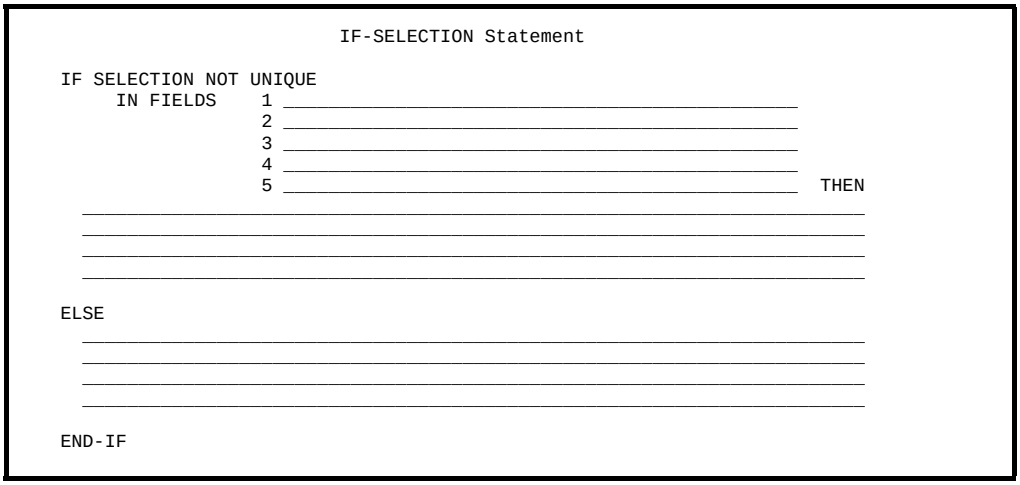
The IF-SELECTION Statement window is displayed when you invoke the If-Selection statement model from the Generation main menu. You can also invoke the IF-SELECTION Statement window from an editor by entering one of the following line commands:

```
.g(IF-SELECTION)
```

or

```
.g(IS)
```

The IF-SELECTION Statement window is displayed:



IF-SELECTION Statement Window

The fields in this window are:

Field	Description
IF SELECTION NOT UNIQUE IN FIELDS	Names of the fields to which the selection processing applies. All fields specified must have the same format. Valid formats are A (alphanumeric), L (logical), or C (attribute control). This is a required field.
THEN	Action performed if all the fields specified in the Selection field variables field have a null value, or if more than one field has a non-null value. This is a required field.
ELSE	Action performed if only one of the fields listed in the Selection field variables field has a non-null value. This is an optional field.

If-Selection Statement Model Example

The following specifications generate an IF STATEMENT NOT UNIQUE statement:

```
                        IF-SELECTION Statement

IF SELECTION NOT UNIQUE
     IN FIELDS    1 #FUNCTION1_____________________________________
                  2 #FUNCTION2_____________________________________
                  3 #FUNCTION3_____________________________________
                  4 _____________________________________________
                  5 _____________________________________________  THEN
  REINPUT 'Mark one and only one function'_____________________________
  MARK *#FUCNTION1 ALARM_______________________________________________
  _____________________________________________________________________
  _____________________________________________________________________

ELSE
  _____________________________________________________________________
  _____________________________________________________________________
  _____________________________________________________________________
  _____________________________________________________________________

END-IF
```

If-Selection Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 IF SELECTION NOT UNIQUE IN
0040    #FUNCTION1
0050    #FUNCTION2
0060    #FUNCTION3 THEN
0070   REINPUT 'Mark one and only one function'
0080   MARK *#FUCNTION1 ALARM
0090 END-IF
```

Include Statement Model

The Include statement model (IN) generates a Natural INCLUDE statement, which merges source lines from an external copycode (C) object type into another object when a program is compiled.

Generating the Default Code into the Editor

To generate an INCLUDE statement, enter one of the following line commands:

```
.g(INCLUDE,)
```

or

```
.g(IN,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 INCLUDE copycode
0040         #parameter1
0050         #parameter2
0060
```

Revise the code to complete the INCLUDE statement.

Using the INCLUDE Statement Window

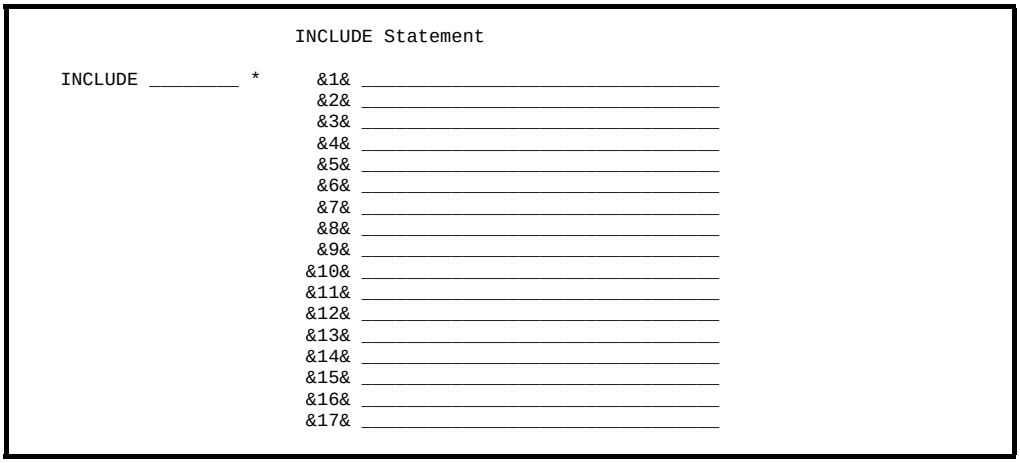
The INCLUDE Statement window is displayed when you invoke the Include statement model from the Generation main menu. You can also invoke the INCLUDE Statement window from an editor by entering one of the following line commands:

```
.g(INCLUDE)
```

or

```
.g(IN)
```

The INCLUDE Statement window is displayed:



INCLUDE Statement Window

The fields in this window are:

Field	Description
INCLUDE	Name of the copycode that has its source included when the program is compiled. This is a required field.
&1&, etc.	Up to 17 dynamic substitution parameters, which are passed in the copycode. This is an optional field. In the copycode, the values are referenced with the "&n&" notation. For every "&n&" notation in the copycode, you must specify a value in the INCLUDE statement. For example, if the copycode contains "&5&", you must pass five parameters with the INCLUDE statement. Values specified in the INCLUDE statement, but not referenced in the copycode, are ignored.

Include Statement Model Example

The following specifications generate an INCLUDE statement:

```
                      INCLUDE Statement

  INCLUDE CCDCOUTM *     &1& #FILE-NAME_____________________
                         &2& #FILE-KEY______________________
                         &3& '10'___________________________
                         &4& _______________________________
                         &5& _______________________________
                         &6& _______________________________
                         &7& _______________________________
                         &8& _______________________________
                         &9& _______________________________
                        &10& _______________________________
                        &11& _______________________________
                        &12& _______________________________
                        &13& _______________________________
                        &14& _______________________________
                        &15& _______________________________
                        &16& _______________________________
                        &17& _______________________________
```

Include Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 INCLUDE CCDCOUTM
0040         #FILE-NAME
0050         #FILE-KEY
0060         '10'
```

Input Statement Model

The Input statement model (IP) generates an INPUT statement, which creates a formatted panel or map for data entry in interactive mode. It can also provide user data for Natural programs executed in batch mode, and can be used in conjunction with the Natural stack facility.

Generating the Default Code into the Editor

To generate an INPUT statement, enter one of the following line commands:

```
.g(INPUT,)
```

or

```
.g(IP,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 INPUT (AD=MI'_' SG=ON ZP=ON IP=ON)
0040       WITH TEXT 'message-text'
0050       ALARM
```

Revise the code to complete the INPUT statement.

Using the INPUT Statement Window

The INPUT Statement window is displayed when you invoke the Input statement model from the Generation main menu. You can also invoke the INPUT Statement window from an editor by entering one of the following line commands:

```
.g(INPUT)
```

or

```
.g(IP)
```

The INPUT Statement window is displayed:

```
                          INPUT Statement

INPUT WINDOW _____________________________________________  NO ERASE _
 AD=M X AD=A _ AD=O _ AD=I _ AD=D X SG=ON _ IP=ON _ ZP=ON _  Filler ... _
 HE= ___________________________ CV= ___________________________

  WITH TEXT
    Msg Number . ____ *
         VARIABLE ____________________________________________________
    Msg Text ... ____________________________________________________
       Text Substitutions . :1:  ____________________________________
                            :2:  ____________________________________
       Field Representation AD=B _ C _ I _ U _ V _

  MARK  POSITION ____________________________________________________
    IN FIELD  Number ..... ____
              Name ....... ____________________________________________
  ALARM _

  USING MAP ________ *
```

INPUT Statement Window

The fields in this window are:

Field	Description
INPUT WINDOW	Name of the window for which the INPUT statement is executed. The window must be defined with a DEFINE WINDOW statement. This is an optional field.
NO ERASE	If this field is marked, the map for the INPUT statement overlays the existing screen without erasing the screen contents.
AD=	Defaults that apply to the INPUT statement. Mark one or more of the following options: • M (both input and output fields) • A (input fields only) • O (output fields only) • I (intensified fields) • D (normal intensity fields)
SG=ON	Display negative sign characters.
IP=ON	Input panel generates prompts.
ZP=ON	Zeros displayed on the generated input panel.
Filler	To display filler characters for all partially filled or null fields, specify the character.

Field	Description (continued)
HE=	Name of a helproutine (enclose constants within single quotes).
CV=	Name of a control variable.
Note:	You must either supply a map name in the USING MAP field or specify the input panel defaults fields.
WITH TEXT	The following fields define the input message and attributes:
Msg Number	To use a message number (which is a constant or a variable) to display message text, specify the message number in this field or the variable containing the message number in the VARIABLE field.
VARIABLE	To use a message number (which is a constant or a variable) to display message text, specify the message number in the Msg Number field or the variable containing the message number in the VARIABLE field.
Msg Text	To use message text, specify it in this field.
Note:	You can use either message numbers or message text, but not both.

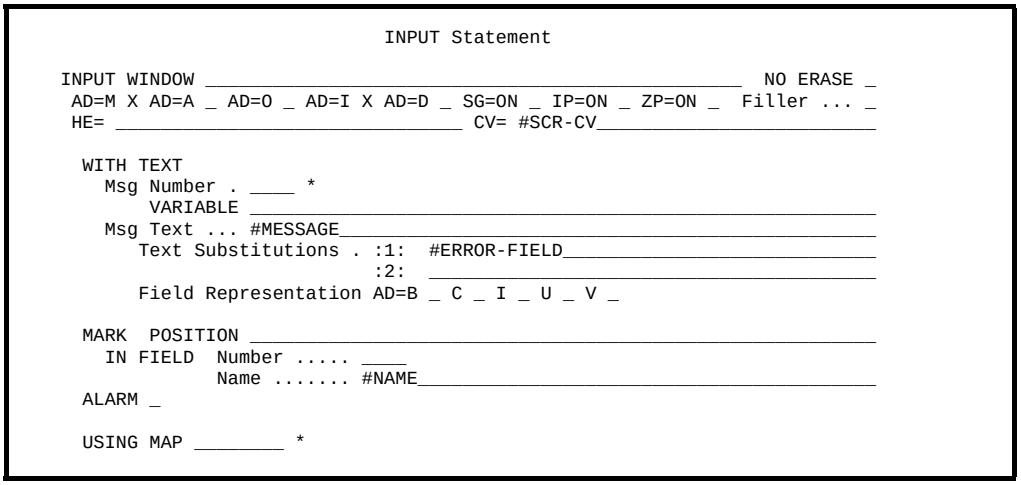
Text Substitutions	To dynamically replace message text, specify the alternative(s) on the two lines provided. For example, if you enter message text as follows:
	`'Name:1:does not exist in file:2:'`
	and the substitution variables are #NAME and #FILE and have the values "Smith" and "EMPLOYEES" at runtime, the INPUT message is displayed as:
	`Name Smith does not exist in file EMPLOYEES`

Field Representation AD=	Codes for display attributes. You can mark the following options: • B (blinking) • C (cursive or italic) • I (intensified) • U (underlined) • V (reverse video)
MARK POSITION	To position the cursor at a particular field, specify the name of the non-protected field.

Field	Description (continued)
IN FIELD Number	To position the cursor at a particular field, specify the number of the non-protected field.
Name	To position the cursor at a particular field, specify the name of the non-protected field.
ALARM	To activate the sound alarm feature of the terminal when the INPUT statement is executed, mark this field.
USING MAP	Name of the map used for input processing. The name must be an alphanumeric constant and no more than eight characters in length. If the name contains an ampersand (&), this character is replaced at runtime by the current value of *Language.

Input Statement Model Example

The following specifications generate an INPUT statement:



Input Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 INPUT (AD=MI'_' CV=#SCR-CV SG=OFF ZP=OFF IP=OFF)
0040       WITH TEXT #MESSAGE
0050                 ,#ERROR-FIELD
0060       MARK *#NAME
```

Move Statement Model

The Move statement model (MO) generates a Natural MOVE statement, which moves the value of an operand to one or more operands (fields or arrays).

Generating the Default Code into the Editor

To generate a MOVE statement, enter one of the following line commands:

```
.g(MOVE,)
```

or

```
.g(MO,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 MOVE value
0040   TO #variable
```

Revise the code to complete the MOVE statement.

Using the MOVE Statement Window

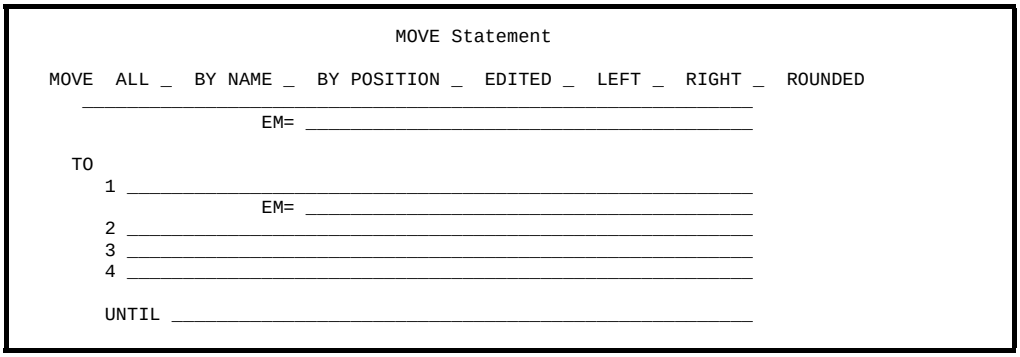
The MOVE Statement window is displayed when you invoke the Move statement model from the Generation main menu. You can also invoke the MOVE Statement window from an editor by entering one of the following line commands:

```
.g(MOVE)
```

or

```
.g(MO)
```

The MOVE Statement window is displayed:



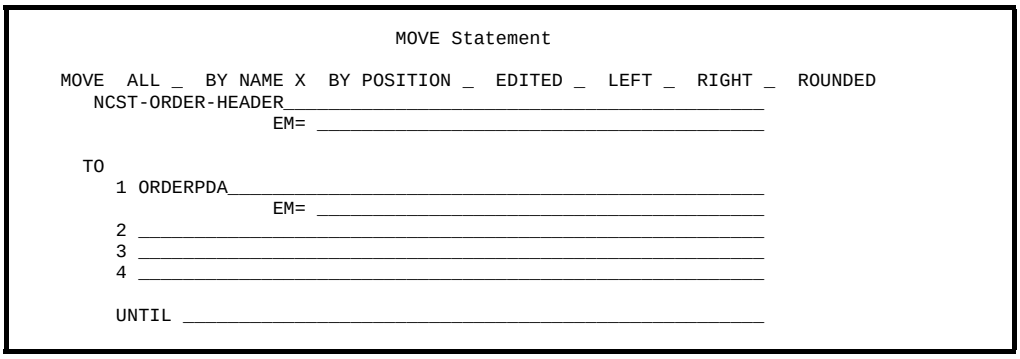
MOVE Statement Window

The fields in this window are:

Field	Description
MOVE	The fields on this line allow you to define the characteristics of the value being moved. In the field on the second line, specify the value (or variable containing the value) that is moved. This is a required field.
ALL	To repeatedly move the value to the TO field until the TO field is filled, mark this field.
BY NAME	To move individual fields from one data structure to another, regardless of their position in the structure, mark this field. (A field is only moved if its name appears in both structures.)
BY POSITION	To move the contents of fields in a group to another group, regardless of the field names, mark this field. The values are moved one field at a time in the same order the fields are defined. The fields may be in any format. The number of fields in each group must be the same and the level structure and array dimensions of the fields must match.
LEFT	To left-justify the data moved to the TO field, mark this field.
RIGHT	To right-justify the data moved to the TO field, mark this field.
ROUNDED	To round a numeric value that is being moved, mark this field.
EM=	To specify an edit mask for the values in the MOVE field, mark this field. If an edit mask is specified for the MOVE field, it is applied to that field and the resulting value is moved to the TO field. The TO field must be A (alphanumeric) or B (binary) format.
TO	Name(s) of the destination variables (where the value specified in the MOVE field is moved). This is a required field.
EM=	To specify an edit mask for the values in the TO field, mark this field. If an edit mask is specified for the TO field, the value in the MOVE field must match the format of that edit mask. The mask information is taken from the MOVE field and the result is moved to the TO field. The value in the MOVE field must be A (alphanumeric) or B (binary) format.
UNTIL	To limit the MOVE ALL operation to a given number of positions in the TO field, specify the number of positions in this field. The MOVE ALL operation is terminated when the specified value is reached. If the value is greater than the length of the TO field, the MOVE ALL operation is terminated when the TO fields are full.

Move Statement Model Example

The following specifications generate a MOVE statement:



Move Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 MOVE BY NAME NCST-ORDER-HEADER
0040   TO ORDERPDA
```

Multiply Statement Model

The Multiply statement model (MU) generates a Natural MULTIPLY statement, which multiplies two operands.

Generating the Default Code into the Editor

To generate a MULTIPLY statement, enter one of the following line commands:

```
.g(MULTIPLY,)
```

or

```
.g(MU,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 MULTIPLY #multiplicand
0040       BY #multiplier
0050   GIVING #result
```

Revise the code to complete the MULTIPLY statement.

Using the MULTIPLY Statement Window

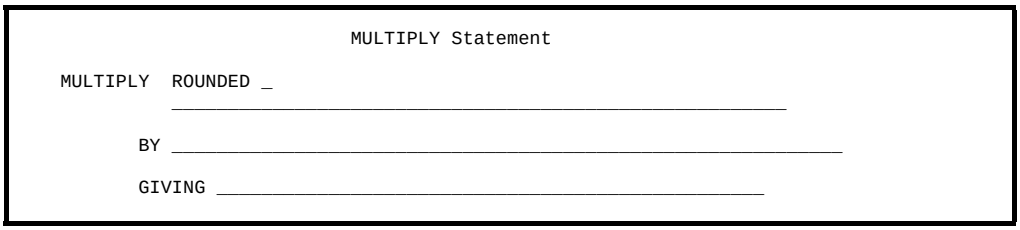
The MULTIPLY Statement window is displayed when you invoke the Multiply statement model from the Generation main menu. You can also invoke the MULTIPLY Statement window from an editor by entering one of the following line commands:

```
.g(MULTIPLY)
```

or

```
.g(MU)
```

The MULTIPLY Statement window is displayed:



MULTIPLY Statement Window

The fields in this window are:

Field	Description
MULTIPLY ROUNDED	To round the result of the MULTIPLY operation, mark this field. In the field below, enter the value that is multiplied by the multiplier. This is a required field.
BY	Value by which the multiplicand is multiplied. This is a required field.
GIVING	Name of the field that stores the result of the multiplication.

Multiply Statement Model Example

The following specifications generate a MULTIPLY statement:

```
                          MULTIPLY Statement

   MULTIPLY  ROUNDED _
             #MONTHLY-SALARY_____________________________________________

          BY 12_________________________________________________________

          GIVING #ANNUAL-INCOME_________________________________________
```

Multiply Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 MULTIPLY #MONTHLY-SALARY
0040       BY 12
0050   GIVING #ANNUAL-INCOME
```

Process-Command Statement Model

The Process-Command statement model (PO) generates a Natural PROCESS COMMAND ACTION statement, which interfaces with the Natural command processor. Once a command processor has been created using the Natural SYSNCP utility, it can be invoked from a Natural program using the PROCESS COMMAND ACTION statement. For information about creating command processors, refer to the Natural documentation.

Generating the Default Code into the Editor

To generate a PROCESS COMMAND ACTION statement, enter one of the following line commands:

```
.g(PROCESS-COMMAND,)
```

or

```
.g(PO,)
```

The following default code is generated:

```
0010 PROCESS command-view ACTION EXEC
0020   USING PROCESSOR-NAME = #Processor-name
0030   COMMAND-LINE (1:1) = #command-line
```

Revise the code to complete the PROCESS COMMAND ACTION statement.

Using the PROCESS-COMMAND Statement Window

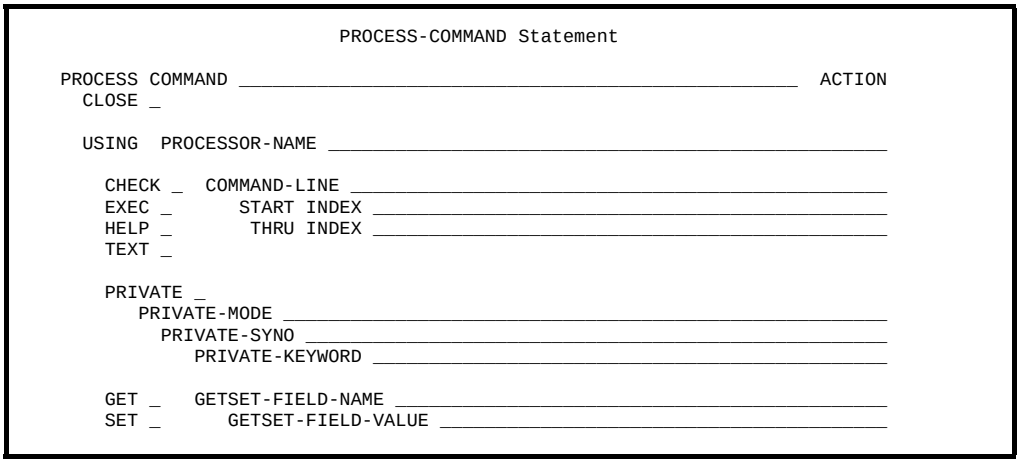
The PROCESS-COMMAND Statement window is displayed when you invoke the Process-Command statement model from the Generation main menu. You can also invoke the PROCESS-COMMAND Statement window from an editor by entering one of the following line commands:

```
.g(PROCESS-COMMAND)
```

or

```
.g(PO)
```

The PROCESS-COMMAND Statement window is displayed:



PROCESS-COMMAND Statement Window

The fields in this window are:

Field	Description
PROCESS COMMAND	We recommend that you use the COMMAND view. A DDM exists with the same name and must be referenced within the DEFINE DATA statement (COMMAND VIEW OF COMMAND, for example).
CLOSE	To terminate the command processor and release the command processor buffer, mark this field. (If the command processor is not released with a PROCESS COMMAND CLOSE statement, a buffer called NCPWORK will remain in your thread.)
USING PROCESSOR-NAME	Name of the command processor for which the PROCESS COMMAND ACTION statement is issued. The specified processor must be cataloged. This field is required for all subcommands except Close.

Field	Description (continued)
CHECK	To check (but not execute) the command line, mark this field. The Check subcommand determines if a command is executable with the PROCESS COMMAND ACTION EXEC statement. This subcommand performs two checks at runtime: • Checks whether the command processor exists in the current library or steplib. • Analyses the command line to determine whether the contents are acceptable.
EXEC	To check and execute the command line, mark this field. This subcommand works exactly the same way as the Check subcommand except the runtime action is executed.
HELP	To return a list of all valid keywords, synonyms, and functions, mark this field. You can use this information to create online help windows, for example. The type of help returned depends on the command line contents.
TEXT	To return general information about the command processor and text associated with a keyword or function, mark this field. The text must have been entered in the keyword or action editor of the Natural SYSNCP utility when the processor was defined.
COMMAND-LINE	Either the name of the command line processed by the command processor (Check and Execute subcommands), or the keyword/command for which user or help text is returned to the program (Text and Help subcommands). This field is required for the Check, Execute, Help, and Text subcommands.
START INDEX	Starting index for the command line.
THRU INDEX	Ending index for the command line.
PRIVATE	To read, delete, add, or modify user-defined synonyms for keywords, mark this field.
PRIVATE-MODE	Code for the action applied to a private synonym. Valid codes are: • R (read) • A (add) • D (delete) • M (modify)
PRIVATE-SYNO	User-defined synonym for an existing keyword.

Field	Description (continued)
PRIVATE-KEYWORD	Keyword for which the synonym applies.
Note:	The PRIVATE-MODE, PRIVATE-SYNO, and PRIVATE-KEYWORD fields are required for the Private subcommand.
GET	To read internal command processor information and current processor settings from the dynamically allocated NCPWORK buffer, mark this field.
SET	To modify the internal command processor settings in the NCPWORK buffer, mark this field.
Note:	You must mark one of the two fields above.
GETSET-FIELD-NAME	Name of the constant or variable that is read or written.
GETSET-FIELD-VALUE	Name of the field that contains the value of the constant or variable specified in the GETSET-FIELD-NAME field.

Note: For more information, refer to PROCESS COMMAND in the Natural documentation.

Process-Command Statement Model Example

The following specifications generate a PROCESS COMMAND ACTION statement:

```
                          PROCESS-COMMAND Statement

PROCESS COMMAND COMMAND_____________________________________________  ACTION
  CLOSE _

  USING  PROCESSOR-NAME 'DEMO'_____________________________________________

    CHECK _  COMMAND-LINE #COMMAND-LINE_____________________________________
    EXEC X      START INDEX 1_______________________________________________
    HELP _       THRU INDEX 1_______________________________________________
    TEXT _

    PRIVATE _
       PRIVATE-MODE _____________________________________________________
         PRIVATE-SYNO _____________________________________________________
            PRIVATE-KEYWORD _______________________________________________

    GET _   GETSET-FIELD-NAME _______________________________________________
    SET _      GETSET-FIELD-VALUE ___________________________________________
```

Process-Command Statement Model Example

Example of code generated from the sample specifications

```
0010 PROCESS COMMAND ACTION EXEC
0020   USING PROCESSOR-NAME = 'DEMO'
0030   COMMAND-LINE (1:1) = #COMMAND-LINE
```

Read Statement Model

The Read statement model (RD) generates a Natural READ statement, which reads records from the database. The records can be retrieved in physical sequence, in Adabas ISN sequence, or in the value sequence of a descriptor (key) field.

Generating the Default Code into the Editor

To generate a READ statement, enter one of the following line commands:

```
.g(READ,)
```

or

```
.g(RD,)
```

The following default code is generated:

```
0010 /*
0020 /* Read view-name
0030 READ1.
0040 READ(10) view-name
0050      BY descriptor STARTING FROM start-value THRU end-value
0060   /*
0070   /*
0080   /*
0090   /*
0100 END-READ /* READ1.
```

Revise the code to complete the READ statement.

Using the READ Statement Window

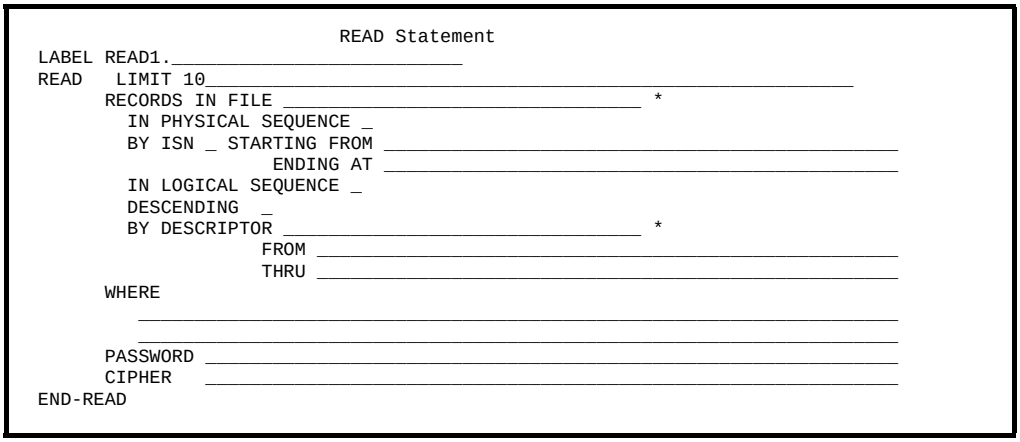
The READ Statement window is displayed when you invoke the Read statement model from the Generation main menu. You can also invoke the READ Statement window from an editor by entering one of the following line commands:

```
.g(READ)
```

or

```
.g(RD)
```

The READ Statement window is displayed:



READ Statement Window

The fields in this window are:

Field	Description
LABEL	Label or reference for the READ statement. To give the READ statement a label or reference, specify the label in this field. This is an optional field.
READ LIMIT	Limit on the number of records read. To limit the number of records read, specify a numeric constant or numeric variable in this field.
RECORDS IN FILE	Name of a view defined in a global data area (GDA) or local data area (LDA). This is a required field.
IN PHYSICAL SEQUENCE	To read records in the order they are physically stored in the database, mark this field.
BY ISN	To read the records in ISN (Adabas Internal Sequence Number) order, mark this field.
STARTING FROM	Starting value for the ISNs.
ENDING AT	Ending value for the ISNs.
IN LOGICAL SEQUENCE	To read records in the order of the default descriptor sequence (as defined in the DDM), mark this field.
DESCENDING	To retrieve records in descending order, mark this field. Otherwise the records are sorted in ascending order.

Field	Description (continued)
BY DESCRIPTOR	To read the records by descriptor (subdescriptor or superdescriptor), specify the name of the descriptor.
FROM	Starting value for the descriptor.
THRU	Ending value for the descriptor.
Note:	You must indicate one of the above options: IN PHYSICAL SEQUENCE, BY ISN, IN LOGICAL SEQUENCE, or BY DESCRIPTOR.
WHERE	Additional selection criteria. To specify additional selection criteria, which are evaluated after a record has been read and before any processing is performed, specify the additional criteria in this field. This is an optional field.
PASSWORD	Password for a password-protected Adabas file.
CIPHER	Cipher key for an enciphered Adabas file.

Read Statement Model Example

The following specifications generate a READ statement:

```
                          READ Statement
LABEL READ-EMP_____________________________
READ   LIMIT   ____________________________________________________________
     RECORDS IN FILE NCST-EMPLOYEES_____________________ *
       IN PHYSICAL SEQUENCE _
       BY ISN _ STARTING FROM ____________________________________________
                 ENDING AT ____________________________________________
       IN LOGICAL SEQUENCE _
       DESCENDING  _
       BY DESCRIPTOR PERSONNEL-ID_____________________ *
                  FROM 'Smith'_____________________________________________
                  THRU 'Smyth'_____________________________________________
     WHERE
        ____________________________________________________________________
        ____________________________________________________________________
     PASSWORD ___________________________________________________________
     CIPHER   ___________________________________________________________
END-READ
```

Read Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /* Read NCST-EMPLOYEES
0030 READ-EMP.
0040 READ NCST-EMPLOYEES
0050      BY PERSONNEL-ID STARTING FROM 'Smith' THRU 'Smyth'
0060 END-READ /* READ-EMP.
```

Read-Work-File Statement Model

The Read-Work-File statement model (RW) generates a Natural READ WORK FILE statement, which reads data from a physical sequential work file. This statement causes a processing loop to be initiated to read all records in the work file.

Generating the Default Code into the Editor

To generate a READ WORK FILE statement, enter one of the following line commands:

```
.g(READ-WORK-FILE,)
```

or

```
.g(RW,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 READ WORK FILE 01
0040      RECORD #record
0050 END-WORK
```

Revise the code to complete the READ WORK FILE statement.

Using the READ-WORK-FILE Statement Window

The READ-WORK-FILE Statement window is displayed when you invoke the Read-Work-File statement model from the Generation main menu. You can also invoke the READ-WORK-FILE Statement window from an editor by entering one of the following line commands:

```
.g(READ-WORK-FILE)
```

or

```
.g(RW)
```

The READ-WORK-FILE Statement window is displayed:

```
                        READ-WORK-FILE Statement

READ WORK FILE  Number ... __  ONCE _
  RECORD _____________________________________________
                                                        OFFSET FILLER
  SELECT _____________________________________________   __     __
         _____________________________________________   __     __
         _____________________________________________   __     __
         _____________________________________________   __     __
         _____________________________________________   __     __

  GIVING LENGTH _____________________________________________________

  AT END OF FILE
    _____________________________________________________________________
    _____________________________________________________________________
  END-ENDFILE
  _____________________________________________________________________
  _____________________________________________________________________
END-WORK
```

READ-WORK-FILE Statement Window

The fields in this window are:

Field	Description
READ WORK FILE Number	Number of the work file that is read (as defined to Natural). This is a required field.
ONCE	To read only one record, mark this field and specify the statement(s) executed at the End of File condition.
RECORD	To read all fields in a record, specify the name of the record in this field. All fields in the record are made available for processing.
SELECT	To read specific fields in a record, specify the fields in the Field name field. You can indicate a starting position for the selected field(s) in the OFFSET or FILLER fields.
OFFSET	Number specified in the OFFSET field indicates the relative starting position of the selected field, in relation to the beginning of the record. (Specify “0” for the first byte of the record.)
FILLER	Number specified in the FILLER field indicates the number of bytes that are skipped in the previously selected field (or the beginning of the record if only one field is selected).

Field	Description (continued)
GIVING LENGTH	To return the actual length of the record being read, specify a numeric variable of I4 format and length in this field. This is an optional field.
AT END OF FILE	If you marked the ONCE field, specify the statement(s) executed at the End of File condition in this field.
Note:	When reading an array from a work file, a variable index range for the array can be specified. For example, the field #ARRAY(#I:#J) can be specified in the SELECT field.

Read-Work-File Statement Model Example

The following specifications generate a READ WORK FILE statement:

```
                        READ-WORK-FILE Statement

READ WORK FILE  Number ... 1_  ONCE _
  RECORD #RECORD_____________________________________
                                                        OFFSET FILLER
  SELECT ___________________________________________     __     __
         ___________________________________________     __     __
         ___________________________________________     __     __
         ___________________________________________     __     __
         ___________________________________________     __     __

  GIVING LENGTH #LENGTH_____________________________________________

  AT END OF FILE
    _________________________________________________________________
    _________________________________________________________________
  END-ENDFILE
  _____________________________________________________________________
  _____________________________________________________________________
END-WORK
```

Read-Work-File Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 READ WORK FILE 01
0040      RECORD #RECORD
0050      GIVING LENGTH #LENGTH
0060 END-WORK
```

Reinput Statement Model

The Reinput statement model (RI) generates a Natural REINPUT statement, which returns to and re-executes an INPUT statement. It displays a message indicating that data input as a result of the previous INPUT statement is invalid.

Generating the Default Code into the Editor

To generate a REINPUT statement, enter one of the following line commands:

```
.g(REINPUT,)
```

or

```
.g(RI,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 REINPUT
0040   WITH TEXT 'message text'
0050   MARK *mark-field-name
0060   ALARM
```

Revise the code to complete the REINPUT statement.

Using the REINPUT Statement Window

The REINPUT Statement window is displayed when you invoke the Reinput statement model from the Generation main menu. You can also invoke the REINPUT Statement window from an editor by entering one of the following line commands:

```
.g(REINPUT)
```

or

```
.g(RI)
```

The REINPUT Statement window is displayed:

```
                            REINPUT Statement

REINPUT FULL _  USING HELP _

  WITH TEXT
    Msg Number . ____ *
        VARIABLE ________________________________
    Msg Text ... ____________________________________________________________
      Text Substitutions . :1: ____________________________________________
                           :2: ____________________________________________
      Field Representation   AD=B _ C _ I _ N _ U _ V _
      Colour Definitions .   CD=BL _ GR _ NE _ PI _ RE _ TU _ YE _

  MARK  POSITION ____________________________________
    IN FIELD  Name ....... ____
              Number ..... ____________________________________  AD= _ CD= __
    IN FIELD  Name ....... ____
              Number ..... ____________________________________  AD= _ CD= __
       Protect unmarked fields ..  AD=P _
  ALARM _
```

REINPUT Statement Window

The fields in this window are:

Field	Description
REINPUT FULL	To apply all modifications (executed after the initial execution of the INPUT statement) to the INPUT statement when it is re-executed, mark this field. All variables on the panel retain their current contents when the REINPUT statement is executed.
USING HELP	To invoke the helproutine defined for the first marked field, mark this field.
WITH TEXT	
Msg Number	To use a message number, either specify a number (as stored in the Natural message file MSGGEN utility) in this field or a variable in the VARIABLE field.
VARIABLE	Name of a variable containing the message number. The specified value accesses the Natural message file to retrieve the message text.
Msg Text	To provide text, which is displayed on the message line, specify the message text in this field. You can assign attributes to the message text using the following fields:

Field	Description (continued)

Text Substitutes	Substitution variable(s) to dynamically replace parts of the message text. For example, if you specify the message text as:
	`'Name:1:does not exist in file:2:'`
	and the substitution variables are #NAME and #FILE and have the values "Smith" and "EMPLOYEES" at runtime, the REINPUT message is displayed as:
	`Name Smith does not exist in file EMPLOYEES`

Field Representation AD=	Display options for message text. Mark one of the following: • B (blinking) • C (cursive or italic) • I (intensified) • U (underlined) • V (reverse video)
Color Definitions CD=	Color attribute for message text. For terminals that support color, mark one of the following: • BL (blue) • GR (green) • NE (neutral/white) • PI (pink) • RE (red) • TU (turquoise) • YE (yellow)
MARK POSITION	Position within a field in which the cursor is placed by default. The Reinput statement model supports the Natural Position option. To position the cursor within a field, specify the numeric constant or the name of a numeric variable in this field. The specified numeric constant or variable must not contain decimal digits.
IN FIELD Name	To position the cursor in a specific field, specify the field name. You can specify a maximum of two field names or a combination of one field name and one field number.
Number	Number of the field in which the cursor is placed by default. To position the cursor to a specified field, specify the field number(s) in the fields provided. You can specify a maximum of two field numbers or a combination of one field number and one field name.

Field	Description (continued)
AD=	Display attributes for each field. To assign display attributes for each field, specify a display code. For a list of the display attribute codes, see the descriptions of the Field Representation fields.
CD=	Color attributes for each field. To assign color attributes for each field, specify a color code. For a list of display attribute codes, see the descriptions of the Color Definitions fields.
Protect unmarked fields	Indicator for whether unmarked fields are protected. To protect all fields when the REINPUT statement is executed, except those specified in the Mark Options field, mark this field.
ALARM	Sound alarm indicator. If this field is marked, the sound alarm feature of the terminal is activated when the REINPUT statement is executed (if your terminal supports the alarm feature). If you do not want the alarm activated, remove the mark from this field.

Reinput Statement Model Example

The following specifications generate a REINPUT statement:

```
                          REINPUT Statement

REINPUT FULL X  USING HELP _

  WITH TEXT
    Msg Number . ____ *
        VARIABLE _____________________________
    Msg Text ... 'Invalid postal code:1:'_____________________________________
      Text Substitutions . :1: #POSTAL-CODE_____________________________
                           :2: _____________________________________________
      Field Representation   AD=B _ C _ I X N _ U _ V _
      Colour Definitions .   CD=BL _ GR _ NE _ PI _ RE _ TU _ YE _

  MARK  POSITION _____________________________________
    IN FIELD  Name ....... ____
              Number ..... #POSTAL-CODE_____________________  AD= I CD= __
    IN FIELD  Name ....... ____
              Number ..... _____________________________  AD= _ CD= __
       Protect unmarked fields ..  AD=P _
  ALARM X
```

Reinput Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 REINPUT FULL
0040   WITH TEXT 'Invalid postal code:1:' (AD=I)
0050             ,#POSTAL-CODE
0060   MARK *#POSTAL-CODE (AD=I)
0070   ALARM
```

Repeat Statement Model

The Repeat statement model (RP) generates a Natural REPEAT statement, which initiates a processing loop. If no logical condition is specified, the loop must be exited by an ESCAPE, STOP, or TERMINATE statement within the loop. If a logical condition is specified, that condition determines when to terminate the loop.

Generating the Default Code into the Editor

To generate a REPEAT statement, enter one of the following line commands:

```
.g(REPEAT,)
```

or

```
.g(RP,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 REPEAT1.
0040 REPEAT
0050   statement1
0060   statement2
0070   ESCAPE BOTTOM(REPEAT1.) IMMEDIATE
0080 END-REPEAT /* REPEAT1.
```

Revise the code to complete the REPEAT statement.

Using the REPEAT Statement Window

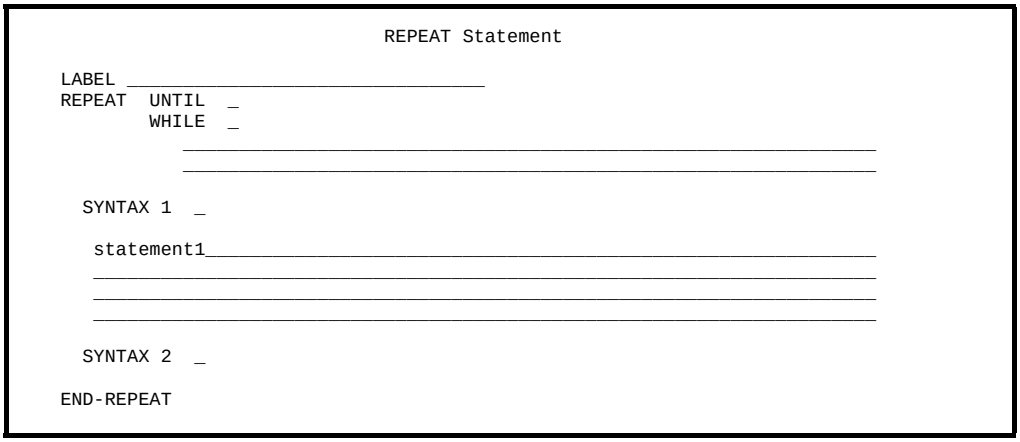
The REPEAT Statement window is displayed when you invoke the Repeat statement model from the Generation main menu. You can also invoke the REPEAT Statement window from an editor by entering one of the following line commands:

```
.g(REPEAT)
```

or

```
.g(RP)
```

The REPEAT Statement window is displayed:



REPEAT Statement Window

The fields in this window are:

Field	Description
LABEL	Label (or reference) for the REPEAT statement. This is an optional field.
REPEAT	
UNTIL	To have the processing loop continue until the logical condition is true, mark this field and specify the logical condition that applies in the field(s) below.
WHILE	To have the processing loop continue as long as the logical condition is true, mark this field and specify the logical condition that applies in the field(s) below.
SYNTAX 1	To execute the specified statements (for either the UNTIL or WHILE logical condition) one or more times, mark this field. In the fields below, specify the statement(s) executed.
SYNTAX 2	To execute the specified statements (for either the UNTIL or WHILE logical condition) zero or more times, mark this field.
Note:	If you mark either the UNTIL or WHILE field, you must also mark either the SYNTAX 1 or SYNTAX 2 field.

Repeat Statement Model Example

The following specifications generate a REPEAT statement:

```
                          REPEAT Statement

LABEL _____________________________
REPEAT  UNTIL  X
        WHILE  _
           *LINE-COUNT > #LOGICAL-PAGESIZE_____________________________
            ___________________________________________________________

  SYNTAX 1  _

   SKIP 1_____________________________________________________________
   ___________________________________________________________________
   ___________________________________________________________________
   ___________________________________________________________________

  SYNTAX 2  X

END-REPEAT
```

Repeat Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 REPEAT
0040   UNTIL *LINE-COUNT > #LOGICAL-PAGESIZE
0050   SKIP 1
0060 END-REPEAT
```

Separate Statement Model

The Separate statement model (SP) generates a Natural SEPARATE statement, which separates the contents of an alphanumeric operand into two or more alphanumeric operands or multiple occurrences in an alphanumeric array.

Generating the Default Code into the Editor

To generate a SEPARATE statement, enter one of the following line commands:

```
.g(SEPARATE,)
```

or

```
.g(SP,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 SEPARATE SUBSTRING (#source-field,#start-position,#length)
0040   LEFT JUSTIFIED INTO #target-fields
```

Revise the code to complete the SEPARATE statement.

Using the SEPARATE Statement Window

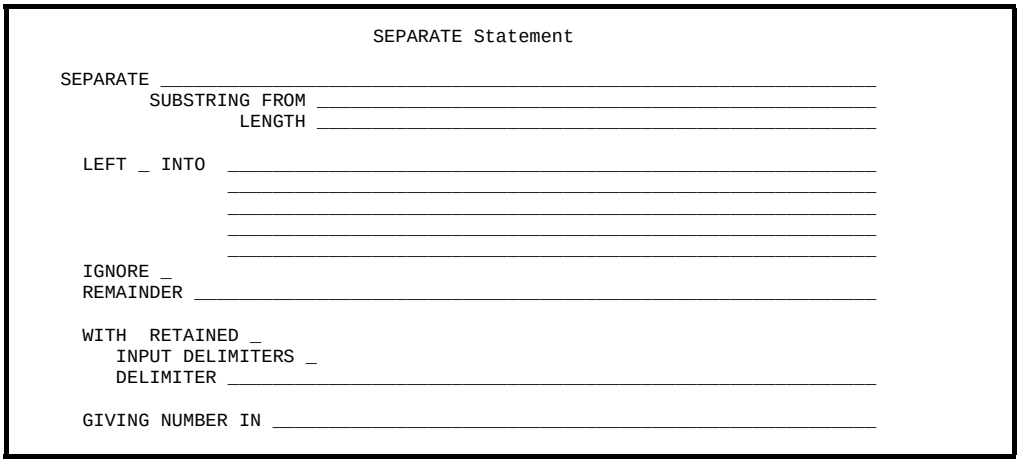
The SEPARATE Statement window is displayed when you invoke the Separate statement model from the Generation main menu. You can also invoke the SEPARATE Statement window from an editor by entering one of the following line commands:

```
.g(SEPARATE)
```

or

```
.g(SP)
```

The SEPARATE Statement window is displayed:



SEPARATE Statement Window

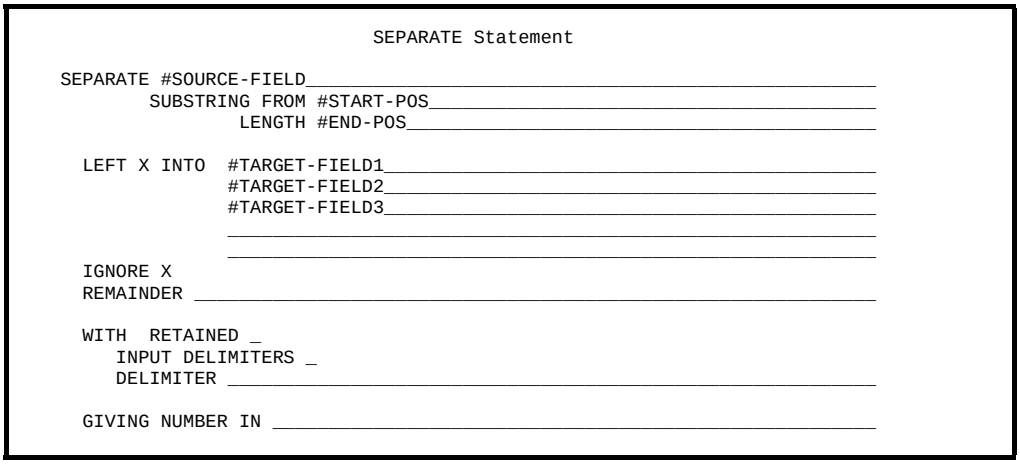
The fields in this window are:

Field	Description
SEPARATE	Name of the alphanumeric operand that is separated. Text must be enclosed within single quotes. This is a required field.
SUBSTRING FROM	To separate a portion of the field, specify the starting position in this field and specify the length of the substring in the LENGTH field.
LENGTH	Number of positions in the separated portion of the field. For example, if #A field contains 'WHATEVER', SUBSTRING(#A,4,3) contains 'TEV'.
LEFT	If this field is marked, leading blanks (which may occur between the delimiter character and the next non-blank character) are removed from the target operand. The result is left-justified.
INTO	Name(s) of the field(s) that receives the separated operands. This is a required field.
IGNORE	If this field is marked, the remainder is ignored when there are not enough target fields to receive the separated results.
REMAINDER	To recognize the remainder, specify the name of the field that receives the remainder of the separated results.

Field	Description (continued)
WITH RETAINED	If this field is marked, each delimiter is placed into the target field.
INPUT DELIMITERS	If this field is marked, the default input delimiter (ID) character is used to indicate the positions at which the value is separated.
DELIMITER	Set of characters used as delimiters. Each one is treated as a separate delimiter character. You can specify an alphanumeric variable containing the set of characters or an alphanumeric constant enclosed within single quotes.
GIVING NUMBER IN	Name of the field that receives the number of filled target fields (including those filled with blanks).

Separate Statement Model Example

The following specifications generate a SEPARATE statement:



Separate Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 SEPARATE SUBSTRING (#SOURCE-FIELD,#START-POS,#END-POS)
0040   LEFT JUSTIFIED INTO #TARGET-FIELD1
0050                       #TARGET-FIELD2
0060                       #TARGET-FIELD3
0070   IGNORE
```

Sequence Statement Model

The Sequence statement model (SQ) repeats a given line with some varying strings as many times as specified, with the changes applied to each variation.

Generating the Default Code into the Editor

To generate a SEQUENCE function, enter one of the following line commands:

```
.g(SEQUENCE,)
```

or

```
.g(SQ,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 /*
0040 Assign initial-value = my-array(1:1)
0050 Assign initial-value = my-array(2:2)
0060 Assign initial-value = my-array(3:3)
0070 /*
0080 SEPARATE SUBSTRING (#source-field,#start-position,#length)
0090   LEFT JUSTIFIED INTO #target-fields
```

Revise the code to complete the SEQUENCE function.

Using the SEQUENCE Statement Window

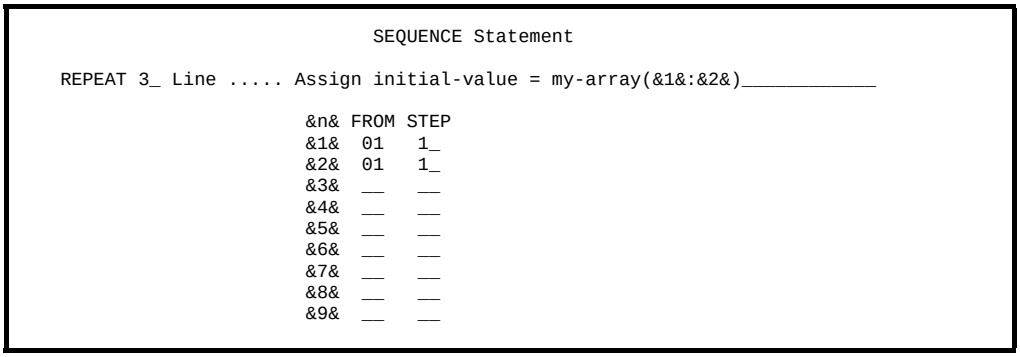
The SEQUENCE Statement window is displayed when you invoke the Sequence statement model from the Generation main menu. You can also invoke the SEQUENCE Statement window from an editor by entering one of the following line commands:

```
.g(SEQUENCE)
```

or

```
.g(SQ)
```

The SEQUENCE Statement window is displayed:



SEQUENCE Statement Window

The fields in this window are:

Field	Description
REPEAT	Number of times the line is repeated. This is a required field.
Line	Character string that is repeated. This is a required field.
FROM	Starting value for the “&1&” to “&9&” variations.
STEP	Amount by which “&1&” is increased. By default, the STEP value is equal to the highest value in the varying string.
Note:	You can specify up to nine FROM and STEP values on the lines provided. The number you can specify depends on how many variation signs you entered in the Line field.

Sequence Statement Model Example

The following specifications generate code that repeats a line containing a sequence:

```
                          SEQUENCE Statement

REPEAT 3_ Line ..... Assign initial-value = my-array(&1&:&2&)___________

                     &n& FROM STEP
                     &1&  01   1_
                     &2&  01   1_
                     &3&  __   __
                     &4&  __   __
                     &5&  __   __
                     &6&  __   __
                     &7&  __   __
                     &8&  __   __
                     &9&  __   __
```

Sequence Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 Assign initial-value = my-array(1:1)
0040 Assign initial-value = my-array(2:2)
0050 Assign initial-value = my-array(3:3)
```

Set-Control Statement Model

The Set-Control statement model (SC) generates a Natural SET CONTROL statement, which performs terminal commands from within a program. Terminal commands perform a wide variety of special-purpose functions.

Note: An overview and complete description of each terminal command are provided in the Natural documentation.

Generating the Default Code into the Editor

To generate a SET CONTROL statement, enter one of the following line commands:

```
.g(SET-CONTROL,)
```

or

```
.g(SC,)
```

The following default code is generated:

```
0010 **
0020 **
0030 SET CONTROL 'U' /* Enable lower-upper translation
```

Revise the code to complete the SET CONTROL statement.

Using the SET-CONTROL Statement Window

The SET-CONTROL Statement window is displayed when you invoke the Set-Control statement model from the Generation main menu. You can also invoke the SET-CONTROL Statement window from an editor by entering one of the following line commands:

```
.g(SET-CONTROL)
```

or

```
.g(SC)
```

The SET-C0NTROL Statement window is displayed:

```
                    SET-CONTROL Statement

Copy Data to Stack or *COM ... SET CONTROL _ CS CC
Frame Characters for Window .. SET CONTROL _ F=chv
Simulate PF\PA-KEY ........... SET CONTROL _ Knn KPn
Message Line Control ......... SET CONTROL _ M
Window Processing ............ SET CONTROL _ W
Control of PF-KEY Lines ...... SET CONTROL _ Y
Copy Contents of Page Buffer . SET CONTROL _ C
Activate Error Processing .... SET CONTROL _ E=ON  _ E=OFF
Lower\Upper Case Translation . SET CONTROL _ L  _ U
Display Stored Screens ....... SET CONTROL _ E
Forms\Screen Mode ............ SET CONTROL _ F
Non-Conversational Control ... SET CONTROL _ N
Delete First Entry from Stack  SET CONTROL _ .P
Suppress Map Printing in Batch SET CONTROL _ Q
Clear Data Buffer ............ SET CONTROL _ R
Read Stack without Deleting .. SET CONTROL _ .S
Clear Source Area ............ SET CONTROL _ Z
Interrupt Current Operation .. SET CONTROL _ .
```

SET-CONTROL Statement Window

Use this window to indicate the conditions for which to generate SET CONTROL statements. When you mark some of the fields and press PF5 (optns), additional specification windows are displayed. Each time an additional window is displayed, mark the options you want to include and press Enter. Any additional windows are displayed automatically until you have filled in the last window for which you marked a field in the SET-CONTROL Statement window.

The fields in the SET-CONTROL Statement window are:

Field	Description
Copy Data to Stack or #COM	If this field is marked, the SET CONTROL statement copies a source line of screen data to the Natural stack or to the system variable *COM. Press Enter or PF5 to display the Copy Data to Stack or *COM window.
Frame Characters for Window	If this field is marked, the SET CONTROL statement changes the appearance of a default window. Press PF5 to display the Frame Characters window.
Simulate PF/PA-KEY	If this field is marked, the SET CONTROL statement assigns the value of PF-keys, activates PF-keys not available on the current terminal hardware, makes function keys available in batch mode, and simulates program attention keys. Press PF5 to display the Simulate PF/PA-KEY window.

Field	Description (continued)
Message Line Control	If this field is marked, the SET CONTROL statement defines attributes for the message line. Press PF5 to display the Message Line Control window.
Window Processing	If this field is marked, the SET CONTROL statement defines the size and positioning of the window. Press PF5 to display the Window Processing window.
Control of PF-KEY Lines	If this field is marked, the SET CONTROL statement assigns display attributes to the function key line. Press Enter or PF5 to display the Control of Function Key Lines window.
Copy Contents of Page Buffer	If this field is marked, the SET CONTROL statement copies the current contents of the screen (contained in the internal page buffer) to the Natural source work area. These contents include all input data and are written to the next available location in the source work area.
Activate Error Processing	If the E=ON field is marked, the SET CONTROL statement activates error processing. If the E=OFF field is marked, the SET CONTROL statement deactivates error processing by *ERROR-TA (restores default error processing).
Lower/Upper Case Translation	If the L field is marked, the SET CONTROL statement translates upper case characters to lower case for alphanumeric input data. If the U field is marked, the SET CONTROL statement translates lower case characters to upper case.
Display Stored Screens	If this field is marked, the SET CONTROL statement suspends normal processing and displays a list of all screens written to the Natural system file. Select screens for display from the list. Each screen is presented in display-only format; all input fields are protected.
Forms/Screen Mode	If this field is marked, the SET CONTROL statement activates forms/screen mode.
Non-Conversational Control	If this field is marked, the SET CONTROL statement indicates that the next output screen will be sent to the terminal without waiting for user response.
Delete First Entry from Stack	If this field is marked, the SET CONTROL statement deletes the first entry on the Natural stack.

Field	Description (continued)
Suppress Map Printing in Batch	If this field is marked, the SET CONTROL statement suppresses the output of maps or screens produced by INPUT statements in batch mode.
Note:	This SET CONTROL statement causes the next INPUT statement not to be processed.
Clear Data Buffer	If this field is marked, the SET CONTROL statement clears the contents of the Natural data buffer.
Read Stack without Deleting	If this field is marked, the SET CONTROL statement non-destructively reads an entry from the Natural stack.
Clear Source Area	If this field is marked, the SET CONTROL statement clears the contents of the Natural source area.
Interrupt Current Operation	If this field is marked, the SET CONTROL statement terminates the active program and returns Natural to command input mode. The statement does not delete the Natural stack.

Copy Data to Stack or *COM Window

Mark the Copy Data to Stack or *COM field and press PF5 (optns) to display the following window:

```
        Copy Data to Stack or *COM

    Source line of screen data to copy
      Specific line ................ _
        Line number ................ ___
      Cursor position line ......... _

    Source Field Specification
      Entire line .................. _
      Intensified fields only ...... _
      Colored fields only .......... _
        Color ...................... __
      Position cursor at field ..... _
      Position cursor at word ...... _

    Target
      Natural data stack ........... _
      *COM system variable ......... _
```

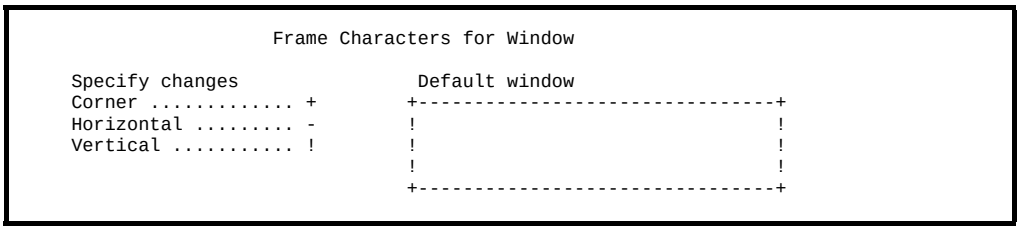
Copy Data to Stack or *COM Window

The fields in this window are:

Field	Description
Source line of screen data to copy	Either mark the Specific line field and specify a line number (from 1 to 24), or copy a line on which the cursor is positioned by marking the Cursor position line field.
Source Field Specification	Copy options for the source field specifications. Mark one of the following options:
Entire line	Copy all protected data from the line.
Intensified fields only	Copy only intensified protected fields.
Colored fields only	Copy protected fields with a specific color. Specify a color in the Color field.
Position cursor at field	Copy the field on which the cursor is positioned.
Position cursor at word	Copy the word (as delimited by blanks or special characters within the field) on which the cursor is positioned.
Target	Target for the copied data. One of the two options (Natural Data Stack or *COM system variable) must be marked.

Frame Characters for Window Window

Mark the Frame Characters for Window field and press PF5 (optns) to display the following window:



Frame Characters for Window Window

Use this window to change the corner, horizontal, and vertical characters that define a default window. The window displays any new characters automatically when you press Enter.

Simulate PF/PA-KEY Window

Mark the Simulate PF/PA-KEY field and press PF5 (optns) to display the following window:

```
                          Simulate PF\PA-KEY

PF1 _  PF4 _  PF7 _  PF10 _  PF13 _  PF16 _  PF19 _  PF22 _  PA1 _  ENTR _
PF2 _  PF5 _  PF8 _  PF11 _  PF14 _  PF17 _  PF20 _  PF23 _  PA2 _
PF3 _  PF6 _  PF9 _  PF12 _  PF15 _  PF18 _  PF21 _  PF24 _  PA3 _
```

Simulate PF/PA-KEY Window

Use this window to identify the key to simulate; mark the key and press Enter.

Message Line Control Window

Mark the Message Line Control field and press PF5 (optns) to display the following window:

```
        Message Line Control

Position
  Top ..... _
  Bottom ... _
  Toggle ... _

Protect-Unprotect
  Toggle ... _

Colour
  BL _ GR _ NE _ PI _ RE _ TU _ YE _
```

Message Line Control Window

The fields in this window are:

Field		Description
Position		Position of the message line. Mark one of the following options:
	Top	Top of screen
	Bottom	Bottom of screen
	Toggle	Switch from top to bottom or vice versa

Field	Description (continued)
Protect-Unprotect	Protection option for the message line.
Toggle	To change from an unprotected message line to a protected one or vice versa, mark this field.
Colour	Colour of the message line. For terminals that support colour, you can assign a colour to the message line by marking one of the following colours: • BL (blue) • GR (green) • NE (neutral/white) • PI (pink) • RE (red) • TU (turquoise) • YE (yellow)

Window Processing Window

Mark the Window Processing field and press PF5 (optns) to display the following window:

```
                          Window Processing

Window size and position on physical screen
  Set to physical screen size .. _
  Top left corner of window .... At position (Line\Column) .... ___ / ___
                                 Top to bottom or vice-versa .. _
                                 At cursor position ........... _
  Bottom right corner at cursor .......... _
  Window line size ....................... ___
  Window height .......................... ___
  Automatic framing .............  ON _  OFF _
Window position on logical page
  Shift top left corner to cursor position _
  Shift left     Line size ...... _  Leftmost ....... _  Positions .. ___
  Shift right    Line size ...... _  Rightmost ...... _  Positions .. ___
  Shift down     Window height .. _  Bottom of page . _  Lines ...... ___
  Shift up       Window height .. _  Top of Page .... _  Lines ...... ___
  Set STAY .......  ON _  OFF _
```

Window Processing Window

The fields in this window are:

Field	Description
Set to physical screen size	If this field is marked, the window size is set to the physical size of the screen.
Top left corner of window	Position of the window in relation to the physical screen. The window size remains the same or is reduced, if necessary. Specify one of the following options: • At position (Line\Column) • Top to bottom or vice-versa • At cursor position
Bottom right corner at cursor	If this field is marked, the bottom right corner of the window is positioned at the cursor position. The top left corner of the window remains unchanged.
Window line size	Line size (horizontal span) of the window. The default is the maximum possible.
Window height	Column size (vertical span) of the window. The default is the maximum possible.
Automatic framing	If ON is marked, the window has a frame. If OFF is marked, the window is displayed without a frame.
Shift top left corner to cursor position	If this field is marked, the top left corner of the window is moved to the cursor position.
Shift left	If one of the fields on this line is marked, the window is moved to the left. • If you mark Line size, the window is moved the number of positions equal to the line size of the window. • If you mark Leftmost, the window is moved to the leftmost position on the screen. • To move the window a certain number of positions to the left, specify the number in the Positions field.
Shift right	If one of the fields on this line is marked, the window is moved to the right. • If you mark Line size, the window is moved the number of positions equal to the line size of the window. • If you mark Rightmost, the window is moved to the right most position on the screen. • To move the window a certain number of positions to the right, specify the number in the Positions field.

Field	Description (continued)
Shift down	If one of the fields on this line is marked, the window is moved farther down on the screen. • If you mark Window height, the window is moved the number of positions equal to the number of lines in the window. • If you mark Bottom of page, the window is moved to the bottom of the page. • To move the window down a certain number of lines, specify the number in the Lines field.
Shift up	If one of the fields on this line is marked, the window is moved farther up on the screen. • If you mark Window height, the window is moved the number of positions equal to the number of lines in the window. • If you mark Top of page, the window is moved to the top of the page. • To move the window up a certain number of lines, specify the number in the Lines field.
Set STAY	If the ON field is marked, control stays on the current page until the end of the page. If a page has not been completely shown, the window scrolls down each time the Enter key is pressed until the end of the page is displayed. The next time the Enter key is pressed, control is returned to the program.
Note:	This does not apply to pages created by an INPUT statement.

Control of Function Key Lines Window

Mark the Control of Function Key Lines field and press Enter or PF5 (optns) to display the following window:

```
          Control of Function Key Lines

   Position
     Top ...... _ Bottom ... _ On line... ___
   Function keys display
     Keys 1-12 .......... _
     Keys 13-24 ......... _
     Toggle ............. _
   Display format
     Sequential ... _      PC-like ...... _
   Function key attributes
     Single Line ........ _
     Double Line ........ _
     Intensify .......... _
     Reverse video ...... _
     Cursor sensitive ... _
     PF-Key Line Colour . _
```

Control of Function Key Lines Window

The fields in this window are:

Field	Description
Position	To position the function key lines in the window, mark one of the following:
Top	Display the function key line at the top of the screen.
Bottom	Display the function key line at the bottom of the screen.
On line	Display the function key line on the specified line.
Function keys display	To display function keys 1 to 12, 13 to 24, or switch back and forth between the two, mark one of the following:
Keys 1-12	Display function keys 1 to 12.
Keys 13-24	Display function keys 13 to 24.
Toggle	Toggle between the two options.
Display format	To display only function keys that have names assigned or the function key line in PC standard, mark one of the following:
Sequential	Display only the function keys that have names assigned.
PC-like	Display the function key line in PC standard format.

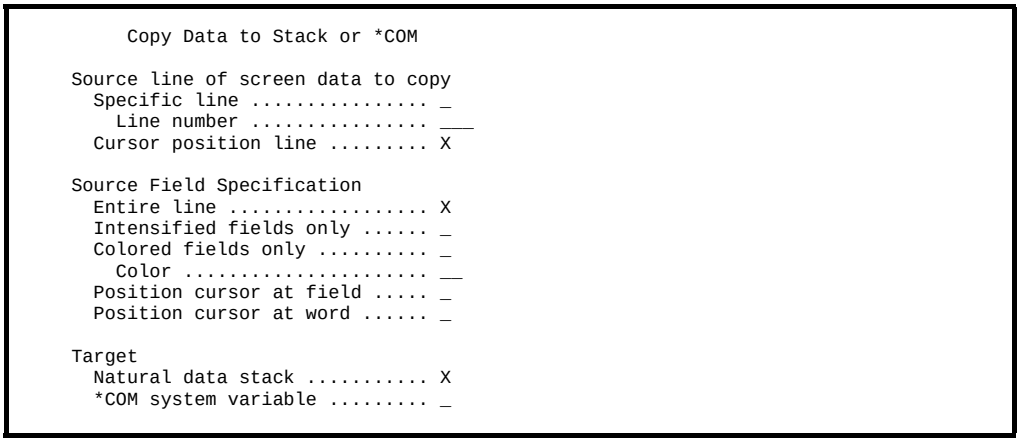
Field	Description (continued)
Function key attributes	
Single Line	Single line display for function keys.
Double Line	Double-line display for function keys.
Intensify	Intensify the keys.
Reverse video	Display the keys in reverse video.
Cursor sensitive	Allows users to activate the function by moving the cursor to the function key and pressing Enter.
PF-Key Line Color	To specify colors for the function key number and name lines, mark this field and press PF5 (optns). A window is displayed in which you can mark your color choices.

Set-Control Statement Model Example

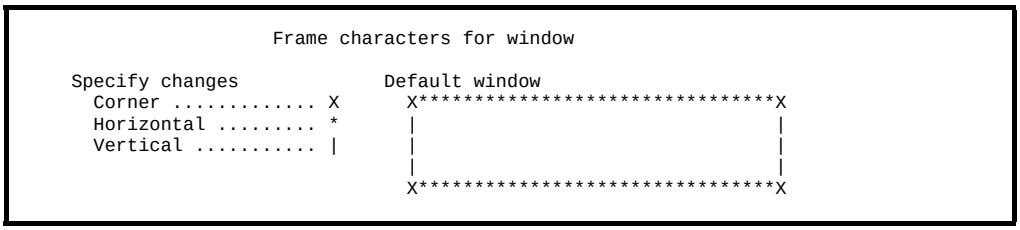
The following specifications generate SET CONTROL statements:

```
                    SET-CONTROL Statement

Copy Data to Stack or *COM ... SET CONTROL X CS CC
Frame Characters for Window .. SET CONTROL X F=chv
Simulate PF\PA-KEY ........... SET CONTROL X Knn KPn
Message Line Control ......... SET CONTROL _ M
Window Processing ............ SET CONTROL X W
Control of PF-KEY Lines ...... SET CONTROL _ Y
Copy Contents of Page Buffer . SET CONTROL _ C
Activate Error Processing .... SET CONTROL X E=ON  _ E=OFF
Lower\Upper Case Translation . SET CONTROL _ L  X U
Display Stored Screens ....... SET CONTROL _ E
Forms\Screen Mode ............ SET CONTROL _ F
Non-Conversational Control ... SET CONTROL _ N
Delete First Entry from Stack  SET CONTROL _ .P
Suppress Map Printing in Batch SET CONTROL _ Q
Clear Data Buffer ............ SET CONTROL _ R
Read Stack without Deleting .. SET CONTROL _ .S
Clear Source Area ............ SET CONTROL X Z
Interrupt Current Operation .. SET CONTROL _ .
```

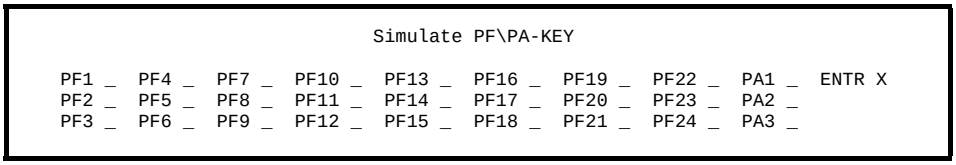
Set-Control Statement Model Example: SET-CONTROL Statement Window



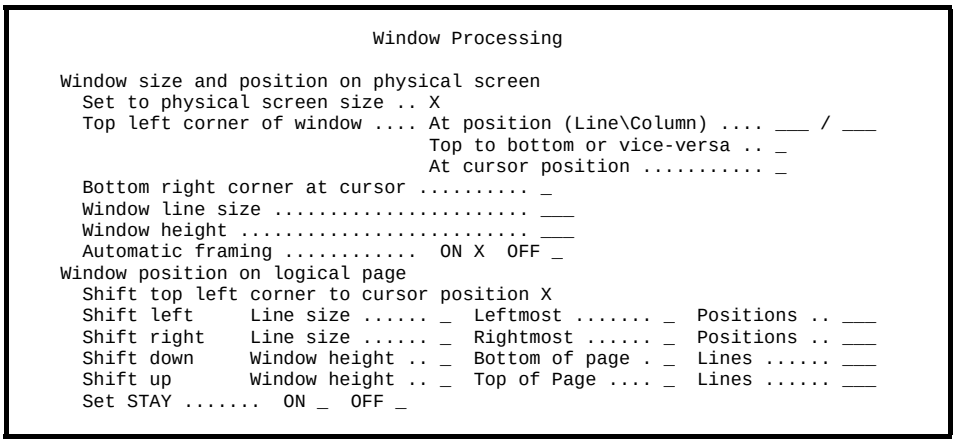
Set-Control Statement Model Example: Copy Data to Stack or *COM Window



Set-Control Statement Model Example: Frame Characters for Window Window



Set-Control Statement Model Example: Simulate PF/PA-KEY Window



Set-Control Statement Model Example: Window Processing Window

Example of code generated from the sample specifications

```
0010 **
0020 **
0030 **
0040 SET CONTROL 'E=ON' /* Activate error processing
0050 SET CONTROL 'U' /* Enable lower-upper transl.
0060 SET CONTROL 'F=X*|' /* Frame character for window
0070 SET CONTROL 'CSCA' /* Copy data to stack/*COM
0080 SET CONTROL 'K0' /* Simulate function keys
0090 SET CONTROL 'WB' /* WINDOW  = SCREEN
0100 SET CONTROL 'W*' /* Logical page positioning
0110 SET CONTROL 'WF'  /* Logical page positioning
0120 SET CONTROL 'Z' /* Clear source area
```

Set-Key Statement Model

The Set-Key statement model (SK) generates a SET KEY statement that assigns functions to video terminal program attention keys (PA*n*), program function keys (PF-keys), and Clear keys. When a SET KEY statement is issued, Natural receives control of the keys and uses the values assigned to the keys. During program execution, the SET KEY statement can:

- Make the keys program-sensitive (available for interrogation by the active program). If a PF-key is program-sensitive, all data entered on the screen is transferred to the program when the user presses the PF-key. Pressing the key has the same effect as pressing Enter.
- Assign a program or command name to a key. When the key is pressed, the current program is interrupted and the assigned program or command (via the Natural stack facility) is invoked. Terminal commands can also be assigned to a key.

Generating the Default Code into the Editor

To generate a SET KEY statement, enter one of the following line commands:

```
.g(SET-KEY,)
```

or

```
.g(SK,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 SET KEY ALL
0040 SET KEY ON
0050 SET KEY OFF
0060 SET KEY COMMAND ON
0070 SET KEY COMMAND OFF
0080 SET KEY NAMED OFF
0090 SET KEY PF1 = HELP NAMED 'help'
0100          PF2 NAMED 'retrn'
0110          PF3 NAMED 'quit'
0120          PF4 NAMED 'test'
0130          PF7 NAMED 'bkwrd'
0140          PF8 NAMED 'frwrd'
0150          PF10 NAMED 'left'
0160          PF11 NAMED 'right'
0170          PF12 NAMED 'main'
```

Revise the code to complete the SET KEY statement.

Using the SET-KEY Statement Window

The SET-KEY Statement window is displayed when you invoke the Set-Key statement model from the Generation main menu. You can also invoke the SET-KEY Statement window from an editor by entering one of the following line commands:

```
.g(SET-KEY)
```

or

```
.g(SK)
```

The SET-KEY Statement window is displayed:

```
                          SET-KEY Statement

SET KEY  ALL _  ON _ OFF _  COMMAND ON _ OFF _  NAMED OFF _
                   COMMAND
  SET KEY   1 ON OFF ON OFF  2 PGM OPERAND    DATA        NAMED        OFF
   >> _1 PF1   _  _   _  _      _  __________ __________ _____________ _
         PF2   _  _   _  _      _  __________ __________ _____________ _
         PF3   _  _   _  _      _  __________ __________ _____________ _
         PF4   _  _   _  _      _  __________ __________ _____________ _
         PF5   _  _   _  _      _  __________ __________ _____________ _
         PF6   _  _   _  _      _  __________ __________ _____________ _
         PF7   _  _   _  _      _  __________ __________ _____________ _
         PF8   _  _   _  _      _  __________ __________ _____________ _
         PF9   _  _   _  _      _  __________ __________ _____________ _
         PF10  _  _   _  _      _  __________ __________ _____________ _
         PF11  _  _   _  _      _  __________ __________ _____________ _
   >>    PF12  _  _   _  _      _  __________ __________ _____________ _
   DYNAMIC     _  _   _  _      _  __________ __________ _____________ _
      VARIABLE ________________________________
   SET KEY ENTR  NAMED ________________ OFF _
```

SET-KEY Statement Window

The fields in this window are:

Field	Description
SET KEY ALL	Mark this field to set options for all the PF-keys. To set options for individual keys, use the fields below this line.
ON	If this field is marked, either the keys are made program-sensitive or the previous functions assigned to the keys are re-activated.
OFF	If this field is marked, the keys are de-activated and control is returned to the TP monitor.
COMMAND ON	If this field is marked, the functions assigned to the keys are re-activated.

Field	Description (continued)
OFF	If this field is marked, the functions assigned to the keys are temporarily de-activated. If the keys were program-sensitive before the functions were assigned, for example, they are program-sensitive again.
NAMED OFF	To delete the names assigned to keys, mark this field.
>> _1	By default, PF-keys 1-7 are displayed in the SET KEY window area. To scroll through the rest of the PF-keys, press PF8 (frwrd) or PF7 (bkwrd). You can also type a PF-key number in the >> field. The PA1, PA2, PA3, and CLR (Clear) keys occur after PF24.
ON	If this field is marked, either the key is made program-sensitive or the previous function assigned to the key is re-activated.
OFF	If this field is marked, the key is de-activated and control is returned to the TP monitor.
COMMAND ON	If this field is marked, the function assigned to the key is re-activated.
COMMAND OFF	If this field is marked, the function assigned to the key is temporarily de-activated. If the key was program-sensitive before the function was assigned, for example, it is program-sensitive again.
PGM	If this field is marked, the key is program-sensitive.
OPERAND	Name of a command or program that is assigned to the key. When the key is pressed, the current program is terminated and the operand assigned to the key is invoked via the Natural stack. You can also pass parameters to the command or program. You can also assign a terminal command to a key. When the key is pressed, that terminal command is executed. **Note:** If the command or program is a constant, it must be enclosed within single quotes.
DATA	Data string assigned to the key. When the key is pressed, the data string is placed in the input field on which the cursor is currently positioned and the data is transferred to the executing program (as if Enter were pressed). **Note:** If the data string is a constant, it must be enclosed within single quotes.

Field	Description (continued)
NAMED	Name assigned to a key. The name is displayed in the PF-key lines on the screen. This feature allows users to easily identify the functions assigned to the keys. The maximum length of a key name is 10 characters. In normal tabular PF-key line format (%YN), only the first five characters are displayed.
Note:	If the name is specified as a constant, it must be enclosed within single quotes.
OFF	To delete the name assigned to a key, mark this field.
DYNAMIC	Instead of assigning a specific key with the SET KEY statement, you can use the Dynamic option with a variable and assign a value (PF*n*, PA*n*, CLR) to the variable in the program. In this way, you can specify a function and make it dependent on the program logic. As with the PF-keys, you can assign attributes to the Dynamic key by marking or entering data in the corresponding fields.
Note:	If you use this option, you must specify the variable name in the VARIABLE field.
VARIABLE	Name of the variable used for the Dynamic option.
SET KEY ENTR NAMED	Name assigned to the Enter key.
OFF	If you want to delete a name that was previously assigned to the Enter key, mark this field.

Specifying Default Functions for PF-Keys

To specify default functions for PF-keys, press PF5 (deflt) in the SET-KEY Statement window. The Default Parameters window is displayed:

```
          Default Parameters

   SET KEY PF1  _ NAMED help
           PF2  _ NAMED retrn
           PF3  _ NAMED quit
           PF4  _ NAMED test

           PF7  _ = %W- _ NAMED bkwrd
           PF8  _ = %W+ _ NAMED frwrd

           PF10 _ = %W< _ NAMED left
           PF11 _ = %W> _ NAMED right
           PF12 _ NAMED main

   SET KEY PF(13:24) = %K(1:12) _

   SET KEY PA1 = %K __
           PA2 = %K __
           PA3 = %K __
           CLR = %K __
```

Default Parameters Window

The fields in this window are:

Field	Description
PF1	To support a help key (PF1), mark this field.
PF2	To support a return key (PF2), mark this field.
PF3	To support a quit key (PF3), mark this field.
PF4	To support a test key (PF4), mark this field.
PF7	Mark one of the following:
=%W-	Use the window command for backward scrolling.
NAMED bkwrd	Support a backward scrolling key.
PF8	Mark one of the following:
= %W+	Use the window command for forward scrolling.
NAMED frwrd	Support a forward scrolling key.

Field	Description (continued)
PF10	Mark one of the following:
= %W<	Use the window command for left scrolling.
NAMED left	Support a left scrolling key.
PF11	Mark one of the following:
= %W>	Use the window command for right scrolling.
NAMED right	Support a right scrolling key (PF11).
PF12	To support a return-to-main key (PF12), mark this field.
PF(13:24) = %K(1:12)	To assign the values for PF-keys 1–12 to PF-keys 13–24, respectively, mark this field.
PA1	To default PA1 to a PF-key, specify a number from 1 to 24. Pressing PA1 is the same as pressing the PF-key with the corresponding number.
PA2	To default PA2 to a PF-key, specify a number from 1 to 24. Pressing PA2 is the same as pressing the PF-key with the corresponding number.
PA3	To default PA3 to a PF-key, specify a number from 1 to 24. Pressing PA3 is the same as pressing the PF-key with the corresponding number.
CLR	To default CLR to a PF-key, specify a number from 1 to 24. Pressing CLR is the same as pressing the PF-key with the corresponding number.

Set-Key Statement Model Example

The following specifications generate a SET KEY statement:

```
                          SET-KEY Statement

SET KEY  ALL _  ON _ OFF _  COMMAND ON _ OFF _  NAMED OFF _
                   COMMAND
  SET KEY   1 ON OFF ON OFF  2 PGM OPERAND    DATA        NAMED        OFF
   >> _1 PF1   X  _   _  _      _  __________ __________ _____________ _
         PF2   X  _   _  _      _  __________ __________ _____________ _
         PF3   _  _   X  _      _  __________ __________ _____________ _
         PF4   _  _   _  _      _  __________ __________ _____________ _
         PF5   _  _   _  _      _  __________ __________ _____________ _
         PF6   _  _   _  _      _  __________ __________ _____________ _
         PF7   _  _   _  _      _  __________ __________ _____________ _
         PF8   _  _   _  _      _  __________ __________ _____________ _
         PF9   _  _   _  _      _  __________ __________ _____________ _
         PF10  _  _   _  _      _  __________ __________ _____________ _
         PF11  _  _   _  _      _  __________ __________ _____________ _
   >>    PF12  _  _   _  _      _  __________ __________ _____________ _
   DYNAMIC     _  _   _  _      _  __________ __________ #DYNAMIC-NAM _
      VARIABLE #DYNAMIC-PFKEY____________________
   SET KEY ENTR  NAMED _______________ OFF _
```

Set-Key Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 SET KEY PF1 = ON
0040         PF2 = ON
0050         PF3 = COMMAND ON
0060 SET KEY DYNAMIC #DYNAMIC-PFKEY NAMED #DYNAMIC-NAM
```

Set-Window Statement Model

The Set-Window statement model (SW) generates a Natural SET WINDOW statement, which activates and deactivates a window.

Generating the Default Code into the Editor

To generate a SET WINDOW statement, enter one of the following line commands:

```
.g(SET-WINDOW,)
```

or

```
.g(SW,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 SET WINDOW 'window-name'
```

Revise the code to complete the SET WINDOW statement.

Using the SET-WINDOW Statement Window

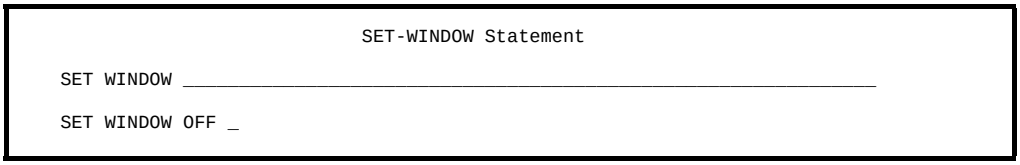
The SET-WINDOW Statement window is displayed when you invoke the Set-Window statement model from the Generation main menu. You can also invoke the SET-WINDOW Statement window from an editor by entering one of the following line commands:

```
.g(SET-WINDOW)
```

or

```
.g(SW)
```

The SET-WINDOW Statement window is displayed:



SET-WINDOW Statement Window

The fields in this window are:

Field	Description
SET WINDOW	Name of the window that is activated.
SET WINDOW OFF	If this field is marked, the currently active window is deactivated.

Set-Window Statement Model Example

The following specifications generate a SET WINDOW statement:

```
                          SET-WINDOW Statement

SET WINDOW #DEMO-WINDOW_____________________________________________

SET WINDOW OFF _
```

Set-Window Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 SET WINDOW '#DEMO-WINDOW'
```

Stack Statement Model

The Stack statement model (SA) generates a Natural STACK statement, which places one of the following into the Natural stack:

- The name of a Natural program to be executed
- The name of a Natural system command
- Data used during the execution of an INPUT statement

Generating the Default Code into the Editor

To generate a STACK statement, enter one of the following line commands:

```
.g(STACK,)
```

or

```
.g(SA,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 STACK TOP DATA 'data'
```

Revise the code to complete the STACK statement.

Using the STACK Statement Window

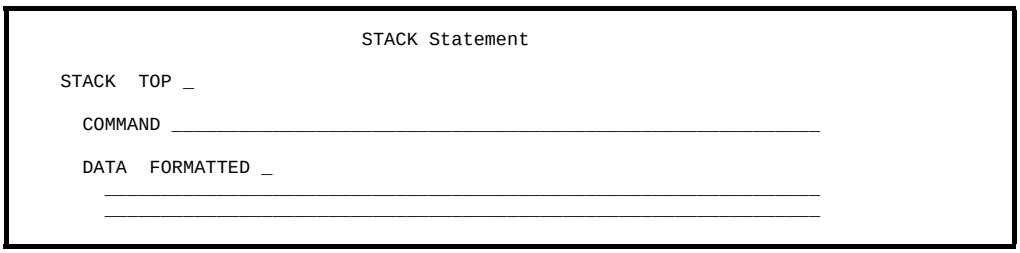
The STACK Statement window is displayed when you invoke the Stack statement model from the Generation main menu. You can also invoke the STACK Statement window from an editor by entering one of the following line commands:

```
.g(STACK)
```

or

```
.g(SA)
```

The STACK Statement window is displayed:



STACK Statement Window

The fields in this window are:

Field	Description
STACK TOP	If this field is marked, data is placed at the top of the Natural stack. If this field is blank, data is placed at the bottom of the stack. This is an optional field.
COMMAND	Name of a command to be executed. Rather than using the command line, you can specify a command in this field. For example, the STACK TOP COMMAND ‘RUN’ command places the RUN command at the top of the stack and executes the command at the point when the command line value is normally executed. To execute a command programmatically, rather than using the direct command line, generate a stack statement that stacks the command.
DATA FORMATTED	If this field is marked, all variables are passed to the next INPUT statement on a field basis. No key assignments or delimiters are interpreted. This is an optional field. If you mark this field, enter the data for the INPUT statement in the fields below.

Stack Statement Model Example

The following specifications generate a STACK statement:

```
                          STACK Statement

STACK  TOP X

  COMMAND 'L F NC#'_____________________________________________

  DATA  FORMATTED _
    ___________________________________________________________
    ___________________________________________________________
```

Stack Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 STACK TOP COMMAND 'L F NC#'
```

Store Statement Model

The Store statement model (SE) generates a Natural STORE statement, which adds a record to the database.

Generating the Default Code into the Editor

To generate a STORE statement, enter one of the following line commands:

```
.g(STORE,)
```

or

```
.g(SE,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 STORE view-name
```

Revise the code to complete the STORE statement.

Using the STORE Statement Window

The STORE Statement window is displayed when you invoke the Store statement model from the Generation main menu. You can also invoke the STORE Statement window from an editor by entering one of the following line commands:

```
.g(STORE)
```

or

```
.g(SE)
```

The STORE Statement window is displayed:

```
                          STORE Statement

STORE RECORD
  IN FILE _____________________________ *
    LABEL _____________________________

  PASSWORD ____________________________________________________________

  CIPHER   ____________________________________________________________

  USING NUMBER ________
```

STORE Statement Window

The fields in this window are:

Field	Description
STORE RECORD	
IN FILE	Name of a view as defined in a global data area (GDA) or a local data area (LDA). This is a required field.
LABEL	Label or reference name for the STORE statement. This is an optional field.
PASSWORD	Password for the file. To store data in an Adabas file that is password-protected, specify the password. This is an optional field.
CIPHER	Cipher key for the file. To store data in an enciphered Adabas file, specify the cipher key. This is an optional field.
USING NUMBER	ISN number for the record. To store a record with the supplied Adabas ISN, specify the number (from 1 to 999999999). If a record with that ISN already exists, an error message is displayed and the program is terminated (unless ON ERROR processing is specified). This is an optional field.

Store Statement Model Example

The following specifications generate a STORE statement:

```
                              STORE Statement

  STORE RECORD
    IN FILE NCST-EMPLOYEES_____________________ *
      LABEL ___________________________________

    PASSWORD ____________________________________________________________

    CIPHER   ____________________________________________________________

    USING NUMBER ________
```

Store Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 STORE NCST-EMPLOYEES
```

Substring Statement Model

The Substring statement model (SS) generates a statement that moves a specified portion (substring) of an alphanumeric field. Without the substring option, the whole content of a field is moved. Use this model if you want to:

- Use a substring
- Move a field to a substring
- Move a substring to a field

Using the Move by Substring Option Window

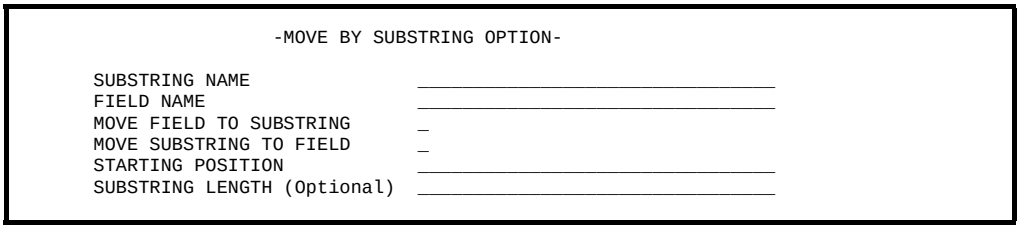
The Move by Substring Option window is displayed when you invoke the Substring statement model from the Generation main menu. You can also invoke the Substring Option window from an editor by entering one of the following line commands:

```
.g(SUBSTRING)
```

or

```
.g(SS)
```

The Substring Option window is displayed:



Substring Option Window

For more information about the parameters for this statement, refer to the MOVE statement in the Natural documentation.

Subtract Statement Model

The Subtract statement model (SB) generates a Natural SUBTRACT statement, which subtracts the values of two operands. For example, the SUBTRACT statement can subtract the value(s) of operand1 from the value of operand2 (operand2 cannot be a constant).

Generating the Default Code into the Editor

To generate a SUBTRACT statement, enter one of the following line commands:

```
.g(SUBTRACT,)
```

or

```
.g(SB,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 SUBTRACT amounts
0040     FROM #variable
```

Revise the code to complete the SUBTRACT statement.

Using the SUBTRACT Statement Window

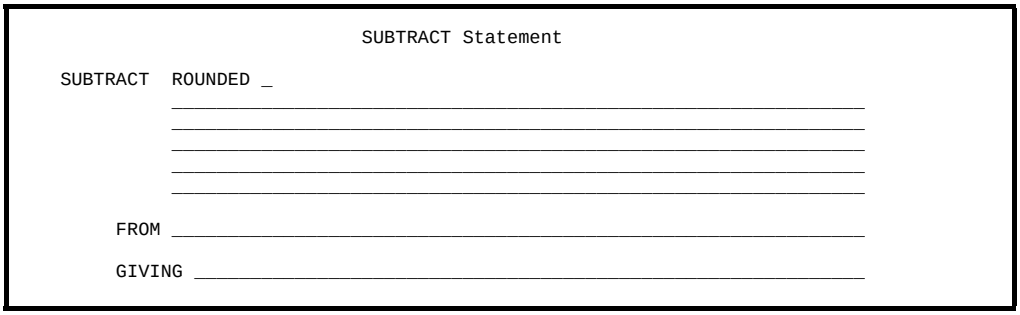
The SUBTRACT Statement window is displayed when you invoke the Subtract statement model from the Generation main menu. You can also invoke the SUBTRACT Statement window from an editor by entering one of the following line commands:

```
.g(SUBTRACT)
```

or

```
.g(SB)
```

The SUBTRACT Statement window is displayed:



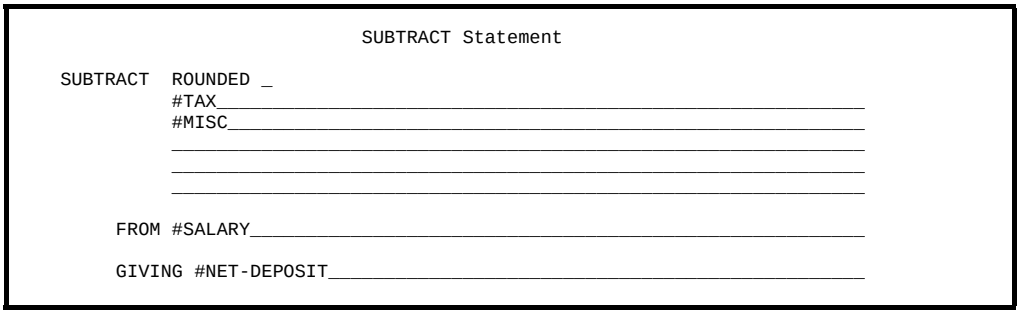
SUBTRACT Statement Window

The fields in this window are:

Field	Description
SUBTRACT ROUNDED	If this field is marked, the result of the SUBTRACT operation is rounded off. For example, if 43.6 is the result, it becomes 44. This is an optional field.
	On the lines provided, enter the value(s) you want to subtract. More than one value may be entered. This is a required field.
FROM	Value to subtract from. The value cannot be a constant. This is a required field.
GIVING	If you specify a value in this field, the result of the operation is stored in that value. If no value is specified, the result is stored in the value in the FROM field. This is an optional field.

SUBTRACT Statement Model Example

The following specifications generate a SUBTRACT statement:



Subtract Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 SUBTRACT #TAX
0040          #MISC
0050     FROM #SALARY
0060   GIVING #NET-DEPOSIT
```

User-Exit Statement Model

The User-Exit statement model (UE) generates user exits. The model is invoked from the User Exit editor and uses text members generated by the user exit models. For information about user exit models, see **Object-Browse Models**, page 293.

Generating Code into the User Exit Editor

➢ To generate user exit code using the User-Exit statement model:

1 Invoke the User Exit editor from the Generation main menu.

2 Enter the following line command:

`.g(User-Exit,`*`user exit model name`*`,`*`text member`*`,`*`text member`*`,...)`

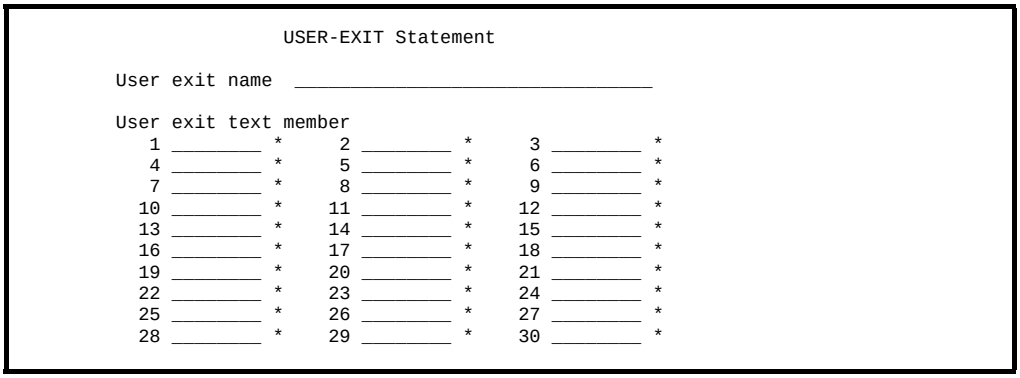
where:

- *user exit model name* is the name of the model used to generate the user exit text members
- *text member* is the name of the user exit text member containing layout specifications (previously generated by one of the user exit models)

  The number of text members you can specify varies depending on the target model. Although the user exit models were designed for the Object-Browse-Dialog model, you can also use the models with other compatible models (you may need to override the user exit names).

  If you do not specify the name of a text member, the User-Exit Statement window is displayed. For a description of this window, see the following section.

User-Exit Statement Window



User-Exit Statement Window

The Natural Construct demo system contains examples of using the User-Exit statement model and several user exit text members. To view the sample code, refer to the NCPRDOBD module. The examples were generated using the User-Exit statement model and the following text members:

- NCPRDEX (generated using the Export-Data-Fields model)
- NCPRDIN (generated using the Input-Key model)
- NCPRDRP (generated using the Report-Data-Fields model)
- NCPRDWF1 and NCPRDWF2 (generated using the Write-Data-Fields model)

View Statement Model

The View statement model (VW) generates the definition of a view, as derived from Predict.

Generating the Default Code into the Editor

If you know the name of the view, enter the following line command:

```
.g(VIEW,viewname)
```

For example:

```
.g(VIEW,NCST_EMPLOYEES)
```

To select a Predict view from a list of available options, enter one of the following line commands:

```
.g(VIEW,)
```

or

```
.g(VW,)
```

The Predict view window is displayed to select a file. For a description of this window, see page 405.

After selecting a file and pressing Enter, the view is generated into the editor.

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 01 NCST-EMPLOYEES VIEW OF NCST-EMPLOYEES
0040   02 PERSONNEL-ID
0050   02 FULL-NAME
0060     03 FIRST-NAME
0070     03 NAME
0080   02 MAR-STAT
0090   02 SEX
0100   02 BIRTH
0110   02 FULL-ADDRESS
0120     03 C*ADDRESS-LINE
0130     03 ADDRESS-LINE(1:5)
0140     03 CITY
0150     03 COUNTRY
0160   02 TELEPHONE
0170     03 AREA-CODE
0180     03 PHONE
0190   02 DEPT
0200   02 REDEFINE DEPT
```

Revise the code as desired.

Using the VIEW Function Window

The VIEW Function window is displayed when you invoke the View statement model from the Generation main menu. You can also invoke the VIEW Function window from an editor by entering one of the following line commands:

```
.g(VIEW)
```

or

```
.g(VW)
```

The VIEW Function window is displayed:

```
                        VIEW Function

  Predict view name ........ _____________________________ *
  Program view name ........ _____________________________

    C* variables ................... X
    Redefined components ........... X
    Superdescriptor fields ......... _
    Natural format specifications .. _
```

VIEW Function Window

The fields in this window are:

Field	Description
Predict view name	Name of the Predict file from which the view will be taken. This is a required field.
Program view name	Name of the view (Natural variable names apply). If you do not specify a name in this field, the name entered in the Predict view name field will be used as the program view name. This is an optional field.
C* variables	By default, this field is marked and the counter fields for periodic groups (PC) and multiple valued (MC) fields are included in the generated view. If you want to suppress this option and not generate any counter fields, specify a blank in this field.
Redefined components	By default, this field is marked and the redefined components for a field are included in the generated view. If you want to suppress this option, specify a blank in this field.
Superdescriptor fields	Mark this field to include superdescriptor or subdescriptor fields in the generated view.
Natural format specifications	Mark this field to generate a structure with fields that match those of the specified DDM.
Note:	If you mark the Natural format specifications field and enter a view name in the Program view name field, the structure name will be the same as the Program view name. If you do not specify a program view name, the structure name will be a DDM name.
Note:	If you mark the Include Natural format specifications field and the C* variables field, the C* variables option is disregarded.

VIEW Statement Model Example

The following specifications generate a VIEW statement:

```
                        VIEW Function

  Predict view name ........ NCST-EMPLOYEES_____________________ *
  Program view name ........ EMP-VIEW_____________________________

    C* variables ................... X
    Redefined components ........... X
    Superdescriptor fields ......... _
    Natural format specifications .. _
```

View Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 01 EMP-VIEW VIEW OF NCST-EMPLOYEES
0040   02 PERSONNEL-ID
0050   02 FULL-NAME
0060     03 FIRST-NAME
0070     03 NAME
0080   02 MAR-STAT
0090   02 SEX
0100   02 BIRTH
0110   02 FULL-ADDRESS
0120     03 C*ADDRESS-LINE
0130     03 ADDRESS-LINE(1:5)
0140     03 CITY
0150     03 COUNTRY
```

Write-Work-File Statement Model

The Write-Work-File statement model (WF) generates a Natural WRITE WORK FILE statement that writes records to a physical sequential work file.

Generating the Default Code into the Editor

To generate a WRITE-WORK-FILE statement, enter one of the following line commands:

```
.g(WRITE-WORK-FILE,)
```

or

```
.g(WF,)
```

The following default code is generated:

```
0010 /*
0020 /*
0030 WRITE WORK FILE 18 VARIABLE
0040       field-names
```

Revise the code to complete the WRITE-WORK-FILE statement.

Using the WRITE-WORK-FILE Statement Window

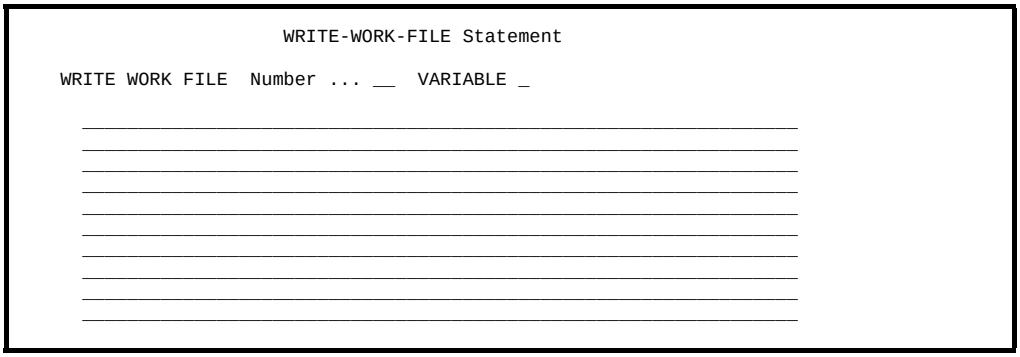
The WRITE-WORK-FILE Statement window is displayed when you invoke the Write-Work-File statement model from the Generation main menu. You can also invoke the WRITE-WORK-FILE Statement window from an editor by entering one of the following line commands:

```
.g(WRITE-WORK-FILE)
```

or

```
.g(WF)
```

The WRITE-WORK-FILE Statement window is displayed:



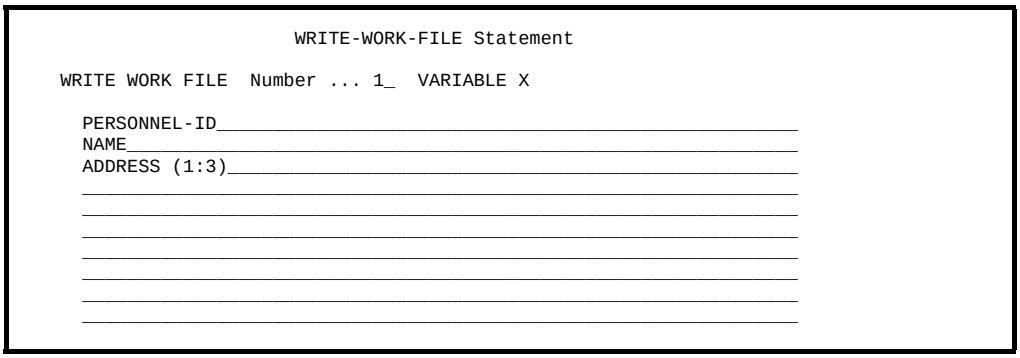
WRITE-WORK-FILE Statement Window

The fields in this window are:

Field	Description
WRITE WORK FILE Number	Number of the work file (as defined in Natural). This is a required field.
VARIABLE	To allow records containing different fields to be written to the same work file (with different WRITE WORK FILE statements), mark this field. This option must be specified in all WRITE WORK FILE statements. The records in the external file will be written in variable format. This is an optional field. Enter the field names in the fields below. Database fields, user-defined variables, and/or fields read from another work file using the READ WORK FILE statement are all valid fields.

Write-Work-File Statement Model Example

The following specifications generate a WRITE WORK FILE statement:



Write-Work-File Statement Model Example

Example of code generated from the sample specifications

```
0010 /*
0020 /*
0030 WRITE WORK FILE 01 VARIABLE
0040       PERSONNEL-ID
0050       NAME
0060       ADDRESS (1:3)
```

JCL MODELS (MAINFRAME)

This chapter describes the two models that generate JCL (Job Control Language). The JCL models, JCL-DOS-NATBATCH and JCL-OS-NATBATCH, generate a JCL member that can be submitted to execute Natural in batch mode.

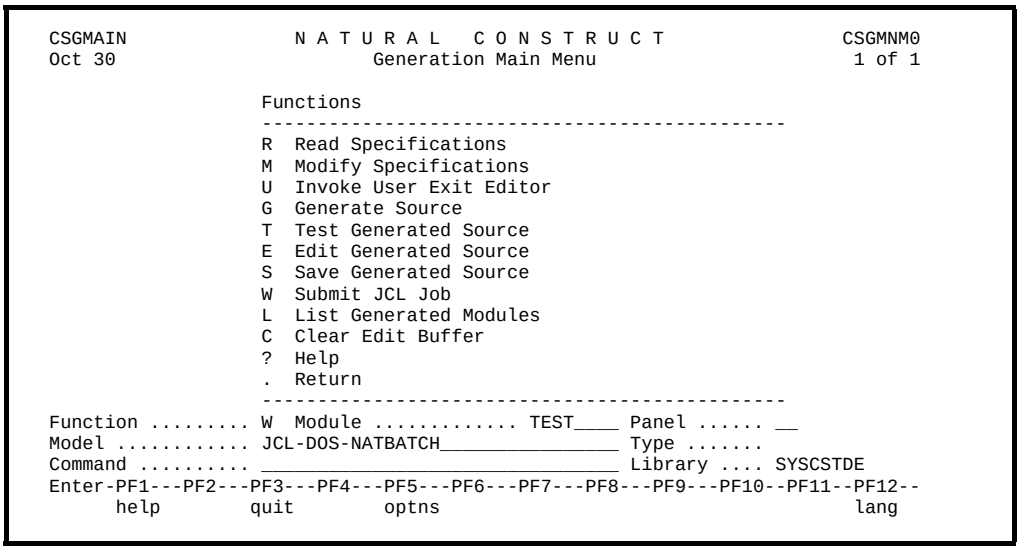
The following topics are covered:

Introduction

The JCL-DOS-NATBATCH model generates jobs to be run under the DOS operating system and the JCL-OS-NATBATCH model generates jobs to be run under a z/OS operating system.

Submit JCL Job Function

The Generation main menu supports the Submit JCL Job function, which is displayed in place of the Stow Generated Source function after you generate the JCL:



Submit JCL Job Function on the Generation Main Menu

Submit the job by entering “W” in the Function field. The source buffer must contain the generated JCL member.

CSUSUB Command

If you are not currently at the Generation main menu, you can use the CSUSUB command to submit a job from the Next prompt or from the editor command line. As with the Submit JCL Job function, the source buffer must contain the JCL member. The JCL member can either be generated by Natural Construct or typed manually by the user.

Both the Submit function and the CSUSUB command call the NATRJE program to route the JCL to the internal reader. To run a job using either method, NATRJE must be available at your installation.

Required Setup for the JCL Models

Before using the JCL models, your Natural Construct administrator must provide site-dependent information. This procedure is described in the following sections.

Setting up the JCL-DOS-NATBATCH Model

The site-dependent information needed to generate DOS JCL with the JCL-DOS-NATBATCH model is stored in the CUNDGLDA local data area (LDA). To generate DOS JCL that fits your installation, define the following in CUNDGLDA:

- The name of the Natural batch module invoked in the EXEC statement. In CUNDGLDA, the #NATBATCH constant must be initialized to the name of the Natural batch module at your installation.
- For Natural work files, the assignment of logical units for each work file. To do this, provide values for the #LOGICAL-FILE(1:32) array in CUNDGLDA. For example, if you provide (020,021,022,023,.....,051), work file 1 is assigned to logical unit SYS020, work file 2 is assigned to logical unit SYS021, etc.
- For each printer:
  - The corresponding logical unit in the #LOGICAL-PRINTER(*) array.
  - The corresponding printer address in the #PRINTER-ADDRESS(*) array.
  - The line size in the #LINE-SIZE(*) array.
  - For Natural and Adabas, the library type and search libraries used in the LIBDEF statement.
  - If required, the SVC and the database numbers for the ADARUN statement.

  After modifying CUNDGLDA, catalog all the model subprograms that contain this LDA (CUNDGLST, CUNDGNAT, CUNDGPS, and CUNDGWFP) and copy all CUND* modules from the SYSCST library to the SYSLIB library.

Setting up the JCL-OS-NATBATCH Model

The site-dependent information needed to generate z/OS JCL with the JCL-OS-NATBATCH model must exist in your library of cataloged procedures under the name “NATBCST”. The information is stored in the CUNOGLDA local data area in the SYSCST library. To generate z/OS JCL that fits your installation, define the following in CUNOGLDA:

- The name of a procedure to invoke Natural in batch (NATBCST, by default). An example of this procedure is shown below.
- If required, the SVC and the database numbers for the ADARUN statement.

  After modifying CUNOGLDA, catalog all the model subprograms that contain this LDA (CUNOGPS and CUNOGEXE) and copy all CUNO* modules from the SYSCST library to the SYSLIB library.

Catalog Procedure to Execute Natural in Batch

To run a job generated with the JCL-OS-NATBATCH model, a catalog procedure similar to the one below must exist in your library of cataloged procedures with the name you specified in CUNOGLDA (by default, NATBCST). Some parameters in the procedure must be changed to fit your installation.

Example of the NATBCST catalog procedure for the JCL-OS-NATBATCH model

```
MEMBER=NATBCST                                              REC=00000001
DSN=SYS1.PROCLIB                                 VOL=SAG140  UNIT=140
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
//NATBCST PROC NATPGM=NATBTCHC,        NATURAL PROGRAM NAME             00000100
//             NATREG=2000K,           REGION SIZE                      00000200
//             NTIME=5,                CPU TIME IN MINUTES              00000300
//             NPARM=,                 NATURAL  PARMS                   00000400
//             SYSDA=SYSDA,            TEMPORARY DISK ALLOCATION        00000500
//             SRTLIB='SYS2.SORTLIB',  SORT LIBRARY NAME                00000600
//             SRTSPCE=4,              SORT WORK SPACE                  00000700
//             NINDX='NATURAL.V21',    NATURAL HI LEVEL INDEX NAME      00000800
//             AINDX='ADABAS.V52',     ADABAS HI LEVEL INDEX NAME       00000900
//             SYSOUT=X,               SYSOUT CLASS                     00001000
//             CMCLASS=X,              SYSOUT CLASS                     00001100
//             SYSUDMP=DUMMY           SYSUDUMP DATA SET                00001200
//*                                                                     00001300
//STEP1   EXEC PGM=&NATPGM,REGION=&NATREG,TIME=&NTIME,                  00001400
//             PARM='&NPARM'                                            00001500
//STEPLIB   DD DSN=&NINDX..LOAD,DISP=SHR                                00001600
//          DD DSN=&AINDX..LOAD,DISP=SHR                                00001700
//*                                                                     00001800
//SORTLIB   DD DSN=&SRTLIB,DISP=SHR                                     00001900
//SYSUDUMP  DD &SYSUDMP                                                 00002000
//*                                                                     00002100
//SYSOUT    DD SYSOUT=&SYSOUT                                           00002200
//SORTOUT   DD DUMMY,DCB=BLKSIZE=80                                     00002300
//DDSORTIN  DD DISP=(,PASS),DSN=&&SORT,UNIT=&SYSDA,                     00002400
//             DCB=RECFM=FB,SPACE=(TRK,(&SRTSPCE,&SRTSPCE))             00002500
//DDSORTUT  DD DISP=(OLD,DELETE),DSN=*.DDSORTIN,VOL=REF=*.DDSORTIN      00002600
//SORTWK02  DD UNIT=&SYSDA,SPACE=(TRK,&SRTSPCE)                         00002700
//SORTWK01  DD UNIT=&SYSDA,SPACE=(TRK,&SRTSPCE)                         00002800
//SORTMSG   DD SYSOUT=&SYSOUT                                           00002900
//*                                                                     00003000
//CMPRINT   DD SYSOUT=&SYSOUT,DCB=BLKSIZE=132                           00003100
//CMPRT01   DD SYSOUT=&CMCLASS,DCB=BLKSIZE=132                          00003200
//CMPRT02   DD SYSOUT=&CMCLASS,DCB=BLKSIZE=132                          00003300
//CMPRT03   DD SYSOUT=&CMCLASS,DCB=BLKSIZE=132                          00003400
//CMPRT04   DD SYSOUT=&CMCLASS,DCB=BLKSIZE=132                          00003500
//CMPRT05   DD SYSOUT=&CMCLASS,DCB=BLKSIZE=132                          00003600
//CMSYNIN   DD DDNAME=SYSIN                                             00003700
 ****** UPD0400 - END OF DATA ******
```

JCL-DOS-NATBATCH Model

This model generates a job to execute Natural in batch under a DOS operating system. You can generate up to three steps, with each step invoking Natural to execute the specified user program(s). You can define up to four work files for each step and specify overriding Natural profile parameters (NATPARM) for each step.

Online help is available to define the NATPARM override and the work files. In addition, you can use a PF-key to copy the work file definition from a previous work file for the same step. You can specify user exits to create a tape initialization step at the beginning of a job, or intermediate steps between the execution of Natural programs. To access the User Exit editor, press PF11 (userX) on the last specification panel.

Parameters for the JCL-DOS-NATBATCH Model

You define the parameters for the JCL-DOS-NATBATCH model on two specification panels: the Standard Parameters panel and the Additional Specifications panel. In addition, there is a help window (the Natural Profile Parameters window) from which you can select Natural profile parameters. These panels and help window are described in the following sections.

Standard Parameters Panel

```
CUNDMA                     JCL-DOS-NATBATCH Model                  CUNDMA0
Jun 10                      Standard Parameters                     1 of 2

 Job name ....... TEST____
 Title .......... ________________________________

 Description .... ____________________________________________________________
                  ____________________________________________________________
                  ____________________________________________________________
                  ____________________________________________________________

 Job Parameters              Printer Parameters
   CLASS  A  (A-Z,0-9)         CLASS  A  (A-Z)
   DISP   D  (D,H,K,L)         COPY   __ (1-99)
   PRI    _  (0-9)             DISP   D  (D,H,K,L)
   USER   SAG__________


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                                          right main
```

Standard Parameters Panel for the JCL-DOS-NATBATCH Model

The fields on this panel are:

Field	Description
Job name	Name of the job to generate (by default, the name entered in the Specification/Program field on the Generation main menu). The generated job is saved in your library under the job name specified. The job name also identifies the job in the Job statement ($$ Jobname JOB, for example).
Title	Title for the generated JCL. (The title also identifies generated objects for the List function.)
Description	Brief description of the JCL. The description is inserted in the banner at the beginning of the control statements.
Job Parameters	Parameters that tell the system how to process the job. These parameters are included as part of the Job statement. You can specify the following parameters:
CLASS	Value that assigns the job a class. By default, “A” is displayed. Valid values are from A to Z or 0 to 9. The class you specify depends on the job characteristics and your installation’s rules for assigning classes.
DISP	Value that controls the disposition of your job. By default, “D” is displayed. Valid values include: • D (deletes the job from the reader queue after it is processed) • H (holds the job in the reader queue, but does not process it) • K (holds the job in the reader queue after it is processed) • L (leaves the job in the reader queue)
PRI	Priority number of the job (0 to 9). The system selects jobs for execution in order of priority within a job class. A job with higher priority is selected first.
USER	By default, the user ID of the person who invoked model. If the user ID contains blanks, enclose it within single quotes.
Printer Parameters	Parameters that control the printer output. You can specify the following parameters:
CLASS	Value that indicates the print class the output is written to. By default, “A” is displayed.

Field	Description (continued)
COPY	Number (from 1 to 99) of copies to be printed.
DISP	Value that controls the print disposition. By default, “D” is displayed. Valid values include: • D (deletes print entry from the reader queue after it is printed) • H (holds print entry in the reader queue, but does not print it) • K (holds print entry in the reader queue after it is printed) • L (leaves print entry in the reader queue)

Additional Parameters Panel

```
CUNDMB                     JCL-DOS-NATBATCH Model                    CUNDMB0
Aug 27                     Additional Parameters                      2 of 2

Number of steps  1
Step number .... 1
NATPARM override ____________________________________________________________ *
                 ____________________________________________________________
Library ........ SAG_
Work Files
1 Natural work file .. __
  Device ID ...... ______
  Unit type ...... ____   (DISK/TAPE)
  Volume or serial ______
  Retention date . ______ (YY/DDD/NUMBER OF DAYS)
  File identifier  ____________________________________________
  Record format .. FB     (VB/V/FB/F/U)
  Block length ... 4628
  Parameters for Disk Files
  File type ............... ____ (SD=SEQUENTIAL,DA=DIRECT ACCESS OR VSAM)
  Relative track - block ... __________
  Number of tracks - blocks  __________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        copy        bkwrd frwrd       left  userX main
```

Additional Parameters Panel for the JCL-DOS-NATBATCH Model

In the top portion of this panel, define the steps for the job. You can define up to three steps (only one step is displayed at a time). In the bottom portion of the panel, define the work files for each step. You can define up to four work files for each step (only one work file is displayed at a time).

The fields on this panel are:

Field	Description
Number of steps	Total number of steps for the job. Each step invokes Natural to execute the specified user program(s). The default is 1.
Note:	You can include steps for other purposes, such as tape initialization, sorts, etc., within user exits. For information, see **User Exits for the JCL-DOS-NATBATCH Model**, page 581.
Step number	Current step number. To display the next step, press PF8 (frwrd) when your cursor is in the top portion of this panel; to return to the previous step, press PF7 (bkwrd).
NATPARM override	Natural parameters you want to override (if any). If you specify more than one parameter, use a comma to separate the parameters. To display the Natural Profile Parameters help window, press PF1 (help) or enter "?" when your cursor is in the first-character position of this field. For a description of this window, see **Natural Profile Parameters Window**, page 578.
Library	Name of the library containing the Natural program executed for the corresponding step. You specify the data and the name of the program(s) within the STEP1-DATA, STEP2-DATA, and STEP3-DATA user exits.
WORK Files	
Natural work file	Number of the current work file. You can define up to four work files for each step. You select the four files from up to 32 available work files. The number of the current work file is displayed to the left of this field. To display the next work file, press PF8 (frwrd) when your cursor is in the bottom portion of this panel, to return to the previous work file, press PF7 (bkwrd).
Device ID	Name of the device for your work file (FBA, 3340, for example). The Device ID value is used in the ASSGN statement. (If the Unit type is DISK or TAPE, you do not need to specify the Device ID.)
Unit type	If the Device ID is a disk, specify "DISK" in this field. If the Device ID is a tape, specify "TAPE".
Volume or serial	Name of the volume for your work file. The volume is a portion of a storage device served by one READ/WRITE mechanism, such as a tape reel or disk pack.

Field	Description (continued)
Retention date	Retention date for the work file. By default, the work file is retained for seven days. You specify the date in yy/ddd format or as a number of days. For example, if you want to retain a work file for the first 90 days of 1996, you specify “96/90\”. If you want to retain a work file for two weeks, you specify “14”.
Note:	The operator will be notified of any attempt to remove the file before the specified retention date.
File identifier	Unique name to identify the file on a volume. If you do not specify a name, the file name (as defined in your VSE/ESA system control statements documentation) is used as the file identifier. The name can be up to 17 characters long and can contain blanks. Natural Construct encloses the name within single quotes in generated code.
Record format	Code for the record format of the work file. By default, “FB” is displayed. Valid record formats are: • VB (variable blocked) • V (variable) • FB (fixed blocked) • F (fixed) • U (undefined)
Block length	Block length for the work file. By default, “4628” is displayed.
Parameters for Disk Files	
File type	File type. Valid file types are: • SD (sequential) • DA (direct access) • VSAM
Relative track - block	Track/block number of where your work file starts. This number is relative to the beginning of the volume.
Number of tracks - blocks	Number of tracks/blocks your work file requires.

Natural Profile Parameters Window

```
CJH-NAT            ***** NATURAL PROFILE PARAMETERS *****          CJH-NAT1
Aug 27                                                             10:35 AM

IM(INPUT Mode).......: _   MADIO(Max Adabas calls between scr I/O):     0
LS(Line size)........: ___ MAXCL(Max prog. calls between scr I/O).:     0
PS(Page size)........: ___ XREF(Active cross-reference)...........: _____
WH(Wait for held rec.): ___ DU(Memory dump).....................: ___
ZD(Zero division chk ): ___

Natural Buffer  Sizes       Natural System Files  DBID..: ___ FNR...: ___
ESIZE..: __                 FDIC..: ________________________________
FSIZE..: __                 FUSER.: ________________________________
SORTSZE: __                 FNAT..: ________________________________
USIZE..: __                 FSEC..: ________________________________
                            LFILE.: ________________________________
                                    ________________________________
                                    ________________________________

Other: ____________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
     help  retrn
```

Natural Profile Parameters Window

Press PF1 (help) in the NATPARM Override field to display the Natural Profile Parameters window. Use this window to define the Natural parameters override. If you entered values in the field before invoking the help window, those values are displayed.

The fields in the Natural Profile Parameters window are:

Field	Description
IM	Default mode for data input. For delimiter mode, specify “D”; for forms mode, specify “F”.
LS	Maximum number of characters per line for Natural DISPLAY and WRITE statements. Together with the PS (page size) parameter, this parameter defines the amount of buffer space acquired during Natural initialization. Valid values are from 35 to 250. If you specify “0”, the output device line size is used.
PS	Maximum number of lines per page for Natural reports created with DISPLAY or WRITE statements. Together with the LS (line size) parameter, this parameter defines the amount of buffer space acquired during Natural initialization. Valid values are from 5 to 250. If you specify “0”, the output device page size is used.

Field	Description (continued)
WH	Action taken if a required record is not available for processing. If you want an error message returned when a record cannot be held, specify “OFF”. If you want a hold request to be attempted 20 times before an error message is returned, specify “ON”.
ZD	Action taken if a division operation with a zero divisor is attempted. To return an error message if an attempt is made to divide by zero, specify “ON”. To return a result of zero for operations with a zero divisor, specify “OFF”.
MADIO	Maximum number of DBMS calls permitted between two screen I/O operations. Valid values are 10 to 32767. If you specify “0”, no limit is in effect. The default value is 0.
MAXCL	Maximum number of program calls permitted between two screen I/O operations. Valid values are 1 to 32767. If you specify “0”, no limit is in effect. The default value is 0.
XREF	Active cross-reference feature. If you do not want active cross referencing, specify “OFF”. To store cross-reference data for database fields, subroutines, and maps in the appropriate Predict entries each time a Natural program is cataloged, specify “ON”. To check whether a Predict entry for the program exists, specify “FORCE”. If an entry exists, CATALOG is allowed and XREF data is stored. If an entry does not exist, CATALOG is not allowed.
DU	Memory dump for abnormal terminations feature. To generate a memory dump if an abnormal termination occurs during the execution of a Natural program, specify “ON”. If you do not want to generate a memory dump, specify “OFF”.
Natural Buffer Sizes	
ESIZE	Size of the user buffer extension area in kilobytes. This area contains Natural source statements, PA/PF key tables, the Natural stack area, global data area, and other work areas. Valid ESIZE values are 1–128.
FSIZE	Size of the storage area for Natural DDMs and Symbol tables in kilobytes. Valid FSIZE values are even numbers from 2–128. (If you specify an odd number, the FSIZE setting is rounded up to the next even number.)
SORTSZE	Amount of storage (in kilobytes) reserved for the sort program. If the necessary amount of storage is not available, the sort is not executed. Valid values are 10–64.

Field	Description (continued)
USIZE	Size of the user buffer area in kilobytes. This area contains data for the execution of Natural programs, including syntax tables and program-generated code. Valid values are 8–64.
Natural System Files	
DBID	Adabas database in which the Natural system files are located.
FNR	Number of the Adabas file used for the Natural system file. Valid values are 0 to 255.
FDIC	Adabas database ID, file number, password, and cipher key for the Predict system file used to retrieve and/or store data. It also stores the Natural DDMs. Valid values are: • DBID (0-254) • File number (0-255) • Password (1-8 characters) • Cipher key (1-8 characters)
FUSER	Adabas database ID, file number, password, and cipher key for the Natural user program system file used to retrieve user programs. For a list of valid values, refer to the FDIC field.
FNAT	Adabas database ID, file number, password, and cipher key for the Natural system file used to retrieve all Natural system programs. For a list of valid values, refer to the description of the FDIC field.
FSEC	Adabas database ID, file number, password, and cipher key for the Natural Security system file used to retrieve all Natural Security information. For a list of valid values, refer to the FDIC field.
LFILE	Value in this field dynamically specifies information about the physical file associated with any logical file. (A logical DDM must have a DBID of 255.) Valid values are: • Logical ID (0-255) • DBID (0-253) • File number (0-255) • Password (1-8 characters) • Cipher key (1-8 characters)
Note:	For more information, refer to the Natural profile parameters in the Natural documentation.

After specifying the appropriate parameters in the Natural Profile Parameters window, press Enter to return the information to the NATPARM Override field. To return to the Additional Specifications panel without entering information, press PF2 (retrn).

User Exits for the JCL-DOS-NATBATCH Model

The user exits listed on the User Exits panel for the JCL-DOS-NATBATCH model are described in the following sections.

CHANGE-HISTORY

This user exit contains a subprogram that keeps a record of changes to the generated JCL. Comment lines are generated which indicate the date, user ID, and a description of the change.

TAPE-INIT

This user exit contains the JCL to initialize a tape/dataset before all other steps are executed.

STEP1-DATA

This user exit specifies the Natural program executed for step 1, as well as any data that is included. If you want to execute multiple programs, specify the name of each program, followed by its corresponding data. This is a required user exit for the JCL-DOS-NATBATCH model.

BETWEEN-STEPS-1-AND-2

This user exit specifies the JCL to execute a non-Natural program between steps 1 and 2. For example, you can use this exit to execute a sorting utility prior to printing a report in step 2 and after creating the data in step 1.

STEP2-DATA

This user exit specifies the Natural program executed for step 2, as well as any data that is included. If you want to execute multiple programs, specify the name of each program, followed by its corresponding data. If you defined step 2, this is a required user exit for the JCL-DOS-NATBATCH model.

BETWEEN-STEPS-2-AND-3

This user exit specifies the JCL to execute a non-Natural program between steps 2 and 3. For example, you can use this exit to execute a sorting utility prior to printing a report in step 3 and after creating the data in step 2.

STEP3-DATA

This user exit specifies the Natural program executed for step 3, as well as any data that is included. If you want to execute multiple programs, specify the name of each program, followed by its corresponding data. If you defined step 3, this is a required user exit for the JCL-DOS-NATBATCH model.

ADDITIONAL-STEPS

This user exit specifies additional job steps, which are executed after step 3.

JCL-DOS-NATBATCH Model Example

To generate the sample JCL described in **JCL-DOS-NATBATCH Model**, page 573, enter the following information on the specifications panels:

```
CUNDMA                     JCL-DOS-NATBATCH Model                    CUNDMA0
Aug 27                      Standard Parameters                       1 of 2

 Job name ....... DREPORT_
 Title .......... Print salary increase rep

 Description .... Prints a report with all the employees, their salaries,
                  and their percentage of salary increases for a given
                  year. The report is ordered by percentage of salary
                  increase. ____________________________________________

 Job Parameters               Printer Parameters
   CLASS  A  (A-Z,0-9)          CLASS  A  (A-Z)
   DISP   D  (D,H,K,L)          COPY   __ (1-99)
   PRI    _  (0-9)              DISP   D  (D,H,K,L)
   USER   DEVLGL__________



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        copy        bkwrd frwrd       left  userX main
```

Standard Parameters Panel for the
JCL-DOS-NATBATCH Model Example

```
CUNDMB                     JCL-DOS-NATBATCH Model                    CUNDMB0
Aug 27                     Additional Parameters                      2 of 2

Number of steps  3
Step number .... 1
NATPARM override ZD=OFF,ESIZE=50,FNAT=(18,5),FUSER=(18,4)_________________ *
                 ____________________________________________________________
Library ........ SAG_____
Work Files
1 Natural work file .. 1_
  Device ID ...... ______
  Unit type ...... Disk   (DISK/TAPE)
  Volume or serial SER100
  Retention date . ______ (YY/DDD/NUMBER OF DAYS)
  File identifier  Salary Data_____________________________
  Record format .. FB     (VB/V/FB/F/U)
  Block length ... 4628
  Parameters for Disk Files
  File type ................ ____ (SD=SEQUENTIAL,DA=DIRECT ACCESS OR VSAM)
  Relative track - block ... 10________
  Number of tracks - blocks  1_________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        hcopy       bkwrd frwrd       left  userX main
```

Additional Parameters Panel for the
JCL-DOS-NATBATCH Model Example – Step 1

```
CJH-NAT            ***** NATURAL PROFILE PARAMETERS *****          CJH-NAT1
Sep 06                                                             11:38 AM

IM(INPUT Mode).....: _   MADIO(Max Adabas calls between scr I/O):     0
LS(Line size)......: ___ MAXCL(Max prog. calls between scr I/O).:     0
PS(Page size)......: ___ XREF(Active cross-reference)...........: _____
WH(Wait for held rec.): ___ DU(Memory dump).....................: ___
ZD(Zero division chk ): OFF

Natural Buffer  Sizes       Natural System Files  DBID..: ___ FNR...: ___
ESIZE..: 50                 FDIC..: _____________________________
FSIZE..: __                 FUSER.: (18,4)______________________
SORTSZE: __                 FNAT..: (18,5)______________________
USIZE..: __                 FSEC..: _____________________________
                            LFILE.: _____________________________
                                    _____________________________
                                    _____________________________

Other: ____________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
     help  retrn
```

Natural Profile Parameters Window for the
JCL-DOS-NATBATCH Model Example

```
CUNDMB                    JCL-DOS-NATBATCH Model                   CUNDMB0
Sep 09                    Additional Parameters                     2 of 2

Number of steps  3
Step number .... 2
NATPARM override IM=D,LS=130,PS=060,ESIZE=50,FNAT=(18,5),FUSER=(18,4)____ *
                ____________________________________________________________
Library ........ SAG_____
Work Files
1 Natural work file .. 2_
  Device ID ...... ______
  Unit type ...... DISK   (DISK/TAPE)
  Volume or serial ______
  Retention date . SER!)) (YY/DDD/NUMBER OF DAYS)
  File identifier  SORTED SALARY DATA_____________________________
  Record format .. FB     (VB/V/FB/F/U)
  Block length ... 4628
  Parameters for Disk Files
  File type ..................... ____ (SD=SEQUENTIAL,DA=DIRECT ACCESS OR VSAM)
  Relative track - block ... 100_______
  Number of tracks - blocks  1_________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        copy        bkwrd frwrd       left  userX main
```

Additional Parameters Panel for the
JCL-DOS-NATBATCH Model Example – Step 2

```
CUNDMB                     JCL-DOS-NATBATCH Model                    CUNDMB0
Sep 09                     Additional Parameters                      2 of 2

Number of steps  3
Step number .... 3
NATPARM override IM=D,LS=130,PS=060,ESIZE=50,FNAT=(18,5),FUSER=(18,4)____ *
                 ____________________________________________________________
Library ........ SAG_____
Work Files
1 Natural work file .. 3_
  Device ID ...... ______
  Unit type ...... DISK   (DISK/TAPE)
  Volume or serial SER100
  Retention date .        (YY/DDD/NUMBER OF DAYS)
  File identifier  SORTED SALARY DATA 2_____________________________
  Record format .. FB     (VB/V/FB/F/U)
  Block length ... 4628
  Parameters for Disk Files
  File type ............... ____ (SD=SEQUENTIAL,DA=DIRECT ACCESS OR VSAM)
  Relative track - block ... 100_______
  Number of tracks - blocks  1_________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        copy        bkwrd frwrd       left  userX main
```

Additional Parameters Panel for the
JCL-DOS-NATBATCH Model Example – Step 3

JCL-OS-NATBATCH Model

This model generates a job to execute Natural in batch under the z/OS operating system. You can generate up to three steps, with each step invoking Natural to execute the specified user program(s). You can define up to four work files for each step and specify overriding Natural profile parameters (NATPARM) for each step.

Online help is available to define the NPARM override and the work files. In addition, a PF-key allows you to copy the work file definition from a previous work file for the same step. You can specify user exits to create a tape initialization step at the beginning of a job, or intermediate steps between the execution of Natural programs. To access the User Exit editor, press PF11 (userX) on the last specification panel.

Parameters for the JCL-OS-NATBATCH Model

Define parameters for the JCL-OS-NATBATCH model on the following specification panels:

- Standard Parameters
- Additional Parameters

In addition, there are six help windows from which you can select specifications:

- Disposition Parameter Options
- Volume Parameter Options
- Unit Parameter Options
- Label Parameter Options
- Space Parameter Options
- DCB Parameter Options

These panels and help windows are described in the following sections.

Standard Parameters Panel

```
CUNOMA                      JCL-OS-NATBATCH Model                      CUNOMA0
Aug 27                       Standard Parameters                        1 of 2

 Job name ....... JCLTest_
 Title .......... _____________________________

 Description .... ____________________________________________________________
                  ____________________________________________________________
                  ____________________________________________________________
                  ____________________________________________________________

 Job Parameters
  CLASS    A   (A-Z,0-9)    REGION _____ K
  MSGCLASS _   (A-Z,0-9)    TIME   _____ (Minutes)
  MSGLEVEL ___ (0-2,0-1)    TYPRUN ____  (HOLD SCAN COPY)
  PRTY     __  (0-13)




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                                          right main
```

Standard Parameters Panel for the JCL-OS-NATBATCH Model

The fields on this panel are:

Field	Description
Job name	Name of the job to generate (by default, the name entered in the Specification/Program field on the Generation main menu). The generated job is saved in your library under the job name specified. The job name also identifies the job in the Job statement (//Jobname JOB, for example).
Title	Title for the generated JCL. (The title also identifies generated objects for the List function.)
Description	Brief description of the JCL. The description is inserted in the banner at the beginning of the control statements.
Job Parameters	Parameters that tell the system how to process the job. These parameters are included as part of the Job statement. You can specify the following parameters:
CLASS	Value that assigns the job a class. By default, “A” is displayed. Valid values are from A to Z or 0 to 9. The class you specify depends on the job characteristics and your installation’s rules for assigning classes.

Field	Description (continued)
MSGCLASS	Character that assigns the job log to an output class. The job log is a record of job-related information for developers. Valid values are A to Z or 0 to 9.
MSGLEVEL	Characters to control the listing of the job log. The characters are entered in the following format: `(statements,messages)` where *statements* indicates which JCL statements the system prints to the job log. Valid values are: • 0 (print only the job) • 1 (print all JCL, JES, and procedure statements) • 2 (print JCL and JES statements) and *messages* indicates which messages the system prints to the job log. Valid values are: • 0 (print only JCL messages) • 1 (print JCL, JES, and operator messages)
PRTY	Number to assign a selection priority to your job. Within a job class, the system selects jobs for execution in order of priority. A job with a higher priority number is selected first. Valid priority numbers are 0 to 13.
REGION	Number of kilobytes the job requires. Valid numbers are 1 to 16383.
TIME	Maximum number of minutes the job requires. The number must be 1 to 1440. If you specify 1440 (1440 equals 24 hours), the job can use the processor for an unlimited amount of time.
TYPRUN	Special job processing features. Valid values are:
SCAN	Scans the job's JCL for syntax errors, without executing the job or allocating devices.
HOLD	Holds the job (before execution) until the operator releases it.
COPY	JES2 copies the input job stream, as submitted, to a sysout dataset and schedules the sysout dataset for output processing. The job is not scheduled for execution.

Additional Parameters Panel

```
CUNOMB                    JCL-OS-NATBATCH Model                    CUNOMB0
Aug 27                    Additional Parameters                     2 of 2

Number of steps  1

Step number .... 1
NATPARM override _____________________________________________________ *
                 _____________________________________________________
Library ........ SAG_____

Work Files

1 Natural work file .. __
  DSN   _____________________________________________
  DISP  NEW,KEEP_____________________ *
  VOL   _____________________________________________________ *
  UNIT  SYSDA_________ *
  LABEL _____________________________________________ *
  SPACE CYL,(5,5),RLSE_____________________________ *
  DCB   _____________________________________________ *

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        copy  hlpwf bkwrd frwrd       left  userX main
```

Additional Parameters Panel for the JCL-OS-NATBATCH Model

In the top portion of this panel, define the steps for the job. You can define up to three steps (only one step is displayed at a time). In the bottom portion of the panel, define the work files for each step. You can define up to four work files for each step (only one work file is displayed at a time).

The fields on this panel are:

Field	Description
Number of steps	Total number of steps for the job. Each step invokes Natural to execute the specified user program(s). By default, “1” is displayed.
Note:	You can include steps for other purposes, such as tape initialization, sorts, etc., within user exits. For more information, see **JCL-OS-NATBATCH Model**, page 586.
Step number	Current step number. To display the next step, press PF8 (frwrd) when your cursor is in the top portion of this panel; to return to the previous step, press PF7 (bkwrd).
NATPARM override	Natural parameters you want to override (if any). If you specify more than one parameter, use a comma to separate the parameters. To display the help window, press PF1 or enter “?” when your cursor is in the first-character position of this field. For a description of this window, see **Natural Profile Parameters Window**, page 578.

Field	Description (continued)
Library	Name of the library containing the Natural program executed for the corresponding step. You specify the data and the name of the program(s) within the STEP1-DATA, STEP2-DATA, and STEP3-DATA user exits.
Work Files	
Natural work file	Number of the current work file. You can define up to four work files for each step. You select the four files from up to 32 available work files. The number of the current work file is displayed to the left of this field. To display the next work file, press PF8 (frwrd) when your cursor is in the bottom portion of this panel, to return to the previous work file, press PF7 (bkwrd).

DSN	Name of a dataset, which can be temporary or permanent. A temporary dataset is created and deleted within the job, while a permanent dataset can be retained after the job is finished. Specify the DSN using the following syntax:
	
	Temporary datasets:
	
	`DSNAME = &&dsname`
	`DSNAME = &&dsname (member)` (partitioned datasets)
	
	Permanent datasets:
	
	`DSNAME = dsname`
	`DSNAME = dsname (member)` (partitioned datasets)

DISP	Status (old, new, or modified) and system disposition (pass, keep, catalog, uncatalog, or delete) for the dataset at the end of the step or if the step abnormally terminates. This parameter is always required unless the dataset is created and deleted in the same step. You specify the DISP parameter using the following syntax: `DISP=(status,normal-disp,abnormal-disp)` To display a help window, press PF1 (help). For a description of the window, see **Disposition Parameter Options Window**, page 592.
VOL	Volume(s) on which the dataset resides or will be created. The volume is a portion of a storage device served by one READ/WRITE mechanism, such as a tape reel or disk pack. You can request a specific volume, multiple volumes, retention of a volume, or a private volume. To display a help window, press PF1 (help). For a description of the window, see **Volume Parameter Options Window**, page 593.

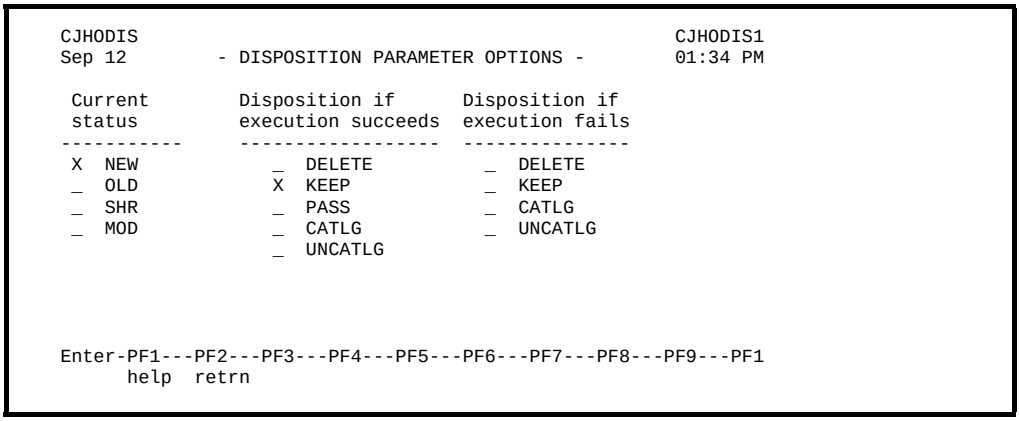
Field	Description (continued)
UNIT	Type of I/O device. You can request the unit by hardware address, device type, or group name. The device type is the numeric model type (3330 disk, for example), and the group name is the name assigned to a group of units (TAPE, DISK, or SYSDA, for example). To display a help window, press PF1 (help). For a description of the window, see **Unit Parameter Options Window**, page 595.
LABEL	Type of label for the dataset, the relative file number on the tape, whether the dataset is protected for input or output, and the retention date or period for the dataset. To display a help window, press PF1 (help). For a description of the window, see **Label Parameter Options Window**, page 596.
Note:	If you enter values in the LABEL field before you invoke the help window, those values are displayed in the window.
SPACE	Amount of space required for a new dataset on a direct access volume. (The Space parameter does not apply to tape volumes.) To display a help window, press PF1 (help). For a description of the window, see **Space Parameter Options Window**, page 597.
DCB	Dataset data control block (DCB) information completed during execution.
Note:	You can create a dataset with the same DCB parameters as a cataloged dataset by either entering the name of the existing dataset or entering a reference to the DD statement containing the desired DCB information. To display a help window, press PF1 (help). For a description of the window, see **DCB Parameter Options Window**, page 599.

Invoking Help for the Work File Definition

To invoke help for a work file definition, press PF6 (hlpwf). This option takes you through a series of help windows to define each of the parameters.

Note: The help window for the Space parameter is not displayed if UNIT=TAPE, DISP=OLD, or DISP=SHR.

Disposition Parameter Options Window



Disposition Parameter Options Window

Note: If you enter values in the DISP field before you invoke the help window, those values are displayed in the window.

By default, the NEW and KEEP fields are marked. Mark one option in each column (Current status, Disposition if execution succeeds, and Disposition if execution fails) and press Enter to return to the Additional Parameters panel with the corresponding character string.

To return to the Additional Parameters panel without modifying the current settings, press PF2 (retrn).

Volume Parameter Options Window

```
CJHOVOL                                                  CJHOVOL1
Sep 12           - VOLUME PARAMETER OPTIONS -            01:36 PM

PRIVATE.....: _  (Exclusive use of volume for the Job)
RETAIN......: _  (Do not demount tape vol. at the end of step)
Seq. number.: __ (Start processing multivolume data set with
                  this volume)
Count.......: __ (Maximum number of volumes allocated to ds.)

Enter serial numbers
SER.........: ______ ______ ______     or

Use the REF parameter to copy the serial number from
an existing data set (specify  REF=dsname)
or from a previous DD statement (specify REF =*.step.ddname)
REF.........: ____________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10-
      help  retrn
```

Volume Parameter Options Window

Note: If you enter values in the VOL field before you invoke the help window, those values are displayed in the window.

Use this window to define the characteristics of the Volume parameter. The fields in this window are:

Field	Description
PRIVATE	To have exclusive use of the volume for the job, mark this field.
RETAIN	To keep the volume mounted until it is used in a subsequent step or until the end of the job, mark this field.
Seq. number	Volume sequence number. Valid values are 1 to 255. If you do not specify a number, processing begins with the first volume.
Count	Maximum number of volumes the dataset requires.

Field	Description (continued)
SER	To request a specific volume, enter the serial numbers in this field. You can specify up to three volumes. A volume serial number is 1 to 6 alphanumeric or national (@, &, #) characters, hyphens, or special characters enclosed within single quotes.
REF	To request the volumes used by a cataloged dataset, specify the following: `REF=`*`dsname`* where *dsname* is the name of the dataset. To request the volumes used by a dataset in a DD statement in the current step, specify the following: `REF=*.`*`ddname`* where *ddname* is the name of the DD statement. To request the volumes used by a dataset in a DD statement in a previous step, specify the following: `REF=*.`*`step`*`.`*`ddname`* where *step* is the name of the step and *ddname* is the name of the DD statement. After specifying the options for the Volume parameter, press Enter to return to the Additional Parameters panel with the corresponding character string. To return to the Additional Parameters panel without making any changes, press PF2 (retrn).

Unit Parameter Options Window

```
CJHOUNI                                                  CJHOUNI1
Sep 12            - UNIT PARAMETER OPTIONS -             01:38 PM

SYSDA  X  or   TAPE  _    Other: ______

For multivolume data sets:
Enter number of volumes that should be mounted
Unit Count........: __   or
Parallel mounting.: _ (obtain unit count from VOLUME parameter)

DEFER.....: _  (Defer mounting the volume until ds. is open)


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF1
      help  retrn
```

Unit Parameter Options Window

Note: If you enter values in the UNIT field before you invoke the help window, those values are displayed in the window.

By default, the SYSDA field is marked. To use another unit type, enter a blank in this field. To use a tape, mark the TAPE field. To use another unit type, enter the name of the unit in the OTHER field.

For multi-volume datasets, you can enter the number of volumes that should be mounted in the Unit Count field. To obtain the unit count from the Vol field, mark the Parallel mounting field.

To defer mounting the volume until after the dataset is open, mark the DEFER field.

After specifying the options for the Unit parameter, press Enter to return to the Additional Parameters panel with the corresponding character string.

To return to the Additional Parameters panel without making any changes, press PF2 (retrn).

Label Parameter Options Window

```
CJHOLAB                                                 CJHOLAB1
Sep 12           - LABEL PARAMETER OPTIONS -            01:41 PM

Sequence number: _____ (Position of data set in volume)

Labels (TAPE and DISK)        Labels (TAPE only)
SL: _ (standard labels)       NL.: _ (no labels)
AL: _ (ISO/ANSI labels)       BLP: _ (bypass label proc.)

PASSWORD _ or NOPWREAD _ (Password protection)
IN _       or OUT _

Expiration Date
RETPD... _____ (Number of days)
EXPDT... _____ (Expiration data in format yyddd,
                ie. Feb 1,1991 is 91032 )
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF1
      help  retrn
```

Label Parameter Options Window

The fields in this window are:

Field	Description
Sequence number	1 to 4 digit sequence number that describes the dataset's position on the volume (in relation to other datasets). The first file on the tape is number 1.
Labels (TAPE and DISK)	
SL	To use standard labels, mark this field.
AL	To use American National Standard labels (ISO/ANSI), mark this field.
Labels (TAPE only)	
NL	To use no labels, mark this field.
BLP	To bypass the label processing, mark this field.
PASSWORD	If a password is required to open or delete the dataset, mark this field.
NOPWREAD	If a password is required to write or delete the dataset, but not to read it, mark this field.
IN	To protect the dataset from being opened for output, mark this field.

Field	Description (continued)
OUT	To protect the dataset from being opened for input, mark this field.
Expiration Date	
RETPD	To assign a retention period to datasets on tape or direct-access volumes, enter the number of days (0 to 9999).
EXPDT	To assign an expiration date to datasets on tape or direct-access volumes, enter the two-digit year and three-digit day. A dataset with an unexpired retention date cannot be modified or deleted. If you do not specify an expiration date, a retention period of zero days is assumed.

After specifying the options for the Label parameter, press Enter to return to the Additional Parameters panel with the corresponding character string.

To return to the Additional Parameters panel without making any changes, press PF2 (retrn).

Space Parameter Options Window

```
CJHOSPA                                                  CJHOSPA1
Sep 12           - SPACE PARAMETER OPTIONS -             01:43 PM

Enter SPACE parameter for disk files only

TRK _ or CYL X or Block size: _____
Primary...: 5____
Secondary.: 5____
Directory/Index: _____ (For partitioned/ISAM data sets)
RLSE......: X (Release allocated space that is not used)
CONTIG....: _ (space allocated to ds. must be contiguous)

If space is specified giving block size
ROUND.....: _ (Round space to an integral number of cylinders)

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF1
      help  retrn
```

Space Parameter Options Window

Note: If you enter values in the SPACE field before you invoke the help window, those values are displayed in the window.

Use this window to specify the parameters for disk files only. The fields in this window are:

Field	Description
TRK	To request space in tracks, mark this field.
CYL	To request space in cylinders, mark this field. By default, this field is marked. To mark another option (TRK or Block size), enter a blank in this field.
Block size	To request space by block size, enter the number of bytes per block. The number is a decimal from 1 to 65,535.
Primary	Number of units (tracks, cylinders, or blocks) to allocate. This space is always allocated on a single volume. By default, “5” is displayed.
Secondary	Number of units (tracks, cylinders, or blocks) to allocate if the primary allocation is exceeded. This space is always allocated on a single volume. By default, “5” is displayed.
Directory/Index	To allocate space for the members of partitioned datasets or index space for ISAM datasets, enter the number of units (tracks, cylinders, or blocks).
RLSE	To have all unused space released when the dataset is closed, mark this field. By default, this field is marked. (This option allows you to allocate more space than may be needed, without wasting space.) If you do not want to include this option, enter a blank.
CONTIG	If you want the Primary space to be allocated only on contiguous tracks and cylinders, mark this field.
ROUND	To ensure that the data is placed on the minimum number of cylinders possible, mark this field. The system calculates the amount of space needed, rounds up to the nearest cylinder, and allocates complete cylinders so the space begins on the first track of a cylinder and ends on the last track of a cylinder. (If contiguous cylinders are required, you must also mark the CONTIG field.)

After specifying the options for the Space parameter, press Enter to return to the Additional Parameters panel with the corresponding character string.

To return to the Additional Parameters panel without making any changes, press PF2 (retrn).

DCB Parameter Options Window

```
CJHODCB                                                  CJHODCB1
Sep 12           - DCB PARAMETER OPTIONS -               01:44 PM

RECFM(Record format)..: ___  (F=Fixed length, V=Variable length,
                              FB=Fixed length and blocked,
                              VB=Variable length and blocked,
                              U=Undefined length records)
LRECL(Record length)..: ___
BLKSIZE(Block size)...: _____ (1-32760)
DSORG(DS organization): ___ (PS=Sequential, PO=Partitioned,
                             IS=Ind. sequential, DA=Direct access
DEN(Density)..........: 0 (0-4. Tape files only)


Use the REF parameter to copy the DCB subparameters from
an existing data set name(specify REF=dsname),
or from a previous DD statement(specify  REF=*.step.ddname)
REF.: ____________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10-
      help  retrn
```

DCB Parameter Options Window

Note: If you enter values in the DCB field before you invoke the help window, those values are displayed in the window.

The fields in this window are:

Field	Description
RECFM	Record format. Specify one or more of the following: • F (fixed length) • V (variable length) • FB (fixed length and blocked) • VB (variable length and blocked) • U (undefined length records)
LRECL	Length in bytes of the logical record for fixed or variable length records. (Omit this option for records with an undefined length.) The LRECL parameter is equal to the record length of fixed length records and equal to the size of the largest record plus 4 bytes for variable length records.

Field	Description (continued)
BLKSIZE	Size of the dataset block in bytes. For fixed length records, it must be a multiple of the value in the LRECL field. For variable length records, it must be equal to or greater than the value in the LRECL field plus 4. For undefined length records, it must be as large as the longest block. The block size can range from 1–32,760 bytes for direct-access storage devices, and 18–32,760 bytes for magnetic tapes.
DSORG	Dataset organization type. Valid types are: • PS (sequential) • PO (partitioned) • IS (indexed sequential) • DA (direct access)
DEN	Density for tape files. Valid values are 0 to 4.
REF	To copy DCB information for a catalogued dataset, specify the following: `DCB=`*`dsname`* where *dsname* is the name of the dataset. To copy DCB information for a dataset in a DD statement in the current step, specify the following: `DCB=*.`*`ddname`* where *ddname* is the name of the DD statement. To copy DCB information for a dataset in a DD statement in a previous step, specify the following: `DCB=*.`*`step.ddname`* where *step* is the name of the step and *ddname* is the name of the DD statement.

After specifying the options for the DCB parameter, press Enter to return to the Additional Parameters panel with the corresponding character string.

To return to the Additional Parameters panel without making any changes, press PF2 (retrn).

User Exits for the JCL-OS-NATBATCH Model

The user exits listed on the User Exits panel for the JCL-OS-NATBATCH model are described in the following section.

CHANGE-HISTORY

This user exit subprogram keeps a record of changes to the generated JCL. Comment lines are generated which indicate the date, user ID, and a description of the change.

TAPE-INIT

This user exit contains the JCL needed to initialize a TAPE/dataset before all other steps are executed.

STEP1-DATA

This user exit identifies the Natural program executed for step 1, as well as any data that is included. This is a required user exit for the JCL-OS-NATBATCH model.

BETWEEN-STEPS-1-AND-2

This user exit specifies the JCL to execute a non-Natural program between steps 1 and 2. For example, you can use this user exit to execute a sorting utility prior to printing a report in step 2 and after creating the data for the report in step 1.

STEP2-DATA

This user exit identifies the Natural program executed for step 2, as well as any data that is included. To execute multiple programs, specify the name of each program followed by its corresponding data. If step 2 is defined, this is a required user exit for the JCL-OS-NATBATCH model.

BETWEEN-STEPS-2-AND-3

This user exit specifies the JCL to execute a non-Natural program between steps 2 and 3. For example, you can use this user exit to execute a sorting utility prior to printing a report in step 3 and after creating the data for the report in step 2.

STEP3-DATA

This user exit identifies the Natural program executed for step 3, as well as any data that is included. To execute multiple programs, specify the name of each program followed by its corresponding data. If step 3 is defined, this is a required user exit for the JCL-OS-NATBATCH model.

ADDITIONAL-STEPS

This user exit specifies additional job steps, which are executed after step 3.

JCL-OS-NATBATCH Model Example

To generate the sample **JCL-OS-NATBATCH Model**, page 586, enter the following information on the specifications panels:

```
CUNOMA                     JCL-OS-NATBATCH Model                     CUNOMA0
Aug 27                      Standard Parameters                       1 of 2

 Job name ....... OSREPORT
 Title .......... Print salary increase rep

 Description .... Prints a report with all the employees, their salaries,
                  and their salary increases for a given year. The ______
                  report is ordered by the percentage of salary increase.
                  ____________________________________________________

 Job Parameters
  CLASS    A   (A-Z,0-9)    REGION 2000_ K
  MSGCLASS _   (A-Z,0-9)    TIME   _____ (Minutes)
  MSGLEVEL 1,1 (0-2,0-1)    TYPRUN ____  (HOLD SCAN COPY)
  PRTY     __  (0-13)




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
right help  retrn quit                                          right main
```

Standard Parameters Panel for the
JCL-OS-NATBATCH Model Example

```
CUNOMB                     JCL-OS-NATBATCH Model                     CUNOMB0
Aug 27                     Additional Parameters                      2 of 2

Number of steps  2

Step number .... 1
NATPARM override IM=D,ZD=OFF,ESIZE=55,FNAT=(18,5),FUSER=(18,4)___________ *
                ____________________________________________________________
Library ........ SAG_____

Work Files

1 Natural work file .. 1_
  DSN    SALARY.DATA_____________________________________
  DISP   NEW,KEEP_____________________ *
  VOL    SER=(SAG140)_____________________________________________ *
  UNIT   SYSDA_________ *
  LABEL  _____________________________________________ *
  SPACE  CYL,(5,5),RLSE_____________________________ *
  DCB    RECFM=FB,LRECL=080,BLKSIZE=00800_____________ *

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        copy  hlpwf bkwrd frwrd       left  userX main
```

Additional Parameters Panel for the
JCL-OS-NATBATCH Model Example — Step 1

```
CJH-NAT            ***** NATURAL PROFILE PARAMETERS *****          CJH-NAT1
Sep 10                                                             10:59 AM

IM(INPUT Mode).....: D   MADIO(Max Adabas calls between scr I/O):     0
LS(Line size)......: ___ MAXCL(Max prog. calls between scr I/O).:     0
PS(Page size)......: ___ XREF(Active cross-reference)...........: _____
WH(Wait for held rec.): ___ DU(Memory dump).....................: ___
ZD(Zero division chk ): OFF

Natural Buffer  Sizes       Natural System Files  DBID..: ___ FNR...: ___
ESIZE..: 55                 FDIC..: _____________________________
FSIZE..: __                 FUSER.: (18,4)______________________
SORTSZE: __                 FNAT..: (18,5)______________________
USIZE..: __                 FSEC..: _____________________________
                            LFILE.: _____________________________
                                    _____________________________
                                    _____________________________

Other: *_____________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn
```

Natural Profile Parameters Window for the
JCL-OS-NATBATCH Model Example

```
CJHODIS                                                  CJHODIS1
Sep 12        - DISPOSITION PARAMETER OPTIONS -          01:49 PM

 Current        Disposition if      Disposition if
 status         execution succeeds  execution fails
-----------     ------------------  ---------------
 X  NEW           _  DELETE           _  DELETE
 _  OLD           X  KEEP             _  KEEP
 _  SHR           _  PASS             _  CATLG
 _  MOD           _  CATLG            _  UNCATLG
                  _  UNCATLG



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF1
      help  retrn
```

Disposition Parameter Options Window for the
JCL-OS-NATBATCH Model Example

```
CJHOVOL                                                  CJHOVOL1
Sep 12           - VOLUME PARAMETER OPTIONS -            01:50 PM

PRIVATE.....: _  (Exclusive use of volume for the Job)
RETAIN......: _  (Do not demount tape vol. at the end of step)
Seq. number.: __ (Start processing multivolume data set with
                  this volume)
Count.......: __ (Maximum number of volumes allocated to ds.)

Enter serial numbers
SER.........: SAG140 ______ ______     or

Use the REF parameter to copy the serial number from
an existing data set (specify  REF=dsname)
or from a previous DD statement (specify REF =*.step.ddname)
REF.........: ________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10-
      help  retrn
```

Volume Parameter Options Window for the
JCL-OS-NATBATCH Model Example

```
CJHOUNI                                                  CJHOUNI1
Sep 10            - UNIT PARAMETER OPTIONS -             11:05 AM

SYSDA  X  or   TAPE  _    Other: ______

For multivolume data sets:
Enter number of volumes that should be mounted
Unit Count.........: __   or
Parallel mounting.: _ (obtain unit count from VOLUME parameter)

DEFER.....: _  (Defer mounting the volume until ds. is open)


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF1
      help  retrn
```

Unit Parameter Options Window for the
JCL-OS-NATBATCH Model Example

```
CJHOLAB                                                  CJHOLAB1
Sep 12           - LABEL PARAMETER OPTIONS -             01:52 PM

Sequence number: _____ (Position of data set in volume)

Labels (TAPE and DISK)        Labels (TAPE only)
SL: _ (standard labels)       NL.: _ (no labels)
AL: _ (ISO/ANSI labels)       BLP: _ (bypass label proc.)

PASSWORD _ or NOPWREAD _ (Password protection)
IN _       or OUT _

Expiration Date
RETPD... _____ (Number of days)
EXPDT... _____ (Expiration data in format yyddd,
                ie. Feb 1,1991 is 91032 )
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF1
      help  retrn
```

Label Parameter Options Window for the
JCL-OS-NATBATCH Model Example

```
CJHOSPA                                                  CJHOSPA1
Sep 12           - SPACE PARAMETER OPTIONS -             01:53 PM

Enter SPACE parameter for disk files only

TRK _ or CYL X or Block size: _____
Primary...: 5____
Secondary.: 5____
Directory/Index: _____ (For partitioned/ISAM data sets)
RLSE......: X (Release allocated space that is not used)
CONTIG....: _ (space allocated to ds. must be contiguous)

If space is specified giving block size
ROUND.....: _ (Round space to an integral number of cylinders)

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF1
      help  retrn
```

Space Parameter Options Window for the
JCL-OS-NATBATCH Model Example

```
CJHODCB                                                    CJHODCB1
Sep 12            - DCB PARAMETER OPTIONS -                01:54 PM

RECFM(Record format)..: FB_  (F=Fixed length, V=Variable length,
                              FB=Fixed length and blocked,
                              VB=Variable length and blocked,
                              U=Undefined length records)
LRECL(Record length)..: 80_
BLKSIZE(Block size)...: 800__ (1-32760)
DSORG(DS organization): ___ (PS=Sequential, PO=Partitioned,
                             IS=Ind. sequential, DA=Direct access
DEN(Density)..........: 0 (0-4. Tape files only)


Use the REF parameter to copy the DCB subparameters from
an existing data set name(specify REF=dsname),
or from a previous DD statement(specify  REF=*.step.ddname)
REF.: ____________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10-
      help  retrn
```

DCB Parameter Options Window for the
JCL-OS-NATBATCH Model Example

```
CUNOMB                     JCL-OS-NATBATCH Model                    CUNOMB0
Aug 27                     Additional Parameters                     2 of 2

Number of steps  2

Step number .... 2
NATPARM override LS=130,PS=060,ESIZE=55,FNAT=(18,5),FUSER=(18,4)_________ *
                ____________________________________________________________
Library ........ SAG_____

Work Files

1 Natural work file .. 2_
  DSN   SORTED.SALARY.DATA_____________________________
  DISP  OLD,DELETE,DELETE_________ *
  VOL   SER=(SAG140)_____________________________________________ *
  UNIT  SYSDA_________ *
  LABEL ____________________________________________ *
  SPACE CYL,(5,5),RLSE_____________________________ *
  DCB   RECFM=FB,LRECL=080,BLKSIZE=00800_____________ *

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        copy  hlpwf bkwrd frwrd       left  userX main
```

Additional Parameters Panel for the
JCL-OS-NATBATCH Model Example — Step 2

USE OF PREDICT IN NATURAL CONSTRUCT

This chapter describes the use of Predict in Natural Construct. Many of the components of Natural Construct access the Predict data dictionary and use Predict information to specify the code that is generated. This chapter also describes the user views and verification rules used in the Natural Construct demo system.

The following topics are covered:

Unix Note:
If you have Entire Net-Work installed, you can access Predict data from a platform other than the Unix system on which Natural Construct is running. This ensures consistency in your Predict data because only one central copy is maintained.

Building Local Views

The views defined in the generated programs correspond to the views specified in Predict. All fields in the Predict view are generated into the program's view. If you do not want to include certain fields, you must either customize the Predict view or leave the fields off the map or display panel.

Use the following Predict options to tailor your views:

- redefined fields
- counter fields
- periodic group structure
- occurrences of MU/PE fields
- Natural synonyms
- DDM prefix

These options are explained in the following sections.

Redefined Fields

All fields redefined in Predict are also redefined in the generated views, making the redefined portions of the field accessible to the user exit code and maps in the generated programs. The elements of a periodic group (PE) can also be redefined. Superdescriptors are also included in the generated views.

If a portion of a field is redefined using the field name of FILLER, it is not displayed in generated INPUT statements. You can initialize the FILLER portion of the prime key field by assigning the initial value to the minimum key value parameter or in the START-OF-PROGRAM user exit.

Note: If a field is redefined more than once, the first redefinition determines the input format for the field. This rule applies recursively to all lower level redefinitions as well.

Counter Fields

Counter fields (C**field-name*) inserted in the generated views for multiple-valued fields (MUs) are defined in Predict with an MC field type (rather than MU). Similarly, counter fields are inserted in the view for periodic groups (PEs) with a PC field type (rather than PE).

Periodic Group Structure

Periodic group fields (PEs) can be structured in the generated views using one of two notations. The value specified for the Index on PE group level field during extended editing determines which notation is used.

Example 1

If the Index on PE group level field is "Y", the PE level is included in the view.

```
01 SAMPLE VIEW OF SAMPLE
  02 PE-GROUP(1:10)
    03 ELEM1
    03 ELEM2
```

Example 2

If the Index on PE group level field is blank, the PE level is not included in the view.

```
01 SAMPLE VIEW OF SAMPLE
  02 ELEM1(1:10)
  02 ELEM2(1:10)
```

This method reduces the size of the format buffer needed during program compilation.

Occurrences of MU/PE Fields

During Predict extended editing of a field, specify the maximum number of occurrences of multiple-valued fields (MUs) and periodic groups (PEs). If not, Natural Construct defaults to the Adabas maximum of 191 for an MU and 99 for a PE. For browse panels, you can return a single variable occurrence of an MU or PE field.

➢ To return a single variable occurrence of an MU or PE field:

1 Invoke extended editing for the field.

2 Invoke the second Modify Field panel (by entering "Y" in the Attr field on the first Modify Field panel). Select 3GL specification from the Additional Attributes window.

3 Specify an Indexed By name value.
If you specify #PANEL, for example, a single occurrence of the MU/PE field is returned by the browse program (based on the current left/right panel number).

Natural Synonyms

If you specify a Natural synonym in Predict during extended field editing, the synonym is used instead of the Predict field name.

DDM Prefix

If you specify a prefix when generating a Natural data definition module (DDM), each field name in the generated DDM is a combination of the specified prefix and the Predict field name.

Internally, Natural Construct works with the Predict field names. Predict field names are displayed on the field selection panels. DDM prefixes are added only at code generation time.

DB2 Users

When the generation defaults for a DDM have the value “D” for Date/Time/Timestamp representation, Predict generates D, T, and TS formats (respectively). Natural Construct acknowledges this default and also generates D, T, and TS formats. When the generation defaults for a DDM have the value “A”, both Predict and Natural Construct generate A10, A8, and A26 formats (respectively).

Note: If these formats are changed in Predict, Natural Construct acknowledges the changes immediately. However, the DDM must be regenerated before Natural will pick up the changes.

Note: For information about null indicators and superdescriptor usage, see **Natural DB2 and SQL/DS Users**, page 50.

Other Uses of Predict

In addition to establishing the local views, you can also use Predict for:

- field prompts
- numeric signs
- superdescriptor input format

Field Prompts

When generating field prompts, Natural Construct compresses all headings specified for a field and inserts a space between each one. If a heading does not exist in Predict, the supplied Natural Construct models create a prompt by taking the Predict field name, removing the dashes, and translating the words into mixed case.

Numeric Signs

Users can define packed fields with P or PS format in Predict. If you use PS, a sign position is reserved when the value is entered in the field. Similarly, you can define numeric fields with N or NS format (or U or US).

Superdescriptor Input Format

When you use a superdescriptor or compound key as the key for a maintenance or browse program, it is input as multiple fields. The number, format, and length of the multiple fields are determined by the number of underlying fields that make up the superdescriptor. If these underlying fields have been redefined, the first redefinition is used.

If you specify the multiple prompt option during generation, a separate prompt is built for each element of the superdescriptor.

Note: When you use a superdescriptor or subdescriptor as the prime key field, all fields that make up the descriptor must be defined in the view.

Verification Rules

You can create your own verification rules in the Predict data dictionary and then use them with Natural Construct. Natural Construct supports two types of Predict verification rules. Type N (Natural Construct) rules are incorporated into Object model programs. Type A (Automatic) rules are incorporated into maps where the Map model is supported.

Verification Rule Types

You can create verification rules of the following types:

Rule Type	Description
E	Equal
G	Greater
L	Less
N	Not equal
R	Range of values
T	Table of values
U	User routine
B	Range but not...
I	Not in range

When you use any type of verification rule other than U, you do not have to supply code when defining the rules in Predict. These rule types are simple conditions for which Natural Construct automatically generates the implementation in the appropriate target language, such as Natural (or Visual Basic if you are using Construct Spectrum). Any code you define in Predict for these rule types is ignored — Natural Construct will always implement the rule. To avoid unnecessary coding, use the other rule types (not U) wherever possible.

Use a U rule type when the verification rule is more complex than a simple condition and must be coded by hand in Predict. This rule type is discussed in detail in the next section.

Validating Rules in a Distributed Application

If you plan to divide your application into client and server components, you must decide how to validate verification rules. Regardless of how you partition the application, it is likely that the verification rules for a single-platform application will be valid for a distributed application. A little planning now can make reuse of rules easier.

Although it is good practice to validate verification rules within a single module, such as an object maintenance subprogram, you may want to validate a rule within the client component of an application. This saves the time required to send data to the object maintenance subprogram on the server for validation.

To determine whether to implement a rule within a server component or within both components (client and server), decide what types of information the rule requires to do its validation. For example:

- If the rule must look up data in a related file, it is best implemented within the server component (object maintenance subprogram), where access to the related file is available.
- If the rule is used by both distributed and non-distributed applications, it should be implemented within both the client components and the server components (object maintenance subprogram).

Defining U Rule Types

To minimize the number of rules you must define (for example, rules for the client components, rules for the server components, and rules for both), Natural Construct supports a new syntax convention for defining U rule types. This convention allows a single rule to contain a language-specific client implementation, a Natural implementation, or both.

Define language-specific rules in the Predict editor using code blocks. The syntax is:

```
>>BEGIN RULE <language code>
Language specific implementation of the VE rule here …
>>END-RULE
```

Currently, Natural Construct supports the Natural and Visual Basic programming languages.

Note: Although Natural Construct supports the syntax convention to define Visual Basic rules using Predict, you must use the models supplied with Construct Spectrum to make use of these rules.

Example of creating code blocks for a Visual Basic rule

```
>>BEGIN RULE VB
Visual Basic implementation of the VE rule here …
>>END-RULE
```

Example of creating code blocks for Natural rules

You can delimit Natural rules with code blocks, although this is not required:

```
>>BEGIN RULE Natural
Natural implementation of the VE rule here …
>>END-RULE
```

If any rule code is not delimited with a language-specific code block, Natural Construct assumes the rule was coded in Natural.

Example of creating code blocks for both Visual Basic and Natural rules

Users can also create a single rule consisting of several code blocks for both Visual Basic and Natural:

```
>>BEGIN RULE VB
1st part of Visual Basic implementation of the VE rule…
>>END-RULE
>>BEGIN RULE Natural
1st part of Natural implementation of the VE rule…
>>END-RULE
>>BEGIN RULE VB
2nd part of Visual Basic implementation of the VE rule…
>>END-RULE
** This code is defaulted to be Natural code because it is
** not delimited by a language-specific code block.
2nd part of Natural implementation of the VE rule…
>>BEGIN RULE Natural
3rd part of Natural implementation of the VE rule…
>>END-RULE
```

When combining Visual Basic and Natural rules, you cannot use nested language-specific code blocks. For example:

Use this:	**Not this:**

`>>BEGIN RULE VB`
`1st part of Visual Basic rule...`
`>>END-RULE`
`This VB code block is invalid`
`>>BEGIN RULE NATURAL`
`1st part of NATURAL rule...`
`>>END-RULE`
	`>>BEGIN RULE NATURAL`
	`VE rule...`
	`>>BEGIN RULE VB`
	
	`>>END-RULE`
	`>>END-RULE`

Referential Integrity Rules

Another type of rule is referential integrity. With this type, the generated maintenance programs do not allow a purge to be completed if the values being purged exist in related files. In addition, the generated programs do not allow a record to be added or modified if it contains a value that does not exist in a related table.

DB2 referential integrity rules are automatically processed by all models that support referential integrity rules: Object-Maint-Subp, Object-Maint-Dialog, Object-Maint-Dialog-Subp, Maint, and Batch.

➢ To add a referential integrity rule:

1 Invoke the Predict Relation Maintenance panel.

2 Enter “A” and a relationship ID to display the Add Relation panel.

3 Enter one file name and the prime key for the parent file of the relationship.

4 Enter a second file name and the foreign key for the child file of the relationship.

The relationship type must be N (Natural Construct). For DB2 users, the relationship type must be N or R (Referential constraint). The cardinality of the relationships must be 1 or C for the parent file and CN or N for the child file.

Example of cardinality of relationships in files

```
Customer                Order
CUST-NO (N5)            ORDER-NO (N7)
NAME (A20)              CUST-NUM (N5)
ADDRESS (A20)           VALUE (N7.2)
```

The prime key of Customer is CUST-NO and the corresponding foreign key in Order is CUST-NUM.

Assume the following relationship is specified:

```
15:33:26                  P r e d i c t  4.1.2                      10-19
                           Add a Relationship
Relationship .... CUSTOMER-ORDER
Type ...........* N  Natural Construct
Keys ..                                                        Zoom: N

Cardinality ..* 1  : CN
File 1                                          Minimum   Average    Maximum
  File ID ....* CUSTOMER                        1         1.00       1
  Field ID ...* CUST-NO
File 2
  File ID ....* ORDER                                     50.00
  Field ID ...* CUST-NUM
Constraint attributes
  Update type .....* R  Restrict
  Delete type .....* R  Restrict
  Constraint name ..

Comments     Zoom: N



EDIT:   Owner: N   Desc: Y
```

Predict Add a Relationship Panel

For this relationship, Natural Construct ensures that:

- The customer maintenance program does not purge a customer if orders exist for that customer.
- Orders cannot be added if the customer does not exist.

Types of Relationships Processed

Only the following types of relationships are processed by the supplied models:

- For all users, type N (Natural Construct) relationships.
- For DB2 users, type N (Natural Construct) and type R (Referential constraint) relationships.

Log Fields in Natural Construct

To use certain built-in features of the Natural Construct Maint model, several Predict fields are required. These fields all begin with the LOG- prefix. Natural Construct ignores all Predict fields beginning with LOG- when generating standard input and display panels. (DB2 fields begin with the LOG_ prefix.)

LOG-COUNTER Field

LOG-COUNTER is an optional field that can be added to the Predict definition of a file accessed by the Natural Construct Maint model. Use this field when the maintenance program you are generating will reread the record after the INPUT statement for an update or delete action. This ensures that the record is not held by Adabas across the INPUT statement when displaying records. Instead, each time an update to a record occurs, the LOG-COUNTER field is adjusted by one.

After a record is reread for updating, the value of the current LOG-COUNTER field is compared to the value of the previous LOG-COUNTER field (which was recorded when the record was initially read). If the value changed, an intervening update by another user occurred. When this happens, a message is displayed instructing the user to obtain a fresh copy of the record and reapply the updates.

A difference between the two types of maintenance programs is only noticed if two users try to simultaneously access the record at the same time. Without a LOG-COUNTER field, Adabas automatically backs one user out of the database. With a LOG-COUNTER field, one user is informed of intervening updates and asked to try the update again.

Users can be timed out when a LOG-COUNTER field is used. This occurs when you hold a record from the time the record is read until either the record is updated or the user leaves the maintenance program.

DB2 Users

If more than one user maintains a table at the same time, a LOG_COUNTER field must exist in the table to protect against the loss of intervening updates.

DL/1 or VSAM Users

If more than one user maintains a file at the same time, a LOG-COUNTER field must exist in the file to protect against the loss of intervening updates.

Other Log Fields

The remaining log fields are used by the Maint and Object-Maint-Subp models to implement Natural Construct's automatic update logging feature. To request automatic logging during the generation of a maintenance program, specify a log file to store the logging information. The log file may contain all or some of the same fields as the file being logged. All common fields in the two files are written to the log file.

The following log fields must be included in the log file. The field order, as well as the two-character Adabas field names, may be arbitrarily chosen.

Log Field	Description
LOG-ACTION (A1)	Action that triggered the generation of the log record. This field contains your installation's update action associated with added, modified, or purged records.
LOG-USER (A8)	User ID of the person who performed the action. The value that assigns this field in the maintenance program comes from the PASS.##USER global variable (which is one of the Natural Construct required global variables). When implementing logging, the PASS.##USER variable should be assigned upon entering the system. The value assigned can come from the Natural *INIT-USER system variable or any other user-defined source.
LOG-TID (A8)	ID of the terminal on which the action occurred. This value comes from the Natural *INIT-ID system variable.
LOG-DATE (N8)	Date the action occurred.
LOG-TIME (N7)	Time the action occurred.
Note:	In the redefinition, the LOG-TIME field is split into TIME (four digits) and FILLER (three digits). Consequently, users only have to enter the time within a one-minute accuracy to specify a starting time when using the LOG-BY-TIME superdescriptor.
Note:	The LOG-TIME field can optionally be defined with format T. If the LOG-TIME field is defined with format T, LOG-DATE is not necessary because the date is embedded in the time variable.

Optional Superdescriptors

To view log records in a logical sequence, it may be desirable to include superdescriptors in the log file (these superdescriptors need not exist in the file being logged). The components of these superdescriptors vary significantly, depending on the nature of the file being logged.

Example of log superdescriptors

The following example demonstrates how the log superdescriptors can be defined in a file. In this example, the goal is to log changes to the employees file, as well as display the logs in time sequence and in PERSONNEL-ID sequence. The following superdescriptors are included in the EMPLOYEES-LOG file:

```
  ——- Field name ——------——   -Start-   -End-
1: LOG-DATE                      1        8
2: LOG-TIME                      1        7
3:
4:
5:

  —-— Field name ——------——    -Start-   —End—
1: PERSONNEL-ID                  1        8
2: LOG-DATE                      1        8
3: LOG-TIME                      1        7
4:
5:
```

Predict Field Description Help for Objects

You can access Predict extended field descriptions for passive field-level help. When help is requested within a data object, Predict's field descriptions are hidden from the dialogs. An algorithm is provided that follows a sequence of steps to display help information: CD-HELPR.

Example of CD-HELPR invoked for an attribute of an object PDA

```
ORDERPDA.ORDER-NUMBER(HE='CD-HELPR',=)
```

CD-HELPR checks the Natural Construct system file for database help with the ORDERPDA major component and the ORDER-NUMBER minor component. If no help is found, the following priorities are observed:

1 Exact match
Find the ORDER-NUMBER field in the ORDERPDA file.

2 Field match with file keyword/comment match
Find the ORDER-NUMBER field belonging to a file that has an ORDERPDA keyword or contains ORDERPDA as part of its comments.

3 Standard file
Find the ORDER-NUMBER field in a Predict standard file.

4 Physical file
Find the ORDER-NUMBER field in a file of type Adabas, DB2, VSAM, or DL/1.

5 Any file
Find any Predict ORDER-NUMBER field that has a description.

6 No available help
No help is available for the specified field.

IMS DL/1 Support

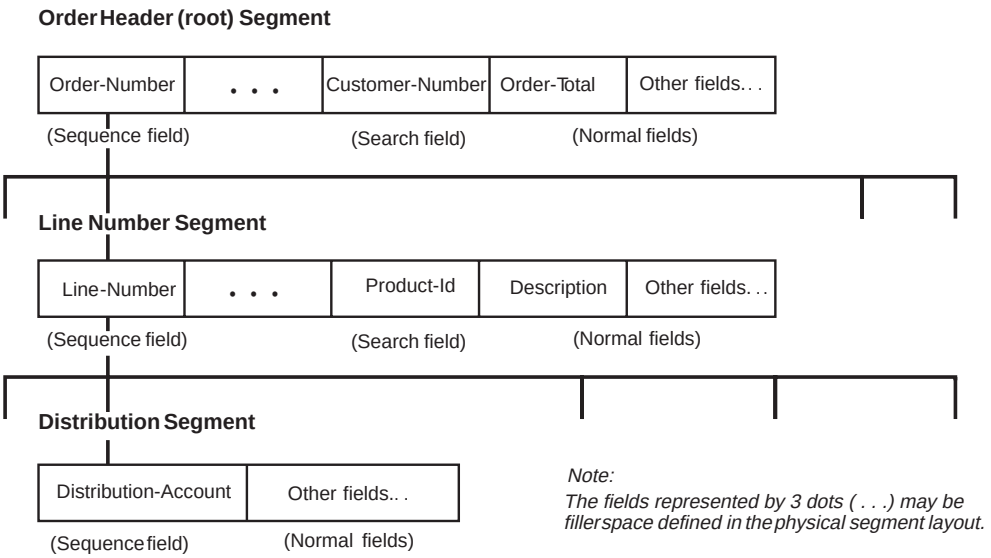
All Natural Construct models support the generation of programs that access IMS files. As with all other supported DBMS, the model specification panels are DBMS-independent.

Predict allows you to document three IMS file types: I, J, and K (for more information, refer to the Predict documentation). Maintenance models support only type I and J files, because type K files could contain fields that overlap each other in the master file. Browse models support all three file types.

IMS Databases

Information Management System (IMS) is classified as a hierarchical database. The physical design may be thought of as a tree structure where the nodes are called “segments” and the roots are called “root segments” in IMS terminology. Generally, a DDM is generated for each segment (that is, 1 segment = 1 file in FDIC). In addition, any child segment DDMs will contain fields from their ancestor segments that could be specified in a segment search argument. These are needed to trace the hierarchic path from the root segment down to the current segment (that is, Child segment DDMs inherit all the key fields from their ancestors).

Example of an IMS Database



Example of an IMS Database

The only method of documenting an IMS database in Predict is to incorporate the database into Predict using the incorporation function (for more information, refer to the Predict documentation). Incorporation of the database results in the creation of three segment (type I) files, one for each node in the tree structure. These files may not be modified. No fields may be added or deleted other than by modifying the IMS database and then re-incorporating the database (all changes are then rippled to related type J and K files).

Generally, type I files only document descriptor fields for the segment.

Example of descriptor types

```
Q  Sequence Fields
D  Search Fields
A  Alternate Index Fields
```

These fields are not necessarily located contiguously within the physical segment layout. Any fields that are not descriptor fields are called UDFs (user defined fields). These fields are documented in the type J fields that are related to a master I file. A type I file may have many type J files associated with it.

Type J files are analogous to Adabas user views. Each type J file documents a type I file and is called a *segment layout*. Field definitions of a segment layout have the same structure as definitions for a sequential field — the position of a field cannot be specified directly but is determined by its offset. The offset is calculated from the lengths of the fields already defined.

Because type J files define the contiguous physical layout of the segment and the key fields defined in the related I master file may not necessarily be in sequential order in the physical layout, care must be taken when defining UDFs in type J files to ensure that they do not unintentionally overlap key fields. To display the offsets of key fields for a segment in Predict, use the List File with Children function.

UDF Classifications Allowed

Natural Construct allows three classifications of UDFs: Dummy fields, Normal fields, and UDF redefinition of key fields.

Dummy Fields

Dummy fields denote unused areas in a segment. A field name of *dummy* can be specified more than once in the layout if necessary. Natural Construct ignores dummy fields when generating user views or object PDAs.

Normal Fields

Normal fields are any fields that do not overlap key fields (for more information about key field IMS offset, refer to the I master file section in the Predict documentation). These fields are not used for search criteria.

UDF Redefinition of Key Fields

A UDF is considered a redefinition of a key if it has the same length and offset as defined in the key field of the I master file. The redefinition field and the field name of the key being defined may have different field names.

When generating user views or object PDAs, Natural Construct always includes all segment ancestor keys and the current segment keys unless a segment key has been redefined as a UDF. In this case, the UDF redefinition is used in the view in place of the key field name that it has redefined.

Natural Construct also recognizes an IMS key field that was redefined by a UDF to be a group field — as long as the lengths of the group field components add up to the length of the IMS key field. Natural Construct rejects any UDF key redefinitions that do not exactly match in length the IMS key field that starts at the same offset.

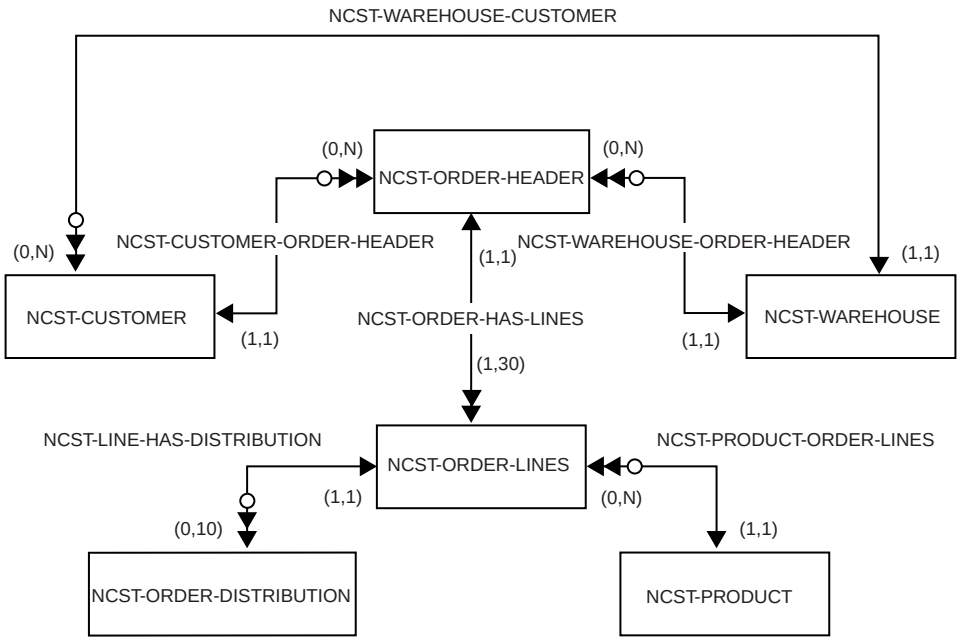
If you specify a UDF key field redefinition, you can use the UDF name as the key field name on Natural Construct's specification panel, even though they are not marked as descriptors in the DDM. Natural Construct recognizes the redefinitions and retrieves the appropriate DDM key names to build the search criteria.

Note: Use caution when defining non-key UDFs as a group field. If the type J file only contains one group field intended to define all non-key UDFs for the segment, Natural Construct will not verify that the group field does not overlap any IMS key fields.

Natural Construct Demo System

The model examples in this documentation are based on an order entry system in the Natural Construct demo system. The demo library name is SYSCSTDE (Adabas), SYSCSTD2 (DB2), SYSCSTDV (VSAM), or SYSCSTDS (Unix).

The demo system accesses files containing customer, warehouse, product, and order information. The following diagram illustrates the order entry files (user views) and their relationships:



Order Entry System — Files and Relationships

The user views, relationships, and Predict rules used by the application exist in Predict under the names listed on the following pages. The output examples used in this documentation are based on these user views, relationships, and verification rules.

User Views

File	Contents
NCST-ORDER-HEADER	Basic data for the order, such as the date, customer ID, warehouse ID, and total amount of the order.
NCST-ORDER-LINES	Lines for each order, identifying the products ordered.
NCST-ORDER-DISTRIBUTION	Ledger information belonging to an order. For each order line, there is a set of records in this file that hold the accounts and the amount that should be posted to the General Ledger. There must be at least one record in the NCST-ORDER-DISTRIBUTION file for each order line. An order object is made up of a header file, a number of product lines, and the account distribution information for each line.
NCST-CUSTOMER	Information about all customers, such as the customer ID, name, and address.
NCST-PRODUCT	Information related to the products, such as the product ID, description, reorder point, and manufacturer's address.
NCST-WAREHOUSE	Information about the warehouses, such as the warehouse ID, name, and address.

Relationships

Relationship	Description
NCST-ORDER-HAS-LINES	Describes the relationship between the order header and order lines. Each order has one or many lines, but each line corresponds to only one order.
NCST-PRODUCT-ORDER-LINES	Describes the relationship between each order line and the product on that line. Each line has one product, but a product can be ordered on many lines.
NCST-LINE-HAS-DISTRIBUTION	Describes the relationship between the order lines and the distribution information. Each line has at least one distribution record attached to it, but each distribution record corresponds to only one line of an order.
NCST-CUSTOMER-ORDER-HEADER	Describes the relationship between the customer and the order. Each order is issued to only one customer, but a customer can have many orders.

Relationship	Description (continued)
NCST-WAREHOUSE-ORDER-HEADER	Describes the relationship between the warehouse and the order. Each order corresponds to only one warehouse, but a warehouse contains products for many different orders.
NCST-WAREHOUSE-CUSTOMER	Describes the relationship between the warehouse and the customer. Each customer can have one default warehouse ID, but a warehouse can be used by many customers.

Predict User Views

The user views displayed on the following pages are listed in Predict.

Example of NCST-CUSTOMER user view

```
Ty L             Name                    F   Length  D U DB S
-- - ------------------------------------ -   ------  - - -- -
   1 CUSTOMER-NUMBER                      N      5.0  D   XY N
   1 BUSINESS-NAME                        A     30.0  D   XZ N
   1 PHONE-NUMBER                         N     10.0      X0 N
GR 1 MAILING-ADDRESS                                     X1
   2 M-STREET                             A     25.0      X2 N
   2 M-CITY                               A     20.0      X3 N
   2 M-PROVINCE                           A     20.0      X4 N
   2 M-POSTAL-CODE                        A      6.0      X5 N
GR 1 SHIPPING-ADDRESS                                    X6
   2 S-STREET                             A     25.0      X7 N
   2 S-CITY                               A     20.0      X8 N
   2 S-PROVINCE                           A     20.0      X9 N
   2 S-POSTAL-CODE                        A      6.0      YA N
   1 CONTACT                              A     30.0      YB N
   1 CREDIT-RATING                        A      3.0      YC N
   1 CREDIT-LIMIT                         P     11.2      YD N
   1 DISCOUNT-PERCENTAGE                  P      3.2      YE N
   1 CUSTOMER-WAREHOUSE-ID                A      3.0  D   YF N
   1 CUSTOMER-TIMESTAMP                   T               YG N
```

Example of NCST-ORDER-HEADER user view

```
Ty L             Name                    F   Length  D U DB S
-- - ------------------------------------ -   ------  - - -- -
*
*    NCST-ORDER-HEADER
*
   1 ORDER-NUMBER                         N      6.0  D   XA N
   1 ORDER-AMOUNT                         P     13.2      XB N
   1 ORDER-DATE                           N      8.0      XC N
   1 ORDER-CUSTOMER-NUMBER                N      5.0  D   XD N
   1 ORDER-WAREHOUSE-ID                   A      3.0  D   XE N
   1 INVOICE-NUMBER                       N      6.0      XF N
   1 ORDER-TIMESTAMP                      T               XG N
MC 1 DELIVERY-INSTRUCTIONS                A     60.0      XH N
SP 1 CUSTOMER-DATE                        B     13.0  D   XI N
```

Example of NCST-ORDER-LINES user view

```
Ty L             Name                    F   Length  D U DB S
-- - ------------------------------------ -   ------  - - -- -
*
*    NCST-ORDER-LINES
*
   1 LINE-ORDER-NUMBER                    N      6.0      XJ N
   1 LINE-NUMBER                          N      2.0      XK N
   1 ORDER-PRODUCT-ID                     A      6.0  D   XL N
   1 LINE-DESCRIPTION                     A     40.0      XM N
   1 QUANTITY                             P      9.0      XN N
   1 UNIT-COST                            P      7.2      XO N
   1 TOTAL-COST                           P      9.2      XP N
SP 1 ORDER-LINE-KEY                       N      8.0  D   XQ N
SP 1 PRODUCT-ORDER                        A     14.0  D   XR N
```

Example of NCST-ORDER-DISTRIBUTION user view

```
Ty L             Name                    F   Length  D U DB S
-- - ----------------------------------- -   ------  - - -- -
*
*    NCST-ORDER-DISTRIBUTION
*
   1 DIST-ORDER-NUMBER                   N      6.0      XS N
   1 DIST-LINE-NUMBER                    N      2.0      XT N
   1 DIST-NUMBER                         N      2.0      XU N
RE 1 ACCOUNT                             A      9.0      XV N
   2 COST-CENTER                         A      2.0
   2 ACCT                                A      4.0
   2 PROJECT                             A      3.0
   1 DIST-AMOUNT                         P      9.2      XW N
SP 1 ORDER-DIST-KEY                      B     10.0  D   XX N
```

Example of NCST-WAREHOUSE user view

```
Ty L             Name                    F   Length  D U DB S
-- - ----------------------------------- -   ------  - - -- -
*
*    NCST-WAREHOUSE
*
   1 WAREHOUSE-ID                        A      3.0  D   YH N
   1 WAREHOUSE-DESCRIPTION               A     30.0      YI N
GR 1 WAREHOUSE-ADDRESS                                   YJ
   2 WAREHOUSE-STREET                    A     20.0      YK N
   2 WAREHOUSE-CITY                      A     20.0      YL N
   2 WAREHOUSE-PROVINCE                  A     20.0      YM N
   2 WAREHOUSE-POSTAL-CODE               A      6.0      YN N
```

Example of NCST-PRODUCT user view

```
Ty L             Name                    F   Length  D U DB S
-- - ----------------------------------- -   ------  - - -- -
*
*    NCST-PRODUCT
*
   1 PRODUCT-ID                          A      6.0  D   YO N
   1 PRODUCT-DESCRIPTION                 A     50.0      YP N
   1 PRODUCT-REORDER-POINT               P      7.0      YQ N
   1 PRODUCT-UNIT-COST                   P      7.2      YR N
   1 PRODUCT-MANUFACTURER                A     30.0      YS N
GR 1 MANUFACTURER-ADDRESS                                YT
   2 MANUFACTURER-STREET                 A     20.0      YU N
   2 MANUFACTURER-CITY                   A     20.0      YV N
   2 MANUFACTURER-PROVINCE               A     20.0      YW N
   2 MANUFACTURER-POSTAL-CODE            A      6.0      YX N
```

These user views apply to Adabas users. For DB2 users, field names with a dash must be replaced by an underscore (for example, CUSTOMER-NUMBER becomes CUSTOMER_NUMBER). Certain fields may also have null indicators. For DB2 and VSAM users, at least one field in each file must be defined as unique. (For example, in the CUSTOMER file, CUSTOMER-NUMBER should be defined as unique.)

Predict Relationships

The following examples show the relationships between files and how the relationships appear in Predict.

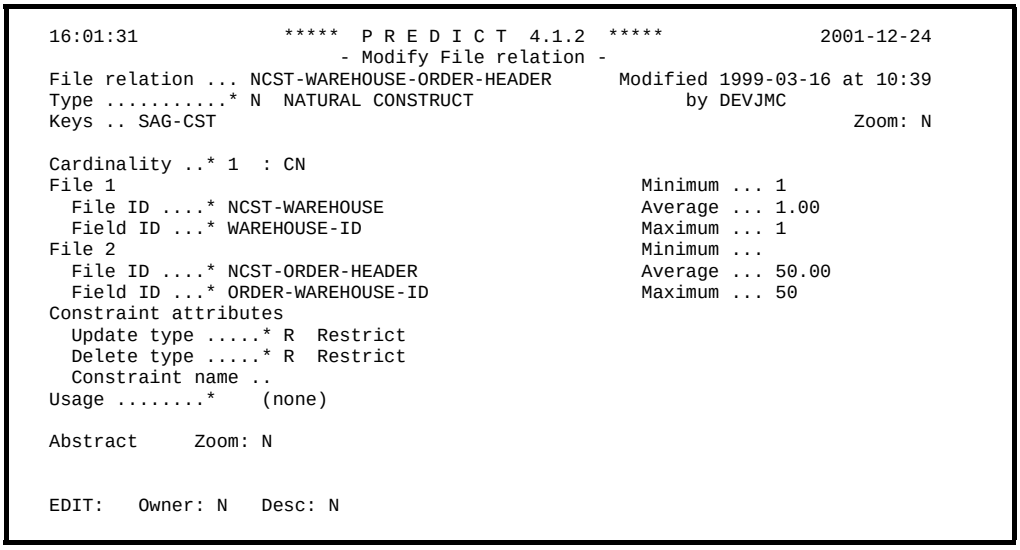
NCST-CUSTOMER-ORDER-HEADER

```
16:01:31            *****  P R E D I C T  4.1.2  *****              2001-12-24
                          - Modify File relation -
File relation ... NCST-CUSTOMER-ORDER-HEADER       Modified 1999-03-16 at 10:39
Type ...........* N  NATURAL CONSTRUCT                   by DEVJMC
Keys .. SAG-CST                                                      Zoom: N

Cardinality ..* 1  : CN
File 1                                              Minimum ... 1
  File ID ....* NCST-CUSTOMER                       Average ... 1.00
  Field ID ...* CUSTOMER-NUMBER                     Maximum ... 1
File 2                                              Minimum ...
  File ID ....* NCST-ORDER-HEADER                   Average ... 50.00
  Field ID ...* ORDER-CUSTOMER-NUMBER               Maximum ... 50
Constraint attributes
  Update type .....* R  Restrict
  Delete type .....* R  Restrict
  Constraint name ..
Usage ........*    (none)

Abstract     Zoom: N


EDIT:   Owner: N   Desc: N
```

NCST-CUSTOMER-ORDER-HEADER Relationship

The NCST-CUSTOMER-ORDER-HEADER relationship specifies that an order cannot be added or updated without specifying a valid customer number from the NCST-CUSTOMER file, and a customer cannot be deleted if orders for that customer exist in the NCST-ORDER-HEADER file. This is called an inter-object relationship because the NCST-CUSTOMER and NCST-ORDER-HEADER files are parts of different objects. Notice that both the Update and Delete constraint types are restricted. For more information, see **Design Methodology**, page 141.

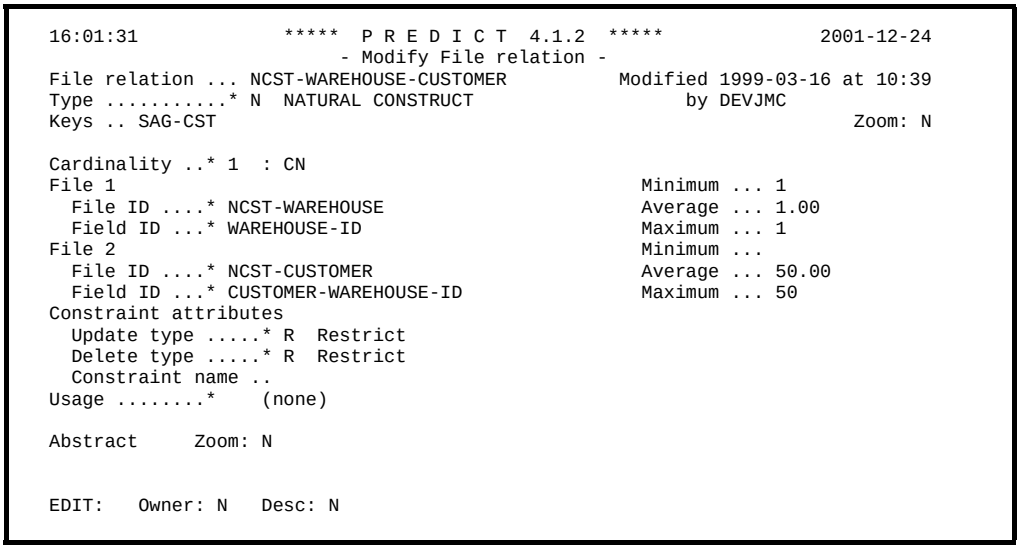
NCST-WAREHOUSE-ORDER-HEADER



NCST-WAREHOUSE-ORDER-HEADER Relationship

The NCST-WAREHOUSE-ORDER-HEADER relationship specifies that an order cannot be added or updated without specifying a valid warehouse ID from the NCST-WAREHOUSE file. A warehouse cannot be deleted if orders for that warehouse exist in the NCST-ORDER-HEADER file. This is called an inter-object relationship.

NCST-WAREHOUSE-CUSTOMER



NCST-WAREHOUSE-CUSTOMER Relationship

The NCST-WAREHOUSE-CUSTOMER-HEADER relationship specifies that a customer cannot be added or updated without specifying a valid warehouse ID from the NCST-WAREHOUSE file. A warehouse cannot be deleted if customers for that warehouse exist in the NCST-CUSTOMER file. This is called an inter-object relationship.

NCST-ORDER-HAS-LINES

```
16:10:29            *****  P R E D I C T  4.2.1  *****             2001-12-24
                         - Modify File relation -
File relation ... NCST-ORDER-HAS-LINES             Modified 1998-08-24 at 11:37
Type ...........* N  NATURAL CONSTRUCT                   by DEVSS
Keys .. SAG-CST                                                    Zoom: N

Cardinality ..* 1  : CN
File 1                                               Minimum ... 1
  File ID ....* NCST-ORDER-HEADER                    Average ... 1.00
  Field ID ...* ORDER-NUMBER                         Maximum ... 1
File 2                                               Minimum ...
  File ID ....* NCST-ORDER-LINES                     Average ... 5.00
  Field ID ...* ORDER-LINE-KEY                       Maximum ... 30
Constraint attributes
  Update type .....* N  re-Number suffix
  Delete type .....* C  Cascade
  Constraint name ..
Usage ........*    (none)

Abstract     Zoom: N


EDIT:   Owner: N   Desc: N
```

NCST-ORDER-HAS-LINES Relationship

The NCST-ORDER-HAS-LINES relationship identifies NCST-ORDER-HEADER and NCST-ORDER-LINES as being parts of the same object (intra-object relationship) because the Delete constraint type is Cascade and the Cardinality is 1:N. The generated object subprogram ensures that an order has at least one line and no more than 30 lines.

The Update constraint type N (Re-Number suffix) indicates that the suffix portion of ORDER-LINE-KEY represents a line number (rather than a user-modifiable data field) and line numbers will be renumbered if a line is removed during the updating of an order. For more information about intra-object relationships and object definitions, see **Design Methodology**, page 141.

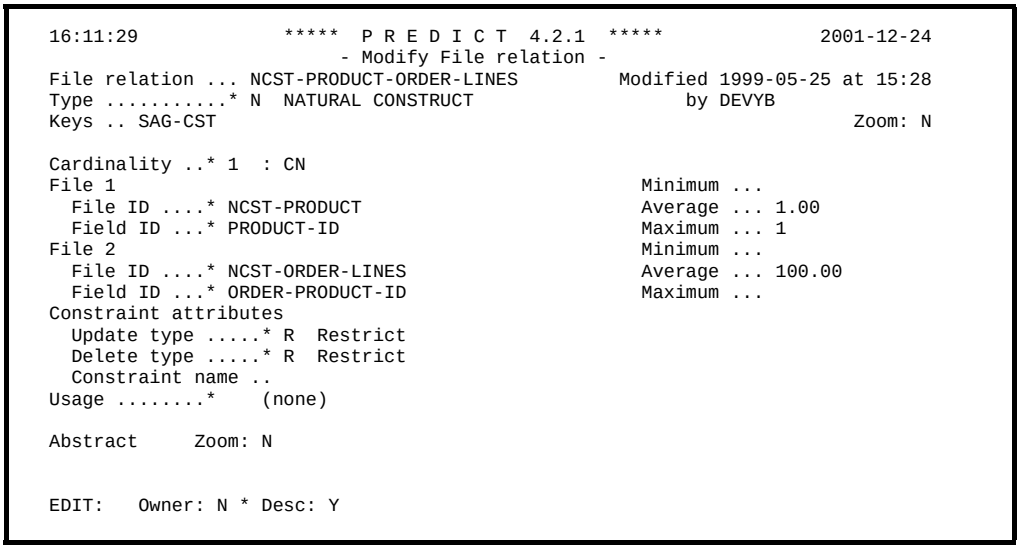
NCST-LINE-HAS-DISTRIBUTION

```
16:10:58            *****  P R E D I C T  4.2.1  *****             2001-12-24
                         - Modify File relation -
File relation ... NCST-LINE-HAS-DISTRIBUTION       Modified 1999-03-15 at 10:12
Type ...........* N  NATURAL CONSTRUCT                   by DEVJMC
Keys .. SAG-CST                                                    Zoom: N

Cardinality ..* 1  : CN
File 1                                              Minimum ... 1
  File ID ....* NCST-ORDER-LINES                    Average ... 1.00
  Field ID ...* ORDER-LINE-KEY                      Maximum ... 1
File 2                                              Minimum ...
  File ID ....* NCST-ORDER-DISTRIBUTION             Average ... 5.00
  Field ID ...* ORDER-DIST-KEY                      Maximum ... 10
Constraint attributes
  Update type .....* N  re-Number suffix
  Delete type .....* C  Cascade
  Constraint name ..
Usage ........*    (none)

Abstract     Zoom: N


EDIT:   Owner: N   Desc: N
```

NCST-LINE-HAS-DISTRIBUTION Relationship

NCST-LINE-HAS-DISTRIBUTION is also an intra-object relationship, identifying NCST-ORDER-DISTRIBUTION as part of the order object.

NCST-PRODUCT-ORDER-LINES



NCST-PRODUCT-ORDER-LINES Relationship

The NCST-PRODUCT-ORDER-LINES relationship specifies that an order line must contain a valid product, and a product in the NCST-PRODUCT file cannot be deleted if it is still referenced in the NCST-ORDER-LINES file.

Predict Verification Rules

The following figures illustrate the NCST-NON-BLANK verification rule used in the demo system:

```
13:55:13                  P r e d i c t  4.1.2                      10-19
                          Display Verification                   Page:   1
Verification ID .... NCST-NON-BLANK
Status ............. Natural Construct      Added: 07-13 at 12:54 by SAG
                                         Modified: 10-19 at 13:54 by SAG
Verification attributes
  Format ........ Alphanumeric
  Compat. format
  Type .......... User routine
  Message nr ....
  Replacement 1 .
  Replacement 2 .
  Replacement 3 .
  Message text ..
Processing rule
  0010 IF &1& = ' ' THEN
  0020   ASSIGN MSG-INFO.##MSG = 'MUST NOT BE BLANK'
  0030 END-IF
***** End of Report *****
MORE
```

NCST-NON-BLANK Verification Rule

The NCST-NON-BLANK verification rule states that an alphanumeric field cannot be given a blank value.

```
13:57:33                  P r e d i c t  4.1.2                      10-19
                      List Verification with Parents              Page:   2
Verification ID .... NCST-NON-BLANK
Status ............. Natural Construct
File ID                           Field ID                       F   Length
* NCST-CUSTOMER                   CONTACT                        A     30.0
NCSTVSAM-CUSTOMER                 CONTACT                        A     30.0
```

NCST-CUSTOMER File Using the NCST-NON-BLANK Rule

The NCST-NON-BLANK verification rule is linked to the Contact field of both the NCST-CUSTOMER file and the NCSTVSAM-CUSTOMER file.

The following example shows the NCST-NUMERIC-NONZERO verification rule:

```
15:14:34                  P r e d i c t  4.1.2                      10-18
                          Display Verification                    Page:   2
Verification ID .... NCST-NUMERIC-NONZERO
Status ............. Natural Construct       Added: 08-25 at 09:41 by SAG
                                           Modified: 08-25 at 09:42 by SAG
Verification attributes
  Format ........ Numeric
  Compat. format
  Type .......... User routine
  Message nr ....
  Replacement 1 .
  Replacement 2 .
  Replacement 3 .
  Message text ..
Processing rule
  0010 IF &1& LE 0 THEN
  0020   ASSIGN MSG-INFO.##MSG = 'value must be greater than zero'
  0030 END-IF
***** End of Report *****
MORE
```

NCST-NUMERIC-NONZERO Verification Rule

This verification rule states that a numeric field cannot be given a zero value. The NCST-NUMERIC-NONZERO verification rule is linked to the Dist-Amount field of the NCST-ORDER-DISTRIBUTION file.

```
15:25:46                  P r e d i c t  4.1.2                      08-24
                      List Verification with Parents              Page:   2
Verification ID .... NCST-NUMERIC-NONZERO
Status ............. Natural Construct
File ID                          Field ID                       F   Length
* NCST-ORDER-DISTRIBUTION        DIST-AMOUNT                    P      9.2
```

NCST-ORDER-DISTRIBUTION File Using NCST-NUMERIC-NONZERO Rule

Note: The Status field for the verification rules must be Natural Construct. See the examples above and on the previous page.

CREATING NATURAL COMMAND PROCESSORS

This chapter provides a brief overview of Natural command processors, with reference to features that are specific to Natural Construct-generated applications.

The following topics are covered:

Introduction

The Natural SYSNCP utility allows you to create and maintain Natural command processors. These processors are made up of two parts: the development component and the runtime component.

Development Component

To create the development component of a command processor, use the SYSNCP utility to define the commands (combinations of keywords) and actions. Consequently, when a command is executed, the associated action is performed.

The SYSNCP utility generates decision tables from your command definitions. These tables are stored in a Natural “PROCESSOR” type member and determine what happens when a command is entered. For details about creating Natural command processors, refer to the SYSNCP chapter in the Natural documentation.

Runtime Component

The PROCESS COMMAND statement invokes the command processor within a Natural program. This statement includes the name of the SYSNCP table used to handle the data input by users. For more information, refer to the Natural documentation.

What is a Command?

A command is made up of any sequence of values entered on the Direct Command line which are recognized and processed by an application. Most commands contain the following two elements:

- Function
  One or more valid keywords (such as BROWSE FILE).
- Command data
  Variable information attached to a function (such as the name or ID number of a customer).

Commands are always executed from within an application — the place from which the command is executed is called the location. Commands take the user from one location to another. In Natural Construct-generated applications, the user is always returned to the main menu location after a command action has completed successfully.

Commands must be labelled as either global or local. Global commands may be executed from any location, whereas local commands may only be executed from pre-defined locations. We recommend that you make all processor commands globally accessible.

Creating a Natural Command Processor

Several distinct steps are required to create a command processor. These include defining the header, keywords, and functions, then linking each action to a function. The SYSNCP utility provides different editors to define the keywords, functions, and runtime actions.

By following the steps outlined in the Natural Utilities documentation, you can create a Natural command processor that allows users to navigate through an application more easily with the use of processor commands. Some conventions for creating command processors are presented in **Design Methodology**, page 141. These conventions adhere to the recommended design methodology for Natural Construct-generated applications.

Header Maintenance Facility

To define global settings for a command process, use the header maintenance function. There are six maintenance panels, all described in the Natural Utilities documentation. The following example shows the fourth Processor Header Maintenance panel:

```
16:46:40             ***** Natural Command Processor *****               10-30
NCP-M005              - Processor Header Maintenance 4 -                   SAG

Modify Header                    Name NCPDEMO  Library SYSCSTDE DBID 6   FNR 4
Created by SAG      Date 03-02                           Current Status Obj-Prep

Command Data Handling:
----------------------
   Data Delimiter ............... #

   Data allowed .................. S
   More than one item allowed ... N
   Max. length of each item ..... 99
   Item must be numeric ......... N
   Put to TOP of STACK .......... Y

   If Error, drop all data ...... Y




Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Cmd   Exit  Last  List  Flip  -     +                       Canc
```

Processor Header Maintenance Panel 4

The following two fields pertain to Natural Construct-generated applications. Set these fields as follows:

Field	Description
Data Allowed	One-character alphanumeric code indicating if data can be input at runtime. To move the data to the Natural stack (where it is allowed as input at runtime), enter “S”.
Note:	For more information, refer to the Natural documentation.
Put to TOP of STACK	To place the data at the top of the Natural stack, enter “Y”.

Function Editor

Use the function editor to compose valid commands and to specify whether a command is accessible globally or locally. The editor automatically generates all possible combinations of keywords from those stored in the keyword editor:

```
16:49:56              ***** Natural Command Processor *****                10-30
NCP-M400                     - Function Editor -                            SAG
Edit Global Combinations       Name NCPDEMO  Library SYSCSTDE DBID 6   FNR 4

Global
I Ac   Action           Object           Addition         Global Local Any Loc.
- --   ---------------- ---------------- ---------------- ------ ----- --------
       ADD
       ADD              CUSTOMER                            Yes
       ADD              ORDER                               Yes
       ADD              PRODUCT                             Yes
       ADD              TEXT
       ADD              WAREHOUSE                           Yes
       BROWSE
       BROWSE           CUSTOMER                            Yes
       BROWSE           ORDER                               Yes
       BROWSE           PRODUCT                             Yes
       BROWSE           TEXT
       BROWSE           WAREHOUSE                           Yes
Repos:                                                    ------ ----- --------

Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Cmd   Exit  Last  List  Flip             Top               Canc
```

Function Editor

To comply with Natural Construct standards, note the following two features of the function editor:

- The function editor allows you to selectively enable each keyword combination that will be a valid function in your application.
- To allow commands access from anywhere in the application, define all selected functions as global functions.

Runtime Action Definition Window

After you validate keyword combinations by creating functions (commands) in the function editor, link these functions to one or more runtime actions. Runtime actions are made up of one or more steps that are carried out when a command is issued.

Use the runtime action editor to define the actions that are carried out when a command is issued from a specific location:

```
                      Runtime Action Definition

Location ....  < Global >
Command ..... DISPLAY PRODUCT

Keep Location .... S
Data allowed ..... S   More than one .... Y   Max. Lgth ........ 99
Numeric .......... N   TOP of STACK ..... Y   Error: Drop ...... Y

A  Runtime Action Definition
-  ------------------------------------------------------------------
1  ACTION=DISPLAY_____________________________________________________
2  FETCH=NCTFRPRD_____________________________________________________
_  ___________________________________________________________________
_  ___________________________________________________________________
_  ___________________________________________________________________
_  ___________________________________________________________________
_  ___________________________________________________________________
_  ___________________________________________________________________
```

Runtime Action Definition Window

The runtime action definition table header consists of the bold lines in the above example. The values in the A fields represent the valid action codes, while the values in the Runtime Action Definition fields accept the input to be processed.

Enter the abbreviation for an action in the A field and any additional information needed to perform the runtime action (any parameters accompanying the action, for example) in the Runtime Action Definition field. (See the table on the following page for a complete list of these options.)

Note: The sequence in which actions are performed at runtime is determined by the order they are entered in the editor (top to bottom). If a FETCH is specified, all actions below are ignored.

The following table lists possible action codes and their runtime action definitions:

Action Code	Runtime Action Definition
V	Specifies no runtime action. This is the default value.
T	Specifies the text that can be read at runtime using the TEXT or GET subcommand of the PROCESS COMMAND statement.
M	Allows the command line to be modified (data is placed on the command line).
C	Places this command at the top of the Natural stack. If an asterisk (*) is entered, the name of the program that issued this PROCESS COMMAND statement is placed on the top of the stack.
D	Places this data at the top of the Natural stack.
F	FETCHes the specified Natural program.
S	Issues the Natural STOP statement at runtime.
E	Moves the specified value immediately to the Natural *ERROR-NR system variable.
R	Specifies the return code entered in the RETURN-CODE field.
1 to 9	Indicates the text string whose value is entered in the RESULT-FIELD field(s).
*	Indicates a comment line.

Implementing Object Maintenance Dialog Actions

Natural Construct recognizes specially coded runtime actions that implement the concepts of action code inheritance, object maintenance dialog name encapsulation, and functional security. For information, see **Design Methodology**, page 141.

The following example shows the Runtime Action Definition window for the DISPLAY PRODUCT runtime action:

```
                      Runtime Action Definition

Location ....  < Global >
Command ..... DISPLAY PRODUCT

Keep Location .... S
Data allowed ..... S   More than one .... Y   Max. Lgth ........ 99
Numeric .......... N   TOP of STACK ..... Y   Error: Drop ...... Y

A  Runtime Action Definition
-  ------------------------------------------------------------------
1  ACTION=DISPLAY_____________________________________________________
2  FETCH=NCTFRPRD_____________________________________________________
_  ___________________________________________________________________
_  ___________________________________________________________________
_  ___________________________________________________________________
_  ___________________________________________________________________
_  ___________________________________________________________________
_  ___________________________________________________________________
```

Runtime Action Definition Window
for the DISPLAY PRODUCT Runtime Action

You can code runtime actions for DISPLAY *OBJECT* commands as illustrated in the above example.

Command names and action codes are language independent, but should conform to the codes found in the #CODES table for the CDACT subprogram. For more information, see **#Action Parameters Panel**, page 228.

The first line in the Runtime Action Definition field indicates the type of action being performed. In the example on the previous page, the DISPLAY action is specified, followed by the name of an object maintenance dialog program to be FETCHed. (FETCHing this program processes the action.) Natural Construct-generated programs recognize the ACTION and FETCH keywords when processing command processor commands and place the current language action code on the Natural stack before FETCHing the specified object maintenance dialog program.

Because the Data Allowed field is set to “S”, the data displayed by the FETCHed program may have been placed on top of the Natural stack prior to executing the actions. For example, if the DISPLAY PRODUCT 123456 command was executed, the object maintenance dialog program is FETCHed with two items on the stack: “DIS” and “123456”.

Note: You should only use DISPLAY OBJECT commands to FETCH a maintenance dialog program.

Adding Runtime Actions

This section outlines how to add runtime actions for a maintenance dialog program. The following example shows runtime actions specific to the ADD action when using Natural Construct-generated applications:

```
                      Runtime Action Definition

Location ....  < Global >
Command ..... ADD PRODUCT

Keep Location .... S
Data allowed ..... S   More than one .... Y   Max. Lgth ........ 99
Numeric .......... N   TOP of STACK ..... Y   Error: Drop ...... Y

A  Runtime Action Definition
-  ------------------------------------------------------------------
1  ACTION=ADD_____________________________________________________
M  DISPLAY PRODUCT_____________________________________________________
_  _____________________________________________________
_  _____________________________________________________
_  _____________________________________________________
_  _____________________________________________________
_  _____________________________________________________
_  _____________________________________________________
```

Runtime Action Definition Window
for the ADD PRODUCT Runtime Action

Runtime actions for any object-dialog action other than DISPLAY can encapsulate the actual name of the object maintenance dialog program by always executing the DISPLAY *OBJECT* command to invoke the object-dialog indirectly. This avoids having to code a FETCH statement within the object-dialog for every command, restricting the FETCH to DISPLAY commands only. Additionally, this minimizes the impact of program name changes.

Runtime actions should be coded as in the previous example.

On the first line of the Runtime Action Definition field, specify the runtime action as follows:

`ACTION=`*`ACTION-NAME`*

where *ACTION-NAME* is a valid action name. To specify the action code, this name may be shortened to any length that uniquely identifies the action.

During execution of a command processor command, these runtime actions are interpreted by the CDNCP subprogram. The subprogram then places the corresponding action code at the top of the Natural stack in the format:

`ACTION:`*`ACTION-NAME`*

If any data was previously stacked (because the Data Allowed field was set to “S”), it is compressed with code and replaced on the stack. After the execution of line 1, for example, issuing the ADD PRODUCT 123456 command results in one item being placed on the stack (ACTION:ADD,123456, for example).

On the second line, the “M” runtime action code executes the DISPLAY PRODUCT command. This command FETCHes the same object maintenance dialog program for all actions that operate on the same object.

A secondary feature associated with coding runtime actions is the effect on functional security. All object maintenance dialog programs CALLNAT the CDACT subprogram, which returns a list of allowable action codes in the current runtime language.

The CDACT subprogram recognizes any action codes on the top of the stack with the ACTION prefix as action requests from the command processor command. (The ACTION text is formatted by the CDNCP subprogram when it processes Natural Construct-formatted command processor runtime actions.) When an action is requested by a command processor command, CDACT returns a list of valid action codes (actions allowed or inherited by the requesting action).

The example on the following page completes the progression of the last two screen examples. Executing the ADD PRODUCT 123456 command invokes the CDNCP subprogram to format the command as ACTION:ADD,123456 and FETCH the object maintenance dialog program (as defined in the DISPLAY PRODUCT runtime action). The CDACT subprogram is then called from the object maintenance dialog program. ACTION:ADD,123456 will be at the top of the stack.

The ADD action inherits the action codes for the MODIFY, BROWSE, NEXT, CLEAR, and DISPLAY actions, and only these action codes (M, B, N, C, and D) are returned to the object maintenance dialog program in the valid actions list.

The following example shows the runtime actions specific to the MAINTAIN action when using Natural Construct-generated applications:

```
                    Runtime Action Definition

Location ....  < Global >
Command ..... MAINTAIN PRODUCT

Keep Location .... S
Data allowed ..... Y   More than one .... Y   Max. Lgth ........ 99
Numeric .......... N   TOP of STACK ..... Y   Error: Drop ...... Y

A  Runtime Action Definition
-  ------------------------------------------------------------
1  ACTION=*_____________________________________________________
M  DISPLAY PRODUCT_____________________________________________
_  _____________________________________________________________
_  _____________________________________________________________
_  _____________________________________________________________
_  _____________________________________________________________
_  _____________________________________________________________
_  _____________________________________________________________
```

Runtime Action Definition Window
for the MAINTAIN PRODUCT Runtime Action

You can code runtime actions for object-dialog commands that place no restriction on the allowable actions as shown in the example above.

On line 1, ACTION=* indicates that no restrictions apply to the list of valid action codes returned to the object-dialog-maintenance program.

Note: CDACT always returns a three-character action code on top of the Natural stack when processing action codes requested by a command processor command.

USE OF EXTERNAL OBJECTS

This chapter describes how Natural Construct-generated applications use external objects to provide standard functionality for all generated applications.

The following topics are covered:

Development Libraries

When Natural Construct is installed, the source and object code for all modules is stored in the SYSCST library (source code for CDU* modules is not supplied). The object code for subprograms, helproutines, and data areas, as well as the source code for map layouts and copycode, is stored in the SYSTEM library. For most of the supplied objects, there is no need to make a separate copy for each application unless you want to change the functionality of an object.

One object that must be copied into each development library is the global data area (GDA) used by the generated applications.

Production Libraries

When applications generated with the supplied modules move into production, some of the external objects the application uses may also have to be copied into the production library (or steplib). This is normally done once by your Natural Construct administrator so all subsequent applications can use these modules.

Note: If you modify a supplied object, copy it into the production application library where the object is used so as not to impact users of the SYSTEM library version.

The generated applications do not require source code when they are run in production.

You must copy the modules for all programs, subprograms, helproutines, and maps beginning with CD from the SYSCST library into your production library (or steplib).

Description of Supplied External Objects

This section describes the external objects supplied with Natural Construct, as well as the relationships among them. The supplied external objects fall into one of the following categories:

- Helproutines and related objects
- Subprograms and related objects
- Common data areas
- Programs
- Map layouts

Helproutines and Related Objects

Applications generated with the supplied models contain the code necessary to integrate with the Natural Construct Help Text subsystem. The following modules provide passive (descriptive) help in your generated applications:

Help Module	Description
CD-HELP	Subprogram that displays help.
CD-HELPL	Subprogram that displays help for large help pages (more than 18 lines per page).
CD-HELPR	Helproutine used as the HE parameter for fields and maps. This invokes CD-HELP, CD-HELPL, or CD-HPRED.
CD-HPRED	Subprogram that displays Predict field descriptions.
CDACTHLP	Helproutine that invokes the CD-HELP subprogram for selected action codes. If desired, you can associate this helproutine with an action field and pass the helproutine the set of valid actions. For example: #ACTION (HE='CDACTHLP',#VAL-ACT)
CDACTHL2	Helproutine that invokes the CD-HELP subprogram for selected action codes. If desired, you can associate this helproutine with an action field and pass the helproutine the set of valid actions. For example: #ACTION (HE='CDACTHL2, CDACTA.#APPL-ACTIONS(*,*))

Help Module	Description (continued)
CDDCHELP	Helproutine used as the HE parameter on the direct command line for maps and INPUT statements in generated applications. If a Natural command processor is active, CDDCHELP invokes the CDNCPH subprogram to display all available commands. For example: CDGETDCA.#DIRECT-COMMAND (HE='CDDCHELP')
CDNCPH	Browse select subprogram that displays the commands available on the direct command line when a Natural command processor is in use (invoked by CDDCHELP).

Subprograms and Related Objects

To provide standard functions within each Natural Construct-generated application, the applications share the following subprograms and related objects:

Subprogram	Uses PDA	Uses Map	Function
CDACT	CDACTA		Controls the function/action/ selection codes used to provide language independence, central control and standardization, and (optionally) security functions.
CDCHWIN	CDCHWINA CDPDA-M		Dynamically changes the window size and location in generated browse objects.
CDENVIR	CDENVIRA		Captures and restores the current environment to preserve calling object settings, such as window and page size.
CDFLIP	CDFLIPA CDPDA-M	CDFLIP11 CDFLIP21	Provides standard flip key processing to alter the format of the PF-key display, or remove the PF-key display to reveal command or other information.
CDGETDC	CDGETDCA		Analyzes the direct command line to get the next command value.
CDHCOPY	CDHCOPYA CDPDA-M	CDHCOPY1	Collects hardcopy information for generated browse programs.

Subprogram	Uses PDA	Uses Map	Function (continued)
CDMNTDF	CDMNTDFA		Sets the default behavior for a maintenance dialog.
CDMNTPR	CDMNTPDFA CDPDA-M CDPDA-D	CDMNPR1	Sets the preferred behavior for a maintenance dialog.
CDNCP	CDPDA-D CDPDA-M		Analyzes the direct command line in applications using a Natural command processor. Executes the entered command if it is a valid processor command. If not, it passes control back to CDNCPX.
CDNCPX	CDPDA-D CDPDA-M		Attempts to process a direct command as a Natural command processor command.
CDPASSW	#SYSTEM (A32) #PGM (A8) CDPDA-M		Prompts the user for a valid password if password checking is specified. This routine requires modification after installation.
CDSETGB	CDSETGBA		Sets the global page size.
CDSETPS	CDSETPSA		Sets the global page size for browse programs, based on the current window settings.
CDUMSG	CDPDA-M		Returns a short message from the Natural system file for error programs.
CDUMSGU	CDPDA-M		Returns a short message from a specified user application and/or specified language.
CDUPARM	CDUPARMA CDPDA-M		Determines current Natural parameter settings for the CDENVIR subprogram. No source is provided for this module.
CDUSAA	CDASAA		Builds an input prompt in ISA-SAA standard format.

Common Data Areas

Natural Construct-generated objects use common data areas to pass information between objects.

Note: The source and object code for each of the parameter data areas (PDAs) is located in both the SYSTEM and SYSCST libraries.

The supplied common data areas include:

Data Area	Type	Description
CDAOBJ2 CDAOBJ	Parameter	Contains standard information exchanged between object subprograms and invoking objects. Used by all object subprograms.
	Note:	We recommend that you use the CDAOBJ2 PDA with new applications. CDAOBJ is supplied to ensure backward compatibility with your existing applications.
CD--ACTM	Local	Contains action definitions used by object browse dialogs.
CDBRPDA	Parameter	Contains fields that control the behavior of the browse object.
CDDIALDA	Local	Contains local data used by dialog objects. This data area includes such things as help parameters.
CDERRLD2 CDERRLDA	Local	Inputs the values stacked by Natural prior to invoking an error transaction.
	Note:	We recommend that you use the CDERRLD2 LDA with new applications. CDERRLDA is supplied to ensure backward compatibility with your existing applications.
CDGDA	Global	Contains the minimum required variables for generated applications. The standard global data area (GDA) can be copied, renamed, and extended as desired.
CDKEYLDA	Local	Contains PF-key names and settings. Use this data area for all dialog objects.
CDLDA-M	Local	Temporarily saves message information.

Data Area	Type	Description (continued)
CDPDA-D	Parameter	Contains standard parameters for dialog objects (must match the DIALOG-INFO structure of the application's GDA).
CDPDA-M	Parameter	Contains standard parameters for exchanging message information (must match the MSG-INFO structure of the application's GDA).
CDPDA-P	Parameter	Shares common global information with subprograms. This data area will vary for each application and must match the PASS structure in the application's GDA.
CD--PFKM	Local	Contains PF-key definitions used by object browse dialogs.
CDSELPDA	Parameter	Contains standard parameters used by selection objects.

Programs

The following programs supply standard functionality to generated applications:

Function	Description
CDERROR	Supplies the standard error processor (invoked by the error transaction). Copy this program for each application (so the GDA matches that of the application).
CDERRTA	Supplies the standard error transactions.
CD-QUIT	Supplies the standard application quit processing, invoked when the Quit key is pressed.

For more information about error processing, see **Design Methodology**, page 141.

SUPPLIED GENERATION UTILITIES

This chapter describes the various utilities supplied with Natural Construct for use in the Generation subsystem.

The following topics are covered:

Using Utilities to Transfer Between Platforms

This section explains how to transfer data across dissimilar platforms (for example, from Unix to mainframe).

Natural Construct's import and export utilities read and write data from and to work file 1. This is true for each of the following utilities:

Utility	Described in
CSFLOAD CSFUNLD	**Supplied Administration Utilities**, *Natural Construct Administration and Modeling*
CSHLOAD CSHUNLD	**Utilities**, *Natural Construct Help Text*
CSMLOAD CSMUNLD	**Multiple Model Import Utility**, page 662 **Multiple Model Export Utility**, page 661

A work file written on one platform (such as Unix) can be read on another platform (such as mainframe) if the following conditions are met:

- The work file must be an ASCII file. For example:

Platform	How to save as an ASCII file
Mainframe	Before running the utility, define work file 1 as a PC file and activate PC Connection (translates from EBCDIC to ASCII).
Unix	Set the work file specification in your NATPARM to any extension other than "SAG".

- When transferring the work file between platforms, select the appropriate translator. For example, the file transfer method you select to move a file from a PC to a Unix machine must translate the PC's CR/LFs to CRs.

Multiple Model Export Utility

The CSMUNLD program exports selected models from a model file (with the related subprogram specifications that exist in the subprogram file) to work file 1. A report of the exported models is written to DEVICE LPT1.

CSMUNLD accepts up to 100 model names in the form:

```
Model:_____________________________
```

In the Model field, enter the name of the model you want exported to work file 1. Model names are composed of the model type, major, and minor.

Valid inputs are:

Input	Description
*	Exports all models to work file 1.
Maint	Exports the Maint model to work file 1.
BR*	Exports all models beginning with “BR” to work file 1.
.	Terminates the input screen.

Note: Up to 24 of the model names entered in the input field are automatically echoed on the screen.

Multiple Model Import Utility

The CSMLOAD program imports selected models (with the related subprogram specifications) from a work file (work file 1) to a model file and subprogram file, respectively. A report of the imported models is written to DEVICE LPT1.

CSMLOAD accepts up to 100 model names and replace options in the form:

```
Model:_____________________________     Replace Option:_  (Y/N)
```

Enter each model name, one name at a time. As you enter the names, they are automatically displayed on the panel.

Note: To replace the existing models with models with the same names in work file 1, mark the Replace Option field. If you do not want to replace the existing models, leave the Replace Option field blank.

Examples of input combinations

Input	Description
*	Imports all models from work file 1. If a model already exists, it is not replaced.
Maint	Imports the Maint model from work file 1. If the Maint model already exists, it is not replaced.
Maint Y	Imports the Maint model from work file 1. If the Maint model already exists, it is replaced.
BR*	Imports all models beginning with “BR” from work file 1. If the model already exists, it is not replaced.
.	Terminates the input screen.
Note:	When running in batch mode, the CSMLOAD utility will terminate with RC=0 if an error occurs due to problems with the internal layout structure of work file 1. To terminate the batch Natural session with RC=99, add “Y” to the end of the last model input combination (for example: BR*,,Y).

INCLUDE Code Insertion Utility

Many Natural Construct models generate programs that contain INCLUDE statements. These statements simplify program maintenance and ensure program standardization.

However, INCLUDE statements may hinder program debugging since not all source is visible. For this reason, Natural Construct provides the CSUINCL utility. The CSUINCL utility inserts all INCLUDE members into a program, thereby displaying all program statements and resolving the line numbers.

The CSUINCL utility is stored in the SYSCST and SYSTEM libraries.

➢ To use the CSUINCL utility:

1 Read the program containing the INCLUDE statements into the edit buffer.

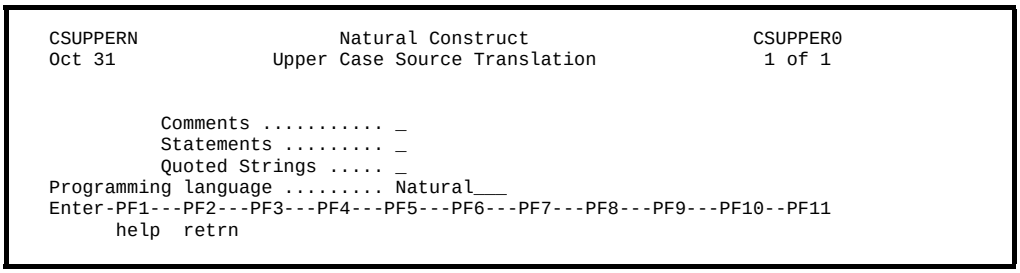
2 Issue the CSUINCL command.

JCL Submit Utility (Mainframe)

The CSUSUB utility submits the contents of the source buffer to the internal reader. This utility requires the availability of the NATRJE module.

Upper Case Translation Utility

This utility translates the contents of the source buffer into upper case. This allows you to print generated programs on printers that do not support lower case characters. To invoke this utility, enter “CSUPPER” from the application library.



Upper Case Source Translation Window

The fields in this window are:

Field	Description
Comments	To translate lower case comments into upper case, mark this field.
Statements	To translate lower case statements (including variables), mark this field.
Quoted Strings	To translate lower case quoted strings into upper case, mark this field.
Programming language	Programming language in which the source code currently in the edit buffer was written.

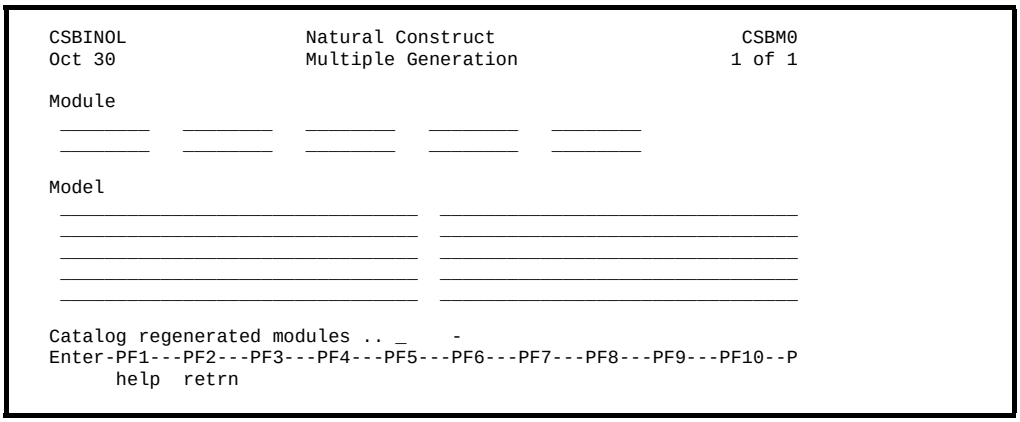
Multiple Generation Utility

Natural Construct supplies the NCSTBGEN multiple generation utility to regenerate multiple Natural modules. You can run this utility either online or in batch. The following section describes the utility.

Online Multiple Generation Utility

To invoke online multiple generation, enter “NCSTBGEN” from the application library. The Multiple Generation window is displayed:

Note: To run NCSTBGEN, the SYNERR profile must be turned ON before Natural is started.



Multiple Generation Window

Using the Multiple Generation Utility

When using this utility, keep the following information in mind:

- You can specify up to 10 module names and the corresponding model names; you must specify at least one module and corresponding model name. You can either enter the complete module and model names or the minimum number of characters required to uniquely identify the module and model.
- You can specify more than 10 module or model names using a wildcard character (*, for example). Type the prefix characters of the module or model name followed by the wildcard character.
- Only those modules previously generated with the specified model are regenerated.
- You can catalog all regenerated modules by marking the Catalog regenerated modules field. If you do not want to catalog the regenerated modules, leave the field blank.

The following examples are valid requests for module/model regeneration:

Enter:		To
Module: Model:	EMPLOYM Maint	Regenerate the EMPLOYM module generated with the Maint model.
Module: Model:	EMP* Maint	Regenerate all modules beginning with EMP generated with the Maint model.
Module: Model:	EMP* VEHICLE Maint Browse*	Regenerate all modules beginning with EMP characters, as well as the VEHICLE module, generated with the Maint or Browse models (any model beginning with Browse).
Module: Model:	* *	Regenerate and catalog all modules generated from all models within the current library.

Online Multiple Generation Process

During the generation process, the Status windows are displayed:

```
+-----------------------------------------------------------+ ----------+
| CSBDISP           Natural Construct           CSBDISP0 |     CSBM0 |
| Oct 31              Status Window               1 of 1 |  1 of 1   |
|                                                          |           |
|             Module        Status                         |           |
|             NCPDEMO  Skipped                             | _________ |
|             NCQUIT   Skipped                             | _________ |
+-----------------------Software AG Canada-----------------------------+
|                              Status Window                           |
| --> PREGEN CUSCPR                                                    |
| <-- PREGEN CUSCPR                                                    |
| --> SAVE CUSCS                                                       |
|                                                                      |
+----------------------------------------------------------------------+
|             NCSELH   Skipped                             |           |
|             NCTFPRDB Skipped                             |           |
|             NCTFPRD1 Skipped                             | --PF10--P |
|             NCTFRPRD Skipped                             |           |
|             NCTFWHSB Skipped                             |           |
|             NCTFWHSE Skipped                             | ----------+
|             NCTFWHS1 Skipped                             |
|                                                          |
+----------------------------------------------------------+
```

Status Windows

These windows allow you to view the progress of your multiple generation job. The smaller window displays each step of the generation procedure, with an appropriate message (for example, “PREGEN CUSCPR”). This window is identical to the one displayed during the generation of an object from the Generation main menu.

The larger window displays status messages that indicate the outcome of the generation process for each selected module. The status messages displayed are the same as those on the Multiple Generation Summary report. For a list of messages that may be displayed, see **Multiple Generation Summary Report**, page 667.

Multiple Generation Summary Report

A Multiple Generation Summary Report is created when the generation process is complete. This report lists the inputs entered (module and model names) and the generation results. Any of the following messages may be displayed:

- The selected object has been Cataloged after successful generation.
- The selected object has been Generated.
  If the Catalog option is marked, the program was generated successfully but cannot be cataloged.
- The selected object has been Skipped.
  This message is displayed when the object was not generated by Natural Construct or if you did not specify a corresponding model for the object.
- A Natural Construct error has occurred for the selected object.
  This message is displayed when Natural Construct errors occur, such as when a program is missing a required user exit.
- A Natural error has occurred.
  A problem, such as ESIZE exceeded for the selected object, may have occurred during execution.
- A CAT error has occurred.
  The object was successfully generated, but cannot be cataloged.

Report Format

The Multiple Generation Summary Report is displayed in the following format:

Status	Module	Model
Cataloged	module	model
Total Generated Cataloged...*nnn*		
Generated Catalog Error	module	model
Total Generated Catalog Error...*nnn*		
Natural Error	module	model
Total Natural Error...*nnn*		
Construct Error	module	model

Status	Module	Model (continued)
Total Construct Error...*nnn*		
Generated	module	model
Total Generated...*nnn*		
Generated Error	module	model
Total Generated Error...*nnn*		
Natural Error	module	model
Total Natural Error...*nnn*		
Construct Error	module	model
Total Construct Error...*nnn*		
Skipped	module	model
Total Skipped...*nnn*		

Note: A blank in the Model column indicates the module was not generated by a Natural Construct model.

Batch Regeneration Utility

You can run the NCSTBGEN multiple generation utility in batch. This program regenerates multiple modules (generated using the specified models) and produces a report listing the names of the generated modules.

Note: While using NCSTBGEN to regenerate multiple modules in batch mode, object generic subprograms may not be regenerated. For example, if the parameters have been categorized (i.e., defined within user exits), you must regenerate the PARAMETER-DATA user exit from the client.

You can use keyword or delimiter mode, as shown in the following examples.

Example of using keyword mode

```
NCSTBGEN
PROGRAM=GL*
PROGRAM=AP*
MODEL=OBJECT*
CATALOG=Y
FIN
```

Note: You can enter program names without the PROGRAM keyword, as long as you enter the MODEL keyword with the model names. If neither keyword is entered, Natural Construct assumes that each entry is a program.

Example of using delimiter mode

```
NCSTBGEN
GL*,OBJECT*,Y
AP*
.
```

When you run this utility in batch, keep the following information in mind:

- You can enter up to 10 program (module) names and the corresponding model names.
- You must enter at least one model name or an asterisk (*) to represent all models.
- To catalog all generated modules, enter either “Y” or “CATALOG=Y”.
- To terminate the process, enter either “FIN” or “.” (period).

Trace and Debug Utilities

The CSUDEBI, CSUDEBUG, and CSUDEB utilities allow you to trace and debug Natural programs. These utilities are ideal for diagnosing problems within server programs or batch programs when access to the Natural debugger is not available. To invoke a utility, enter the utility name on the command line.

CSUDEBI Utility

The CSUDEBI utility is the first component of the debug facility. It generates trace statements into the current source area. Because you can use the CSUDEBUG utility to trigger debug output externally, these statements can remain there indefinitely. You can also add hand-coded statements to display variable contents.

The following example shows Natural statements listed in the CSUDEBI utility window:

```
Mark desired DEBUG locations
  Before   Statement     After  |  Before   Statement     After
           Start of pgm    X    |    X      PERFORM         X
           ACCEPT               |           READ
    X      CALL            X    |           END-READ
    X      CALLNAT         X    |           READ WORK
           ESCAPE               |           END-WORK
    X      FETCH                |           REJECT
    X      FETCH RETURN    X    |           REPEAT
           FIND                 |           END-REPEAT
           END-FIND             |           SELECT
           HISTOGRAM            |           END-SELECT
           END-HISTOGRAM        |    X      STOP
           IF                   |    X      TERMINATE
           ON ERROR        X    |           END
  Debug copycode member: CCDEBUG
```

CSUDEBI Utility Window

Use this window to select the locations where you want to add debug statements.

CSUDEBUG Utility

The CSUDEBUG utility is the second component of the debug facility. It allows you to activate and deactivate trace output, as well as specify where it is to be written (source area, screen, printer, etc.). You can also filter output to display trace output for a particular set of programs or statements.

Main Debug Options Window

The following example shows the Main Debug Options window:

```
                Main Debug Options
Source update options ...
  Mark to renumber line references _
Runtime options ...
  Select the desired output media ...
     X Disable debugging
     _ Write to screen (prt 0)   _ Write to source area
     _ Write to printer(1)       _ Write to work file 1
     _ Write to printer(2)       _ Write to work file 2
     _ Write to printer(3)       _ Write to work file 3
     _ Write to printer(4)       _ Write to work file 4
     _ Write to printer(5)       _ Write to work file 5
  Enter the following to force a runtime error...
     Program name .. ________     Line number ... 0000
     Error number .. 0000         Skip n ........ ___0
  Program filters ...
     ________ ________ ________ ________ ________
  Mark to set trace mask _ (No current mask)
```

Main Debug Options Window

The fields in this window are:

Field	Description
Source update options	
Mark to renumber line references	Automatically renumbers line references when you add your own lines to the program.
Runtime options	
Select the desired output media	Destination for the output.
Program name	Name of the program from which to trigger the error.
Line number	Number of the line on which to trigger the error program. If this field is zero, the error is triggered on the first program line traced.
Error number	Error number to trigger.
Skip n	Number of occurrences before a runtime error is triggered. Replace “n” with a numeric value.

Field	Description (continued)
Program filters	Programs for which to filter trace output. This option filters trace output for selected programs. To select a range of programs, use wildcard characters. For more information, see **Wildcard Selection**, page 89.
Mark to set trace mask	To indicate which statements/locations you want to trace, mark this field and press Enter. The Mark Settings window is displayed (for a description of this window, see the following section).

Mark Settings Window

The Mark Settings window displays the statements/locations in the program you want to trace:

```
                      Mask Settings
Mark the statements/locations to be traced ...
_ Start of Program  _ Before ACCEPT     _ After  ACCEPT
_ Before CALL       _ After  CALL       _ Before CALLNAT
_ Before ESCAPE     _ Before FETCH      _ Before FETCH RET
_ Before FIND       _ After  FIND       _ Before END-FIND
_ Before FOR        _ After  FOR        _ Before END-FOR
_ Before HISTOGRAM  _ After  HISTOGRAM  _ Before END-HIST
_ Before IF         _ After  IF         _ After  ON-ERROR
_ Before PERFORM    _ After  PERFORM    _ Before READ
_ Before END-READ   _ After  END-READ   _ Before READ-WORK
_ Before END-WORK   _ After  END-WORK   _ Before REJECT
_ Before REPEAT     _ After  REPEAT     _ Before END-REPEAT
_ Before SELECT     _ After  SELECT     _ Before END-SELECT
_ Before STOP       _ Before TERMINATE  _ Before END
_ Before CALL ETB   _ After  CALL ETB
```

Mask Settings Option

Use this window to mark the statements or locations you want to trace.

CSUDEB Utility

The CSUDEB utility is the third component of the debug facility. It allows you to remove all generated debug statements from the program currently in the source area when the debug code is no longer needed.

APPENDIX A — GLOSSARY OF TERMS

The following terms are used throughout this documentation:

Term	Definition
Browse program	A program that retrieves records from a specified file. Once retrieved, you can select a record for processing. Sometimes referred to as a query program.
Browse a file	View the records in a specified file.
Code frame	A block of code that performs a specified function. Code frames are one of the major components of Natural Construct models.
Constant	A value that is always the same.
Copycode	Static code that is provided for you to copy and use in INCLUDE statements.
Cursor-sensitive or Cursor sensitivity	Select using a cursor, such as moving your cursor to an item and pressing Enter. If you are using Natural on a PC to access Natural Construct, you can double-click with the mouse to select.
Data area	A Natural module in which data is stored. For example, a parameter data area stores parameters that are passed between subprograms and a global data area stores data that is used by all programs in an application.
Enter	To type a value in a field and press the Enter key.
Enter key	The key you press to execute a module. Sometimes referred to as the Return key.
Execute	Start or display a program, menu, panel, editor, utility, etc. Also referred to as invoke.
Field	An area in a window or on a panel that either displays information or requires the user to input information.
Function	A process. For example, the Maintain Models function on the Administration main menu.

Term	Definition (continued)
Helproutine	A Natural module that displays a help panel.
Invoke	Start or display a program, menu, panel, editor, utility, etc. Also referred to as execute.
Issue	Execute a command by typing the command name and pressing Enter.
Mark a field	Type any non-blank character in the field.
Note:	You may also be required to press the Enter key.
Menu	A panel or program that displays the available functions (processes) and allows you to select one to use.
Model	A Natural Construct template used to record specifications and generate source into a Natural buffer.
Module	Any Natural object that is generated by Natural Construct.
Object	Any entity that represents a business function and is used by Natural Construct.
Optional field	A field for which input is not required.
Panel	A screen or map.
Parameter	A value for a field.
PF-key	A program function key. To perform the associated function, you press that key. For example, pressing PF1 (help) displays help information.
Program	A block of code that performs a function. For example, a subprogram, subroutine, helproutine, etc. Also referred to as a module.
Query program	See the description for a browse program.
Required field	A field for which input is required.
Return code	The code you enter in the input field on a menu to return to the previous panel. The return code on Natural Construct menus is a . (period).
Return key	See the description for the Enter key.
Scroll	To move forward (down), backward (up), left, or right through the information displayed on a panel or in a window.

Term	Definition (continued)
Specify	Supply a value for an input field. For example, by typing a value in the field and pressing Enter or by marking the field.
Subprogram	A self-contained block of code that is called via parameters by a program to perform a function.
Subroutine	A block of code within a larger block of code that is referenced one or more times. A subroutine is typically used to perform repetitive tasks or to isolate a specific task.
Substitution parameters	Parameters that always have the same format and different values at generation time.
Terminate	End your Natural Construct session.
User exit	An area within the program code that is reserved for user-defined functions. Within these areas, you can change the generated functions as desired. User exit code is preserved when the program is regenerated.
Utility	A program supplied to perform a specific function. For example, the model load utility.
Variable	A value that represents one of many possible values. The actual value can be supplied by Natural when the program is executed or supplied by other variables (either user-supplied or derived).
Window	A separate, self-contained area displayed on a panel. For example, a help window.

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z