Natural Business Services

Natural Construct Administration and Modeling

Version 8.2.1

November 2013

Natural Business Services

This document applies to Natural Business Services Version 8.2.1.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: NBS-CSTADMIN-821-20131119

Table of Contents

Preface

Natural Construct Administration and Modeling explains how to access and use the Administration subsystem of Natural Construct. This section explains how information is presented for different platforms, as well as the purpose and structure of *Natural Construct Administration and Modeling*. It includes an overview of conventions used in this documentation and information about other resources you can use to learn more about Natural Construct.

Note: Although the screen examples used in this documentation are from a mainframe environment, the information applies to all server environments.

This documentation is intended for Natural Construct administrators who want to:

- Maintain the existing models, code frames, and control record for their companies
- Create new models
- Use the utilities provided with Natural Construct

This documentation assumes that, as a Natural Construct administrator, you have extensive knowledge of Natural and the Natural Construct Generation subsystem.

Natural Construct Administration and Modeling covers the following topics:

Introduction to Natural Construct	Contains a general description of Natural Construct and the basic information you need to use the Administration subsystem.
Using the Administration Subsystem	Describes how to use the Administration subsystem to define custom models and maintain the models Natural Construct uses to generate programs.
Using the Code Frame Editor	Describes the Code Frame editor, as well as the line and edit commands you can use in the editor.
Creating New Models	Describes the procedure for creating a new Natural Construct model.
New Model Example	Contains a step-by-step example of how to create a new model using the procedure described in *Creating New Models*.
CST-Clear Model	Describes the model that generates clear subprograms for your models.
CST-Document Model	Describes the model that generates documentation subprograms for your models.
CST-Frame Model	Describes the model that generates frame subprograms for your models.
CST-Modify and CST-Modify-332 Models	Describes the model that generates maintenance subprograms for your models.
CST-Panel Model	Describes the model that generates panels for a Windows interface.
CST-PDA Model	Describes the model that generates parameter data areas (PDAs) for your models.
CST-Postgen Model	Describes the model that generates post-generation subprograms for your models.

CST-Pregen Model	Describes the model that generates pre-generation subprograms for your models.
CST-Proxy Model	Describes the model that generates a subprogram proxy to convert data between the network transfer format and the native Natural data format used in the subprogram's PDA. This model can generate both a server proxy and a client proxy.
CST-Read Model	Describes the model that generates read subprograms for your models.
CST-Save Model	Describes the model that generates save subprograms for your models.
CST-Shell Model	Describes the model that generates a template for a model subprogram.
CST-Stream Model	Describes the model that generates stream subprograms for your models.
CST-Validate Model	Describes the model that generates validation subprograms for your models.
User Exits for the Administration Models	Describes the user exits supplied for the Natural Construct administration models.
Modifying the Supplied Models	Describes how to modify models supplied with Natural Construct.
External Objects	Describes the supplied subprograms and helproutines.
Supplied Administration Utilities	Describes the supplied utilities for all supported platforms.
Using SYSERR for Multilingual Support	Describes how to use the SYSERR utility to provide multilingual support.
Appendix A: Glossary of Terms	Contains a glossary of terms used throughout this documentation.

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.
Access	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

Term	Description
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.

1 Introduction to Natural Construct

This section introduces you to Natural Construct. It describes how to access the subsystems, use PF-keys, and access the online help. It includes sections on translating to upper case, handling messages, storing saved modules, and using direct commands.

What is Natural Construct?

Natural Construct is a set of tools for application developers. Created for Software AG's Natural/Predict environment, it helps application developers achieve higher productivity goals than are obtainable using Natural and Predict alone. At the same time, Natural Construct helps standardize and control the application development process.

Natural Construct provides a series of models you can use to create different Natural modules (objects). The following table lists the advantages of using Natural Construct-generated modules over modules created in Natural alone:

Advantage	Benefits
Standardization and quality	Create a consistent user interface and code structure.
Reusage	Once your model is tested and debugged, it can be used by multiple users, problem free. Models help share your Natural expertise, making optimal use of available talent.
Increase productivity	These benefits include: ■ Reduce design considerations ■ Speed up implementation ■ Reduce testing requirements
Minimize errors	Avoid errors that are introduced by program cloning.

Natural Construct Subsystems

Natural Construct is comprised of the following subsystems:

Subsystem	Description
Administration	Used by the Natural Construct administrator to define custom models and maintain the models Natural Construct uses to generate programs. The Administration subsystem is described in detail in this documentation.
Generation	Used by the developer to define specifications for the Natural Construct models and generate the following modules: ■ programs ■ subprograms ■ helproutines

Subsystem	Description
	■ subroutines ■ copycode ■ maps ■ parameter data areas ■ local data areas ■ global data areas ■ Predict program descriptions ■ code blocks ■ JCL text (mainframe) ■ user exit code For information about this subsystem, refer to *Natural Construct Generation*.
Help Text	Used by documenters or developers to create and maintain help text at the map and/or input field level. For information about this subsystem, see *Natural Construct Help Text*.

Access Natural Construct

You can access the Administration subsystem in standard or translation mode. Translation mode allows you to create multilingual specification panels for developers, as well as dynamically maintain the panel components.

This section describes how to access each Natural Construct subsystem, how to access the Administration subsystem in standard and translation mode, and how to access the generation facilities from a steplib with Natural Security installed.

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

Natural Construct Libraries

While other Software AG products can be accessed from other libraries, they run exclusively in their own product library (SYSSEC, SYSPAC, SYSDIC, for example). Natural Construct does not run exclusively out of its product library, SYSCST. It also must also run out of the application libraries in the FUSER file.

Typically, Natural Construct developers access Natural Construct using the NCSTG or NCSTH command from any library. These commands invoke modules in the SYSLIB and SYSLIBS libraries. The CD-HELP* modules in the SYSLIBS library provide online help for Natural Construct screens.

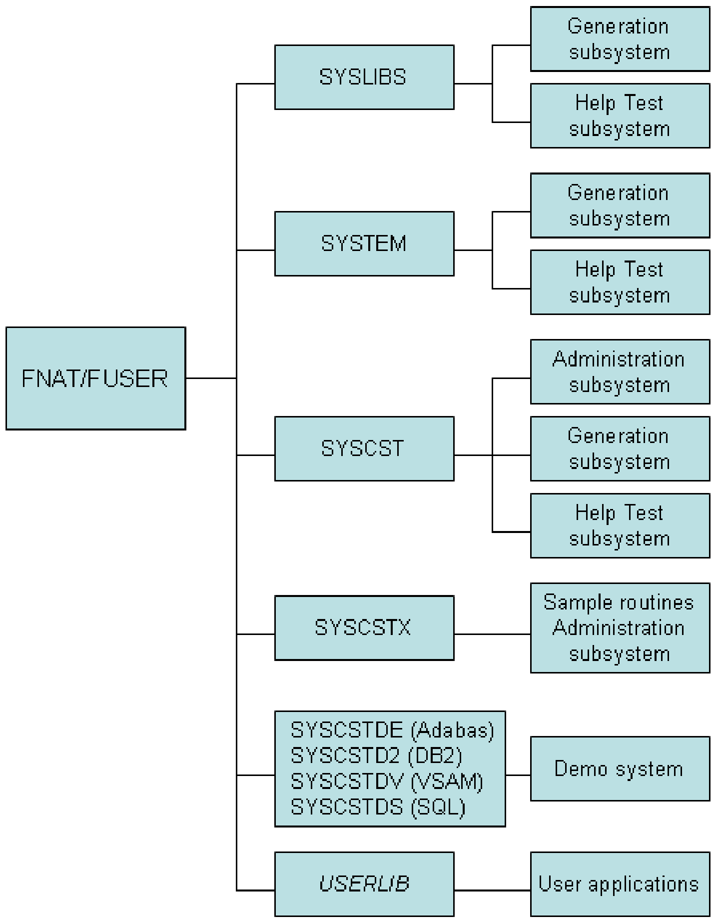
Natural Construct also allows administrators and modelers to customize the standard Natural Construct models. These users access Natural Construct using the CSTG or CSTH command from the SYSCST library. All changes are confined to this library, which allows administrators to test customizations without affecting developers and their applications.

Since administrators can customize help modules like CD-HELP*, copies of these modules are also stored in the SYSCST library. Any changes to these modules do not affect developers because they use the CD-HELP* routines in the SYSLIBS library. Typically, modelers access Natural Construct using the CSTG or CSTH command.

Once a Natural Construct modeler creates or maintains a model, all customized modules must be copied to the appropriate library.

- If the changes apply to the development environment, copy the modules to the SYSLIB or SYSLIBS library.
- If the changes apply to the runtime environment for a Natural Construct-generated application, copy the modules to a library within the application steplib chain (for example, SYSTEM on the FNAT). It is also common to see CD-HELP* routines in the SYSTEM FNAT library.

Copies of Natural Construct are stored in the following libraries:



Each library is available to different users and contains different subsystems. The libraries are:

- SYSLIBS Library
- SYSTEM (FNAT) Library
- SYSCST Library
- SYSCSTX Library
- SYSCSTDE, SYSCSTD2, SYSCSTDV, and SYSCSTDS Libraries
- USERLIB Library

- Execute Generation Facilities from a Steplib with Natural Security Installed

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 entered at the Next prompt to invoke each subsystem:

Authorized Users	Subsystems	Command to Invoke Each Subsystem
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 entered at the Next prompt to invoke each subsystem:

Authorized Users	Subsystems	Command to Invoke Each Subsystem
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 entered at the Next prompt to invoke each subsystem:

Authorized Users	Subsystems	Command to Invoke Each Subsystem
Administrators	Administration	"menu" (standard mode) "menut" (translation mode)
	Generation	"ncstg"
	Help Text	"ncsth"

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.

Execute Generation Facilities from a Steplib with Natural Security Installed

With Natural Security installed, you can access the Natural Construct generation facilities from a steplib. This allows you to override the supplied model subprograms at a higher level steplib without disturbing the modules supplied by Natural Construct.

For example, you can define the following steplibs in your development library:

- *CSTMODS* (your modification library)
- SYSCST
- SYSLIBS
- SYSTEM

Using this configuration, you can easily change your standards without disturbing the supplied modules. To modify any modules in the SYSCST or SYSTEM library that are affected by changes, copy them into the *CSTMODS* library.

Note: You can also define multiple modification libraries in the steplib chain (to reflect corporate versus application standards).

When accessing Natural Construct from a steplib, the highest level steplib should contain a replacement for the NCSTG program. For example:

```
FETCH 'CSTG'
END
```

Otherwise, the NCSTG program invokes the version of Natural Construct stored in the SYSLIBS library.

Note: If Natural Security is not installed, refer to USR1025P in the SYSEXT library for an example of how to set up your steplibs.

Use Standard PF-Keys

Throughout the Natural Construct system, certain PF-keys have standard functions (pressing the PF1 key invokes online help, for example). The PF-key lines, which are typically located at the bottom of panels, display the PF-key functions for that panel.

Notes:

1. PF-keys 13 to 24 are equivalent to PF-keys 1 to 12, respectively. However, only PF1 to PF12 are displayed.
2. You can change the function and/or description associated with each key (for more information, see *Access the Administration Main Menu*). Within this documentation, the default values are used.

The standard PF-keys and functions are:

PF-Key	Name	Function
PF1	help	Displays help for a particular panel or field. ■ When the cursor is in a field followed by an asterisk (*), displays a window from which you can select a valid value for the field. For information, see *Field-Level Help*. ■ When the cursor is in a field not followed by an asterisk (*), displays help information for that field. For information, see *Panel-Level Help*. ■ When the cursor is anywhere on the panel except a field, displays help for the entire panel **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 Function 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.
PF7	bkwrd	Scrolls backward (up) through data.
PF8	frwrd	Scrolls forward (down) through data.
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.

PF-Key	Name	Function
PF12	main	Displays the Natural Construct Administration main menu.

Help and Return Codes on Menus

On each Natural Construct menu, you are given the options "?" and "." 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 terminates the current program and returns you to the previous menu. It is equivalent to pressing PF2 (retrn).

Access Online Help

Natural Construct provides extensive online help. You can display both general help information for each panel (panel-level help) or help for a specific field (field-level help). This section covers the following topics:

- Panel-Level Help
- Field-Level Help

Panel-Level Help

While you are 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 positioned in an input field when you request help, Natural Construct displays help information for that field. For more information, see *Field-Level Help*.

The following example shows the panel-level help for the Administration main menu:

```
                              Panel Help
                       Administration Main Menu

 This menu lists the functions available within the Administration
 subsystem; you use these functions to perform various administrative
 duties within Construct.

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

 For example, you use these functions to:
 - maintain the Construct control record defaults, such as the
   default PF-key settings and dynamic attribute characters
 - maintain the Construct components, such as the code frames and
   subprograms used by each model
 - invoke the supplied utilities to compare models or code frames
 - use the supplied driver programs to invoke many of the internal
   Construct subprograms
 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/CSDMNM0/1                                              ↩
```

- 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 online help, press PF1 (help) in any help window.
- To display information about a topic enclosed within angle brackets (<< >>), move the cursor over the name and press Enter. A window is displayed, containing help information about the selected topic.

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 field-level help displays a selection window containing the valid values for a field. If active help is available, the field is followed by an asterisk (*).

Passive

To display passive field-level help:

1 Move the cursor to any field that is not followed by an asterisk (*).

2 Press PF1 (help).

Note: You can also type a question mark (?) in the first-character position of any field that is not followed by an asterisk (*) and press Enter (mainframe).

Active

To display active field-level help:

1 Move the cursor to a field that is followed by an *.

2 Press PF1 (help).

Note: You can also type a question mark (?) in the first-character position of any field that is followed by an asterisk (*) and press Enter (mainframe).

The following example shows the active help window for the Relationship name field:

```
CPHRL               Natural Construct              CPHRL0
Aug 20         Select Predict Relationship         1 of 1

Relationship                     Relationship type
-------------------------------- ------------------------
NCST-CUSTOMER-ORDER-HEADER       Natural Construct
NCST-LINE-HAS-DISTRIBUTION       Natural Construct
NCST-ORDER-HAS-LINES             Natural Construct
NCST-POLICY-COVERS-VEHICLES      Natural Construct
NCST-POLICY-HAS-INQUIRIES        Natural Construct
NCST-POLICY-IS-FOR-CUSTOMER      Natural Construct
NCST-PRODUCT-ORDER-LINES         Natural Construct
NCST-VEHICLES-HAVE-COVERAGES     Natural Construct
NCST-VEHICLES-MUST-EXIST         Natural Construct
NCST-WAREHOUSE-CUSTOMER          Natural Construct
Relationship  .......... ________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9-
      help  retrn                         bkwrd frwrd
Position cursor or enter screen value to select           ↩
```

To select a value from the help window:

1 Move the cursor to the line containing the value.

2 Press Enter.

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

Convert Text to Upper Case

Natural Construct automatically performs the commands to convert text from lower or mixed case to upper case where appropriate. Headings are displayed exactly as entered (lower or upper case), but if certain specifications must be in upper case, Natural Construct converts them. When Natural Construct ends, the case setting is restored to the default value.

Note: If you are a mainframe user, specify your teleprocessing (TP) monitor's command for lower case. In Com-Plete, for example, issue the LOW command.

Maintain Messages for Generated Programs

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.

- 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

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

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

Store Saved Modules

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

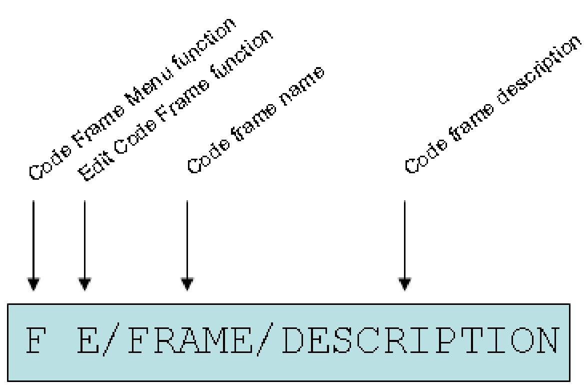
Use Direct Commands

To navigate within the Administration subsystem, you can enter codes on menus, press PF-keys, or issue direct commands. Direct commands take you to any function or menu within the subsystem without using intervening menus. They are useful for experienced users who know the menu structure, valid menu codes, and the required parameters at each menu level. The following example shows the Command line:

```
Command ____________________________________________________________
```

You can string together as many commands as you like. If one of the codes is not valid on the corresponding menu, Natural Construct displays that menu so you can enter a valid code.

The following diagram illustrates a sample direct command:



This direct command accesses the Code Frame Menu (menu code F on the Administration main menu) and the Edit Code Frame function (menu code E on the Code Frame menu) and displays the code frame called FRAME with the description, DESCRIPTION, in the Code Frame editor.

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

When you enter direct 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 direct command, Natural Construct checks the main menu for a match.

You can also issue direct commands at the Natural Next prompt (Direct command box for Unix). While you are in the SYSCST library, for example, you can enter the following direct command to access the Administration subsystem (MENU) and edit the code frame, FRAME, with the description, DESCRIPTION:

```
MENU F E/FRAME/DESCRIPTION
```

2 Using the Administration Subsystem

This section describes how to use the Administration subsystem supplied with Natural Construct. Use the Administration main menu to access the functions available in the Administration subsystem.

Access the Administration Main Menu

To access the Administration main menu:

1 Enter "menu" at the Natural prompt.

The Administration main menu is displayed. For example:

```
CSDMAIN               N a t u r a l   C o n s t r u c t              CSDMNMO
Aug 17                     Administration Main Menu                    1 of 1

                      Functions

                      -----------------------------------------------

                      M  Maintain Models

                      F  Code Frame Menu

                      S  Maintain Subprograms

                      R  Maintain Control Record

                      C  Compare Menu

                      D  Drivers Menu



                      ?  Help

                      .  Return

                      -----------------------------------------------

Function ........... _




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

Note: For a description of the Help and Return functions, see ***Help and Return Codes on Menus***.

2 Enter the corresponding one-character function code in `Function`.

The functions available through the Administration main menu are:

Code	Function	Description
M	Maintain Models	Displays the Maintain Models panel, where you can maintain the components that define a model for the Natural Construct generation process. For information, see ***Maintain Models Function***.
F	Code Frame Menu	Displays the Code Frame menu. Using the functions available through this menu, you can maintain the code frames used by the generation models. For information, see ***Code Frame Menu Function***.
S	Maintain Subprograms	Displays the Maintain Subprograms panel, where you can maintain the modify specification subprograms used by the generation models. For information, see ***Maintain Subprograms Function***.
R	Maintain Control Record	Displays the Maintain Control Record panel, where you can maintain the default values for the Natural Construct control record (PF-keys, dynamic attribute characters, help indicator, etc.). For information, see ***Maintain Control Record Function***.
C	Compare Menu	Displays the Compare menu. Using the functions available through this menu, you can compare code frames used by the models. For information, see ***Compare Menu Function***.
D	Drivers Menu	Displays the Drivers menu. Using the driver programs available through this menu, you can access many of the utility subprograms supplied with Natural Construct. (The source code for these subprograms is not supplied.) For information, see ***Drivers Menu Function***.

Create and Maintain Natural Construct Models

This section describes how to use the Administration main menu to define custom models and maintain the models Natural Construct uses to generate programs. The following topics are covered:

- Maintain Models Function
- Code Frame Menu Function
- Maintain Subprograms Function
- Maintain Control Record Function
- Compare Menu Function

- Drivers Menu Function

Maintain Models Function

To create or maintain a model:

1 Enter "M" in `Function` on the Administration main menu.

The Maintain Models panel is displayed. For example:

```
CSDFM                   N a t u r a l   C o n s t r u c t                CSDFM0
Aug 17                         Maintain Models                            1 of 1

Action .................... __ A,B,C,D,M,N,P,R

Model ..................... BROWSE__________________________

Description ........ *0200.1__________________________________________________
                     BROWSE Program

  PDA name ................. CUSCPDA_     Status window ............ Y

  Programming mode ......... S_           Comment start indicator .. **_

  Type ..................... P Program    Comment end indicator .... ___


  Code frame(s) ............ CSCA?___  CSCB?___  CSCC?___  ________  ________
  Modify server specificatn  CUSCMA__  CUSCMB__  CUSCMC__  CUSCMG__  ________
                             ________  ________  ________  ________  ________
  Modify client specificatn  CUSCMA__  CUSCMB__  CUSCMC__  ________  ________
                             ________  ________  ________  ________  ________

  Clear specification ...... CUSCC___    Post-generation .......... CUSCPS__
  Read specification ....... CUSCR___    Save specification ....... CUSCS___
  Pre-generation ........... CUSCPR__    Document specification ... CUS-D___
Command ........... __________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit  frame                                            main
```

2 Enter the action code in `Action`.

For example, if you are creating a new model, enter A (Add); if you are changing the settings for an existing model, enter M (Modify).

Note: For a description of the actions available, press PF1 (help) when the cursor is in the Action field.

3 Use the following fields to define or modify the model settings:

Field	Description
Model	Name of the model you are creating or maintaining.
Description	Brief description of the model or the SYSERR number that supplies the description. When a module is generated using the specified model, this description is displayed as the first heading on the panel. Because this description is part of the model user interface, you can use SYSERR numbers from the CSTLDA library to support dynamic translation. Within SYSERR, you can also specify substitution variables (instead of hardcoding the message). For example, SYSERR number *0200.1 corresponds to the English text "`:1:`Program". If you specify *0200.1 in this field for the Browse model, Natural Construct replaces `:1:` with the model name and the first panel heading becomes Browse Program. (The actual heading is displayed below this field.) For more information about dynamic translation, see ***Maintenance***.
PDA name	Name of the parameter data area (PDA) for the model. This PDA is passed to the model subprograms to capture model specifications. For more information, see ***Step 1: Define the Scope of the Model***.

Status window	Code that indicates whether the Status window is displayed when a module is generated.
	
	If the code is Y or T, you can press PF5 (optns) while generating the module to display the Status window, which contains information about the generation progress, save, and/or stow functions. You can also decide how the Status window is displayed. The following example uses symbols:
	
	`<-- PREGEN CUMNGPR`
	`--> FRAME CUMN9`
	`    --> FRAME CU--B9`
	
	The following example uses text:
	
	`Ending Pre-generation Subprogram CUMNGPR`
	`  Starting Code Frame CUMN9`
	`  Starting Code Frame CU--B9`
	
	■ To display symbols, enter "Y".
	■ To display text, enter "T".
	■ If you do not want the window displayed, enter "N".
	
	Note: If this field is blank, it defaults to N.

Field	Description
Programming mode	Mode for the resulting code. Valid codes are S (structured), SD (structured data), or R (reporting) mode. All supplied models use structured mode.
Comment start indicator	Set of characters that indicate the beginning of a comment line for the generated module. As required for Natural modules, the default value is **. You can change this value for other supported programming languages.
Type	Code for the type of module generated by this model. Valid module types are: ■ P (program) ■ E (external; non-Natural) ■ * (super model modules) ■ N (subprogram) ■ S (subroutine) ■ H (helproutine) ■ M (map) ■ L (local data area) ■ A (parameter data area) ■ G (global data area) ■ J (JCL statements; mainframe) ■ . (statement code block; .g) ■ T (text) ■ C (copycode) ■ blank (determined when a module is generated using this model; model subprograms must assign the CU—PDA.#PDA-OBJECT-TYPE parameter)
Comment end indicator	Set of special characters that indicate the end of a comment. For some programming languages, this set of characters is required to generate modules. For PL1, for example, the indicator is */.
Code frame(s)	Names of the code frames used to create the specified model (for information, see *Naming Conventions for Code Frames*). The code frames are listed in the sequence they are used during generation. You can specify a maximum of five code frame names for each model; you can only use existing code frames. In addition: ■ You can select a code frame and access the Code Frame editor from this panel. For information, see *Select a Code Frame for Editing*. ■ You can use nested code frames. For information, see *Nested Code Frames*. **Note:** Code frames that are used to generate maps and data areas can only have subprogram and comment lines.

Field	Description
Modify server specificatn	Names of the subprograms executed when the Modify function is invoked by the Natural Construct nucleus for server platform generation. The subprograms are listed in execution sequence. To change the order of execution, change the order of these subprograms. You can specify a maximum of 10 subprograms.
Modify client specificatn	Names of the subprograms executed when the Modify function is invoked by the nucleus for client platform generation. The subprograms are listed in the sequence they are executed. To change the order of execution, change the order of these subprograms. You can specify a maximum of 10 subprograms.
Clear specification	Name of the subprogram executed when the Clear function is invoked by the nucleus. The Clear function is automatically invoked prior to the Read function or when a new model name is specified and the parameter data area (PDA) is different. It is typically used to set default values for the model.
Post-generation	Name of the subprogram executed when the Post-generation function is invoked by the nucleus. This subprogram applies post-generation changes to the generated program. It is typically used to perform model specification substitutions; it is not supported for models that cannot be regenerated.
Read specification	Name of the subprogram executed when the Read function is invoked by the nucleus. It is typically used to retrieve the specifications from a previously-generated module It is not supported for models that cannot be regenerated.
Save specification	Name of the subprogram executed when the Save function is invoked by the nucleus (not supported for models that cannot be regenerated). This subprogram is executed immediately after the pre-generation subprogram is executed. It writes the generation specifications so the generated program can be read using the Read function. If a user marks the Save Specification Only option, this subprogram can be invoked even if generation cannot be completed due to specification errors.
Pre-generation	Name of the subprogram executed when the Pre-generation function is invoked by the nucleus. This subprogram sets up internal variables before the generation process begins. It is typically used to set PDAC- variables for code frame manipulation or to generate a module for simple models.
Document specification	Name of the subprogram executed when the Document function is invoked by the nucleus. This subprogram documents generated modules in Predict as they are saved or stowed.

Select a Code Frame for Editing

You can use the Maintain Models panel to select a code frame for editing.

To select a code frame for editing:

1 Move the cursor over the code frame you want to edit.

2 Press PF4 (frame).

The specified code frame is displayed in the Code Frame editor.

Note: For more information about modifying the supplied code frames, see *Step 5: Create Code Frame(s) and Define the Model*.

Naming Conventions for Code Frames

The following example shows the Maintain Models panel for the Browse model:

```
CSDFM                      N a t u r a l   C o n s t r u c t          CSDFM0
Aug 17                          Maintain Models                       1 of 1

Action ..................... __ A,B,C,D,M,N,P,R
Model ...................... BROWSE__________________________
Description ........ *0200.1____________________________________________________
                     BROWSE Program
  PDA name ................. CUSCPDA_    Status window ........... Y
  Programming mode ......... S_          Comment start indicator .. **_
  Type ..................... P Program   Comment end indicator .... ___

  Code frame(s) ............ CSCA?___  CSCB?___  CSCC?___  ________  ________
  Modify server specificatn  CUSCMA__  CUSCMB__  CUSCMC__  CUSCMG__  ________
                             ________  ________  ________  ________  ________
  Modify client specificatn  CUSCMA__  CUSCMB__  CUSCMC__  ________  ________
                             ________  ________  ________  ________  ________

  Clear specification ...... CUSCC___    Post-generation .......... CUSCPS__
  Read specification ....... CUSCR___    Save specification ....... CUSCS___
  Pre-generation ........... CUSCPR__    Document specification ... CUS-D___
Command ............ _________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit  frame                                         main
```

Notice that the code frame names listed in the Code frame(s) field end with a question mark (?). The question mark indicates a hierarchy in which the code frame with the lowest number at the end of its name is used.

All code frames supplied with Natural Construct end with an 8 (used for code frame fixes supplied between releases) or 9 (used for original code frames supplied with Natural Construct). To define a custom code frame for your model, copy the supplied code frame, change the 8 or 9 to a lower number (from 1 to 7), and modify the code frame as desired. The next time Natural Construct calls that code frame, the one with the lowest number is used.

For example, you can copy the CSCA9 code frame, change the name to CSCA7, and edit it as desired. The next time Natural Construct calls CSCA?, CSCA7 is used.

The naming conventions for code frames are:

- The first character in a code frame name is always C.
- The second and third characters are reserved for the two-character model identifiers, such as MN for Menu or dash (—) for generic code frames used by multiple models.
- The fourth character is a single letter from A-Z indicating a position within a series of code frames.
- The fifth, sixth, and seventh characters are optional. They indicate specific functions that are typically performed by nested code frames, such as wildcard support.
- The last character must be a number from 1-9, with 9 reserved for the Natural Construct-supplied code frames and 8 reserved for any future updates.

Note: The last character refers to the last position in the code frame name, which may or may not be the eighth physical position.

Use Nested Code Frames

When code frames are referenced in code (nested code frames), their names also end with the question mark character. For example, the CSLBA9 code frame for the Browse-Select model contains the nested code frame CS-BA?:

```
Code Frame ......... CSLBA9                                          SIZE 17120
Description ........ Browse-Select* model main body                  FREE 82673
>                                              > + ABS X X-Y _ S 214  L 1
Top...+....1....+....2....+....3....+....4....+....5....+....6....+....7.. T C
  PROG.
  REPEAT /* Repeat loop to allow escape of program from within subroutine.
  *
  ******************** Start of Main Program Logic *********************
  *
    RESET #FIRST-&UQ-FOUND #REDISPLAY-SCREEN #MATCH-FOUND
  CS-BA?                                                                F
  NOT PROCESS-SELECTION-COLUMN AND PROCESS-SELECTED-RECORD                1
    /*                                                                    "
    /* reposition to selected field if cursor selection                   "
    IF #CURS-LINE > #FIRST-ACTION-LINE                                    "
      IF #SEL-TBL.#&UQ(#CURS-LINE) NE #NULL-&UQ                           "
        ASSIGN #FORWARD = FALSE                                           "
        ASSIGN #MATCH-FOUND = FALSE                                       "
        ASSIGN #START.#KY = #SEL-TBL.#KY(#CURS-LINE)                      "
        ASSIGN #START.#&UQ = #SEL-TBL.#&UQ(#CURS-LINE)                    "
      END-IF                                                              "
    END-IF                                                                "
  ....+....1....+....2....+....3....+....4....+....5....+....6....+....7.. T  ↩
```

Code Frame Menu Function

Use this function to access the Code Frame menu.

To access the Code Frame menu:

1 Enter "F" in `Function` on the Administration main menu.

  The Code Frame menu is displayed. For example:

```
CSMMAIN                N a t u r a l   C o n s t r u c t                CSMMNM0
Aug 17                           Code Frame Menu                          1 of 1

                     Functions

                     -----------------------------------------------

                     E  Edit Code Frame

                     S  Save Code Frame

                     L  List Code Frames

                     P  Purge Code Frame

                     C  Clear Edit Buffer

                     H  Print Saved Code Frame




                     ?  Help

                     .  Return

                     -----------------------------------------------

Function ........... _

Code Frame ......... ________

Description ........ _____________________________________________


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

2 Enter the one-character function code in `Function`.

The functions available through this menu are:

- Edit Code Frame
- Save a Code Frame
- List Code Frames for Selection
- Purge a Code Frame

- Clear Edit Buffer
- Print Saved Code Frame

Note: For a description of the Help and Return functions, see ***Help and Return Codes on Menus***.

Edit Code Frame

Use this function to:

- Create a New Code Frame
- Modify an Existing Code Frame

Create a New Code Frame

To create a new code frame:

1 Enter "E" in `Function` on the Code Frame menu.

The Code Frame editor is displayed. For example:

```
 Code Frame .........                                                SIZE    ↩

 Description ........                                               FREE 56825
 >                                             > + ABS X X-Y _ S       L     ↩

   ....+....1....+....2....+....3....+....4....+....5....+....6....+....7.. T C
                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

                                                                             ↩

   ....+....1....+....2....+....3....+....4....+....5....+....6....+....7.. T ↩
↩
```

2 Type the code frame name in `Code Frame`.

3 Type a brief description of the code frame in `Description`.

4 Use the editor to create the code frame.

The Code Frame editor supports all edit commands except the RUN, CHECK, TEST, STOW, and SAVE command. For more information about the Code Frame editor, see ***Using the Code Frame Editor***.

5 Enter "." (period) at the > prompt to return to the Code Frame menu.

For information on saving the code frame, see ***Save Code Frame***.

Modify an Existing Code Frame

To modify an existing code frame:

1 Type "E" in `Function` on the Code Frame menu.

2 Type the code frame name in `Code Frame`.

3 Optionally, type a brief description of the code frame in `Description`.

4 Press Enter.

The specified code frame in displayed in the Code Frame editor.

5 Modify the code frame.

The Code Frame editor supports all edit commands except the RUN, CHECK, TEST, STOW, and SAVE command. For more information about the Code Frame editor, see ***Using the Code Frame Editor***.

6 Enter "." (period) at the > prompt to return to the Code Frame menu.

For information on saving the code frame, see ***Save Code Frame***.

Note: For more information about modifying the supplied code frames, see ***Step 5: Create Code Frame(s) and Define the Model***.

Save a Code Frame

Use this function to save the code frame that is currently in the edit buffer to the Code Frame file.

To save the code frame:

- Enter "S" in `Function` on the Code Frame menu.

  If the specified code frame name already exists, `Code Frame exists. Press Enter to confirm replace` is displayed. You can either change the name or press Enter to update the existing code frame.

List Code Frames for Selection

Use this function to display a list of available code frames for selection.

To list the available code frames for selection:

1 Enter "L" in `Function` on the Code Frame menu.

The Select Frames window is displayed. For example:

```
CSMLIST                    Natural Construct                        CSMLIST0
Oct 07                       Select Frames                            1 of 1

Frame    Description                                 User     Date      Time
-------- ---------------------------------------- -------- --------- ------
C--BAN9  Standard banner                             SAG      Sep 30,13 09:55
CBAA9    Batch define data area                      SAG      Sep 30,13 09:55
CBAB9    Batch initial setup                         SAG      Sep 30,13 09:55
CBAC9    Batch main body                             SAG      Sep 30,13 09:55
CBOA9    Object Browse Subp define data area         SAG      Sep 30,13 09:55
CBOB9    Object Browse Subp main body                SAG      Sep 30,13 09:55
CBRA9    Object Browse Static main body              SAG      Sep 30,13 09:55
CCNA9    Callnat main body                           SAG      Sep 30,13 09:55
CDRA9    Driver main body                            SAG      Sep 30,13 09:55
CETA9    Extendable Input main body                  SAG      Sep 30,13 09:55
CFMA9    Maint define data area                      SAG      Sep 30,13 09:55
 Frame .... ________ Detail ..... _ Scan for ... __________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn                         bkwrd frwrd
Position cursor or enter screen value to select                            ↩
```

This window displays the following information:

- Each code frame name in alphabetical order
- Brief description of the corresponding code frame
- User ID for the user who last saved the corresponding code frame
- Date the corresponding code frame was last saved
- Time the corresponding code frame was last saved

2 Type the name of the code frame in `Frame`.

Note: If you enter the name of a code frame that is not currently displayed, the list is repositioned.

Optionally, you can mark `Detail` and type a value to scan for in `Scan for`. Detail lines are displayed for code frames containing the scanned value only.

3 Press Enter.

Purge a Code Frame

Use this function to permanently remove a code frame from the Code Frame file.

Note: You cannot purge a code frame if it is currently used in a model.

To purge a code frame:

1. Type "P" in `Function` on the Code Frame menu.
2. Type the name of the code frame in `Code Frame`.
3. Optionally, type a brief description of the code frame in `Description`.
4. Press Enter.

   A confirmation window is displayed to confirm the purge.

Clear Edit Buffer

Use this function to clear the current values from the Code Frame editor.

To clear the edit buffer:

- Enter "P" in `Function` on the Code Frame menu.

Print Saved Code Frame

Use this function to print a hardcopy of the specifications for a code frame that has been saved.

Note: To use this function, you must have access to Com-Plete, CMS, TSO, or CICS with Natural/AF or Com-Pose. For more information, see ***Frame Hardcopy Utility***.

To print a hardcopy of a saved code frame:

1. Type "H" in `Function` on the Code Frame menu.
2. Type the name of the code frame in `Code Frame`.
3. Optionally, type a brief description of the code frame in `Description`.
4. Press Enter.

Maintain Subprograms Function

Use this function to maintain the modify specification subprograms used by the generation models.

To maintain the modify specification subprograms for a model:

1 Enter "S" in `Function` on the Administration main menu.

The Maintain Subprograms panel is displayed. For example:

```
CSDFSP              N a t u r a l   C o n s t r u c t              CSDFSP0
Aug 17                      Maintain Subprograms                      1 of 1

Action ....................... __ A,B,C,D,M,N,P,R

Subprogram .................. ________

Description ........ ___________________________________________________________



PF-keys Used

Backward - Forward ........... _

Test ......................... _


Assign to #PDA-PF-AVAILABLE1 . __________

Assign to #PDA-PF-AVAILABLE2 . __________

Assign to #PDA-PF-AVAILABLE3 . __________


Optional Window Settings

Window height ................ ___

Window width ................. ___


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

Use this panel to maintain the PF-key and window settings for the model subprograms. The Natural Construct nucleus uses these settings to determine the window size and PF-key functions for the model maintenance panels and sample subprograms.

Caution: You cannot change these settings for model subprograms shipped with Natural Construct; you can only change the settings for model subprograms you create.

2 Type an action code in `Action`.

For a description of the available actions, press PF1 (help) when the cursor is in the field.

3 Type the name of the subprogram in `Subprogram`.

4 Press Enter.

The PF-key and window settings for the model are displayed.

Maintain Control Record Function

Use this function to maintain the default PF-key numbers and names, special characters, and dynamic attribute settings for Natural Construct.

Note: These settings are for Natural Construct only, not for Natural Construct-generated programs.

To maintain the control record:

1 Enter "R" in `Function` on the Administration main menu.

The Maintain Control Record panel is displayed. For example:

```
CSCTRL               N a t u r a l   C o n s t r u c t                 CSCTRL0
Aug 17                    Maintain Control Record                        1 of 1
                                                                               ↩
PF-key Assignments                                     Dynamic Attributes      ↩

Main ............. PF 12 NAMED *0031.5___ main         Intensify .......... < ↩

Return ........... PF 2_ NAMED *0031.2___ retrn        Blue ............... _ ↩

Quit ............. PF 3_ NAMED *0031.3___ quit         Green .............. _ ↩

Test ............. PF 4_ NAMED *0031.4___ test         White .............. _ ↩

Backward ......... PF 7_ NAMED *0032.2___ bkwrd        Pink ............... _ ↩

Forward .......... PF 8_ NAMED *0032.1___ frwrd        Red ................ _ ↩

Move left ........ PF 10 NAMED *0032.3___ left         Turquoise .......... _ ↩

Move right ....... PF 11 NAMED *0032.4___ right        Yellow ............. _ ↩

Help ............. PF 1_ NAMED *0031.1___ help         Special Hardware       ↩

User exit ........ PF 11 NAMED *0032.5___ userX        Blinking ........... _ ↩

Help indicator ............... *0033.1___ *            Italic ............. _ ↩

Underscore character ......... *0033.2___ ----         Underline .......... _ ↩

Of indicator (eg., 1 of 2) ... *0033.3___ of           Reverse video ...... _ ↩

Disable indicator ............ *0033.4___ -                                   ↩

Scroll indicator ............. *0033.5___ >>          Default return ..... > ↩

Position indicator(s) ........ *0034/4___ 1   2   3   4   5   6   7   8   9   ↩

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

2 Use the fields on this panel to specify settings for the control record.

The fields on this panel are:

Column Heading	Field	Description
PF-key Assignments	PF*n*	PF-key numbers for the corresponding functions. For each function (Main, Return, Quit, etc.), specify the number of the PF-key that performs the function. These functions are: ■ Main (invokes main menu) ■ Return (displays previous panel) ■ Quit (terminates current session) ■ Test (invokes the Test function) ■ Backward (scrolls backward/up through data) ■ Forward (scrolls forward/down through data) ■ Move left (scrolls to panel on the left of current panel) ■ Move right (scrolls to panel on the right of current panel) ■ Help (invokes help for current panel) ■ User exit (invokes the User Exit editor) **Note:** Only PF-keys 1 through 12 are defined. PF-keys 13 to 24 are equivalent to PF-keys 1 to 12, respectively.
	NAMED	PF-key names for the corresponding functions or the SYSERR numbers that supply the names. The current names are displayed on the right (main, retrn, quit, etc.). Because PF-key settings are part of the user interface, you can specify a SYSERR number from the CSTLDA library as the PF-key name. For example, SYSERR number *0031.5 corresponds to the English text "main". If you specify *0031.5 in one of the NAMED fields, the corresponding PF-key name is "main".
	Help indicator	Character used to indicate that help is available for a panel field (the default is *) or the SYSERR number that supplies the character. The indicator is placed in a separate prompt to the right of the input field.
	Underscore character	One- to 4-character set used to create the underscore line for panel text (the default is ----) or the SYSERR number that supplies the character set. The specified set is repeated until all spaces are filled (80, by default). For example, if "----" is specified, the underscore line is displayed as: - Or if "++" is specified, the underscore line is: ++ ++ ++ ++ ++ ++ ++ ++ ++ ++
	Of	indicator Character(s) used to indicate the current panel and the number of additional panels (the default is "of" as in "1 of 2") or the SYSERR number that supplies the character(s).

Column Heading	Field	Description
	Disable indicator	Character used to indicate that an option is unavailable on a panel (the default is -) or the SYSERR number that supplies the character.
	Scroll indicator	Character(s) used to indicate that scrolling is available for a field on a panel (the default is >>) or the SYSERR number that supplies the character(s).
	Position indicator(s)	Characters used to indicate a position in a series of positions (the defaults are 1 to 10) or the SYSERR number that supplies the characters. If you are not using SYSERR, change the default characters by typing the new characters on the lines below this field.
Dynamic Attributes		Default dynamic attributes. You can specify up to four attributes, one of which must be the return to normal display attribute (see the description for the Default return field). The attributes are:
	Intensify	Character used to intensify text.
	Blue	Blue display for color terminals.
	Green	Green display for color terminals.
	White	White display for color terminals.
	Pink	Pink display for color terminals.
	Red	Red display for color terminals.
	Turquoise	Turquoise display for color terminals.
	Yellow	Yellow display for color terminals.
Special Hardware		Options available for terminals with special hardware. **Note:** Due to hardware restrictions, you may not be able to use all the options listed. For more information, refer to *DY Session Parameter* in the *Natural Parameter Reference* documentation. The special hardware options are:
	Blinking	Support for blinking.
	Italic	Support for italic.
	Underline	Support for underline.
	Reverse video	Support for reverse video.
Default return		Character used to return to normal (default) display; the default is >. A character must be specified in this field.

Note: For more information on using SYSERR, see ***Using SYSERR for Multilingual Support***.

Compare Menu Function

Use this function to access the Compare menu.

To access the Compare menu:

1 Enter "C" in `Function` on the Administration main menu.

The Compare menu is displayed. For example:

```
CSDCMMF               N a t u r a l   C o n s t r u c t               CSDCMMF0
Aug 08                           Compare Menu                             1 of 1

                     Functions

                     ----------------------------------------------

                     M  Compare Models

                     F  Compare Frames









                     ?  Help

                     .  Return

                     ----------------------------------------------

Function ........... _




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

2 Enter the one-character function code in `Function`.

The functions available through this menu are:

- Compare Models

- Compare Frames

Note: For a description of the Help and Return functions, see ***Help and Return Codes on Menus***.

Compare Models

Use this function to:

- Compare a Model in Different Files
- Compare Two Models in the Same File
- Compare a Range of Models in Different Files

To access the Compare Models function:

- Enter "M" in `Function` on the Compare menu.

  The Compare Models panel is displayed. For example:

```
CSDCMP                 N a t u r a l   C o n s t r u c t              CSDCMP10
Apr 02                             Compare Models                       1 of 1
             Old                               New
Model ...... ________________________________  ________________________________
Database ... _____                             _____
File ....... _____                             _____
Version .... _____                             _____
Command ............ _________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                                                    main
```

Note: The Old and New designation does not limit the comparison to old and new versions of the same model.

Compare a Model in Different Files

Use this function to compare the components of a model in different files. You can compare the same model or different models. In the following example, the same model is compared.

To compare the same model in different files:

1. Type the name of the model in `Old Model` and `New Model` on the Compare Models panel.
2. Type the database identification (DBID) number for the Natural Construct system file for the first model in `Old Database`.
3. Type the DBID for the second model in `New Database`.

4 Type the Natural Construct file number for the first model in `Old File`.

5 Type the Natural Construct file number for the second model in `New File`.

6 Type the Natural Construct version number for the first model in `Old Version`.

7 Type the Natural Construct version number for the second model in `New Version`.

For example:

```
CSDCMP                N a t u r a l   C o n s t r u c t              CSDCMP10
Apr 02                          Compare Models                         1 of 1
             Old                             New
Model ...... BROWSE__________________________ BROWSE__________________________
Database ... 18___                           18___
File ....... 116__                           120__
Version .... 5.3.2                           8.2.1
Command ........... __________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                                                  main
```

8 Press Enter.

The Show Model Differences window is displayed, showing the differences between the two models. For example:

```
CSDCMPD                  Natural Construct
Aug 08                 Show Model Differences

Old ........ 5.3.2                            New ........ 8.2.1
                    BROWSE                       BROWSE
------------------- ---------------------------- ------------------------
Description ......  *0200.1                      *0200.1
Save subpr .......  CUSCGST                      CUSCS
Pre-generate .....  CUSCGPR                      CUSCPR
Post-generate ....  CUSCGPS                      CUSCPS
Document .........  CUSCDOC1                     CUS-D
Modify   1                                       CUSCMA
Modify   2                                       CUSCMB
Modify   3                                       CUSCMC
Frame .......... 1  CUBANNER                     CSCA?
Frame .......... 2  CUSCDA                       CSCB?
Frame .......... 3  CUSCC1                       CSCC?
Frame .......... 4  CUSCC2
Frame .......... 5  CUSCC3                                             ↩
```

Compare Two Models in the Same File

Use this function to compare the components of two models in the same file.

To compare two models in the same file:

1 Type the name of the first model in `Old Model` on the Compare Models panel.

2 Type the name of the second model in `New Model`.

3 Type the database identification (DBID) number for the Natural Construct system file for the models in `Old Database`.

4 Type the Natural Construct file number for the models in `Old File`.

5 Type the Natural Construct version number for the models in `Old Version`.

For example:

```
CSDCMP                 N a t u r a l   C o n s t r u c t            CSDCMP10
Apr 02                          Compare Models                        1 of 1
             Old                           New
Model ...... BROWSE__________________________ BROWSE-SELECT___________________
Database ... 18___                         _____
File ....... 121__                         _____
Version .... 5.3.2
Command ........... __________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                                                   main
```

6 Press Enter.

The Show Model Differences window is displayed, showing the differences between the two models. For example:

```
CSDCMPD                    Natural Construct
Aug 08                  Show Model Differences

Old ........ 5.3.2                            New ........ 8.2.1
                    BROWSE                       BROWSE-SELECT
                    ---------------------------- ------------------------
Clear subpr ......  CUSCC                        CUSLC
Pre-generate .....  CUSCPR                       CUSLPR
Post-generate ....  CUSCPS                       CUSLPS
Modify   Host   2   CUSCMB                       CUSLMB
Modify   Host   4   CUSCMG                       CUSLMD
Modify   Host   5                                CUSCMG
Modify   4                                       CUSLMF
Frame .......... 1  CSCA?                        CSLA?
Frame .......... 2  CSCB?                        CSLB?
Frame .......... 3  CSCC?                        CSLC?
Date .............  Jul 31,2013                  Oct 21,2013
Time .............  10:09.510                    10:09.510
User .............  SAG                          SAG                   ↩
```

Compare a Range of Models in Different Files

Use this function to compare the components for a range of models in different files. You can compare the same range of models or a different range. In the following example, the same range is compared.

To compare a range of models in different files:

1. Type the starting value for the range in `Old Model` on the Compare Models panel.

   The starting value can be either the name of a model or the first few characters in the name. You can also limit the range by entering the wildcard character (*) with the model name. For example, if you enter Browse*, all the Browse models are compared. For information about using wildcard characters, refer to *Wildcard Selection*, *Natural Construct Generation*.
2. Type the database identification (DBID) number for the first range of models in `Old Database`.
3. Type the DBID for the second range in `New Database`.
4. Type the Natural Construct file number for the first range of models in `Old File`.
5. Type the Natural Construct file number for the second range in `New File`.
6. Type the Natural Construct version number for the first range of models in `Old Version`.
7. Type the Natural Construct version number for the second range in `New Version` field.
8. Press Enter.

   The Show Model Differences window is displayed, showing the differences between the two ranges of models. For a description of this window, see ***Compare a Model in Different Files***.

Compare Frames

Use this function to:

- Compare Two Code Frames in Different Files
- Compare All Frames For Two Models
- Compare a Range of Frames in Different Files

The models containing the code frames can reside in different system files. You can also compare all code frames and nested code frames for a model. The code frames can be different code frames in the same file, the same code frames in different files, or different code frames in different files. Results are presented code frame by code frame.

For information on comparing code frames in batch mode, see ***Comparison Utilities***.

To access the Compare Frames panel:

- Enter "F" in `Function` on the Compare menu.

  The Compare Frames panel is displayed. For example:

```
CSDCMP              N a t u r a l   C o n s t r u c t          CSDCMP20
Aug 08                          Compare Frames                     1 of 1




             Old                              New

Model ...... ________________________________ ________________________________
Frame ...... ________                         ________

Database ... ___                              ___

File ....... ___                              ___

Version ....










Command ......... __________________________________________________________

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

Note: The Old and New designation does not limit the comparison to old and new versions of the same model or code frame.

Compare Two Code Frames in Different Files

Use this function to compare two code frames in different files. You can compare the same code frame or different code frames. In the following example, the same code frame is compared.

To compare the same code frame in different files:

1. Type the name of the code frame in `Old Frame` and `New Frame` on the Compare Frames panel.
2. Type the database identification (DBID) number for the Natural Construct system file for the first frame in `Old Database`.
3. Type the DBID for the second frame in `New Database`.
4. Type the Natural Construct file number for the first frame in `Old File`.
5. Type the Natural Construct file number for the second frame in `New File`.
6. Type the Natural Construct version number for the first frame in `Old Version`.
7. Type the Natural Construct version number for the second frame in `New Version`.

   For example:

```
CSDCMP                N a t u r a l   C o n s t r u c t              CSDCMP20
Aug 08                           Compare Frames                         1 of 1




            Old                            New

Model ...... ______________________________ ______________________________
Frame ...... CUBADA9_                      CBAA9___

Database ... 18_                           18_

File ....... 116                           121

Version .... 5.3.2                         8.2.1










Command ......... ____________________________________________________

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

8 Press Enter.

The Summary Report window is displayed, showing the differences between the two frames. For example:

```
 CSDCMPFD                   Natural Construct                        CSDCMP
                              Summary Report


 Old version .... 5.3.2              New version .... 8.2.1
       Frame .... CUBADA9                  Frame .... CBAA9


  Old    New    Matched    Deleted    Inserted            Comments
 ------ ------ ---------- ---------- ---------- ---------------------------
   284    292     284          0          8    Frames do not match
 Press ENTR to continue or any PF-key to retrn                             ↩
```

The Summary Report window displays the following information:

- Version numbers
- Name of each code frame
- Number of lines of code for each code frame
- Number of lines that match
- Number of lines removed from the first code frame
- Number of lines added to the second code frame
- Whether the code frames match (in this example, they do not match)

9 Press Enter.

The Compare Frames window is displayed, showing a line-by-line comparison. For example:

```
Oct 07                        Natural Construct                       04:15 PM
                               Compare Frames                         PAGE: 1
  Old version .... 5.3.2                            New version .... 8.2.1    ↩

                                                                              ↩

                                CUBADA9/CBAA9                               T C
   ---------------------------------------------------------------------- - -
 +  C--BAN?                                                                 F
 =  DEFINE DATA                                                               ↩

 =  GDA-SPECIFIED                                                             ↩
1
 =          _______________ 33 more equal lines _______________           ↩

 =    ET-SPECIFIED                                                            ↩
2
 =    01 #HOLD-COUNT(P3)                                                      ↩
"
 +    01 #WRITE-LINE(A30)                                                     ↩

 =                                                                          *
 =  Secondary file 1 key for ADABAS, VSAM, DB2                              *
 =          _______________ 161 more equal lines _______________          ↩

 =    01 #INPUT1                                                              ↩
"
 =      KEY-IS-REDEFINED OR KEY-IS-COMPOUND                                   ↩
3
 +      02 #INPUT1-FIELDS(&KEY-NAT-FORMAT)                                    ↩
"
 +      02 REDEFINE #INPUT1-FIELDS                                            ↩
"
 =  CUBAGRED REDEFINE-INPUT-KEY                                             N "
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
frwrd       retrn             top   hcopy       frwrd                         ↩
 ↩
```

The lines in the code frames that match are marked with an equal sign (=). Lines that are in the first code frame, but not in the second, are marked with a minus sign (-). Lines that are in the second code frame, but not in the first, are marked with a plus sign (+).

- To scroll forward (down) through the information, press Enter or PF8 (frwrd).
- To return to the first line, press PF5 (top).
- To return to the Compare Frames panel, press PF2 (retrn).
- To print a hardcopy of the Code Frame Compare Utility panel, press PF6 (hcopy).

  For more information on printing a hardcopy of a code frame, see ***Print Saved Code Frame***.

Compare All Frames For Two Models

Use this function to compare all the code frames used by two models.

To compare all the code frames used by two models:

1. Type the name of the first model in `Old Model` on the Compare Frames panel.
2. Type the name of the second model in `New Model`.
3. Type the database identification (DBID) number for the Natural Construct system file for the first model in `Old Database`.
4. Type the DBID for the second model in `New Database`.
5. Type the Natural Construct file number for the first model in `Old File`.
6. Type the Natural Construct file number for the second model in `New File`.
7. Type the Natural Construct version number for the first model in `Old Version`.
8. Type the Natural Construct version number for the second model in `New Version`.

   For example:

```
CSDCMP                N a t u r a l   C o n s t r u c t            CSDCMP20
Aug 08                           Compare Frames                      1 of 1




            Old                            New

Model ...... BROWSE________________________ BROWSE-SELECT_________________
Frame ...... ________                       ________

Database ... 18_                            18_

File ....... 116                            121

Version .... 5.3.2                          8.2.1









Command ......... ________________________________________________________

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

9 Press Enter.

The Summary Report window is displayed, showing the differences between the two models.

10 Press Enter.

The Compare Frames window is displayed, showing a line-by-line comparison. For a description of the Summary Report and Compare Frames window, see ***Compare Two Code Frames in Different Files***.

Compare a Range of Frames in Different Files

Use this function to compare the components for a range of frames in different files. You can compare the same range of frames or a different range. In the following example, the same range is compared.

To compare a range of frames in different files:

1 Type the starting value for the range in `Old Frame` on the Compare Frames panel.

The starting value can be either the name of a code frame or the first few characters in the name. You can also limit the range by entering the wildcard character (*) with the code frame name. For example, if you enter CFM*, all code frames that begin with CFM are compared. For more information on using wildcards, refer to *Wildcard Selection*, *Natural Construct Generation*.

2 Type the Database identification (DBID) number for the first range of frames in `New Database`.

3 Type the DBID for the second range in `Old Database`.

4 Type the Natural Construct file number for the first range of frames in `New File`.

5 Type the Natural Construct file number for the second range in `Old File`.

For example:

```
CSDCMP               N a t u r a l   C o n s t r u c t            CSDCMP20
Aug 08                         Compare Frames                          1 of 1




            Old                            New

Model ...... ________________________________ ________________________________
Frame ...... CG______                       ________

Database ... 18_                            18_

File ....... 116                            121

Version ....










Command ......... ___________________________________________________________

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

6 Press Enter.

The Select Frames window is displayed. For example:

```
CSDCMPF                  Natural Construct                         CSDCMF0
Oct 07                     Select Frames                            1 of 1

  Frame              Old                   New
  ----------------- -------------------- --------------------
_ CGMA9              DATE: 13-09-03 09:46 DATE: 13-10-27 15:03
_ CGOA9              DATE: 13-09-30 09:55 DATE: 13-10-27 15:03
_ CGPA9              DATE: 13-09-30 09:55 DATE: 13-10-27 15:03
_ CGRA9              DATE: 13-09-30 09:55 DATE: 13-10-27 15:03
_ CGSA9              DATE: 13-09-30 09:55 DATE: 13-10-27 15:03
_ CHDA9              DATE: 13-09-30 09:55 DATE: 13-10-27 15:03
  CMDA9              Does not Exist       DATE: 13-10-27 15:03
_ CMNA9              DATE: 13-09-30 09:55 DATE: 13-10-27 15:03
_ CN-BAN9            DATE: 13-09-30 09:55 DATE: 13-10-27 15:03
_ CNDA9              DATE: 13-09-30 09:55 DATE: 13-10-27 15:03
_ CNOA9              DATE: 13-09-30 09:55 DATE: 13-10-27 15:03
_ COBA9              DATE: 13-10-07 11:12 DATE: 13-10-27 15:03
Code frame name .... CFM_____
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF
      help  retrn                          bkwrd frwrd
Position cursor or enter screen value to select                           ↩
```

Use this window to select frames and display the comparison information.

7 Type "C" in the input field for any code frame.

8 Press Enter.

The Summary Report window is displayed, showing the differences between the two ranges of frames.

9 Press Enter.

The Compare Frames window is displayed, showing a line-by-line comparison. For a description of the Summary Report and Compare Frames window, see ***Compare Two Code Frames in Different Files***.

Drivers Menu Function

Use this function to access the Drivers menu, which provides access to various utility subprograms supplied with Natural Construct.

▶ To access the Drivers menu:

1 Enter "D" in `Function` on the Administration main menu.

The Drivers Menu panel is displayed. For example:

```
CTEMENU             N a t u r a l   C o n s t r u c t                CTEMNMO
Oct 31                        Drivers Menu                            1 of 1

                     Functions
                     -----------------------------------------------
                     P  Predict-Related Drivers Menu
                     N  Natural-Related Drivers Menu
                     M  Miscellaneous Drivers Menu







                     ?  Help
                     .  Return
                     -----------------------------------------------
Function ........... __
Command ........... ______________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                                               lang  ↩
 ↩
```

The drivers used to access the utilities are grouped according to what kind of subprogram they invoke. For a description of each menu function and the subprogram it invokes, refer to the applicable *Drivers Menu Option* section in **External Objects**.

2 Enter the one-character function code in `Function`.

> **Note:** For a description of the Help and Return functions, see ***Help and Return Codes on Menus***.

Multilingual Support for Natural Construct

You can install Natural Construct in static (single) or dynamic (multiple) language mode. If dynamic language mode is installed, you can change your *Language value at runtime and display text in another supported language. You can also use the Natural SYSERR utility to add translations for the supplied text or change the supplied text to suit your organization's standards.

- For information on installing Natural Construct in dynamic language mode, see the installation documentation.
- For information on installing Natural Construct in static language mode, see ***Static (One-Language) Mode***.
- For more information on using SYSERR, see ***Using SYSERR for Multilingual Support***.

Libraries Supplying Multilingual Text

In dynamic language mode, all text displayed by Natural Construct is supplied by the Natural SYSERR utility from the following libraries:

- CSTLDA (all panel and window text)
- CSTMSG (all message text)

Natural Construct checks the value of the *Language variable to determine which language to display and retrieves the text for that language from the appropriate library.

Note: For information on the SYSERR utility, refer to the Natural utilities documentation.

Display Text in Another Language

To change the text displayed on panels to another supported language:

1 Press PF12 (lang) on the Administration main menu.

The Language Preference window is displayed. For example:

```
CSULPS               Natural Construct               CSULPS0
Aug 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              ↩
```

2 Select the desired language.

The main menu is displayed in the selected language.

English (*Language 1) is the default language for Natural Construct. Although other languages are listed in the Language Preference window, you must add the translations for those languages in SYSERR.

If you do not provide translated text for a selected language, Natural Construct determines which language to display based on a user-defined hierarchy of language numbers (defined in the DEFAULT-LANGUAGE field in the CNAMSG local data area for the CNUMSG subprogram). For more information, see *CNUMSG Utility*.

Maintain Panel and Message Text

To define the text for another language, you must first change the *Language value in the Language Preference window. For information, see *Display Text in Another Language*.

To add text for another language or modify the supplied text:

- Use the SYSERR utility to add translations or modify the supplied text for all Natural Construct screens. Using the SYSERR utility is the quickest way to translate text on all panels.

  Or:

- Use the Administration subsystem in translation mode to dynamically add translations or modify the supplied text. Typically, you would use translation mode to fine tune translations that were added using the SYSERR utility. This allows you to view the translation in the context of the entire panel. For information about translation mode, see *Access the Administration Main Menu in Translation Mode*.

Access the Administration Main Menu in Translation Mode

To help maintain the text for Natural Construct panels, windows, and messages, the Administration subsystem is also available in translation mode. Translation mode allows you to change the text supplied in the Natural SYSERR utility without leaving Natural Construct. You can change the text displayed on the Administration main menu, as well as on panels and help or selection windows for each function available through the Administration main menu.

You can also change the text displayed on the Generation and Help Text subsystem screens. For information, see *Translate Text for the Generation Subsystem* and *Translate Text for the Help Text Subsystem*.

The current value of the *Language variable determines whether you can maintain text for the current language or for another language.

To Invoke in translation mode:

- Enter "menut" at the Natural prompt.

The Administration main menu is displayed. For example:

```
CSDMAIN              N a t u r a l   C o n s t r u c t               CSDMNMO
Aug 08                    Administration Main Menu                    1 of 1

                     Functions

                     ----------------------------------------------

                     M  Maintain Models

                     F  Code Frame Menu

                     S  Maintain Subprograms

                     R  Maintain Control Record

                     C  Compare Menu

                     D  Drivers Menu

                     H  Help Text Main Menu

                     G  Generation Main Menu

                     ?  Help

                     .  Return

                     ----------------------------------------------

Function .......... _




Command ..... ________________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                                                  lang
```

Use this panel to access the Natural Construct Administration functions in translation mode. Notice that functions are also available to access the Help Text and Generation main menus in translation mode.

Note: Although the panels look the same in translation mode, they do not perform the same functions. For example, edit checks are not performed on input data. We recommend that you do not use translation mode for maintenance functions, such as defining a new model; use translation mode for translation functions, such as editing text in the current language or creating multilingual specification panels and messages.

This section covers the following topics:

- Use Translation Mode

Use Translation Mode

Translation mode uses the same series of panels and windows used throughout Natural Construct. All translatable text is cursor sensitive. When you select the text and press Enter, the Translate Short Message window is displayed. You can identify translatable text by the difference in color or intensification.

Note: If you use Entire Connection to access Natural Construct, you can display the Translate Short Message window by double-clicking on translatable text.

You can translate two types of text:

- Screen text (text displayed on panels and in windows), which is stored in the CSTLDA library in SYSERR
- Message text, which is stored in the CSTMSG library in SYSERR

Each Natural Construct panel or window is associated with a local data area (LDA) that initializes the screen prompt variables. In translation mode, these variables are initialized to a SYSERR number and the actual text values are retrieved at runtime (based on the current value of the Natural *Language system variable).

You can use SYSERR numbers for some or all screen prompts. If you specify text as an initial value, Natural Construct displays the text as entered and the prompt cannot be dynamically translated.

When you use a SYSERR number instead of text, Natural Construct retrieves the corresponding text from the CSTLDA library (for prompts) or the CSTMSG library (for messages) in SYSERR. All changes to the values stored in SYSERR are automatically applied to the panels and messages the next time they are invoked.

Note: For more information on substitution variables, refer to *REINPUT Statement*, *Natural Statements* documentation.

This section describes how to perform the following tasks:

- Translate Text for the Generation Subsystem
- Translate Text for the Help Text Subsystem
- Edit Text in the Current Language
- Translate Text to Another Language

- Use Substitution Variables

Translate Text for the Generation Subsystem

To translate text for the Generation subsystem:

1. Type "G" in `Function` on the Administration main menu in translation mode.
2. Press Enter.

   The Generation main menu is displayed in translation mode.
3. Translate the text as desired.

Translate Text for the Help Text Subsystem

To translate text for the Help Text subsystem:

1. Type "H" in `Function` on the Administration main menu in translation mode.
2. Press Enter.

   The Help Text main menu is displayed in translation mode.
3. Translate the text as desired.

Edit Text in the Current Language

Using translation mode, you can dynamically edit the text displayed on Natural Construct panels in the current language — without invoking the Natural map or code editor. For example, you can change the field prompt values to match your organization's conventions.

To edit text in the current language:

1. Invoke in translation mode.
2. Access the panel you want to translate.
3. Move the cursor to the prompt text you want to change (not a blank input line).
4. Press Enter.

   The Translate Short Message window is displayed. For example:

```
CSUTLATE                    Natural Construct
Aug 08                   Translate Short Message                    1 of 1

Language Short Message ( CSTLDA1116 )
-------- ....+....1....+....2....+....3....+....4....+....5....+....6....+

English  Action/Subprogram                                         /+26
```

This window provides quick access to the SYSERR numbers and text. Any changes made to the text in this window are automatically applied in SYSERR. The "/+26" value in this window indicates there are up to 26 characters available for each text segment that is to be translated. For more information on using the Translate Short Message window, see *Context Translation*.

Note: Take care when changing the text for SYSERR numbers that are used on other panels.

5 Edit the SYSERR text as desired.

6 Press Enter.

The panel for which you are translating text is displayed, showing the edited text.

Translate Text to Another Language

Use translation mode to add translations for prompt text on panels and windows. For example, you can create specification panels in French (*Language 3).

To translate text to another language:

1 Invoke in translation mode.

2 Press PF12 (lang).

The Language Preference window is displayed. For a description of this window, see *Display Text in Another Language*.

3 Move the cursor to the line containing the language for which you want to translate text.

4 Press Enter.

The Administration main menu is displayed.

5 Display the panel you want to translate.

For this example, the Maintain Models panel is translated to French.

6 Move the cursor over the prompt text you want to change (not a blank input line).

7 Press Enter.

The Translate Short Message window is displayed. For example:

```
CSUTLATE                    Natural Construct
Oct 07                    Translate Short Message                    1 of 1

Language Short Message ( CSTLDA1116 )
-------- ....+....1....+....2....+....3....+....4....+....5....+....6....+

English  Action/Subprogram                                          /+30
Francais                                                               ↩
```

8 Type the French equivalent under the English text.

The "/+30" value in this window indicates that you can use up to 30 characters for each text segment that is to be translated.

9 Press Enter.

The panel for which you are translating text is displayed, showing the translated text.

10 Repeat steps 6 through 9 until all text is translated.

You can translate text on any Natural Construct panel or window by invoking that panel or window and performing the translation procedure.

Note: To display the Generation and Help Text subsystem screens, see ***Translate Text for the Generation Subsystem*** and ***Translate Text for the Help Text Subsystem***.

Use Substitution Variables

Within SYSERR, you can provide text in different languages for each SYSERR number. For even greater reusability, you can use a substitution variable (such as `:1:`) with the text. Typically, the :*n*: variables are used in messages and the prompt is substituted for the :*n*: value. The actual text displayed depends on the value of the *Language variable for the user who accessed the panel.

Note: For more information on substitution variables, refer to *REINPUT Statement*, *Natural Statements* documentation.

Access and Use the Sample Exit Subprograms

Natural Construct supplies several sample exit subprograms you can use to:

- Implement security
- Restrict access to various Natural Construct modules (models, code frames, model subprograms, help text members)
- Define model aliases for use in the Generation subsystem
- Provide user-defined defaults

Tip: Always keep a backup copy of your modified sample exit subprograms.

The Natural Construct installation tape contains the sample exit subprograms. The subprograms are initially loaded into the SYSCSTX library, which is created during installation.

To modify a sample exit subprogram:

1. Use the SYSMAIN utility to copy the subprogram to the SYSCST library.
2. Modify the subprogram as desired.
3. Use SYSMAIN to copy the object code to the library indicated in *Supplied Sample Exit Subprograms*.

This section covers the following topics:

Supplied Sample Exit Subprograms

The following table lists each sample exit subprogram, the library in which Natural Construct will search for the subprogram, and the function supported by the subprogram. When a user selects a module and action, Natural Construct checks the library indicated below and invokes the applicable subprogram. The supplied subprograms are:

Subprogram	Library	Function
CSXAUEXT	SYSLIBS	Support for model alias names.
CSXCNAME	SYSLIBS	Security for the Generation main menu (before the post-generation subprogram is invoked).
CSXDEFLT	SYSLIBS	User-defined default values for generation models.
CSXDUEXT	SYSCST	Security for the Administration main menu.

Subprogram	Library	Function
CSXFUEXT	SYSCST	Security for the Code Frame menu.
CSXHUEXT	SYSLIBS	Security for the Help Text main menu.
CSXMUEXT	SYSCST	Security for the Maintain Model function.
CSXPSCHG	SYSLIBS	Security for the Generation main menu (after all substitution values are generated into the program).
CSXSECX	SYSLIBS	Support for customized security routines.
CSXTRANS	SYSLIBS	Support for special processing before an END or BACKOUT TRANSACTION statement is issued. Uses the same parameters as CSXSECX, with the addition of a timestamp parameter.
CSXSUEXT	SYSCST	Security for the Maintain Subprograms function.

Define Default Specifications

Natural Construct reads the default specifications for a model into the editor whenever the clear subprogram is invoked for a model. This occurs when the:

- Clear Specifications and Editor function is invoked and a model name is specified
- Modify Specifications function is invoked for a new model

To set default values for the model parameters, edit the clear subprogram for the model.

This section covers the following topics:

- Determine the Name of the Clear Subprogram
- Set the Default Specification Values
- Use CSXDEFLT Overrides
- Assign Your Own Defaults
- Use Predict Keywords

Determine the Name of the Clear Subprogram

To determine the name of the clear subprogram for the model:

1. Logon to the SYSCST library.
2. Enter the following on the command line:

   ```
   Menu,M
   ```

   The Maintain Models panel is displayed. For example:

```
CSDFM                  ***** Natural Construct *****               CSDFM0
Aug 18                        Maintain Models                       1 of 1

Action .................... ___ A,B,C,D,M,N,P,R

Model ..................... ________________________________

Based on model ............ ________________________________

Description ........ ________________________________________________________


 PDA name ................ ________    Status window ........... _

 Programming mode ......... __        Comment indicators ....... ___ \ ___
 Type ..................... _         Programming Language ..... ________ *
 Code frame(s) ........... ________  ________  ________  ________  ________

 Modify server ........... ________  ________  ________  ________  ________

                           ________  ________  ________  ________  ________

 Modify client ........... ________  ________  ________  ________  ________

                           ________  ________  ________  ________  ________

 Clear ................... ________    Post-generation .......... ________

 Read .................... ________    Save ..................... ________

 Pre-generation .......... ________    Document ................. ________

 Validate ................ ________    Stream ................... ________

Command ........... ______________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit  frame                                          main
```

3 Enter "B" in Action.

The Select Models window is displayed.

4 Select the model name.

The information for that model is displayed. For example:

```
CSDFM                   ***** Natural Construct *****                 CSDFM0
Aug 18                         Maintain Models                        1 of 1

Action .................... ___ A,B,C,D,M,N,P,R

Model ..................... OBJECT-BROWSE-DIALOG____________

Based on model ............ ________________________________

Description ........ *0201.1__________________________________________________
                     OBJECT-BROWSE-DIALOG Subprogram

 PDA name ................ CUBDPDA_    Status window ........... N

 Programming mode ......... S_         Comment indicators ....... **_ \ ___
 Type ..................... N Subprog.  Programming Language ..... NATURAL_ *
 Code frame(s) ........... CBDA?___  CBDB?___  ________  ________  ________

 Modify server ........... CUBDMA__  CUBDMB__  ________  ________  ________

                           ________  ________  ________  ________  ________

 Modify client ........... WCNBDMA_  WCNBDMB_  ________  ________  ________

                           ________  ________  ________  ________  ________

 Clear ................... CUBDC___    Post-generation .......... CUBDPS__

 Read .................... CUBDR___    Save ..................... CUBDS___

 Pre-generation .......... CUBDPR__    Document ................. CUBDD___

 Validate ................ CUBDVAL_    Stream ................... CUBDT___

Command ........... ___________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit  frame                                           main

Model OBJECT-BROWSE-DIALOG displayed successfully
```

In this example, the clear subprogram is called CUBDC and the PDA name is CUBDPDA.

Set the Default Specification Values

To set the default specification values for a model:

1. Log onto the SYSCST library.
2. Edit the clear subprogram for the model.

   For example, the default values in the CUBDC subprogram for the Object-Browse-Dialog model are:

   ```
   IF #PDAX-DESCRIPTION(1) = ' ' THEN
        #PDAX-DESCRIPTION(1) :=
             'This dialog is used for the object browse ...'
     END-IF
   ```

3. Compile CUBDC.
4. Use the SYSMAIN utility to copy the object code for the clear subprogram to the SYSLIBS library.

   The new defaults will now be used.

Use CSXDEFLT Overrides

When there is a default specification value that affects several models, you can set this value in the supplied CSXDEFLT subprogram. This subprogram provides default values for model parameters that can be overridden on the specification panels, as well as internal model parameters that are not displayed on the panels.

Tip: Natural Construct has identified the most common parameters that fit this category. To see what they are, invoke CSUGETDF from the SYSCST library.

To change the default values of these parameters, edit CSXDEFLT in the SYSCSTX library. For example, to change DATE-EDIT-MASK (by default, LLL' 'ZD','YY) to 08 Aug11, change CSXDEFLT as follows:

```
VALUE 'DATE-EDIT-MASK'
    CSADEFLT.PARM-VALUE := 'YY'',''LLL'' ''ZD'
CSADEFLT.PARM-VALUE := 'LLL'' ''ZD'',''YY'  ↩
```

To use the new default values, CSXDEFLT must exist in the SYSLIBS library and the model's clear subprogram must call this subprogram. For an example of calling CSXDEFLT, refer to the CUFMC clear subprogram in the SYSCST library. For example:

```
INCLUDE CCDEFLTA '''DATE-EDIT-MASK''' '#PDA-DATE-EDIT-MASK'
```

Three modules in CUFMC are used to query the defaults: CCDEFLTN, CCDEFLTA, and CCDEFLTL.

The supplied INCLUDE code members retrieve the default parameter values by issuing a CALLNAT to the CSUDEFLT sample exit subprogram. Prior to returning the defaults, CSUDEFLT checks to see whether the values have been overridden by the user-defined CSXDEFLT subprogram. If so, the overridden values are returned to the model.

Normally, the model's clear subprogram requests the default values and the returned values are copied to the model parameter data area (PDA). This way, the overhead of retrieving the defaults is only incurred when the user switches to another model or issues a Clear request.

To simplify the interface to CSUDEFLT, Natural Construct supplies three parameterized copycode members. Which copycode member you choose depends on the format of the field for which you are providing defaults. The copycode members are:

Copycode Member	Description
CCDEFLTA	Provides default values for alphanumeric fields.
CCDEFLTL	Provides default values for logical fields.
CCDEFLTN	Provides default values for numeric fields.

Each copycode member accepts two parameters. The format of the second parameter determines which of the copycode members to use:

- The first parameter identifies the default value; this value is passed to CSXDEFLT as the CSADEFLT.PARM-NAME variable. The exact name must appear in the DECIDE statement for CSXDEFLT.
- The second parameter defines the variable to which the default value is assigned (this is typically a variable in the model PDA). The variable is assigned the value returned in CSADEFLT.PARM-VALUE.

Example of retrieving an alphanumeric default value:

```
/*
/* Assign default date edit mask to (alphanumeric) model PDA variable
INCLUDE CCDEFLTA '''DATE-EDIT-MASK''' 'CUMNPDA.#PDA-DATE-EDIT-MASK'
```

For a list of parameters that can be modified by CSXDEFLT, refer to the CSUGETDF program. CSUGETDF also indicates which parameters are currently being overridden by CSXDEFLT. The CSXDEFLT source code contains a description of the parameters.

Example of increasing the size of the left or right prompt on panels:

You can use the CSXDEFLT sample exit subprogram to increase the size of the #RIGHT-PROMPT or #LEFT-PROMPT variable in generated browse, maintenance, or batch programs. For example:

```
VALUE 'RIGHT-PROMPT-LENGTH'
  CSADEFLT.PARM-VALUE := '9'
```

If you increase the prompt length to more than 9 characters, you must also change the size of two variables in the CSUMORE generation utility subprogram in the SYSCST library. Typically, the #PROMPT value should be two characters bigger than the biggest prompt size and the #LITERAL value should be the same size as #PROMPT. For more information, see ***CSUMORE Subprogram***.

> **Note:** If you change the prompt length in CSXDEFLT, you must also change the #RIGHT-PROMPT and/or #LEFT-PROMPT variable on existing maps and then regenerate the modules.

This section covers the following topics:

- Modify the CSXDEFLT Subprogram
- Enable NATdoc Generation
- Modify the DEFAULT Keyword
- Use *ISN as a Unique Primary Key for Maintenance

Modify the CSXDEFLT Subprogram

To modify CSXDEFLT:

1 Logon to the SYSCSTX library.

During installation, the CSXDEFLT subprogram is installed in the SYSCSTX library.

2 Edit and save the CSXDEFLT subprogram.

3 Use the Natural SYSMAIN utility to copy CSXDEFLT to the SYSCST library.

4 Catalog CSXDEFLT in the SYSCST library.

5 Use SYSMAIN to copy the CSXDEFLT object code to the SYSLIBS library.

Enable NATdoc Generation

For subprograms used in Eclipse, you can enable NATdoc generation. This allows the comments in the generated PDAs to be used for Eclipse help.

To enable NATdoc generation:

1 Modify the CSXDEFLT subprogram.

For information, see ***Modify the CSXDEFLT Subprogram***.

2 Remove the comment indicators from the following code:

```
* VALUE 'USE-NATDOCS'
 *   CSADEFLT.PARM-VALUE := TRUE
```

Modify the DEFAULT Keyword

To modify the DEFAULT keyword:

1. Modify the CSXDEFLT subprogram.

   For information, see *Modify the CSXDEFLT Subprogram*.
2. Change the value of the DEFAULT-SPECIFICATION-KEYWORD parameter.

Use *ISN as a Unique Primary Key for Maintenance

For information, see *Use *ISN as the Unique Primary Key for Maintenance*.

Assign Your Own Defaults

You can define default values at the corporate level. For example, you can use the export data function to default information such as the export work file number and the delimiter character. To implement the defaulting mechanism, refer to the following code example. The example illustrates how a work file number and column delimiter values are defaulted.

Example of assigning corporate defaults in the clear subprogram:

```
** We want to default two internal variables: #WORKFILE-NR and
** #COLUMN-DELIMITER
   DEFINE DATA
     LOCAL USING CSADEFLT              /* Must include user default
                                       /* interface LDA
     LOCAL
     01 #WORKFILE-NR(N2) INIT<5>       /* Assign fallback default "5"
     01 #COLUMN-DELIMITER(A1) INIT<','>/* Assign fallback default ","
     01 #PERFORMANCE(L) INIT<FALSE>    /* Assign fallback default
                                       /* "FALSE"
   END-DEFINE
** Assign corporate default overrides if available
   INCLUDE CCDEFLTN '''WORKFILE-NUMBER-PC-DOWN''' #WORKFILE-NR
   INCLUDE CCDEFLTA '''WORKFILE-DELIMITER-CHAR''' #COLUMN-DELIMITER
   INCLUDE CCDEFLTL '''PERFORMANCE''' #PERFORMANCE
** Note that there are 3 separate INCLUDE members: one for numeric
** defaults (CCDEFLTN), one for alphanumeric defaults (CCDEFLTA), and
** one for logical defaults (CCDEFLTL)
** Continue normal processing and the initial values may have been
** overridden by a corporate-supplied defaulting routine.
```

Notes:

1. To apply the changes corporation-wide, you must add the initial variable name and its initial value in the CSXDEFLT sample exit subprogram.

2. The internal defaulting mechanism may be affected when you use this defaulting mechanism to initialize the specification panel default keyword. Use the same keyword for both mechanisms. The specification panel default keyword overrides the internal default keyword.

After adding your own parameters, modify CSUDEFLT (so the CSUGETDF subprogram can add the new parameters to the #PARM-LIST) and then set the #MAX-DEFAULTS setting (for example, if you add one parameter, add one to the #MAX-DEFAULTS value).

You can also override changes the programmer has made and insist on certain values by including statements that assign values to the model PDA in the post-generation subprogram for the model, instead of the clear subprogram. Alternatively, you can hard code a search and replace option. For example, you can create your own copy of CCSETKEY and call it MYSETKEY. To do this, add the line STACK TOP DATA FORMATTED 'CCSETKEY' 'MYSETKEY' in the post-generation subprogram. All instances of CCSETKEY in the code will be replaced by MYSETKEY.

Use Predict Keywords

You can use Predict keywords to define default values for some model input parameters (for example, primary key fields, logical hold fields, and object descriptions). If default values have been specified in Predict, Natural Construct fills in the default values when the model is accessed. This reduces the number of specifications developers must provide when using the model.

This section covers the following topics:

- Define a Default Primary Key
- Define a Default Logical Hold Field
- Define a Default Object Description

Define a Default Primary Key

You can define a default value for a primary key by specifying a descriptor name in the Sequence field for the file in Predict. Natural Construct observes the following priorities when defaulting a primary key name for a file:

1. If the value of the default Sequence field for the file is unique and a valid descriptor, Natural Construct uses this value as the primary key.
2. If the value of the default Sequence field is not unique, Natural Construct reads through the file and uses a unique descriptor field value as the primary key.
3. If the file does not have a unique descriptor field, but has only one descriptor field, Natural Construct assumes the field value is unique and uses it as the primary key.

Define a Default Logical Hold Field

You can define a default value for the logical hold field by attaching a keyword called "HOLD-FIELD" to the field in Predict.

Note: You may have to first define the HOLD-FIELD keyword in Predict using Keyword Maintenance.

Natural Construct observes the following priorities when defaulting a hold field name for a file:

1. If the HOLD-FIELD keyword is attached to a field that meets the format criteria for a hold field, Natural Construct uses this field as the logical hold field.
2. If a field name contains any of the following strings, it is used as the logical hold field:
   - HOLDFIELD
   - HOLD-FIELD
   - HOLD_FIELD
   - TIMESTAMP
   - TIME-STAMP
   - TIME_STAMP
   - LOGCOUNTER
   - LOG-COUNTER
   - LOG_COUNTER
3. If a field meets the format criteria for a hold field, Natural Construct uses this field as the logical hold field.

Define a Default Object Description

You can define a default value for the object description by specifying the default value in the Literal Name field for the file in Predict. Natural Construct uses this value as the object description when the file is referenced in messages. If the value is "Customer", for example, messages are displayed as "Customer not found" or "Customer displayed".

3 Using the Code Frame Editor

A code frame is the basic building block of a model. It provides a rudimentary outline of the code generated by the model. Code frames may contain condition codes to generate blocks of code conditionally. They may also contain subprograms used to generate more complex blocks of code.

Access the Code Frame Editor

There are three methods you can use to access the Code Frame editor. These methods are:

- From the Administration Main Menu
- From the Command Line
- From the Maintain Models Panel

From the Administration Main Menu

To access the Code Frame editor from the Administration main menu:

1 Type "F" in `Function`.

2 Press Enter.

The Code Frame menu is displayed. For example:

```
CSMMAIN              N a t u r a l   C o n s t r u c t               CSMMNMO
Jul 05                        Code Frame Menu                          1 of 1
                                                                            ↩
                     Functions                                              ↩
                     ----------------------------------------------         ↩
                     E  Edit Code Frame                                     ↩
                     S  Save Code Frame                                     ↩
                     L  List Code Frames                                    ↩
                     P  Purge Code Frame                                    ↩
                     C  Clear Edit Buffer                                   ↩
                     H  Print Saved Code Frame                              ↩
                                                                            ↩
                                                                            ↩
                     ?  Help                                                ↩
                     .  Return                                              ↩
                     ----------------------------------------------         ↩
Function .......... _                                                       ↩
Code Frame ........ ________                                                ↩
Description ....... ____________________________________________            ↩
                                                                            ↩
Command ........... ________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                                                  main ↩
 ↩
```

For information about the functions available through this menu, see ***Code Frame Menu Function***.

3 Type "E" in `Function`.

Tip: To edit an existing code frame, type the name of the code frame in `Code Frame` before accessing the Code Frame editor.

4 Press Enter.

The Code Frame editor is displayed. For example:

```
 Code Frame .........                                                 SIZE    ↩
 
 Description ........                                                FREE 61361
 >                                              > + ABS X X-Y _ S      L      ↩
 
   ....+....1....+....2....+....3....+....4....+....5....+....6....+....7.. T C
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
                                                                              ↩
   ....+....1....+....2....+....3....+....4....+....5....+....6....+....7.. T ↩
↩
```

For information about modifying the supplied code frames, see ***Edit Code Frame***.

From the Command Line

You can also access the Code Frame editor from the Natural Next prompt (Direct command box for Unix).

To access the Code Frame editor from the command line:

1 Logon to the SYSCST library.

2 Enter the following command:

```
MENU F E/framename/framedescription
```

From the Maintain Models Panel

You can also access the Code Frame editor from the Maintain Models panel.

To access the Code Frame editor from the Maintain Models panel:

1 Access the Administration main menu.

For information, see *Access the Administration Main Menu*.

2 Enter "M" in `Function`.

The Maintain Models panel is displayed.

Note: For a description of this panel, see *Maintain Models Function*.

3 Move the cursor over the code frame you want to edit.

4 Press PF4 (frame).

The specified code frame is displayed in the Code Frame editor.

Note: For information about editing code frames, see *Edit Code Frame*.

Features of the Code Frame Editor

The following example shows the CSLC9 code frame in the Code Frame editor:

```
Code Frame ......... CSLC9                                              SIZE 29281
Description ........ Browse-Select* model subroutines                   FREE 29520
>                                              > + ABS X X-Y _ S 408  L 1
Top...+....1....+....2....+....3....+....4....+....5....+....6....+....7.. T C
  *
  * Subroutines (in alphabetical order).
  *
  CHECK-WILD-CHARACTER                                                        1
  ***********************************************************************    "
  DEFINE SUBROUTINE CHECK-WILD-CHARACTER                                      "
  ***********************************************************************    "
  *                                                                           "
  * Check for wild characters in the input key and                            "
  * reset minimum and maximum values for the key accordingly                  "
  RESET #WILD-CHAR #LAST-POS                                                  "
  FOR #WINDX = 1 TO 3                                                         "
    EXAMINE #INPUT.#CHAR-ARRAY(*) FOR                                         "
           CDWILDA.#WILD-CARD-CHARS(#WINDX) GIVING INDEX #FIRS-POS(#WINDX)    "
  END-FOR                                                                     "
  /* Find the first wild character                                            "
  FOR #WINDX = 1 TO 2                                                         "
     IF #FIRS-POS(#WINDX) = 1 THRU #FIRS-POS(#WINDX + 1) OR                   "
  ....+....1....+....2....+....3....+....4....+....5....+....6....+....7.. T  ↩
```

The Code Frame editor supports all generic Natural edit commands except the RUN, CHECK, TEST, STOW, and SAVE commands. This editor has no line numbers, but it does have two extra fields to the right of the edit area: T (Type) and C (Condition). Natural Construct uses these fields to control the generation process for each code frame.

The fields in the Code Frame editor are

Field	Description
Code Frame	Name of the code frame currently in the editor (the name specified in `Code Frame` on the Code Frame menu).
Description	Brief description of the code frame.
SIZE	Size of the code frame (in bytes).
FREE	Number of bytes currently available in the editor.
>	Command line prompt, at which you can: ■ Enter "Q", "QUIT", or "." to close the editor. ■ Issue an edit command (for a list of the edit commands, see ***Edit Commands***.

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>+</td><td>Direction indicator. The plus sign (+) indicates that the ADD, MOVE, COPY, INSERT, and SCAN commands operate in a forward (from top to bottom) direction. To have the commands operate in a backward direction (from bottom to top), type a minus sign (-) over the plus sign.

Edit commands use the direction indicator to determine whether to place lines before the first line in the editor or after the last line. For example, using the ADD edit command and a + indicator adds lines after the last line in the editor; using the ADD edit command and a - direction indicator adds lines before the first line in the editor.</td></tr>
<tr><td>ABS</td><td>Absolute field, which is used in conjunction with the SCAN and CHANGE edit commands. When this field is marked, the system scans for or changes the specified characters, including those within words. If you specify a blank in this field, the system scans for or changes the specified characters only if they are a separate entity (delimited by blanks or special characters).</td></tr>
<tr><td>X-Y</td><td>X and Y delimiters for a block of code. To confine SCAN and CHANGE commands to code within an X-Y delimited range, mark this field. Code outside the X-Y range is not affected.</td></tr>
<tr><td>S</td><td>Total number of lines of code currently in the editor.</td></tr>
<tr><td>L</td><td>Number of the first line currently displayed in the editor.</td></tr>
<tr><td>T</td><td>Editor line type. Valid line types are:
N

Indicates that this is a subprogram line and the specified Natural subprogram is invoked during generation. If you specify "N", the line is automatically formatted as follows:<pre><code>Subprogram: ________   Parameter: _______   N</code></pre>Type the name of the subprogram in <code>Subprogram</code>. If the subprogram is invoked more than once or in multiple code frames, you can specify a constant in <code>Parameter</code> (the constant is placed in the #PDA-FRAME-PARM field in the CU—PDA parameter data area). The subprogram can test this field to determine where the subprogram is invoked.
F

Indicates that this is a secondary (nested) code frame line and the specified code frame is invoked during generation. The names of nested code frames should all end with a question mark (?). This naming convention greatly reduces the time and effort required to modify code frames.
U

Indicates insertion points where developers can insert user exit code. (You can specify additional attributes using the .E command after the line is specified.)
*

Indicates code frame comments, which are not used by the generated module.
B

Indicates that blank lines are valid and will be generated into the source area. This line type is used to explicitly hold blank line positions. Natural Construct will not change the contents</td></tr>
</table>

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>of any B type line. If text is entered on a B type line, the text is generated; if a B type line is blank, a blank line is generated.

Note: Natural code does not require blank lines, whereas many scripting languages use the blank line concept extensively.

■ X

Indicates that the text portion of the line must contain the name of a user exit, and the code in the C field must be a number from 1 to 9. If the user exit exists in the User Exit editor when the program is generated, this line indicates that the condition is True.

■ blank

Indicates that this line is constant text and is inserted directly in the generated program, based on the value in <code>C</code>. Whenever a code frame is updated, Natural Construct compresses blank lines and lines marked with B.</td></tr>
<tr><td>C</td><td>Condition level of the corresponding lines. Valid levels are:

■ n (1–9)

Indicates a new condition for this level. The conditions are Boolean combinations of the condition constants specified for the generator. If the condition specified on the line is True, all subsequent code with quotation marks (") is included in the generated program. You can nest conditions by specifying a number greater than 1. (For information about setting up conditions for your generators, see Use Code Frame Conditions.)

■ "

Indicates that text on this line is a continuation of the previous block of code and subject to the last condition specified.

■ blank

Indicates that the corresponding line is constant text and is included unconditionally.</td></tr>
</table>

This section covers the following topics:

- Use Commands in the Code Frame Editor
- Change the PF-Key Profile for the Current Session
- Save the Contents of the Edit Buffer

- Create GUI Sample Subprograms

Use Commands in the Code Frame Editor

This section describes how to use commands in the Code Frame editor. The following topics are covered:

- Order of Command Execution
- Line Commands
- Edit Commands
- Positional Edit Commands

Order of Command Execution

The Code Frame editor executes commands in the following order:

1. Processes text modifications.
2. Executes line commands.

   These commands are specified in the text area of the editor and are preceded with a period (.E, for example).
3. Executes edit commands.

   These commands are specified at the > prompt (ADD, for example).

Line Commands

Within the Code Frame editor, you can issue line commands to copy, move, and delete lines of code. Line commands must be entered in the first column position of a line in the edit area (not at the > prompt) and must begin with a period (.).

Note: Except for the .L command, you should only issue line commands on modified code after you press Enter.

If the direction indicator is + (indicating from top to bottom), the copied, moved, or inserted lines are placed below the line on which the command is entered. If the direction indicator is - (indicating from bottom to top), the lines are placed above the line on which the command is entered.

Note: To avoid shifting the T (Type) and C (Condition) fields, the SHIFT, .J, and .S commands are not available in the Code Frame editor.

The line commands applicable in the Code Frame editor are:

Command	Function
.C(*nn*)	Copies the current line *nn* times, where *nn* is the number of times. The default is one time.
.CX(*nn*)	Copies the line marked X *nn* times, where *nn* is the number of times. The default is one time.
.CY(*nn*)	Copies the line marked Y *nn* times, where *nn* is the number of times. The default is one time.
.CX-Y(*nn*)	Copies the block delimited by X and Y *nn* times, where *nn* is the number of times. The default is one time.
.D(*nn*)	Deletes *nn* lines, where *nn* is the number of lines. The default is one line.
.E	Specifies additional attributes for user exits. If the corresponding line is type U (user exit point), you can specify additional attributes for the user exit by issuing the .E command.
.G(*model*, *parameters*)	Invokes the Natural Construct Generation subsystem.
.I(*nn*)	Inserts *nn* lines, where *nn* is the number of lines. The default is 9 lines; the maximum is 9 lines. The Code Frame editor suppresses unused lines unless they are marked with a B line type.
.IF (*code frame name*)	Inserts the specified code frame on the line below the line on which the command is specified. **Note:** The direction indicator has no effect on this command.
.I(*member*,*startline*,*number of lines*)	Places a member from the current library onto a specified line in the editor. You can also specify a starting line and the total number of lines to include.
.L	Restores the line on which the command is specified to its previous state. (This command is similar to the LET edit command, except it applies to one line only.)
.MX	If the direction indicator is +, this command moves the line marked X to the line below the one on which .MX is specified. If the indicator is -, this command moves the line marked X to the line above.
.MY	If the direction indicator is +, this command moves the line marked with Y to the line below the one on which .MY is specified. If the direction indicator is -, this command moves the line marked Y to the line above.
.MX-Y	Moves the block of lines delimited by the X and Y markers. If the direction indicator is +, this command moves the block to the line below the one on which .MX-Y is specified. If the direction indicator is -, this command moves the block to the line above.
.N	Marks the line for the POINT edit command (for information on the POINT command, see *Positional Edit Commands*).
.P	Moves the line on which the command is specified to the top of the panel.
.W(*nn*)	Inserts *nn* blank lines in the editor, where *nn* is the number of lines. The default is 9 lines. Whenever the code frame is updated, Natural Construct suppresses any unused lines unless they are marked as B line types.

Command	Function
.X	Marks a line, or marks the beginning of a block of lines, that ends with a line marked Y.
.Y	Marks a line, or marks the end of a block of lines, that begins with a line marked X.

Edit Commands

Edit commands are specified at the command prompt (>). These commands are:

Command	Function
ADD	Adds 9 blank lines to the editor.
CHANGE	Scans for text and replaces it with the specified value. The syntax is: `CHANGE 'scanvalue'replacevalue'` You can use any special character as a delimiter, as long as you do not use the same character within the command. **Note:** Unless X and Y line commands limit the range, this edit command performs changes to the entire edit buffer.
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 markers, inclusively.
END	Ends the edit session and invokes the previous menu.
EX	Deletes all lines before the X marker.
EY	Deletes all lines after the Y marker.
EX-Y	Deletes all the lines before the X marker and after the Y marker.
HELP	Displays help text for the Code Frame editor.
LET	Restores lines to their previous state, should you inadvertently change them. Specify the command before pressing Enter. (This command is similar to the .L line command, but applies to the entire buffer.)
LIST	Lists the current contents of the Main buffer.
PROFILE	Invokes a window in which you can modify PF-key settings and edit specifications for the current edit session (see *Change the PF-Key Profile for the Current Session*).
QUIT or .	Ends the edit session and invokes the previous menu.
READ *`program`*	Reads the Natural source for *`program`* into the edit buffer.
RESET	Clears the X and Y markers.
SCAN	Scans for data in the edit area in the following ways: `SCAN 'scanvalue`

Command	Function
	Scans for text within the delimiters. `SCAN scan value` Scans for the entire text after the SCAN keyword, including spaces. **Note:** You must use delimiters for scan values that begin with a non-alphanumeric character. If the direction indicator is "+", the scan begins at the first line displayed on the panel and continues to the end of the text. If the indicator is "-", the scan begins at the last line and continues to the beginning. When the scan value is found, "S" is displayed in the left column next to the target line(s). **Note:** You can also limit the scan range by marking the X-Y field at the top of the Code Frame editor. For a description of this field, see ***Features of the Code Frame Editor***.
UPPER	Invokes a window in which you can specify one or more of the following translation options: ■ Comments Translates all lower case text in comments (text preceded by *, **, or /*). ■ Statements Translates all lower case text in statements, including variables. ■ Quoted strings Translates all lower case text in quoted strings. ■ Programming Translates text for the programming language specified.
*	Redisplays the last command issued.

Positional Edit Commands

If the code frame in the edit buffer is too large to be displayed in its entirety on the panel, you can issue edit commands at the command prompt (>) to scroll through the code:

Command	Function
+*nnnn* or -*nnnn*	Scrolls forward (+) or backward (-) *nnnn* lines.
+H or -H	Scrolls forward (+) or backward (-) half a panel.
+P or -P	Scrolls forward (+) or backward (-) one panel. **Note:** If the code was not changed, you can press Enter to scroll forward one panel.
BOTTOM or ++	Scrolls forward to end of code frame.

Command	Function
POINT	Scrolls line on which the .N line command is specified to top of panel.
TOP or –	Scrolls backward to top of panel.
X or Y	Scrolls to the line marked X or Y.
nnnn	Scrolls to the *nnnn* line.

Change the PF-Key Profile for the Current Session

You can change the PF- and PA-key settings, the number of updates before an automatic save, and the name of the recovery member. Any changes to the current profile take effect immediately and remain in effect for the duration of the current edit session. These changes do not affect the Natural edit profile.

To change the PF-key profile for the current session:

1 Enter "PROFILE" at the > prompt in the Code Frame editor.

The Maintain Current PF-Key Profile window is displayed. For example:

```
  CS-PROF                   Natural Construct                   CS-PRFMO
  Jun 20             Maintain Current PF-Key Profile              1 of 1

   PF1 =  -______________  PF2 =  T______________  PF3 =  B______________
   PF4 =  -H_____________  PF5 =  +H_____________  PF6 =  +P_____________
   PF7 =  N______________  PF8 =  _______________  PF9 =  Q______________
   PF10=  _______________  PF11=  _______________  PF12=  _______________
   PF13=  _______________  PF14=  _______________  PF15=  _______________
   PF16=  _______________  PF17=  _______________  PF18=  _______________
   PF19=  _______________  PF20=  _______________  PF21=  _______________
   PF22=  _______________  PF23=  _______________  PF24=  _______________
   PA1 =  _______________  PA2 =  SCAN___________  PA3 =  _______________

  Auto save numbers ............      In member .......... EDITWORK
 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11-
       help  retrn
 Changes DO NOT affect your edit profile outside Construct
```

This window displays the various settings in effect for the current edit session. The PF-key settings for the Natural Construct editors are determined in the same manner as those for the Natural editor. If you have a profile that corresponds to your user ID, Natural Construct will use those defaults.

2 Change the settings as desired.

The fields in this window are:

Field	Description
PF-*nn* or PA-*n*	Functions assigned to the PF- and PA- keys. You can add new functions by typing a command next to the desired key, or modify existing functions by typing a new command over the one displayed.
Auto save numbers	Number of updates allowed before the source is automatically saved. If this field is blank or 0 (zero), Natural Construct does not automatically save work.
In member	Name of the program that is overwritten each time the specified number of updates is exceeded (by default, EDITWORK). To change the name of the program, type a new name over the one displayed. If this field is blank, Natural Construct does not automatically save work.

Save the Contents of the Edit Buffer

The Natural Construct editors can automatically save work in the edit buffer after a certain number of updates. The number specified in `Auto save numbers` in the Maintain Current PF-Key Profile window determines how often the work is saved. If this field is blank, Natural Construct does not automatically save work. You can also use `In member` in the Maintain Current PF-Key Profile window to specify the name of the recovery member where you want your work saved.

To recover edits, the value in `Auto save numbers` must not be blank or 0 (zero) and the value in `In member` must be specified. For information, see ***Change the PF-Key Profile for the Current Session***.

Tip: Save your work using a unique recovery member name, such as your user ID. This way, your work will not be overwritten by another user using the same recovery member name in the same library.

To retrieve lost code:

1. Access the Code Frame editor.

   For information, see ***Access the Code Frame Editor***.
2. Read EDITWORK into the edit buffer (or whatever name you specified as your recovery member name in the Maintain Current PF-Key Profile window).
3. Re-specify the description, as it is not saved in the recovery member.

Create GUI Sample Subprograms

Sample subprograms are invoked from a user exit. These subprograms help the developer create user exit code by providing a starting sample. The GUI sample subprogram is a client version of the mainframe sample subprogram — minus the input statements. When Natural Construct generates a model on the client, it bypasses the mainframe sample subprogram and reads the GUI sample subprogram instead.

4 Creating New Models

This section describes the procedure to create a new Natural Construct model and contains information about testing the components of a model and debugging a model. In addition, it describes special considerations for building statement models and presents a summary of tips and precautions. This section also provides information about the utility subprograms and helproutines supplied with Natural Construct. These utilities can help you create your new model.

Components of a Natural Construct Model

A Natural Construct model is the combination of several components which, when used together, generate a Natural module. Natural Construct provides models you can use to help generate many of these components. The following table lists the components of a Natural Construct model, as well as the name of the model you can use to generate each component (if applicable):

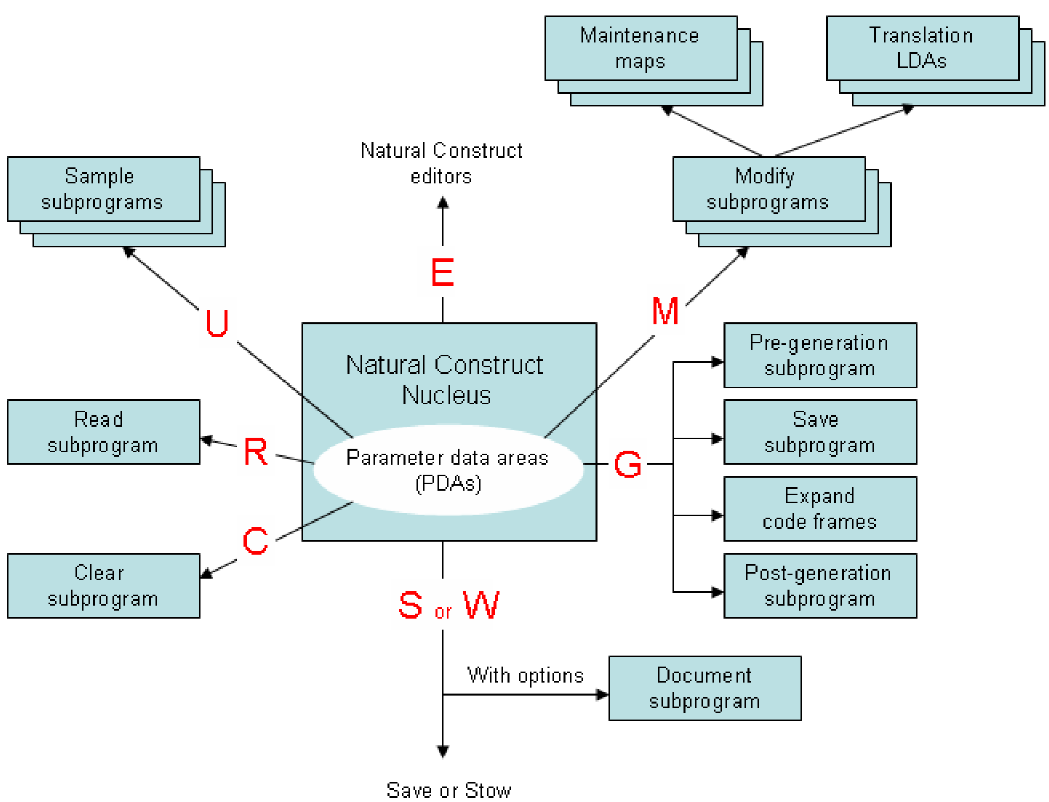
Component	Model Used to Generate
Code frames	None (either create manually or copy and modify existing).
Model PDA	CST-PDA model (described in ***CST-PDA Model***).
Translation LDAs for dynamic translation	None (either create manually or copy and modify existing).
Maintenance maps	Map model (described in *Natural Construct Generation*).
Maintenance subprogram(s)	CST-Modify or CST-Modify-332 model (described in ***CST-Modify and CST-Modify-332 Models***).
Pre-generation subprogram	CST-Pregen model (described in ***CST-Pregen Model***).
Generation subprograms	CST-Frame model (described in ***CST-Frame Model***).
Post-generation subprogram	CST-Postgen model (described in ***CST-Postgen Model***).
Clear subprogram	CST-Clear model (described in ***CST-Clear Model***).
Save subprogram	CST-Save model (described in ***CST-Save Model***).
Read subprogram	CST-Read model (described in ***CST-Read Model***).
Sample subprogram(s)	CST-Frame model (described in ***CST-Frame Model***).
Documentation subprogram	CST-Document model (described in ***CST-Document Model***).
Stream subprogram	CST-Stream model (described in ***CST-Stream Model***).
Validation subprogram	CST-Validate model (described in ***CST-Validate Model***).

How the Natural Construct Nucleus Executes a Model

The Natural Construct nucleus is a sophisticated driver program that assembles the model components and sets them in motion. Although it invokes the subprograms at the appropriate time in the generation process and performs the functions common to all models, it is not aware of the code generated by the models.

The nucleus communicates with the model subprograms through standard parameter data areas (PDAs). These PDAs contain fields assigned by Natural Construct, as well as fields that are redefined as required by a model.

The generation process uses each model component at a different time. The following diagram illustrates the components of a model and how they interact with each other and the nucleus. The large letters in red correspond to the function codes a user enters on the Generation main menu to invoke the corresponding subprogram(s):



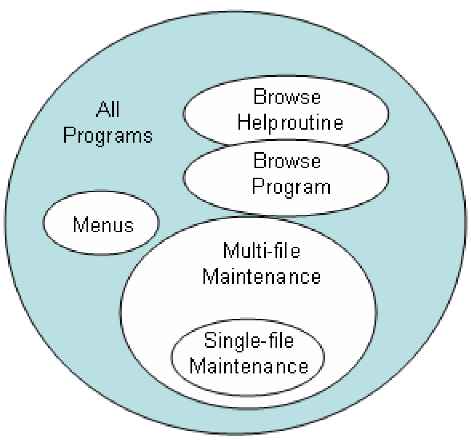
Build a New Model

This section describes how to build a new Natural Construct model. These steps are:

- Step 1: Define the Scope of the Model
- Step 2: Create the Prototype
- Step 3: Scrutinize the Prototype
- Step 4: Isolate the Parameters in the Prototype
- Step 5: Create Code Frame(s) and Define the Model
- Step 6: Create the Model PDA
- Step 7: Create the Translation LDAs and Maintenance Maps
- Step 8: Create the Model Subprograms

Step 1: Define the Scope of the Model

Before you can build the new model, you must decide what type of module the model will generate. The following diagram illustrates the varying scope and overlapping functionality of different module types:



Is the Scope Too Broad?

If your model contains many parameters (one that generates complex modules with broad functionality), it may:

- Confuse and frustrate developers
- Lengthen the time it takes developers to specify parameters
- Require complex code frames with many conditions
- Make the model so flexible that generated code may deviate from standards

For example, the model should not allow developers to define PF-keys used for standard features (these should be standardized across all applications). On the other hand, these models can be very powerful and flexible — once the developer is familiar with them.

Is the Scope Too Narrow?

If your model contains few parameters (one that generates simple modules with narrow functionality), it may:

- Make the model inflexible
- Limit the model's usefulness

On the other hand, these models are simple to use and easy to maintain.

What to Generate and Why

Typically, models generate Natural source code — but the possibilities are endless. Natural Construct was designed to generate text in any form: Unix scripts, JCL, COBOL, Visual Basic, C++, HTML scripts, etc.

As a general rule, you will want your models to generate common modules that cannot be parameterized at execution time. This type of module often involves file accesses or compile-time statements, such as:

- map names
- parameter lists
- FORMAT statements
- I/O statements
- file definitions

Alternately, you may want the model to generate modules that can be parameterized at execution time but are hardcoded for performance reasons (menus, for example).

Step 2: Create the Prototype

Once you determine the purpose and scope of the model, you can create a Natural module (program, subprogram, map, etc.) to base your model on. This module should perform all the functions you defined for the scope of the model.

If the scope contains mutually-exclusive options, you should prepare several prototypes. For example, if the Natural code to maintain a file with a superdescriptor is significantly different from the code that maintains a file with a descriptor, create two prototypes. If possible, generate the more complex prototype first and add the simpler prototype later.

Step 3: Scrutinize the Prototype

After creating your prototype Natural program, perform the following checks:

- Ensure that the program is fully commented
- Check the code indentation
- Check the clarity of the program
- Ensure that the program conforms to standards
- Evaluate the efficiency of the program
- Ensure that variable names are sorted

After you have scrutinized the prototype as thoroughly as possible, have someone else perform the same checks and tests.

Step 4: Isolate the Parameters in the Prototype

The basic premise behind program generation is to take a working module that performs a fixed function and generalize the module so it performs varying functions based on parameter values. To isolate the parameters:

- Determine Which Elements Need to be Parameterized
- Remove Redundant Parameters
- Choose Between Compile Time and Runtime

Determine Which Elements Need to be Parameterized

The first step is to determine which program lines remain constant in the generalized module and which lines vary. If the prototype reads a file and displays information, for example, the file and information varies with each generation. Therefore, this information must be parameterized. To make the prototype easier to generate, try to reduce the number of parameters in your prototype without affecting the functionality.

Remove Redundant Parameters

Programs often contain several instances of the same parameter. These can be reduced to a single instance of the parameter by using a constant variable. Consider the following examples:

<table>
<tr><th>Redundant Parameters</th><th>Single Parameter</th></tr>
<tr><td><pre>DEFINE DATA LOCAL
01 #A(A1/1:50
.
.
END-DEFINE
.
.
IF #A(#CUR:50) NE ' ' THEN
FOR #I = #CUR TO 50
etc.</pre></td><td><pre>DEFINE DATA LOCAL
01 #ASIZE(P3) CONST<50>
01 #A(A1/1:#ASIZE)
.
END-DEFINE
.
.
IF #A(#CUR:#ASIZE) NE ' ' THEN
FOR #I = #CUR TO #ASIZE
etc.</pre></td></tr>
</table>

This technique makes the prototype easier to generate, since there are fewer parameter instances. In addition, the generated programs are easier to read, since it is more obvious that the constant value always refers to the same thing.

Choose Between Compile Time and Runtime

Ensure that your prototype does not contain hardcoded parameters that could easily be calculated at runtime. Consider the following examples:

<table>
<tr><th>Unnecessary Constant</th><th>Determine at Runtime</th></tr>
<tr><td><pre>DEFINE DATA LOCAL
   01 #MAX-LINES(P3) CONST <15>
   01 #LINE-NR(P3/1:#MAX-LINES)
INIT<1,2,3,4,5,6,7,8,9,10,11,12,13,
               15>
END-DEFINE</pre></td><td><pre>DEFINE DATA LOCAL
   01 #MAX-LINES(P3) CONST <15>
   01 #LINE-NR(P3/1:#MAX-LINES)
   01 #I (P3)
END-DEFINE
FOR #I = 1 TO #MAX-LINES
   ASSIGN #LINE-NR (#I) = #I
END-FOR</pre></td></tr>
</table>

Both the INIT statement on the left and the FOR loop on the right initialize an array with consecutive numbers. However, the code on the right does not vary based on the value of #MAX-LINES. No special processing is required to generate the code on the right, as it is constant for each generation. To make the prototype more flexible and easier to generate, use Natural system variables to determine the values at runtime.

Note: Ensure you do not sacrifice program efficiency to achieve this goal.

Once you have written and tested your prototype, save it in the SYSCST library.

Step 5: Create Code Frame(s) and Define the Model

This section covers the following topics:

- Create the Code Frames
- Define the Model

Create the Code Frames

If the prototype program is large, you can create multiple code frames with a portion of the program in each code frame. You can also use nested code frames.

To create the code frames:

1. Invoke the Code Frame editor.
2. Read your prototype into the editor.
3. Determine the parameters required for the code frame.

   These include substitution parameters, code frame conditions, generation subprograms, nested code frames, and user exits. The following example shows a code frame in the Code Frame editor:

```
 Frame .............. PRSLCC9                                       SIZE 1125
 Description ........ Browse Select Code©) Inline Subroutines       FREE 59940
 >                                       > + ABS X X-Y X S 18   L 1   ↩

 All...+....1....+....2....+....3....+....4....+....5....+....6....+....7.. T C
   *                                                                       ↩

   * Subroutines (in alphabetical order).                                  ↩

   * Check wildcard processing                                           *
   CHECK-WILD-CHARACTER                                                    ↩
1
   CUSLCWC?                                                              F "
   * Initializations                                                     *
   CUSLCI?                                                               F
    Subprogram: CUSCGBND Parameter: INITIALIZE                           N
   * Initialize the input key to the minimum key value specified           ↩

     ASSIGN #INPUT.&PRIME-KEY = #MIN-KEY-VALUE                             ↩

   Process Selected Column or Record                                     *
   PROCESS-SELECTION-COLUMN OR PROCESS-SELECTED-RECORD                     ↩
1
   CUSLCPS?                                                              F "
   * Final Processing                                                    *
   CUSLCFP?                                                              F
   MISCELLANEOUS-SUBROUTINES                                             U
   PERFORM FINAL-PROCESSING                                                ↩

   END                                                                     ↩

   ....+....1....+....2....+....3....+....4....+....5....+....6....+....7.. T ↩
↩
```

For a description of the Code Frame editor, see ***Using the Code Frame Editor***. For information about edit commands, see ***Edit Commands***.

The code frame example above demonstrates different methods of supplying parameters for a code frame. These methods are:

- Use Substitution Parameters
- Use Parameters Supplied by Generation Subprograms
- Use Parameters Supplied by Nested Code Frames
- Use Parameters Supplied by User Exits

- Use Code Frame Conditions

Use Substitution Parameters

One type of code frame parameter is substitution parameters. These parameters are always present in the same format, but their values change. You can usually assign substitution parameters by replacing the values with unique substitution strings. To identify a parameter as a substitution, use an ampersand (&) at the beginning of the substitution string in the editor.

The code frame example above contains the following substitution parameter:

```
* Initialize the input key to the minimum key value specified
  ASSIGN #INPUT.&PRIME-KEY = #MIN-KEY-VALUE
```

Values are substituted after the module is fully generated. The unique identifier (&PRIME-KEY in the example above) is substituted for the derived value by placing the unique identifier and the value in the Natural stack.

Note: For more information about substitution during the post-generation phase, see ***Post-Generation Subprogram***.

The following stipulations apply:

- Substitution parameters cannot span multiple lines.
- Substitution parameters always begin with an ampersand (&).
- The substitution string can be up to 32 characters in length.
- The substitution value can be up to 72 characters in length.

The name of the parameter should correspond to the name of the model PDA variable that supplies the value. For example, &VAR is assigned the value of #PDA-VAR or #PDAX-VAR. Following this naming convention makes it easier to generate the model subprograms using the supplied models. For more information about the model PDA, see ***Model PDA***.

Use Parameters Supplied by Generation Subprograms

A generation subprogram can supply the code frame parameters. When a substitution parameter spans more than one line, varies in length, or performs complex calculations (centering, for example), you can supply the parameters in a generation subprogram.

An example of this type of parameter is a file view where the developer specifies the name of the file to use. Instead of supplying a list of the fields in the view, you can specify the name of a subprogram to supply this list.

To indicate that a subprogram is called on this line, enter "N" (Natural subprogram) in the corresponding T (Type) field. To pass a parameter to the subprogram, specify the parameter value after the subprogram name. The parameter can be a literal string, 1–32 characters in length.

Natural Construct passes the following structures to each generation subprogram:

- Model PDA (CU*xx*PDA), containing model-specific parameters
- CSASTD, containing the standard messaging parameters
- CU—PDA, containing the standard generation parameters (the #PDA-FRAME-PARM field in this PDA passes the parameter literal string)

The following code frame line indicates that the CUSCGBND subprogram is invoked from this point in the code frame and passed the INITIALIZE value:

```
Subprogram: CUSCGBND Parameter: INITIALIZE                    N
```

Because code frame parameters are supplied in a generation subprogram, the same subprogram can be invoked several times within the code frame. The subprogram uses the value of the passed parameter to determine what to generate each time.

Use Parameters Supplied by Nested Code Frames

Another method of supplying parameters to a code frame is to use nested code frames. As with generation subprograms, nested code frames can perform substitutions on lines of varying length. In fact, nested code frames have all substitution options available to the calling code frame. For example, a nested code frame can have substitution parameters, generation subprograms, and its own nested code frames.

All code frames supplied with Natural Construct end with 9 (see the description of the Code frame(s) field in ***Maintain Models Function***) and 8 is reserved for any future updates. When you reference a code frame from within another code frame, use a question mark (?) instead of 9. The ? indicates a hierarchy structure in which Natural Construct uses the code frame with the lowest number during generation.

For specific hardcoded references, you can specify a nested code frame without using the question mark (?) — but if you want to change what the nested code frame generates, you must modify every calling code frame and its reference. When you use the question mark (?) character, Natural Construct automatically calls your new version of the nested code frame.

Note: To make nested code frames more reusable across multiple models, it is important to use the same naming conventions. In this way, the nested code frame logical and substitution parameters are always available within the model PDAs.

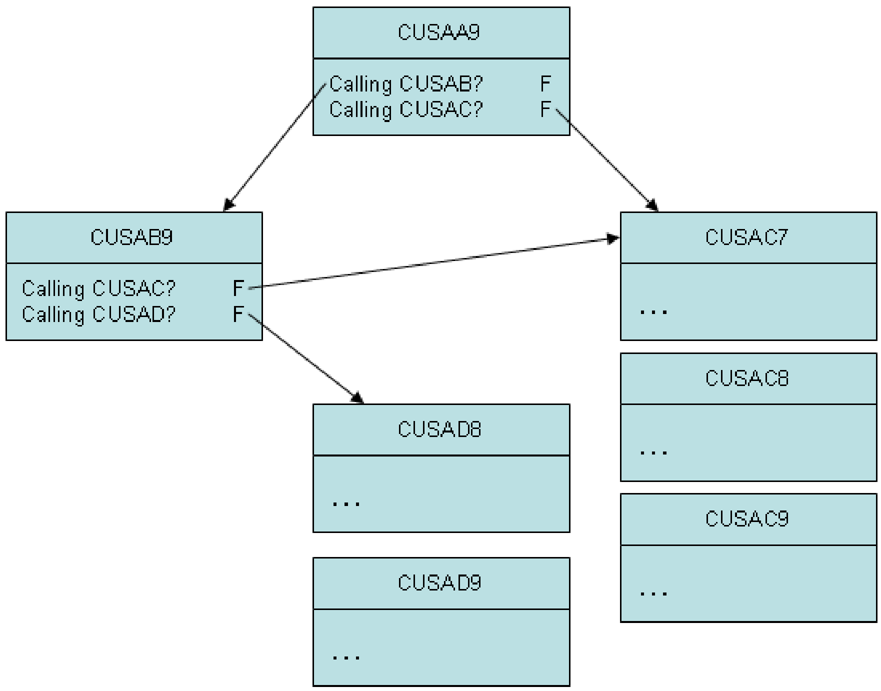
To indicate that another code frame is called on a Code Frame editor line, enter "F" in the corresponding T (Type) field. The following code frame line indicates that the CUSLCI*n* code frame supplies parameters for the code frame, where *n* is a number from 1 to 9:

```
CUSLCI?                                                       F
```

To modify a supplied code frame, copy the code frame, change the 9 to a lesser number from 1 to 7 (8 is used for code frame fixes supplied between releases), and modify the code frame as desired.

The next time Natural Construct calls that code frame, the one you created with the lesser number is used. For example, you can copy the CUSLCI9 code frame, change the name to CUSLCI7, and edit it as desired. The next time Natural Construct calls CUSLCI?, CUSLCI7 is used.

In the following example, the CUSAA9 code frame has two nested code frames (CUSAB? and CUSAC?). The arrows indicate which code frame is used:



Tip: Ensure that you do not create endless loops within nested code frames; endless loops result when a code frame calls itself, either directly or indirectly as a nested code frame.

Use Parameters Supplied by User Exits

Parameters for a code frame can also be supplied by user exits. User exits provide maximum flexibility for defining parameters because parameters are specified in the form of embedded Natural code. User exits allow programmers/analysts to provide specialized portions of code at various points within the generated module.

To supply parameters for a code frame through a user exit:

1. Enter the name of the user exit in the text portion of a line.
2. Enter "U" in the corresponding `T` (Type) field.
3. Optionally, you can specify additional attributes by entering ".E" at the beginning of the user exit line.

For example:

```
 Frame .............. CUSLD9                                           SIZE 5973
 Description ........ Browse Select Subp. Define Data Area             FREE 54796
 >                                              > + ABS X X-Y _ S 102  L 1   ↩

 Top...+....1....+....2....+....3....+....4....+....5....+....6....+....7.. T C
   CU--B?                                                                 F
   DEFINE DATA                                                                ↩

   GDA-SPECIFIED                                                              ↩
1
     GLOBAL USING &GDA &WITH-BLOCK                                            ↩
"
     PARAMETER                                                                ↩

     01 #PDA-KEY(&PARM-NAT-FORMAT)  /* Start/Returned key.                    ↩

     VARIABLE-MIN-MAX AND PREFIX-IS-PDA-KEY                                   ↩
1
     01 REDEFINE #PDA-KEY                                                     ↩
"
       02 #PDA-KEY-PREFIX(&PREFIX-NAT-FORMAT)                                 ↩
"
     PARAMETER USING CDSELPDA /* Selection info                               ↩

     PARAMETER USING CU—PDA  /* Global parameters                             ↩

     PARAMETER USING CSASTD   /* Message information                          ↩

   .eRAMETER-DATA                                                         U
     LOCAL USING CDDIALDA /* Used by dialog objects.                          ↩

     LOCAL USING CDENVIRA /* Used to capture/restore previous environment.    ↩

   DIRECT-COMMAND-PROCESSING                                                  ↩
1
     LOCAL USING CDGETDCA /* Used to get direct command info.                 ↩
"
   MULTIPLE-WINDOWS                                                           ↩
1
   ....+....1....+....2....+....3....+....4....+....5....+....6....+....7.. T
CUSLD9 read                                                                   ↩
 ↩
```

4 Press Enter.

The Maintain User Exit window is displayed. For example:

```
CSMUSEX                       Natural Construct
Jul 05                        Maintain User Exit                       1 of 1
 User exit name ......... START-OF-PROGRAM
 Code frame name ........ COBB9    Conditional  N
 User exit required ..... _
 Generate as subroutine . _
 Sample subprogram ...... ________        GUI sample subprogram .. ________
 Default user exit code .
   *_______________________________________________________________________
   * Specify code to be executed at the beginning of the object subprogram.
   * This might include security checking logic.___________________________
  _________________________________________________________________________
  _________________________________________________________________________
  _________________________________________________________________________
  _________________________________________________________________________
  _________________________________________________________________________
  _________________________________________________________________________
  _________________________________________________________________________
 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
       help  retrn
```

Use this window to specify information about the user exit. The fields in this window are:

Field	Description
User exit name	Name of the user exit.
Code frame name	Name of the code frame for the user exit.
Conditional	Condition code for the user exit. If the user exit is conditional (required only under certain conditions), "Y" is displayed. If it is not conditional, "N" is displayed.
User exit required	If this field is marked, the user exit is required; if this field is blank, the user exit is optional.
Generate as subroutine	If the user exit is used in more than one place in the module, enter "Y". The code is generated as an inline subroutine. During generation, Natural Construct places the code in a subroutine with the same name as the user exit. This allows you to execute the code several times using a PERFORM `user-exit-name` statement. If the user exit is optional, the PERFORM statement can be conditional on the presence of the user exit itself (for information, see ***Use Code Frame Conditions***). Regardless of whether user exits are generated as subroutines or embedded code, use the DEFINE EXIT keyword to specify all user exits.
Sample subprogram	If a subprogram contains the sample code for the user exit, enter the name of the subprogram. The sample code is generated after the developer enters the SAMPLE command in the User Exit editor and selects an exit. Natural Construct passes three parameter data areas (PDAs) to each sample subprogram: the model PDA, CU—PDA, and CSASTD. For more information, see ***Step 6: Create the Model PDA***.

Field	Description
	Note: The SAMPLE command is executed automatically when you enter "U" on the Generation main menu or press PF11 (userX) on the last specification panel for a model that supports user exits, but none have been specified.
GUI sample subprogram	GUI sample subprogram invoked when the code is being generated from the client. This subprogram should not display input panels. If the sample subprogram does not use input panels, it can be used in the GUI sample subprogram. If the sample subprogram includes input panels, create a copy and modify it to use the defaults.
Default user exit code	If complex processing or calculations are not required, you can enter up to 10 lines of sample code. This code becomes the default sample code for this user exit. **Note:** If you specify a sample subprogram name and provide default user exit code, Natural Construct generates the user exit code before it generates the sample subprogram code.

Use Code Frame Conditions

Frequently, a block of statements is inserted in a program based on a condition or combination of conditions specified in the code frame. In the following example, the `INPUT WITH TEXT+MSG USING MAP '&MAP-NAME' INPUT` statement is generated if a map is used. Otherwise, the `INPUT(AD=OI)` statement is generated:

```
Top...+....1....+....2....+....3....+....4....+....5....+....6....+....7.. T C
MAP-USED       1
INPUT WITH TEXT + MSG USING MAP '&MAP-NAME'       "
ELSE       1
INPUT(AD=OI) *PROGRAM #HEADER1       "
/ *DATX #HEADER2 *TIMX       "
```

Note: To identify a condition line, enter a number in the C (Condition) column in the Code Frame editor. Number "1" initiates a new condition; higher numbers represent nested conditions that are only evaluated if all active lower conditions are True.

To identify a statement as conditional, enter """" in the C column. The corresponding statement is included in the generated module only if the current condition is True.

When you use code frame conditions, consider the following points:

- The names of conditions must correspond to the names of logical variables defined in the model PDA, with the #PDAC- prefix removed. (For more information about the model PDA, see *Step 6: Create the Model PDA*.) The MAP-USED condition, for example, corresponds to the `#PDAC-MAP-USED` logical variable.

Note: These condition variables must be part of the redefinition of the `#PDA-CONDITION-CODES` field in the model PDA.

- When Natural Construct generates a module, it checks the condition code values to determine whether the condition is True. It then resets the conditions before invoking the maintenance subprograms. Condition codes should be selectively set to True by either the pre-generation subprogram or one of the maintenance subprograms.
- Conditions can be negated, ANDed and ORed (in order of precedence).
- Conditions can be nested and ELSEed (ELSE refers back to the previous condition at the same level number).
- The RETURN-TO-CONDITION keyword can close levels of conditioning.
- A special condition line can check for the existence of a specific user exit. To specify this type of condition, enter the name of the user exit as the condition value and specify a line type of "X". These conditions cannot be negated, ANDed, or ORed, but can be nested. They do not require a corresponding `#PDAC` variable.

The following example shows code frame conditions:

```
Frame ..............ABC                                    SIZE 68
Description ........Example of conditions               FREE 36676
  >                                      > + ABS X X-Y _ S 21 L 1
Top.+...1...+...2...+...3...+...4...+...5...+...6...+...7..   T C Notes
MAP-USED                                                           1
INPUT WITH TEXT + MSG USING MAP '&MAP-NAME'1                     " 1
ELSE                                                               1
INPUT(AD=OI) *PROGRAM #HEADER1                                   " 2
/ *DATX #HEADER2 *TIMX                                           " 2
ROOM-FOR-SKIP                                                      2
/                                                                " 3
RETURN-TO-CONDITION                                                1
/ 20T #FUNCTION-HEADING                                          " 2
 NOT MAP-CONTAINS-PARAMETERS                                       2
 CODE1-SPECIFIED                                                   3
/ 16T #CODE(1) 20T #FUNCTION(1)                                  " 4
 CODE2-SPECIFIED                                                   3
/ 16T #CODE(2) 20T #FUNCTION(2)                                  " 5
     .
     .
     .
 CODE12-SPECIFIED                                                  3
/ 16T #CODE(12) 20T #FUNCTION(12)                                " 6
 RETURN-TO-CONDITION                                               2
/ 11T 'Code:' #CODE(AD=M)                                        " 7
 ELSE                                                              2
Subprogram: CUMNGIN  Parameter                                N  " 8
RETURN-TO-CONDITION                                                1
21/1 'Direct Command:' #COMMAND(AD=M)                            " 2
RESET +MSG                                                         9
AFTER-INPUT
AFTER-INPUT                                                   X    1
PERFORM AFTER-INPUT                                              " 10
```

Higher-level numbers (nested conditions) are always joined with an AND statement to previous lower condition numbers.

Notes

The lines of code corresponding to each note number in the above example are inserted into the generated module when the following Boolean conditions are met:

Note Number	Boolean Condition
1	#PDAC-MAP-USED = TRUE
2	#PDAC-MAP-USED = FALSE
3	#PDAC-MAP-USED = FALSE and #PDAC-ROOM-FOR-SKIP = TRUE

Note Number	Boolean Condition
4	#PDAC-MAP-USED = FALSE and #PDAC-MAP-CONTAINS-PARAMETERS = FALSE and #PDAC-CODE1-SPECIFIED = TRUE
5	#PDAC-MAP-USED = FALSE and #PDAC-MAP-CONTAINS-PARAMETERS = FALSE and #PDAC-CODE2-SPECIFIED = TRUE
6	#PDAC-MAP-USED = FALSE and #PDAC-MAP-CONTAINS-PARAMETERS = FALSE and #PDAC-CODE12-SPECIFIED = TRUE
7	#PDAC-MAP-USED = FALSE and #PDAC-MAP-CONTAINS-PARAMETERS = FALSE
8	#PDAC-MAP-USED = FALSE and #PDAC-MAP-CONTAINS-PARAMETERS = TRUE
9	Line is inserted unconditionally.
10	Line is inserted only when the AFTER-INPUT user exit is specified in the User Exit editor before the module is generated.

Define the Model

Use the Maintain Models panel to define your model.

To display the Maintain Models panel:

1. Log onto the SYSCST library.
2. Enter "MENU" at the Next prompt (Direct Command box for Unix).

   The Administration main menu is displayed.
3. Enter "M" in `Function`.

   The Maintain Models panel is displayed. For example:

```
CSDFM                    N a t u r a l   C o n s t r u c t            CSDFM0
Aug 17                          Maintain Models                       1 of 1

Action .................... __ A,B,C,D,M,N,P,R

Model ..................... ________________________________

Description ........ __________________________________________________________


  PDA name ................. ________     Status window ............ _

  Programming mode ......... __           Comment start indicator .. ___

  Type ..................... _            Comment end indicator .... ___



  Code frame(s) ............ ________  ________  ________  ________  ________
  Modify server specificatn  ________  ________  ________  ________  ________
                            ________  ________  ________  ________  ________
  Modify client specificatn  ________  ________  ________  ________  ________
                            ________  ________  ________  ________  ________


  Clear specification ...... ________    Post-generation .......... ________
  Read specification ....... ________    Save specification ....... ________
  Pre-generation ........... ________    Document specification ... ________
Command ........... ___________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit  frame                                           main
```

Use this panel to specify the names of the model components (the generation subprograms require this model definition); the specified components do not have to currently exist. When naming the model components, use the naming conventions described in the following section.

For a description of the Maintain Models panel, see ***Maintain Models Function***.

Naming Conventions for Model Components

Standardizing the names of the various components of a model makes it easier to write and debug models. Supplied model subprograms, maps, and data areas are typically named CU*xx*, where *xx* uniquely identifies each model and *y* identifies each panel. When naming model components, we recommend the following naming conventions:

Name	Model Component
CU*xx*PDA	Parameter data area.
CU*xx*R	Read subprogram.
CU*xx*C	Clear subprogram.
CU*xx*MA	First maintenance subprogram.
CU*xx*MA*n*	Map associated with the first maintenance subprogram. ■ To display a map based on the current value of the *Language system variable, use a *Language value in the last position of the map name. ■ To support dynamic translation, use a zero (0) in the last position of the map name.
CU*xx*MAL	Translation local data area (LDA) associated with the first maintenance subprogram. A translation LDA contains the names of all variables that are initialized to the maintenance map text and can be translated. You cannot dynamically translate a map to another language unless the module that invokes the map has a corresponding translation LDA.
CU*xx*MB	Second maintenance subprogram.
CU*xx*MB*n*	Map associated with the second maintenance subprogram.
CU*xx*MBL	Translation LDA associated with the second maintenance subprogram.
CU*xx*S*yyy*	Sample user exit code subprograms, where *yyy* is a 1–3 character suffix that uniquely identifies each sample subprogram. For example, the CUFMSRIN sample subprogram supplies REINPUT statements for the Maint model (if required).
CU*xx*G*yyy*	Generation subprograms, where *yyy* is a 1–3 character suffix that uniquely identifies each generation subprogram. For example, the CUMNGGL subprogram generates parameter variables for the Menu model (when a length and format are specified).
CU*xx*PR	Pre-generation subprogram.
CU*xx*PS	Post-generation subprogram.
CU*xx*S	Save subprogram.
CU*xx*D	Documentation subprogram.
WCN*xx*M*y*	Construct Program Generation plug-in maintenance subprogram.
WCD*xx*	Construct Program Generation plug-in dialog.

To modify the supplied Natural Construct models, copy the subprograms and change the prefix from CU (or WC) to CX. This way, you can identify the modified subprograms and include any changes in future versions of Natural Construct.

After defining a model, it can be used in the Generation subsystem.

Step 6: Create the Model PDA

All models require three parameter data areas (PDAs). Two of the data areas are supplied with Natural Construct and the model PDA is user-created for each individual model.

PDAs pass information between the nucleus and the model and code frame subprograms. Every model subprogram uses the following external PDAs:

PDA	Description
Model PDA	User-created and named CU*xx*PDA, where *xx* uniquely identifies the model. This PDA contains variables and conditions specific to the model. It is the only PDA you must create. Use the CST-PDA model to create the model PDA (see ***Parameters for the CST-PDA Model***).
CU—PDA	Supplied with Natural Construct.
CSASTD	Supplied with Natural Construct.

These PDAs must contain the following fields:

PDA	Required Fields and Format
Model PDA (varies for each model)	#PDA-CONDITION-CODES (L/1:75) #PDA-USER-AREA (A100/1:40)
CU--PDA (same for every model)	#PDA-MODE (A2) #PDA-OBJECT-TYPE (A1) #PDA-MODIFY-HEADER1 (A60) #PDA-MODIFY-HEADER2 (A54) #PDA-LEFT-PROMPT (A11) #PDA-LEFT-MORE-PROMPT (A9) #PDA-RIGHT-PROMPT (A11) #PDA-RIGHT-MORE-PROMPT (A9) #PDA-PHASE (A1) #PDA-DIALOG-METHOD (I1) #PDA-TRANSLATION-MODE (L) #PDA-USERX-NAME (A10) #PDA-PF-NAME (A10/1:12) #PDA-MAIN-NAME (A10)

PDA	Required Fields and Format
	#PDA-RETURN-NAME (A10)
	#PDA-QUIT-NAME (A10)
	#PDA-TEST-NAME (A10)
	#PDA-BACKWARD-NAME (A10)
	#PDA-FORWARD-NAME (A10)
	#PDA-LEFT-NAME (A10)
	#PDA-RIGHT-NAME (A10)
	#PDA-HELP-NAME (A10)
	#PDA-AVAILABLE1-NAME (A10)
	#PDA-AVAILABLE2-NAME (A10)
	#PDA-AVAILABLE3-NAME (A10)
	#PDA-PF-NUMBER (N2/1:12)
	#PDA-MAIN (N2)
	#PDA-RETURN (N2)
	#PDA-QUIT (N2)
	#PDA-TEST (N2)
	#PDA-BACKWARD (N2)
	#PDA-FORWARD (N2)
	#PDA-LEFT (N2)
	#PDA-RIGHT (N2)
	#PDA-HELP (N2)
	#PDA-AVAILABLE1 (N2)
	#PDA-AVAILABLE2 (N2)
	#PDA-AVAILABLE3 (N2)
	#PDA-PF-KEY (A4)
	#PDA-PF-MAIN (A4)
	#PDA-PF-RETURN (A4)

PDA	Required Fields and Format
	#PDA-PF-QUIT (A4) #PDA-PF-TEST (A4) #PDA-PF-BACKWARD (A4) #PDA-PF-FORWARD (A4) #PDA-PF-LEFT (A4) #PDA-PF-RIGHT (A4) #PDA-PF-HELP (A4) #PDA-PF-AVAILABLE1 (A4) #PDA-PF-AVAILABLE2 (A4) #PDA-PF-AVAILABLE3 (A4) #PDA-TITLE (A25) #PDA-GEN-PROGRAM (A8) #PDA-MODEL-VERSION (N2.2) #PDA-HELP-INDICATOR (A4) #PDA-USER-DEFINED-AREA (A1/1:100) #PDA-UNDERSCORE-LINE (A80) #PDA-RIGHT-PROMPT-OF (A4) #PDA-DISPLAY-INDICATOR (A4/1:10) #PDA-CURS-FIELD (I4) #PDA-CV1 (C) #PDA-CV2 (C) #PDA-CV3 (C) #PDA-CV4 (C) #PDA-CV5 (C) #PDA-CV6 (C) #PDA-CV7 (C) #PDA-CV8 (C)

PDA	Required Fields and Format
	#PDA-SCROLL-INDICATOR (A4) #PDA-DYNAMIC-ATTR-CHARS (A1/1:13) #PDA-FRAME-PARM (A32) #PDA-SYSTEM (A32)
CSASTD (same for every model)	MSG (A79) MSG-NR (N4) MSG-DATA (A32/1:3) RETURN-CODE (A1) ERROR-FIELD (A32) ERROR-FIELD-INDEX1 (P3) ERROR-FIELD-INDEX2 (P3) ERROR-FIELD-INDEX3 (P3) **Note:** The CSASTD PDA is used by every model. It passes messages between subprograms and is typically used for error handling.

The following sections describe the layout of these PDAs.

Model PDA

The following example shows a model PDA:

```
Parameter CUETPDA   Library SYSCST                               DBID  19 FNR  28
Command                                                                       > +
I T L Name                             F Leng Index/Init/EM/Name/Comment
Top - -------------------------------- - ---- --------------------------------
    1 CUETPDA                                 /* Construct Model PDA
    2 #PDA-CONDITION-CODES             L      (1:75) /* Conditions in frames
  R 2 #PDA-CONDITION-CODES                    /* REDEF. BEGIN : #PDA-CONDITION
    3 #PDAC-USE-MSG-NR                 L      /* TRUE IF MESSAGE NUMBERS ARE U
    3 #PDAC-FILE-NAME-SPECIFIED        L
    3 #PDAC-FIELD-NAME-SPECIFIED       L
    3 #PDAC-PDA-SPECIFIED              L
    3 #PDAC-COMPLEX-FIELD              L      /* Field is a PE, MU a STRUCT or
  *                                           /* REDEFINE
    3 #PDAC-SCROLLING                  L      /* Scrolling
    3 #PDAC-NATURAL-WINDOWS            L      /* Set window sizes
    3 #PDAC-WINDOW-LENGTH              L      /* Set window line length
    3 #PDAC-WINDOW-COLUMN              L      /* Set window column height
    3 #PDAC-WINDOW-BASE                L      /* Set window base
    3 #PDAC-DEFINE-WINDOW              L      /* Generate DEFINE WINDOW
    2 #PDA-USER-AREA                   A  100 (1:40) /* Area for INPUT and der
  R 2 #PDA-USER-AREA                          /* REDEF. BEGIN : #PDA-USER-AREA
    3 RESET-STRUCTURE                         /* Use for resetting non-alpha
  *                                           /* fields in Clear Subprogram.
    4 #PDAX-DESCS                      A   55 (1:4) /* description
    4 #PDAX-USE-MSG-NR                 L
  *
  *   Modify screen 2
    4 #PDAX-PDA                        A    8 /* PDA with display info.
    4 #PDAX-FILE-NAME                  A   32 /* File name
    4 #PDAX-FIELD-NAME                 A   32 /* Field name
    4 #PDAX-MAP-NAME                   A    8 /* Input using map
    4 #PDAX-LINES-PER-SCREEN           N    3 /* Number of lines per screen
  *
  *   used to generate a
  *   DEFINE WINDOW statement.
    4 DEFINE-WINDOW-INFO
    5 #PDAX-WINDOW-SIZE                A    6 /* Window size
  R 5 #PDAX-WINDOW-SIZE                       /* REDEF. BEGIN : #PDAX-WINDOW-S
    6 #PDAX-WINDOW-SIZE-WIDTH          N    3 /* Window size width
    6 #PDAX-WINDOW-SIZE-HEIGHT         N    3 /* Window size height
    5 #PDAX-WINDOW-BASE                A    6 /* Window base
  R 5 #PDAX-WINDOW-BASE                       /* REDEF. BEGIN : #PDAX-WINDOW-B
    6 #PDAX-WINDOW-BASE-LINE           N    3 /* Window base line
    6 #PDAX-WINDOW-BASE-COLUMN         N    3 /* Window base column
    5 #PDAX-WINDOW-FRAME-OFF           L      /* Window frame off
    5 #PDAX-WINDOW-TITLE               A   65 /* Window title
    5 #PDAX-WINDOW-CONTROL-SCREEN      L      /* Window control screen on
    5 #PDAX-DEFINE-WINDOW              L      /* Use DEFINE WINDOW statement
    4 #PDA-FIELD-TYPE                  A    2 /* Field type: GR,PE,PC,MU,MC
  *                                           /* S(Structure), F(Single Field)
  *                                           /* R(REDEFINE)
    4 #PDA-FIELD-REDEFINED             L
```

```
    4 #PDA-LEVEL-NUMBER                N    1
    4 #PDA-FIELD-FORMAT                A    1
    4 #PDA-FIELD-LENGTH                N  3.1
  R 4 #PDA-FIELD-LENGTH
    5 #PDA-UNITS                       N    3
    5 #PDA-DECIMALS                    N    1
    4 #PDA-FROM-INDEX                  N    5 (1:3)
    4 #PDA-THRU-INDEX                  N    5 (1:3)
    4 #PDA-FIELD-RANK                  N    1
    4 #PDA-FILE-CODE                   P    8 /* file code for security check
    4 #PDA-MAX-LINES                   N    5 /* Num. of occurrences for PE/MU
    4 #PDA-WFRAME                      A    1 /* Parameters for window setting
    4 #PDA-WLENGTH                     A    3
    4 #PDA-WCOLUMN                     A    3
    4 #PDA-WBASE                       A    7                                ↩
```

The fields in the model PDA are described in the following sections.

#PDA-CONDITION-CODES

This field (L/1:75) is an array of condition codes that allow you to define up to 75 logical conditions for each model. The field is usually redefined into separate logical variables, one for each condition variable used by the model code frames. The name of the logical condition variable in the PDA must be the same as the condition, with a #PDAC- prefix added.

When a module is generated, the condition values are checked to determine whether the condition is True. The conditions are reset before the maintenance subprograms are invoked. Along with the pre-generation subprogram, the maintenance subprograms assign all True condition values.

Note: To make nested code frames more reusable across multiple models, it is important to use exactly the same naming conventions. In this way, the nested code frame logical and substitution parameters are always available to the model PDAs.

#PDA-USER-AREA

This field (A100/1:40) defines a large block of data that is passed between the Natural Construct nucleus and the model subprograms. Always redefine this field into separate fields that refer to the module being generated. The following information can be passed:

- Data entered by the developer on a maintenance panel. The names of the fields that receive the parameters should be prefixed by #PDAX- and appear first in the redefinition of #PDA-USER-AREA. Usually, the values for these fields are written as comments at the beginning of the generated program. This allows Natural Construct to read the parameters for subsequent regeneration.
- You can also group a series of related parameters into a single external parameter by redefining the #PDAX- variable into sub-fields. This technique reduces the number of comment lines at the beginning of a generated program.

Note: This technique should only be used when the length of the sub-fields does not change.

- Data calculated during the generation process and shared with the model subprograms. The variable names should be prefixed by #PDA- and appear second in the redefinition of #PDA-USER-AREA (after the #PDAX- variables).
- The pre-generation subprogram assigns these internal generation variables; all subsequent code frame and model subprograms can use the values.
- When you use substitution parameters in code frames, a variable with the same name and a #PDAX- or #PDA- prefix should be in the redefinition of the #PDA-USER-AREA variable. For example, the &MAX-SELECTIONS substitution parameter value should be supplied by the #PDA-MAX-SELECTIONS variable or the #PDAX-MAX-SELECTIONS variable.

  Note: To make nested code frames more reusable across multiple models, it is important to use exactly the same naming conventions. In this way, the nested code frame logical and substitution parameters are always available to the model PDAs.

CU—PDA

The following example shows the CU—PDA data area:

```
Parameter CU—PDA   Library SYSCST                              DBID  19 FNR  28
Command                                                                      > +
I T L Name                             F Leng Index/Init/EM/Name/Comment
Top - -------------------------------- - ---- ---------------------------------
  *   Parameters used by all user
  *   subprograms
  *
    1 CU—PDA
  *
  *   Parameters used by generating
  *   subprograms
    2 #PDA-MODE                        A    2 /* R=Report, S=Struct, SD=Str data
    2 #PDA-OBJECT-TYPE                 A    1 /* P=Program, N=Subprogram,etc.
  *
  *
  *   Parms used by modify screens
    2 #PDA-MODIFY-HEADER1              A   60 /* First heading on modify scr
    2 #PDA-MODIFY-HEADER2              A   54 /* Second heading on modify scr
    2 #PDA-LEFT-PROMPT                 A   11 /* Date
  R 2 #PDA-LEFT-PROMPT
    3 #PDA-LEFT-MORE-PROMPT            A    9
    2 #PDA-RIGHT-PROMPT                A   11 /* n of n
  R 2 #PDA-RIGHT-PROMPT
    3 #PDA-RIGHT-MORE-PROMPT           A    9
    2 #PDA-PHASE                       A    1 /* Modify, Generate, Clear etc.
    2 #PDA-DIALOG-METHOD               I    1 /* See CSLMMETH
  *                                           /* 1 = Input + Validate
  *                                           /* 2 = Input no validate
  *                                           /* 3 = Validate no input
  *                                           /* 4 = Validate input on error
    2 #PDA-TRANSLATION-MODE            L      /* Translation mode
  *
  *   The following PF key variables          are only required if the modify
  *   or sample program requires the          use of additional PF keys other
  *   than the standard MAIN, RETURN,         QUIT, HELP keys.
  *
  *   Place the following key names at        the bottom of map instead of
  *   using the KD option. The modify         program should reset the keys
  *   that are not being used or              assign the available key names
  *   to set additional keys.
  *
    2 #PDA-USERX-NAME                  A   10 /* User Exit name.
    2 #PDA-PF-NAME                     A   10 (1:12)
  R 2 #PDA-PF-NAME                            /* REDEF. BEGIN : #PDA-PF-NAME
    3 #PDA-MAIN-NAME                   A   10 /* Main menu key name.
    3 #PDA-RETURN-NAME                 A   10 /* Return key name.
    3 #PDA-QUIT-NAME                   A   10 /* Quit key name.
    3 #PDA-TEST-NAME                   A   10 /* Test key name.
    3 #PDA-BACKWARD-NAME               A   10 /* Bkwrd key name.
    3 #PDA-FORWARD-NAME                A   10 /* Frwrd key name.
    3 #PDA-LEFT-NAME                   A   10 /* Left key name.
    3 #PDA-RIGHT-NAME                  A   10 /* Right key name.
```

```
   3 #PDA-HELP-NAME                  A   10 /* Help key name.
   3 #PDA-AVAILABLE1-NAME            A   10 /* Not used by default.
   3 #PDA-AVAILABLE2-NAME            A   10 /* Not used by default.
   3 #PDA-AVAILABLE3-NAME            A   10 /* Not used by default.
 *
 *   This array contains the PF-KEY         number associated with each
 *   standard key setting as well as        the numbers of the available
 *   numbers for non-standard key           use.
   2 #PDA-PF-NUMBER                  N    2 (1:12)
 R 2 #PDA-PF-NUMBER                         /* REDEF. BEGIN : #PDA-PF-NUMBER
   3 #PDA-MAIN                       N    2 /* Main menu key number.
   3 #PDA-RETURN                     N    2 /* Return key number.
   3 #PDA-QUIT                       N    2 /* Quit key number.
   3 #PDA-TEST                       N    2 /* Test key number.
   3 #PDA-BACKWARD                   N    2 /* Bkwrd key number.
   3 #PDA-FORWARD                    N    2 /* Frwrd key number.
   3 #PDA-LEFT                       N    2 /* Left key number.
   3 #PDA-RIGHT                      N    2 /* Right key number.
   3 #PDA-HELP                       N    2 /* Help key number.
   3 #PDA-AVAILABLE1                 N    2 /* Not used by default.
   3 #PDA-AVAILABLE2                 N    2 /* Not used by default.
   3 #PDA-AVAILABLE3                 N    2 /* Not used by default.
 *
 *   This array corresponds to the          above array except the 'PF'
 *   'PF' string prefixes the key           for easy comparison to *PF-KEY.
   2 #PDA-PF-KEY                     A    4 (1:12)
 R 2 #PDA-PF-KEY                            /* REDEF. BEGIN : #PDA-PF-KEY
   3 #PDA-PF-MAIN                    A    4 /* PFnn where nn = main key.
   3 #PDA-PF-RETURN                  A    4
   3 #PDA-PF-QUIT                    A    4
   3 #PDA-PF-TEST                    A    4
   3 #PDA-PF-BACKWARD                A    4
   3 #PDA-PF-FORWARD                 A    4
   3 #PDA-PF-LEFT                    A    4
   3 #PDA-PF-RIGHT                   A    4
   3 #PDA-PF-HELP                    A    4
   3 #PDA-PF-AVAILABLE1              A    4 /* Not used by default.
   2 #PDA-CV3                        C      /* Special characters in T mode
   2 #PDA-CV4                        C      /* Column headings in T mode
   2 #PDA-CV5                        C      /* CV 5
   2 #PDA-CV6                        C      /* CV 6
   2 #PDA-CV7                        C      /* CV 7
   2 #PDA-CV8                        C      /* CV 8
   2 #PDA-SCROLL-INDICATOR           A    4 /* Scroll region indicator
 *
 *   Dynamic attribute characters
 *   from the control record. The
 *   following index values represent
 *   1=Default, 2=Intensify, 3=Blink,       4=Italics, 5=Underline,
 *   6=Reversed, 7=Blue, 8=Green,           9=White, 10=Pink, 11=Red,
 *   12=Turquoise, 13=Yellow.
   2 #PDA-DYNAMIC-ATTR-CHARS         A    1 (1:13)
```

```
*
*   Passed parameter from code frame
  2 #PDA-CV6                        C       /* CV 6
  2 #PDA-CV7                        C       /* CV 7
  2 #PDA-CV8                        C       /* CV 8
  2 #PDA-SCROLL-INDICATOR           A     4 /* Scroll region indicator
*
*   Dynamic attribute characters
*   from the control record. The
*   following index values represent
*   1=Default, 2=Intensify, 3=Blink,        4=Italics, 5=Underline,
*   6=Reversed, 7=Blue, 8=Green,            9=White, 10=Pink, 11=Red,
*   12=Turquoise, 13=Yellow.
  2 #PDA-DYNAMIC-ATTR-CHARS         A     1 (1:13)
*
*   Passed parameter from code frame
  2 #PDA-FRAME-PARM                 A    32
  2 #PDA-SYSTEM                     A    32 /* System must exist in dict.
*
```

The fields in CU—PDA are described in the following sections:

- #PDA-MODE
- #PDA-OBJECT-TYPE
- #PDA-MODIFY-HEADER1
- #PDA-MODIFY-HEADER2
- #PDA-LEFT-PROMPT
- #PDA-RIGHT-PROMPT
- #PDA-PHASE
- #PDA-DIALOG-METHOD
- #PDA-TRANSLATION-MODE
- #PDA-USERX-NAME
- #PDA-PF-NAME
- #PDA-PF-NUMBER
- #PDA-PF-KEY
- #PDA-TITLE
- #PDA-GEN-PROGRAM
- #PDA-MODEL-VERSION
- #PDA-HELP-INDICATOR
- #PDA-USER-DEFINED-AREA
- #PDA-UNDERSCORE-LINE
- #PDA-RIGHT-PROMPT-OF
- #PDA-DISPLAY-INDICATOR
- #PDA-CURS-FIELD
- #PDA-CVn
- #PDA-SCROLL-INDICATOR
- #PDA-DYNAMIC-ATTR-CHARS
- #PDA-FRAME-PARM

- #PDA-SYSTEM

#PDA-MODE

This field (A2) identifies the programming mode. The value for this field is the programming mode specified on the Maintain Models panel. Valid values for this field are S (structured), SD (structured data), and R (reporting) mode.

#PDA-OBJECT-TYPE

This field (A1) identifies the type of module generated. The value for this field is the module type specified on the Maintain Models panel. This field is useful when a model subprogram is associated with multiple models that use different module types. In this case, the presence or format of certain generated code may be dependent on the type of module generated.

#PDA-MODIFY-HEADER1

This field (A60) contains the description specified on the Maintain Models panel. Maintenance panels use the #HEADER1 variable for the first heading, instead of #PDA-MODIFY-HEADER1. If #HEADER1 has not been assigned a value, it is assigned the contents of #PDA-MODIFY-HEADER1.

#PDA-MODIFY-HEADER2

This field (A54) contains the description specified on the Maintain Models panel. Maintenance panels use the #HEADER2 variable for the second heading, instead of #PDA-MODIFY-HEADER2. If #HEADER2 has not been assigned a value, it is assigned the contents of #PDA-MODIFY-HEADER2.

#PDA-LEFT-PROMPT

This field (A11) is redefined into the #PDA-LEFT-MORE-PROMPT field (A9). The #PDA-LEFT-MORE-PROMPT field indicates the current date. Place this field as an output field in the top left corner of all maintenance panels. (If you require more than nine bytes, you can use the full length of A11.)

#PDA-RIGHT-PROMPT

This field (A11) is redefined into the #PDA-RIGHT-MORE-PROMPT field (A9). The #PDA-RIGHT-MORE-PROMPT field indicates the current panel and the total number of panels (1 of 4, for example). Place this field as an output field in the top right corner of all maintenance panels. (If you require more than nine bytes, you can use the full length of A11.)

#PDA-PHASE

This field (A1) identifies the current phase of the Natural Construct nucleus (see the CSLPHASE data area for an example). Valid values for this field are A (post-generation), B (batch), C (clear), D (default), G (generation), L (translate), M (modify), P (pre-generation), R (read), U (sample user exit), and V (save). The value for this field is typically controlled by the Natural Construct nucleus and should not be manipulated locally.

Note: Maintenance subprograms are also invoked prior to SAMPLE processing in the User Exit editor (in which case, the phase is U) and prior to the generation phase (in which case, the phase is G).

Since some subprograms are invoked during more than one phase, this field activates the subprogram logic for the current phase. For example, the maintenance subprograms performed during the maintenance phase (M) are invoked (with data stacked) during the generation (G) and sample user exit (U) phases. It may be inappropriate for the maintenance subprogram to perform certain processing during any of these phases.

#PDA-DIALOG-METHOD

This field (I1) is reserved for future use.

#PDA-TRANSLATION-MODE

This field (L) is reserved for future use.

#PDA-USERX-NAME

This field (A10) is for internal use only.

#PDA-PF-NAME

This field (A10/1:12) is an array containing the names of the standard PF-keys and is redefined into the following fields (A10):

Field	Description
#PDA-MAIN-NAME	Main menu key name.
#PDA-RETURN-NAME	Return key name.
#PDA-QUIT-NAME	Quit key name.
#PDA-TEST-NAME	Test key name.
#PDA-BACKWARD-NAME	Backward key name.
#PDA-FORWARD-NAME	Forward key name.
#PDA-LEFT-NAME	Left key name.
#PDA-RIGHT-NAME	Right key name.
#PDA-HELP-NAME	Help key name.
#PDA-AVAILABLE1-NAME	Not used (by default).
#PDA-AVAILABLE2-NAME	Not used (by default).
#PDA-AVAILABLE3-NAME	Not used (by default).

The names are in the same order as the key settings specified on the Natural Construct control record. The name for PF1 is stored in the first position, PF2 is stored in the second position, etc.

You can define special PF-keys for maintenance subprograms (or sample generation subprograms) by specifying the desired PF-key values and names on the Maintain Subprograms panel (S function on the Administration main menu).

Occasionally, a subprogram may need to modify its PF-key assignments based on internal program functions and parameter values. If this is the case, place this array of PF-key names on the model panels and set the appropriate PF-key names (assuming your model supports variable PF-keys).

If a subprogram requires PF-keys for non-standard functions that are not known at compile time, display this array on the map (instead of using the SET KEY statement and the KD option of the FORMAT statement).

#PDA-PF-NUMBER

This field (N2/1:12) is an array containing the PF-keys that support the standard PF-key functions and is redefined into the following fields (N2):

Field	Description
#PDA-MAIN	Main menu key number.
#PDA-RETURN	Return key number.
#PDA-QUIT	Quit key number.
#PDA-TEST	Test key number.
#PDA-BACKWARD	Backward key number.
#PDA-FORWARD	Forward key number.
#PDA-LEFT	Left key number.
#PDA-RIGHT	Right key number.
#PDA-HELP	Help key number.
#PDA-AVAILABLE1	Not used (by default).
#PDA-AVAILABLE2	Not used (by default).
#PDA-AVAILABLE3	Not used (by default).

The values in this array assign a PF-key function to a PF-key number (for indexing on the #PDA-PF-NAME table). The first occurrence contains the PF-key number associated with the "main" function, the second occurrence contains the PF-key number associated with the "return" function, etc.

To include additional PF-keys, use the PF-key corresponding to the numbers assigned to #PDA-AVAILABLE1 through #PDA-AVAILABLE3.

#PDA-PF-KEY

This field (A4) is an array corresponding to the #PDA-PF-NUMBER array (see ***#PDA-PF-NUMBER***) except the values have a PF- prefix. This makes it easy to compare the value of a *PF-KEY system variable to one of the following fields (A4):

Field	Description
#PDA-PF-MAIN	PF*nn*, where *nn* is the main menu key number.
#PDA-PF-RETURN	PF*nn*, where *nn* is the return key number.
#PDA-PF-QUIT	PF*nn*, where *nn* is the quit key number.
#PDA-PF-TEST	PF*nn*, where *nn* is the test key number.
#PDA-PF-BACKWARD	PF*nn*, where *nn* is the backward key number.
#PDA-PF-FORWARD	PF*nn*, where *nn* is the forward key number.
#PDA-PF-LEFT	PF*nn*, where *nn* is the left key number.

Field	Description
#PDA-PF-RIGHT	PF*nn*, where *nn* is the right key number.
#PDA-PF-HELP	PF*nn*, where *nn* is the help key number.
#PDA-PF-AVAILABLE1	Not used (by default).
#PDA-PF-AVAILABLE2	Not used (by default).
#PDA-PF-AVAILABLE3	Not used (by default).

Note: The PF-key variables defined in this PDA allow your models to automatically use the PF-key values and names specified on the Natural Construct control record. If you do not require this flexibility, use hardcoded PF-key values and names.

#PDA-TITLE

This field (A25) contains the title of the module that is generated, which is required for the generation process. The title identifies the module for the List Generated Modules function on the Generation main menu. Place this field on the model maintenance panels.

#PDA-GEN-PROGRAM

This field (A8) contains the name of the module that is generated, read, or saved. The value for this field is the module name specified on the Generation main menu. Place this field on the first maintenance panel for the model.

#PDA-MODEL-VERSION

This field (N2.2) contains the number of the Natural Construct version used to generate the model.

#PDA-HELP-INDICATOR

This field (A4) contains the help indicator for maps. The value for this field is the help indicator specified on the control record, such as an asterisk (*).

#PDA-USER-DEFINED-AREA

This field (A1/1:100) is available to the user.

#PDA-UNDERSCORE-LINE

This field (A80) contains the 1- to 4-character set used to create the underscore line for text on maps. The specified set is repeated until all spaces are filled (80, by default). The value for this field is the underscore character set specified on the Natural Construct control record.

For example, if "----" is specified, the underscore line is:

Or if "++" is specified, the underscore line is:

++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++

#PDA-RIGHT-PROMPT-OF

This field (A4) contains the text used in the right prompt for maps. The value for this field is the *of* indicator specified on the Natural Construct control record (1 of 4, for example).

#PDA-DISPLAY-INDICATOR

This field (A4/1:10) is an array corresponding to the position indicators used on maps. The values for this field are the position indicators specified on the Natural Construct control record (1, 2, 3..., for example).

#PDA-CURS-FIELD

This field (I4) contains the cursor position for dynamic translation on maps.

#PDA-CVn

These fields ©) are control variables (#PDA-CV1 through #PDA-CV8) used on maps to dynamically control the text displayed on a panel. These control variables are:

Control Variable	Description
#PDA-CV1	Controls field prompts.
#PDA-CV2	Controls prompt headings.
#PDA-CV3	Controls special characters.
#PDA-CV4	Controls column headings.
#PDA-CV5	Not currently used.
#PDA-CV6	Not currently used.
#PDA-CV7	Not currently used.
#PDA-CV8	Not currently used.

#PDA-SCROLL-INDICATOR

This field (A4) contains the scroll region indicator(s) used on maps. The value for this field is the character(s) specified on the Natural Construct control record (>>, for example).

#PDA-DYNAMIC-ATTR-CHARS

This field (A1/1:13) is an array containing the default dynamic attribute characters. The values for this array are the dynamic attributes specified on the Natural Construct control record. Dynamic attribute characters allow the developer to embed special characters within text that change how the text is displayed.

These dynamic attribute characters correspond to the following index occurrences:

Attribute	Index Occurrence
Default return	01
Intensify	02
Blinking	03
Italics	04
Underline	05
Reverse video	06
Blue	07
Green	08
White	09
Pink	10
Red	11
Turquoise	12
Yellow	13

The CSUDYNAT subprogram uses these settings for the Natural dynamic attribute parameter (DY=). For more information, see *CSUDYNAT Subprogram*.

#PDA-FRAME-PARM

This field (A32) contains different values depending on the type of subprogram. The Natural Construct nucleus can set this field before the code frame subprograms are invoked; this field is always set before the sample user exit subprograms are invoked.

For code frame generation subprograms, this field contains the value of the constant literal entered in the subprogram line in the code frame (next to the Parameter prompt). For sample user exit subprograms, this field contains the name of the user exit for which the sample was invoked.

#PDA-SYSTEM

This field (A32) contains the default system name when Predict program entries are generated from within Natural Construct. (Programmers/analysts can document generated modules in Predict by pressing the optns PF-key on the Generation main menu before saving or stowing the module.) Place this field on the first maintenance panel for the model.

Any supplied model that generates a dialog also uses this field as part of the key to access help information. The system value corresponds to the Major component of the help key.

CSASTD PDA

The CSASTD PDA is used by every model. It passes messages between subprograms and is typically used for error handling. CSASTD PDA contains the fields described in the following sections:

- MSG
- MSG-NR
- MSG-DATA
- RETURN-CODE
- ERROR-FIELD
- ERROR-FIELD-INDEXn

MSG

This field (A79) is used with the RETURN-CODE field (see *RETURN-CODE*) to pass messages between the Natural Construct nucleus and the model subprograms. It should be displayed on the message line of all maintenance panels and reset after all inputs.

MSG-NR

This field (N4) is not currently used.

MSG-DATA

This field (A32/1:3) contains the values for embedded substitution strings. If a message contains the `:1:`, `:2:`, or `:3:` substitution strings, you can supply values to these strings in MSG-DATA(1), MSG-DATA(2), and MSG-DATA(3), respectively.

RETURN-CODE

This field (A1) is used with the MSG field (see ***MSG***). When a module is generated, the model subprograms or related code frame subprograms may encounter problems. When this happens, the subprogram should assign the RETURN-CODE field before returning to the Natural Construct nucleus. It should also assign an error message to the MSG field.

If the value assigned to the RETURN-CODE field is blank (informational message) or W (warning message), a warning is issued by Natural Construct and a message is displayed in the Status window. The developer can either ignore the warning and continue the generation process or terminate generation.

If the value assigned to the RETURN-CODE field is C (communication error) or E (error), the error message is displayed but the developer cannot continue the generation process.

The CSLRCODE local data area contains valid return codes for the RETURN-CODE field.

ERROR-FIELD

This field (A32) identifies a field in error. The field name is displayed with the error message.

ERROR-FIELD-INDEXn

These fields (P3) identify occurrences of fields in error. If the error field is an element of an array, they identify the specific occurrence of the field in error.

Step 7: Create the Translation LDAs and Maintenance Maps

After defining the parameters and creating the parameter data area (PDA) for the model, you may want to create translation LDAs to support multilingual specification panels and the maintenance maps (panels) to accept parameters from the developer. These procedures are described in the following sections:

- Format of the Translation LDAs
- Maintenance Maps

Format of the Translation LDAs

To support multilingual text and messages, each maintenance panel can use up to five translation local data areas (LDAs). These LDAs contain the names of the fields that can be translated. You cannot display a panel in another language unless the module that invokes the panel has a corresponding translation LDA.

All translation LDAs must have following format:

```
Local     CUBAMAL   Library SYSCST                              DBID  18 FNR   4
Command                                                                      > +
 I T L Name                             F Leng Index/Init/EM/Name/Comment
 All - -------------------------------- - ---- ---------------------------------
   * * **SAG TRANSLATION LDA
   * * * used by map CUBAMA0.
     1 CUBAMAL
     2 TEXT                                    /* Corresponds to syserr message
     3 #GEN-PROGRAM                     A   20 INIT<'*2000.1,.'>
     3 #SYSTEM                          A   20 INIT<'*2000.2,.'>
     3 #GDA                             A   20 INIT<'*2000.3,.'>
     3 #TITLE                           A   20 INIT<'*2001.1,.'>
     3 #DESCRIPTION                     A   20 INIT<'*2001.2,.'>
     3 #GDA-BLOCK                       A   20 INIT<'*2001.3,.'>
   R 2 TEXT
     3 TRANSLATION-TEXT
     4 TEXT-ARRAY                       A    1 (1:120)
     2 ADDITIONAL-PARMS
     3 #MESSAGE-LIBRARY                 A    8 INIT<'CSTLDA'>
     3 #LDA-NAME                        A    8 INIT<'CUBAMAL'>
     3 #TEXT-REQUIRED                   L      INIT<TRUE>
     3 #LENGTH-OVERRIDE                 N    10 /* Explicit length to translate  ↩
```

In this example, the fields in CUBAMAL correspond to the following fields on the Standard Parameters panel for the Batch model:

Field Name in LDA	Field Name on Panel
#GEN-PROGRAM	Module
#SYSTEM	System
#GDA	Global data area
#TITLE	Title
#DESCRIPTION	Description
#GDA-BLOCK	With block

When naming your translation LDAs, we recommend using the name of the module that uses the LDA and adding an "L" in the last character position. For example, the CUBAMA maintenance subprogram uses the CUBAMAL translation LDA.

The sum of the lengths of all fields in the translation LDA must match the length of the text array. In the CUBAMAL example, each of the six fields has a length of 20 and the text array is 1:120 (6 x 20).

To support multilingual specification panels, use SYSERR numbers to assign the INIT values for the translation LDA fields. The translation LDAs are passed through the CSUTRANS utility, which expects the structure shown in the CUBAMAL example. CSUTRANS also expects the SYSERR INIT values in the following format:

Position	Format
Byte 1	Must be an asterisk (*).
Bytes 2–5	Must be numeric and represent a valid SYSERR number. The first five bytes are mandatory (bytes 1–5); these values are used to retrieve the text associated with the corresponding SYSERR number and the current value of the *Language Natural system variable. If the text for the current language is not available, CSUTRANS follows a modifiable hierarchy of *Language values until text is retrieved (you can define this hierarchy in the DEFAULT-LANGUAGE field within the CNAMSG local data area). As the original development language, English (*Language 1) should always be available. **Note:** CSUTRANS does not perform any substitutions using `:1::2::3:`. To perform substitutions, call the CNUMSG subprogram.
Byte 6	Can be a period (.), which indicates that the next byte is a valid position value.
Byte 7	Can be a position value. Valid values are 1 to 9, A (byte 10), B (byte 11), C (byte 12), D (byte 13), E (byte 14), F (byte 15), and G (byte 16). For example, *2000.2 identifies the text for SYSERR number 2000, position 2 (as delimited by "/" in SYSERR). If the message for SYSERR number 2000 is Module/System/Global data area, only System is retrieved. If you reference the same SYSERR number more than once in a translation LDA, define the INIT values on consecutive lines to reduce the number of calls to SYSERR; the position values for a SYSERR number can be referenced in any order. To minimize confusion, we recommend that you use the *.n* notation even when there is only one message for the SYSERR number.
Byte 8	Can be a comma (,), which indicates that the next byte or bytes contain special format characters. Values specified before the comma (,) indicate what text to retrieve; values specified after the comma indicate how the text is displayed. **Note:** Although you can use a comma in byte 6 (instead of a period), we recommend that you always use the *.n* position indicator in bytes 6 and 7.
Byte 9	After the comma, can be one of the following: ■ . Indicates that the first position after the field name is blank and the remainder of the field prompt is filled with periods. For example, Module name ....: ■ + Indicates that the text is centered using the specified field length override (see description of Byte 10). If you do not specify the override length, Natural Construct uses the actual field length. ■ < Indicates that the text is left justified (this is the default). ■ >

Position	Format
	Indicates that the text is right justified. ■ / Indicates that a length override value follows.
Bytes 10–16	After the / override length indicator (see above), indicates the actual override length in bytes.

If you want to use the override length notation (*0200.4,+/6, for example) and the LDA field is too small (A6, for example), you can define a larger field (A12, for example), redefine it using a shorter display value, and then use the override length notation:

```
01  FIELD-NAME              A1  INIT<'*0200.4+/6'>
01  Redefine #FIELD-NAME
    02  #SHORT-FIELD-NAME   A6
```

Note: For more information, see ***Use SYSERR References***.

Maintenance Maps

Normally, each maintenance subprogram is associated with a different maintenance map. You can use a layout map as a starting layout for your maintenance maps and then list the model PDA fields in the Map editor and select the desired fields. For a standard maintenance map, use the CDLAY layout map. For a multilingual maintenance map, you can also use the CDLAY layout map and remove all text except the lines containing the first and second headings. (For an example of a multilingual maintenance map, refer to CU--MA0 in the SYSCST library.)

You can also use the Map model to create maintenance maps. For a description, refer to the applicable section in *Natural Construct Generation*.

Step 8: Create the Model Subprograms

You can use the supplied models to generate the subprograms described in this step. For a detailed description of a model, refer to the applicable section in this documentation. The model generation models are described in the order they are implemented during the generation process.

Maintenance Subprograms

Generated using the CST-Modify model, these subprograms receive the specification parameters (#PDAX variables in the model PDA) from the developer and should ensure that the parameters are valid. Maintenance subprograms can also set condition codes and assign derived PDA variables.

Maintenance subprograms are executed in the same order as they appear on the Maintain Models panel. Usually, there is one maintenance subprogram for every left/right (horizontal) maintenance panel. Data edits should only be applied if the developer presses Enter or PF11 (right). Either the maintenance subprogram or the maintenance map can validate the parameters.

You should only trap PF-keys that perform specialized functions related to the panel. If you want the PF-key settings to be dependent on the default settings specified on the control record, the subprogram should not contain hardcoded PF-keys (check the PF-key values using the variables specified in CU—PDA).

You can define special PF-keys and window settings for each maintenance subprogram (see *Maintain Subprograms Function*).

Note: A maintenance subprogram can test the value of CU—PDA.#PDA-PHASE to identify the phase during which it was invoked.

References

- For an example of a generated maintenance subprogram, refer to CUMNMA and CUMNMB in the SYSCST library.
- For information about the CST-Modify model, see *CST-Modify Model*.

When are Maintenance Subprograms Invoked?

The Natural Construct nucleus invokes the maintenance panels in the following situations:

Generation Main Menu

When the developer supplies the following input on the Generation main menu:

Field	Input
Function	M
Module	TEST
Model	*model name*

The nucleus invokes the first maintenance panel for the specified model.

- If the developer presses Enter or PF11 (right) on the first panel, the nucleus invokes the second panel; if there are no other panels, the nucleus invokes the Generation main menu.

When the developer supplies the following input on the Generation main menu:

Field	Input
Function	M
Module	TEST
Panel	2
Model	*model name*

The nucleus invokes the second maintenance panel for the specified model.

- If the developer presses Enter or PF11 (right) on the second panel, the nucleus invokes the second panel; if there are no other panels, the nucleus invokes the Generation main menu.
- If the developer presses PF10 (left), invokes the second panel and displays the message: Beginning of specification panels.

When the developer supplies the following input on the Generation main menu:

Field	Input
Function	G
Module	TEST
Model	*model name*

The nucleus invokes all maintenance panels for the specified model to ensure that all parameters have been edited before generation. The input panels are not displayed unless an error is encountered.

User Exit Editor

When the developer supplies the following input on the command line in the User Exit editor:

```
> SAMPLE
```

The nucleus invokes all maintenance panels for the specified model to ensure that all parameters have been edited before generation. The input panels are not displayed unless an error is encountered.

Pre-Generation Subprogram

Generated using the CST-Pregen model, this subprogram is invoked either after all maintenance subprograms have been executed during the generation phase or after the SAMPLE command has been issued from the User Exit editor. It is the first user subprogram invoked. It assigns all True condition values, based on user-supplied input parameters or other calculated values.

Note: All #PDAC- condition values are reset before the generation process is started.

This subprogram should also calculate the values of any #PDA variables required by subsequent generation subprograms. For simple models that do not have code frames, this subprogram can also perform the functions of a generation subprogram. (Condition code values and derived fields can also be assigned within the maintenance subprograms.)

References

- For an example of a generated pre-generation subprogram, refer to CUMNPR in the SYSCST library.
- For more information about the CST-Pregen model, see *Parameters for the CST-Pregen Model*.

Generation Subprograms

Because the lengths and contents of certain code frame parameters change based on user-supplied input values or information in Predict, these parameters must be supplied by the generation subprograms. These subprograms write statements to the Natural edit buffer, based on user-supplied input parameters or other calculated values.

To write to the edit buffer, include a `DEFINE PRINTER(SRC=1) OUTPUT 'SOURCE'` statement in the subprogram that routes the output to the source work area. To allow models to be ported to multiple platforms, use the CU--DFPR copycode member to define the SRC printer.

All `WRITE (SRC)`, `DISPLAY (SRC)`, and `PRINT (SRC)` statement output for your print file is written to the edit buffer. Use the NOTITLE option on each of these statements. If a DISPLAY statement is used in the subprogram, also use the NOHDR option.

Tip: When trailing blanks should be suppressed in variable names, the PRINT statement can be a useful alternative to the WRITE statement. However, you may want to increase the line length of the edit buffer when using the PRINT statement, so variable names are not split at the "-" character.

Because generation logic can be highly complex, these subprograms allow ultimate flexibility. However, they are less maintainable than code frame statements since you must change Natural programs to modify the generated code.

Generation subprograms can also accept the #PDA-FRAME-PARM constant code frame parameter in CU—PDA. This parameter allows a subprogram to be invoked several times within the generation process. Each time the generation subprogram is invoked, it can use the value of this parameter to determine what to generate.

To invoke the generation subprograms, specify line type N in the T (type) column in the Code Frame editor. You can also specify the constant parameter value on this line.

The following example of the Code Frame editor shows the code frame in which the CUMYGVAR subprogram is invoked. The DEFINE and INIT parameters are passed to this subprogram:

```
Frame ..............GENSUBP                                         SIZE 172
Description ........Example of generation subprogram                FREE 36572
>                                             > + ABS X X-Y _ S 21   L 1
  ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..T C
Subprogram: CUMYGVAR Parameter: DEFINE                    N
 .
 .
 .
Subprogram: CUMYGVAR Parameter: INIT                      N
```

Note: For an example of a generated generation subprogram, refer to CUMNGGL in the SYSCST library.

Post-Generation Subprogram

Generated using the CST-Postgen model, this subprogram provides the values for the substitution parameters in the code frames identified by an ampersand (&). When the developer enters "G" in Function on the Generation main menu, this subprogram is invoked as the final stage of the generation process.

During generation, code lines specified in the code frame are written to the edit buffer, as well as the output of the generation subprogram in the code frame. Substitution parameters are included in the edit buffer exactly as they appear in the code frame.

After the Generation phase, the content of the edit buffer can be the following:

```
>                                     > + Program     : ABCSUBS  Lib: CSTDEV
All    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
DEFINE DATA LOCAL
  01 #MAX-LINES(P3) CONST<&MAX-SELECTIONS>
  01 #LINE-NR(P3/1:#MAX-LINES)
  01 #I(P3)
END-DEFINE
FOR #I = 1 TO #MAX-LINES
  ASSIGN #LINE-NR(#I) = #I
END-FOR
.
.
       ....+....1....+....2....+....3....+....4....+....5....+... S 10   L 1   ↩
```

The post-generation subprogram substitutes the code frame parameters with the corresponding substitution values by stacking the substitution parameters and their corresponding values. Use the STACK TOP DATA FORMATTED statement to stack these values. For example:

```
DEFINE DATA
  PARAMETER USING CUMYPDA
  PARAMETER USING CU—PDA
  PARAMETER USING CSASTD
END-DEFINE
**
** Stack change commands
STACK TOP DATA FORMATTED '&KEY' #PDAX-KEY
STACK TOP DATA FORMATTED '&KEY-FORMAT' #PDA-KEY-FORMAT
END
```

- #PDAX-KEY must contain the &KEY substitution parameter value.
- #PDA-KEY-FORMAT must contain the &KEY-FORMAT substitution parameter value.

Stack Order of Substitution Parameters

Stacked parameters build a series of CHANGE commands that are applied by the nucleus after the post-generation subprogram is finished executing. To change the substitution variables embedded within a longer string, these CHANGE commands use the ABS (Absolute) option. If one substitution variable is a substring of another substitution variable, stack the longer substitution variable last. Since the STACK TOP option supplies the substitution values, the changes to the longer substitution value are applied first. For example:

```
STACK TOP DATA FORMATTED '&KEY' #PDAX-KEY
STACK TOP DATA FORMATTED '&KEY-FORMAT' #PDA-KEY-FORMAT
```

Blanks versus Nulls

By default, the substitution parameter is replaced by one blank character if the second parameter (the substituted value) is blank. If you want to replace a blank substitution value with a null string, use the following notation:

```
STACK TOP DATA FORMATTED '&FILE-PREFIX' #PDA-FILE-PREFIX 'NULL'
```

Clear Subprogram

Generated using the CST-Clear model, this subprogram resets the #PDA-USER-AREA variables in the model PDA. Only non-alphanumeric variables are reset. The clear subprogram can also assign initial default values for user parameters.

Notes:

1. If you do not specify a clear subprogram, the Clear function on the Generation main menu sets #PDA-USER-AREA to blanks.
2. The edit buffer is always cleared, regardless of whether the model uses a clear subprogram.

When are Clear Subprograms Invoked?

The Natural Construct nucleus invokes the clear subprogram in the following situations:

- When the developer invokes the Clear Edit Buffer function on the Generation main menu.
- When the developer changes the model name and the new model uses a different PDA.
- Immediately before the Read Specifications function is executed on the Generation main menu.

The following example shows a the code generated for a clear subprogram:

```
DEFINE DATA
  PARAMETER USING CUMYPDA
  PARAMETER USING CU—PDA
  PARAMETER USING CSASTD
END-DEFINE
```

```
**
**Initialize non-alpha fields and set default values.
RESET #PDAX-MAX-PANELS #PDA-KEY-LENGTH
ASSIGN #PDAX-GDA = 'CDGDA'
ASSIGN #PDA-SYSTEM = *LIBRARY-ID
END
```

Save Subprogram

Generated using the CST-Save model, this subprogram writes the specification parameters to the edit buffer. To read a previously-generated program, the model must have both a save and a read subprogram. The save subprogram must contain a separate WRITE statement for each specification parameter (#PDAX variable). Use the equal (=) notation to include the variable name with the contents of the variables. For example:

```
WRITE(SRC) NOTITLE '=' #PDAX-variable-name
```

Note: Use a separate WRITE statement for each element of an array.

The following example shows a the code generated for a save subprogram:

```
DEFINE DATA
  PARAMETER USING CUMYPDA
  PARAMETER USING CU—PDA
  PARAMETER USING CSASTD
  LOCAL
  01 #I(P3)
  01 #TEMP(A25)
END-DEFINE
**
DEFINE PRINTER (SRC=1) OUTPUT 'SOURCE'
FORMAT(SRC) LS=150
**
** Write out parameters to be saved.
WRITE(SRC) NOTITLE '=' #PDAX-GDA
WRITE(SRC) NOTITLE '=' #PDAX-MAIN-MENU-PROGRAM
WRITE(SRC) NOTITLE '=' #PDAX-QUIT-PROGRAM
FOR #I = 1 TO 4
  IF #PDAX-DESC(#I) NE ' ' THEN
    COMPRESS '#PDAX-DESC(' #I '):' TO #TEXT LEAVING NO
    PRINT(SRC) NOTITLE #TEXT #PDAX-DESC(#I)
  END-IF
END-FOR
END
```

Note: When compressing an index value that can be more than one digit in length, redefine a numeric index with an alpha string and compress the alpha string to preserve leading zeros.

Natural Construct changes the output of the subprogram to:

```
**SAG variable-name: variable contents
```

For example, `#PDAX-MAP-NAME: MYMAP` becomes `**SAG MAP-NAME: MYMAP`. The lines containing the **SAG parameter values are placed at the beginning of the generated module.

Read Subprogram

Generated using the CST-Read model, this subprogram reads the specification parameters for a generated module. It contains a series of INPUT statements that accept the data previously placed in the Natural stack. The read subprogram is invoked when the developer invokes the Read Specifications function on the Generation main menu.

Before the read subprogram is invoked, all **SAG parameter values are placed on the Natural stack. The read subprogram repeats a series of INPUT statements to accept the stacked parameters and assign them to the correct PDA variables. This subprogram must correspond to the save subprogram that writes the **SAG parameter lines. The read subprogram can also read common parameters from a different model.

Notes:

1. Natural Construct invokes the clear subprogram before invoking the read subprogram. It is not necessary to save null parameter values.
2. For an example of a generated read subprogram, refer to CUMNR in the SYSCST library.

Sample User Exit Subprograms

Generated using the CST-Frame model, these subprograms help the developer create user exit code by providing a starting sample. The subprograms can be simple or complicated, depending on the model.

When creating a sample subprogram, you can include additional parameters to give the developer more control over what is generated into the user exit. To pass additional information to the sample subprogram, use the CU—PDA.#PDA-FRAME-PARM variable.

All maintenance subprograms and the pre-generation subprogram are automatically invoked before the sample subprograms are invoked. This ensures that the current specification parameters are valid and the conditions are set.

To define a sample subprogram, enter ".E" at the beginning of a user exit line in the Code Frame editor. For information, see *Use Parameters Supplied by User Exits*.

For an example of a sample subprogram, refer to CUFMSRIN in the SYSCST library.

Documentation Subprogram

Generated using the CST-Document model, this subprogram creates an extended Predict description. To support the generation of a Predict extended description for the generated modules, you must create a documentation subprogram for your model. This subprogram creates a free-form description of the generated module using the information entered on the model specification panels. You can write information in any language for which you have translated help text members. For more information, see ***Using SYSERR for Multilingual Support***.

The documentation subprogram writes the model description to Predict when the developer turns this option on (using the optns PF-key on the Generation main menu) and invokes the Save or Stow function. The functions available on the Generation main menu are described in *Natural Construct Generation*.

Note: For an example of a generated documentation subprogram, refer to CUMND in the SYSCST library.

Test the Model Subprograms

Because a model contains several components, it is often better to test each component individually, or test related subprograms, without the overhead of the Natural Construct nucleus. After defining the model PDA, maintenance maps, and model subprograms, you can test the individual components of the model.

To test the model subprograms:

1 Issue the CSUTEST command from the SYSCST library.

The Single Module Test Program panel is displayed. For example:

```
CSUTEST                ***** Natural Construct *****              CSUTESM1
Oct 09                 - SINGLE MODULE TEST PROGRAM -                    ↩
                                                                         ↩
Code Function           *Model: ________________________________         ↩
---- ------------------   Number all subprograms to be executed          ↩
 R   Release Variables    |                     |                        ↩
 *   Execute All Subp.    V                     |                        ↩
1-9  Execute One Subp.       Clear :            V                        ↩
 E   Edit source             Mod  1:              Mod  6:                ↩
 C   Clear Edit Buffer       Mod  2:              Mod  7:                ↩
 ?   Help                    Mod  3:              Mod  8:                ↩
 .   Terminate               Mod  4:              Mod  9:                ↩
---- ------------------      Mod  5:              Mod 10:                ↩
 _                           Pregen:              Save  :                ↩
                Source       Documt:              Postgn:                ↩
                Lines                                                    ↩
          Total:    0                        Frame Parameter or Exit Name ↩
                          _  Other : ________ ___________________________
                          _  Other : ________ ___________________________
                          _  Other : ________ ___________________________
                          _  Other : ________ ___________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help        quit                                                   ↩
 ↩
```

A typical test will invoke the clear subprogram, one or more maintenance subprograms (indicated by Mod *n*), the pre-generate subprogram, and a generation subprogram (in that order).

Note: This panel is a utility; it is not available in dynamic translation mode.

2 Enter the name of the model in Model.

Note: If the test conditions and variables for the generation subprogram are set in the pre-generation or maintenance subprograms, invoke these subprograms first.

The names of the model subprograms are displayed. For example:

```
CSUTEST               ***** Natural Construct *****              CSUTESM1
Oct 09                 - SINGLE MODULE TEST PROGRAM -                    ↩
                                                                         ↩
Code Function           *Model: BROWSE-SELECT___________________         ↩
---- ------------------   Number all subprograms to be executed          ↩
 R   Release Variables    |                     |                        ↩
 *   Execute All Subp.    V                     |                        ↩
1-9  Execute One Subp.    _  Clear : CUSLC      V                        ↩
 E   Edit source          _  Mod  1: CUSCMA     _ Mod  6: CUSCMG         ↩
 C   Clear Edit Buffer    _  Mod  2: CUSLMB       Mod  7:                ↩
 ?   Help                 _  Mod  3: CUSCMC       Mod  8:                ↩
 .   Terminate            _  Mod  4: CUSLME       Mod  9:                ↩
---- ------------------   _  Mod  5: CUSLMF       Mod 10:                ↩
 _                        _  Pregen: CUSLPR     _ Save  : CUSCST         ↩
                Source    _  Documt: CUSLD      _ Postgn: CUSLPS         ↩
                Lines                                                    ↩
          Total:    0                         Frame Parameter or Exit Name ↩
                         _  Other : ________ ________________________________
                         _  Other : ________ ________________________________
                         _  Other : ________ ________________________________
                         _  Other : ________ ________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help        quit                                                   ↩
New model definition read.                                               ↩
 ↩
```

This window also displays the total number of lines in the source buffer.

3 Type a number beside each subprogram you want to test.

4 Type the same number in the input field below the Code column.

Valid codes are:

Code	Description
R	Resets the parameter data area (PDA) passed to all model subprograms.
*	Executes all model subprograms. Subprograms marked with a number are executed in order from 1 to 9. Code generated into the edit buffer by a subprogram is delimited by comments containing the name of the subprogram.
1–9	Executes the specified model subprogram. To execute a specific subprogram, enter a number from 1 to 9. If you enter 1, for example, all subprograms marked 1 are executed in the same order they are displayed on the panel.
E	Invokes the appropriate Natural editor to edit source.
C	Clears the edit buffer. You should clear the edit buffer before testing the next subprogram.
?	Displays help for the panel.
.	Terminates the Test utility and displays the Natural Next prompt (Direct Command box for Unix).

Note: Optionally, you can enter the names of up to four generation subprograms and code frame parameters or user exits to pass to each subprogram when it is invoked.

5 Press Enter to test the model.

Debug a Model

After creating all the components of a model, you can use several Natural Construct trace facilities to display information about the generation process.

To invoke the trace facilities:

1 Enter the specifications for the model you want to test on the Generation main menu.

2 Press PF5 (optns).

The Optional Parameters window is displayed. For example:

```
CSGOPTS             Natural Construct             CSGOPTS0
Oct 09             Optional Parameters              1 of 1
  Status window ................ _
                    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 quit
```

3 Mark which trace facilities to invoke while debugging the model.

The trace facility options are:

Option	Description
Status window	Displays the Status window during generation. Messages in this window indicate which module is executing at each stage of the generation process. **Note:** The default for this field is determined by the value specified for the Status field on the Maintain Models panel (see ***Maintain Models Function***). The Status window options are: ■ Step Allows you to "step" through the stages of the generation process by pressing Enter; the next message is not displayed until you press Enter. To have the generation process continue unaided, press PF2 (run). ■ Text Displays messages as text (for example, "starting" ... and "ending" ...). If this field is not marked, messages are displayed with graphics "--->" ... (starting) and "<---" ... (ending).
Embedded statements	Writes embedded statements to the source buffer as part of the generated module. These statements indicate where the code originated and the name of the code frame, generation subprogram, or sample subprogram that produced it.
Condition codes	Displays the values of the condition codes in the Condition Codes window after the pre-generation subprogram is invoked.
Post-generation modifications	Displays the values of the code frame substitution parameters, which are identified by an ampersand (&), in the Post-Generation Modifications window during generation. The window is displayed after the post-generation subprogram stacks the substitution values in the code frame.
Specifications only	Saves 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	Documents the saved generated module (program, data area, etc.) in the Predict data dictionary.

4 Type "G" in Function on the Generation main menu.

The following example shows the Status window with graphics instead of text:

```
CSGMAIN             N a t u r a l   C o n s t r u c t          CSGMNM0
  +-------------------------------------------------------------+   1 of 1
  | CSGOPTS             Natural Construct            CSGOPTS0 |
  | Oct 09             Optional Parameters             1 of 1 |
 +----------------------------------------------------------------------+
 | CSGENPGF                    Natural Construct                        |
 | Oct 09                        Status Window                   1 of 1 |
 |                                                                      |
 | <-- SAVE CUGRS                                                       |
 | --> FRAME CUGRF9                                                     |
 |     --> FRAME CU--B7                                                 |
 |                                                                      |
 +----------------------------------------------------------------------+
  |   Document in Predict .......... _                          |
  | Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9- |
  |       help  retrn                                           |
  |                                                             |
  +-------------------------------------------------------------+
Function ......... g___  Module ........... CUMNR___ Panel ...... __
Model  ........... CST-READ_________________________ Type........ Subprogram
Command .......... _________________________________ Library .... SYSCST
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help        quit        optns                                        lang
```

Miscellaneous Tips and Precautions

The following tips and precautions apply when using the model subprograms:

- If you modify the redefinitions in a parameter data area (PDA), recatalog all subprograms that use the PDA. (You can extend redefinitions without recataloging.)
- In the post-generation subprogram, use the `STACK TOP DATA FORMATTED` statement so Natural does not process input delimiter and assign characters.
- In the generation subprograms, use the `NOTITLE` or `WRITE TITLE ' '` statements.
- To remove trailing blanks, use the `PRINT (SRC) NOTITLE` statement.
- If you include `PRINT` statements, be sure to use a long line length (LS=150) so Natural does not break the line on a "-" or other special character.
- To write data without embedded spaces, use an edit mask. For example:

```
PRINT(SRC) NOTITLE #FIELD(EM='UPDATE-VIEW.'X(32)) ...
```

- In user-supplied text strings that are used to build quoted literals, always change single quotation marks to double quotation marks. For example:

```
INCLUDE CU--QUOT              /* Assign #DOUBLE-QUOTE based on ASCII/
                              /* EBCDIC
EXAMINE #PDAX-HEADING FOR ''''
AND REPLACE WITH #DOUBLE-QUOTE
```

  CU--QUOT is supplied with Natural Construct.

  Note: For double-byte languages, such as Kanji, use the CSUEXAM subprogram to perform the Examine and Replace operations.

- Although it is always better to use the `.n` extension when using SYSERR numbers to define field prompts, you can divide the contents of a delimited SYSERR message (indicated by the "/" character) with a single definition — if the field prompts are all the same length and are defined in the LDA one after the other as follows:

```
#FIELD-ONE   A 10  INIT<'*1234'>
#FIELD-TWO   A 10
#FIELD-THREE A 10
```

  If the SYSERR message is *prompt1*/*prompt2*/*prompt3*, the result is #FIELD-ONE = *prompt1*, #FIELD-TWO = *prompt2*, and #FIELD-THREE = *prompt3*.

Implement Your Model

After testing the code frames and model components (data areas, model subprograms, maps, etc.), you are ready to make your model available to developers in the Generation subsystem. To do this, use the SYSMAIN utility to copy all the model components to the SYSLIBS library.

Create Statement Models

Statement models generate portions of code, such as Natural statements, Predict views, and field processing code, which can be used in programs generated by your programmers/analysts.

To create a statement model, specify a period (.) in the Type field on the Maintain Models panel when you define the model. Typically, a statement model uses a parameter data area (PDA), a maintenance subprogram, and a pre-generation subprogram (most do not use code frames). Statement models do not support user exit code. After defining the model and its components, use the SYSMAIN utility to move the model components into the SYSLIBS library.

Statement models are designed to look like the statement syntax they are generating. For example, the If model looks like the IF statement:

```
IF   ____________________________________________________________
THEN ____________________________________________________________
     ____________________________________________________________
ELSE ____________________________________________________________
     ____________________________________________________________
END-IF
```

The screen text looks exactly like the Natural syntax. This also eliminates the need for translation, thus improving performance and screen presentation.

To invoke a statement model, the developer issues the .G line command in the User Exit, code frame, or Natural program editor. Using statement models can give your programmers/analysts a variety of benefits, including:

- Reduce the need to refer to the Natural Statements documentation for the statement syntax.
- Reduce the keystrokes required to code Natural statements, since keywords are automatically generated.
- Generate statements into their programs that have a consistent indentation.
- Allow their programs to perform tedious calculations (centering headings within a window, for example).
- Allow their programs to access system files and automatically retrieve Predict views, SYSERR message numbers, etc.

For information about invoking and using statement models, refer to *Statement Models*, *Natural Construct Generation*.

Code Alignment of Generated Statement Models

By default, Natural Construct aligns the generated block of code so the first generated statement is indented by the same amount as the line on which the .G command was entered. If you do not want your model to use this alignment, generate a line beginning with "**" as the first line of your generated code.

Use the Supplied Utility Subprograms and Helproutines

Natural Construct provides many subprograms and helproutines to simplify and standardize the model creation process. These utilities, which are used by the supplied models, can also be used by your models. The source for these utilities is not supplied.

All subprograms use an external parameter data area (PDA). The source for this PDA is located in the SYSCST library. Use this PDA as the local data area (LDA) in the invoking subprograms to determine required parameters. Parameters are documented within the PDA.

The supplied utilities are divided into categories, based on the type of information they access. The names of these subprograms and helproutines begin with one of the following prefixes:

Prefix	Description
CPU	Predict data retrieval subprograms.
CPH	Predict data helproutines.
CNU	Natural data retrieval subprograms.
CNH	Natural data helproutines.
CSU	Natural Construct utility subprograms.

Note: For more information about the supplied utilities, see ***External Objects***.

5 New Model Example

This section provides a step-by-step example of creating a new model using the procedure described in ***Build a New Model***. The model, Menu, generates a program that displays several choices to a user and allows the user to select one.

Note: For an example of a generated menu program, refer to NCMAIN in the demo library.

Step 1: Define the Scope of the Model

A program generated by the Menu model will provide a list of options and descriptions to the user for selection. The INPUT statement can be generated by Natural Construct or supplied by the developer.

Step 2: Create the Prototype

After defining the scope of the model, create a prototype to handle the most complex function and then refine the prototype to handle the simpler functions.

The following example shows the output from the NCMAIN prototype:

```
NCMAIN                ***** ACME DEPARTMENT STORES *****              NCLAYMN1
Oct 09                       - MAIN MENU -                            04:11 PM

               Code |  Subsystem
              +------+-------------------------------------------------+
              |  C   |  Customer                                       |
              |  T   |  Table Maintenance                              |
              |  O   |  Order                                          |
              |      |                                                 |
              |      |                                                 |
              |      |                                                 |
              |      |                                                 |
              |      |                                                 |
              |      |                                                 |
              |      |                                                 |
              |      |                                                 |
              |  ?   |  Help                                           |
              |  .   |  Terminate                                      |
              +------+-------------------------------------------------+
          Code: __
Direct Command: ____________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        flip                                    main
```

Step 3: Scrutinize the Prototype

After creating the prototype, follow the steps outlined in *Step 3: Scrutinize the Prototype*, to ensure that all of the assumptions are correct and the scope of the model has been addressed.

Step 4: Isolate the Parameters in the Prototype

Next, identify data that must be supplied by parameters.

Parameters for the Program Header

The parameters supplied for the program header are:

- Name of the program being generated.
- Application to which the generated program belongs.
- Date and time the program was generated.
- Title and description of the program.

Parameters for the Program Body

The parameters supplied for the program body are:

- Name of the global data area (GDA).
- Map used by the generated program.
- List of functions and their descriptions.

Step 5: Create a Code Frame and Define the Model

This section describes how to create a code frame and define the model.

Create the Code Frame

Once you have identified all data that must be supplied by parameters, you can create the code frame (CMNA?) for the model. For more information, see *Create the Code Frames*.

Note: For an example of the code frame for the Menu model, display the CMNA? code frame (stored in the SYSCST library) in the Code Frame editor.

To create the code frame:

1. Read the prototype into the Code Frame editor and define the substitution parameters.
2. Create the user exits.

   To allow developers to specify additional parameters, local data, or Natural statements, include the following user exits:

User Exit	Description
CHANGE-HISTORY	Generates comment lines indicating the date and ID of the person who created or modified the program. The developer provides a description of changes.
LOCAL-DATA	Defines additional local variables used in the generated program.
START-OF-PROGRAM	Defines code that is executed once at the beginning of the generated program — after all standard initial values are assigned. For example, this user exit code can initialize input values from globals.
BEFORE-INPUT	Defines code that is executed immediately before the INPUT statement is executed (before each input panel is displayed). For example, this user exit code can issue the `SET CONTROL` statements.
AFTER-INPUT	Defines code that is executed immediately after the INPUT statement is executed (after each input panel is displayed).
BEFORE-PROCESSING-MENU-CODES	Defines code that is executed before the menu code is processed.
SPECIAL-CODE- PROCESSING	Defines code that is executed when a menu code does not FETCH a program.
END-OF-PROGRAM	Contains code that is executed once before the program is terminated. For example, this user exit code can assign a termination message.
SET-PF-KEYS	Defines code that is executed before the PF-keys are set and allows non-standard PF-keys to be added to the program. (The additional PF-keys are defined in the CDKEYLDA local data area.)

3 Create the code frame conditions.

To create conditional code, insert the condition name and condition level number in the code frame. To view some examples of conditional code, display the CMNA? code frame in the Code Frame editor and refer to the following condition names:

- GDA-SPECIFIED
- DIRECT-COMMAND-PROCESSING
- MAP-USED

Define the Model

At this point, you can define the model to Natural Construct using the Maintain Models function on the Administration main menu. For more information, see *Define the Model*.

Model subprograms are prefixed by CUMN, where CU identifies the subprogram as a Natural Construct model subprogram and MN identifies the model (Menu).

Note: The CU prefix is used by the models supplied with Natural Construct. When you create a new model or modify a supplied model, use a CX prefix. For this example, we use a CU prefix.

To add the Menu model to Natural Construct:

1 Invoke the Maintain Models function from the Administration main menu.

2 Specify the parameters on the Maintain Models panel.

For example:

```
CSDFM                 N A T U R A L   C O N S T R U C T                 CSDFM0
Oct 09                         Maintain Models                           1 of 1
                                                                              ↩
Action ..................... __ A,B,C,D,M,N,P,R                               ↩

Model ...................... MENU____________________________                 ↩

Description ........ *0200.1__________________________________________________
                     MENU Program                                             ↩

  PDA name ................. CUMNPDA_    Status window ............ Y         ↩

  Programming mode ......... S_          Comment start indicator .. **_       ↩

  Type ..................... P Program   Comment end indicator .... ___       ↩

                                                                              ↩

  Code frame(s) ............ CMNA?___  ________  ________  ________  ________
  Modify server specificatn  CUMNMA__  CUMNMB__  ________  ________  ________
                             ________  ________  ________  ________  ________
  Modify client specificatn  CUMNMA__  CUMNMB__  ________  ________  ________
                             ________  ________  ________  ________  ________
                                                                              ↩

  Clear specification ...... CUMNC___    Post-generation .......... CUMNPS__
  Read specification ....... CUMNR___    Save specification ....... CUMNS___
  Pre-generation ........... CUMNPR__    Document specification ... CUMND___
Command ........... __________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit  frame                                           main ↩
 ↩
```

Most of the model components are listed on this panel. The components that are not listed are assigned through subprograms or code frames. For example, the CUMNMA0 and CUMNMB0 maps are invoked through the CUMNMA and CUMNMB maintenance subprograms, respectively, and the generation subprogram is assigned through the CMNA? code frame.

Step 6: Create the Model PDA

Use the CST-PDA model in the Generation subsystem to create the parameter data area (PDA) for the model (CUMNPDA).

For an example of the parameter data area for the Menu model, refer to the CUMNPDA parameter data area in the SYSCST library.

To create the model PDA:

1 Type the following parameter values on the Generation main menu:

Parameter	Value
Function	M
Module	CUMNPDA
Model	CST-PDA

2 Press Enter.

The Standard Parameters panel for the CST-PDA model is displayed.

3 Enter "Menu" in Model.

For example:

```
 CUPDMA                  CST-PDA Parameter Data Area                  CUPDMA1
 Oct 09                      Standard Parameters                       1 of 1


  Module ............. CUMNPDA_

  Model .............. Menu____________________________ *







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

The Generation main menu is displayed.

4 Enter "G" in Function.

Natural Construct generates the PDA.

5 Enter "E" in Function.

The Natural data area editor is displayed.

- Each substitution parameter in the model code frame corresponds to a user area variable in the model PDA that has the same name and a #PDAX- or #PDA- prefix.
- Each condition variable in the model code frame corresponds to a condition variable in the model PDA that has the same name and a #PDAC- prefix.

6 Specify the type and length of each #PDAX variable.

7 Add any #PDA variables required by the model.

Step 7: Create Translation LDAs and Maintenance Maps

This section describes how to create the translation LDA and maintenance map for your model.

Create the Translation LDAs

To support dynamic translation of text and messages, you can create up to five translation local data areas (LDAs) for each maintenance map; the module that invokes the map must have a translation LDA. Translation LDAs contain the names of the fields on the map that can be translated. To assign the INIT values for these fields, use SYSERR references.

For an example of the translation LDAs for the Menu model, refer to the CU--MAL and CUMNMBL LDAs in the SYSCST library.

The following example shows a translation LDA:

```
Local      CUXXMAL   Library SYSCST                                   DBID  19 FNR  26
Command                                                                            > +
I T L Name                             F Leng Index/Init/EM/Name/Comment
All - -------------------------------- - ---- --------------------------------
  * * **SAG TRANSLATION LDA
  * * * used by map CUXXMX0.
    1 CUTRMAL
    2 TEXT                                    /* Corresponds to syserr message
    3 #GEN-PROGRAM                     A   20 INIT<'*2000.1,.'>
    3 #TITLE                           A   20 INIT<'*2001.1,.'>
    3 #DESCS                           A   20 INIT<'*2001.2,.'>
    3 #DATA-AREA                       A   20 INIT<'*2097.3,.'>
    3 #LANGUAGE                        A   20 INIT<'*1309.2,.'>
  R 2 TEXT
    3 TRANSLATION-TEXT
    4 TEXT-ARRAY                       A    1 (1:100)
    2 ADDITIONAL-PARMS
    3 #MESSAGE-LIBRARY                 A    8 INIT<'CSTLDA'>
    3 #LDA-NAME                        A    8 INIT<'CUXXMAL'>
    3 #TEXT-REQUIRED                   L      INIT<TRUE>
    3 #LENGTH-OVERRIDE                N    10 /* Explicit length to translate   ↩

----------------------------------------------------------------- S 17    L 1   ↩
```

To create your translation LDAs

1. Copy an existing translation LDA.
2. Define the fields for which you want dynamic translation.

   All translation LDAs must have the format shown in the example above. For more information, see *Step 7: Create the Translation LDAs and Maintenance Maps*.

Create the Maintenance Maps

The model uses one or more maintenance maps to accept parameters from a user. To create the maintenance maps, use one of the following methods:

- Copy an existing maintenance map and modify it to suit your requirements.
- Create the map in the Natural Map editor.
- Create the map using the Natural Construct Map model.

For an example of the maintenance maps for the Menu model, refer to the CU--MA0 and CUMN-MB0 maps in the SYSCST library.

The CU--MA0 maintenance map contains the following input fields:

Field	Description
Module	Name of the menu to be generated.
System	Name of the system (usually the library name).
Global data area	Name of the global data area (GDA) used by this menu program. Developers can display a field-level help window to select a value for this field.
With block	Name of the GDA block used by this menu program (if desired).
Title	Title for the menu program. The title identifies the program for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the menu program. The description is inserted in the banner at the beginning of the program and is used internally for program documentation.
First header	First heading displayed on the generated menu.
Second header	Second heading displayed on the generated menu.
Command	Indicates whether the menu supports a Direct Command line.
Message numbers	Indicates whether the menu uses message numbers or message text.
Password	Indicates whether the menu is password protected.

The CUMNMB0 maintenance map contains the following input fields:

Field	Description
Map layout	Name of the map layout (form) used to create the menu panel. Developers can display a field-level help window to select a value for this field.
Code	1 or 2-character code used to invoke the functions listed on the menu. Each code must have a corresponding function.
Functions	Functions listed on the menu. Each function must have a corresponding code. If desired, developers can change the word, Functions, to another value.
Program Name	Name of the program that is invoked when the corresponding function is selected. Developers can display a field-level help window to select a value for this field.
Optional Parameters	Indicates whether additional input parameters are required (user must enter a value) or optional. Developers can specify a maximum of four additional parameters (using PF5). On the menu, the parameters are displayed as column headings to the right of the Function heading and as input fields below the Code field. If additional parameters are specified, Natural Construct generates a legend ® for Required, O for Optional). The legend is aligned under the first occurrence of a Required or Optional indicator.

Step 8: Create the Model Subprograms

After creating the code frame, PDA, maintenance maps, and translation LDAs for the Menu model, you are ready to create the model subprograms. The following sections describe how to create each of the model subprograms:

- Create the Maintenance Subprograms
- Create the Pre-Generation Subprogram
- Create the Post-Generation Subprogram
- Create the Clear Subprogram
- Create the Save Subprogram
- Create the Read Subprogram
- Create the Generation Subprogram
- Create the Documentation Subprogram
- Test the Model Subprograms

Create the Maintenance Subprograms

Use the CST-Modify model in the Generation subsystem to create the maintenance subprograms (CUMNMA and CUMNMB). These subprograms invoke the CUMNMA0 and CUMNMB0 maps, respectively.

For an example of the maintenance subprograms for the Menu model, refer to the CUMNMA and CUMNMB subprograms in the SYSCST library.

To create the CUMNMA maintenance subprogram:

1 Display the Standard Parameters panel for the CST-Modify model.

2 Specify the following parameters:

```
 CUGIMA                      CST-Modify Subprogram                      CUGIMA0
 Oct 09                       Standard Parameters                        1 of 1

  Module ............. CUMNMA__

  Parameter data area  CUMNPDA_ *


  Title .............. Menu Model Modify Subp___

  Description ........ This subprogram is used as modify panel 1____________
                      1 of 2________________________________________________
                      ______________________________________________________
                      ______________________________________________________

  Map name ........... CU--MA0_ *

  Translation LDAs ... CU--MAL_ ________ ________ ________ ________ *

  Cursor translation . X


  First header ....... ______________________________________________________
  Second header ...... *0311.1,+/54_________________________________________

  Subpanel ........... _

  Window support ..... _

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

3 Generate the subprogram.

For information, refer to *Natural Construct Generation*.

To create the CUMNMB maintenance subprogram:

1 Display the Standard Parameters panel for the CST-Modify model.

2 Specify the following parameters:

```
 CUGIMA                      CST-Modify Subprogram                    CUGIMA0
 Oct 09                       Standard Parameters                      1 of 1

  Module ............. CUMNMB__

  Parameter data area  CUMNPDA_ *


  Title .............. Menu Model Modify Subp___

  Description ........ This subprogram is used as modify panel 2_____________
                     2 of 2_______________________________________________________
                     ____________________________________________________________
                     ____________________________________________________________

  Map name ........... CUMNMB0_ *

  Translation LDAs ... CUMNMBL_ ________ ________ ________ ________ *

  Cursor translation . X


  First header ....... ___________________________________________________________
  Second header ...... *0310.1,+/54_______________________________________________

  Subpanel ........... _

  Window support ..... _

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

3 Generate the subprogram.

For information, refer to *Natural Construct Generation*.

Create the Pre-Generation Subprogram

Use the CST-Pregen model in the Generation subsystem to create the pre-generation subprogram.

For an example of the pre-generation subprogram for the Menu model, refer to the CUMNPR subprogram in the SYSCST library.

To create the CUMNPR pre-generation subprogram:

1 Display the Standard Parameters panel for the CST-Pregen model.

2 Specify the following parameters:

```
 CUGPMA                     CST-Pregen Subprogram                    CUG-MA0
 Oct 09                      Standard Parameters                      1 of 1
                                                                           ↩

  Module ............. CUMNPR__                                            ↩

  Parameter data area  CUMNPDA_ *                                          ↩

                                                                           ↩

  Title .............. Menu Model Pregen Subp                              ↩

  Description ........ Pre-generate subprogram.  ...__________________________
                       Set conditions and assign shared PDA variables.     ↩

                      ________________________________________________________
                      ________________________________________________________
                                                                           ↩

                                                                           ↩

                                                                           ↩

                                                                           ↩

                                                                           ↩

                                                                           ↩

                                                                           ↩

                                                                           ↩

                                                                           ↩

                                                                           ↩

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

3 Generate the subprogram.

For information, refer to *Natural Construct Generation*.

Create the Post-Generation Subprogram

Use the CST-Postgen model in the Generation subsystem to create the post-generation subprogram.

For an example of the post-generation subprogram for the Menu model, refer to the CUMNPS subprogram in the SYSCST library.

To create the CUMNPS post-generation subprogram:

1 Display the Standard Parameters panel for the CST-Postgen model.

2 Specify the following parameters:

```
CUGOMA                     CST-Postgen Subprogram                     CUGOMA0
Oct 09                      Standard Parameters                        1 of 1


 Module ............. CUMNPS__

 Model .............. MENU____________________________ *


 Title .............. Menu Model Post-Gen Subp_

 Description ........ Post-generation parameters for the Menu model.________
                     ____________________________________________________________
                     ____________________________________________________________
                     ____________________________________________________________










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

3 Generate the subprogram.

For information, refer to *Natural Construct Generation*.

Create the Clear Subprogram

Use the CST-Clear model in the Generation subsystem to create the clear subprogram. The Menu model requires a clear subprogram because the #PDA-USER-AREA field is redefined into non-alphanumeric variables (for example, #PDA-USER-PARM-LENGTH and #PDA-CODE-LENGTH) and the `Description` field on the first maintenance panel requires default text.

For an example of the clear subprogram for the Menu model, refer to the CUMNC subprogram in the SYSCST library.

To create the CUMNC clear subprogram:

1 Display the Standard Parameters panel for the CST-Clear model.

2 Specify the following parameters:

```
 CUGCMA                       CST-Clear Subprogram                    CUG-MA0
 Oct 09                       Standard Parameters                      1 of 1
                                                                             ↩
  Module ............. CUMNC___                                              ↩
  Parameter data area  CUMNPDA_ *                                            ↩
                                                                             ↩
  Title .............. Menu Model Clear Subp____                             ↩
 Description ........ Clear specification parameters and assign initial value
                      _____________________________________________________
                      _____________________________________________________
                      _____________________________________________________
                                                                             ↩
                                                                             ↩
                                                                             ↩
                                                                             ↩
                                                                             ↩
                                                                             ↩
                                                                             ↩
                                                                             ↩
                                                                             ↩
                                                                             ↩
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit                                            userX main ↩
 ↩
```

3 Generate the subprogram.

For information, refer to *Natural Construct Generation*.

Create the Save Subprogram

Use the CST-Save model in the Generation subsystem to create the save subprogram. The save subprogram allows the model to read a previously-generated program.

For an example of the save subprogram for the Menu model, refer to the CUMNS subprogram in the SYSCST library.

To create the CUMNS save subprogram:

1. Display the Standard Parameters panel for the CST-Save model.
2. Specify the following parameters:

```
CUGAMA                          CST-SAVE Subprogram                       CUG-MA0
Oct 09                          Standard Parameters                        1 of 1

 Module ............. CUMNS___

 Parameter data area  CUMNPDA_ *


 Title .............. Menu Model Save Subp_____

 Description ........ Save specification parameters for the menu model_______
                     ________________________________________________________
                     ________________________________________________________
                     ________________________________________________________











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

3 Generate the subprogram.

For information, refer to *Natural Construct Generation*.

Create the Read Subprogram

Use the CST-Read model in the Generation subsystem to create the read subprogram.

For an example of the read subprogram for the Menu model, refer to the CUMNR subprogram in the SYSCST library.

To create the CUMNR read subprogram:

1. Display the Standard Parameters panel for the CST-Read model.
2. Specify the following parameters:

```
CUGRMA                         CST-Read Subprogram                      CUG-MA0
Oct 09                         Standard Parameters                       1 of 1

 Module ............. CUMNR___

 Parameter data area  CUMNPDA_ *


 Title .............. Menu Model Read Subp_____

 Description ........ Read parameter specifications _______________________
                    ____________________________________________________________
                    ____________________________________________________________
                    ____________________________________________________________











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

3 Generate the subprogram.

For information, refer to *Natural Construct Generation*.

Create the Generation Subprogram

Use the CST-Frame model in the Generation subsystem to create the generation subprogram.

For an example of the generation subprogram for the Menu model, refer to the CUMNGGL subprogram in the SYSCST library.

To create the CUMNGGL generation subprogram:

1. Display the Standard Parameters panel for the CST-Frame model.
2. Specify the following parameters:

```
 CUGFMA                       CST-Frame Subprogram                    CUG-MA0
 Oct 09                       Standard Parameters                      1 of 1

  Module ............. CUMNGGL_

  Parameter data area  CUMNPDA_ *


  Title .............. Menu Model Frame Subp____

  Description ........ Generation parameter variables (if length and format
                      are specified)__________________________________________
                      ________________________________________________________
                      ________________________________________________________










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

3 Generate the subprogram.

For information, refer to *Natural Construct Generation*.

Create the Documentation Subprogram

Use the CST-Document model in the Generation subsystem to create the documentation subprogram.

Note: For an example of the documentation subprogram for the Menu model, refer to the CUMND subprogram in the SYSCST library.

To create the CUMND documentation subprogram:

1. Display the Standard Parameters panel for the CST-Document model.
2. Specify the following parameters:

```
 CUGDMA                     CST-Document Subprogram                     CUGDMA0
 Oct 09                       Standard Parameters                        1 of 2

  Module ............. CUMND___

  Model .............. Menu____________________________ *

  Maps ............... CU--MAO_ CUMNMBO_ ________ ________ ________ *

                       ________ ________ ________ ________ ________ *

  Translation LDAs ... CU--MAL_ CUMNMBL_ ________ ________ ________ *

                       ________ ________ ________ ________ ________ *


  Title .............. Menu Model Document Subp_

  Description ........ Writes Predict documentation for the Menu model____

                      _______________________________________________________
                      _______________________________________________________
                      _______________________________________________________






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

3 Press PF11 (right).

The Additional Parameters panel is displayed.

4 Specify the following parameters:

```
 CUGDMB                    CST-Document Subprogram                     CUGDMB0
 Oct 09                     Additional Parameters                       2 of 2


   Help Text .......... Type ..... 0

                       Major .... Model___________________________

                       Minor .... Menu____________________________


        Description

   1    ______________________________________________________________________
   2    ______________________________________________________________________
   3    ______________________________________________________________________
   4    ______________________________________________________________________
   5    ______________________________________________________________________
   6    ______________________________________________________________________
   7    ______________________________________________________________________
   8    ______________________________________________________________________
   9    ______________________________________________________________________
   10   ______________________________________________________________________




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

5 Generate the subprogram.

For information, refer to *Natural Construct Generation*.

Test the Model Subprograms

Natural Construct supplies a utility to help test the model subprograms.

To invoke the model subprogram test utility:

1 Log onto the SYSCST library.

2 Enter "CSUTEST" at the Next prompt (Direct Command box for Unix).

The Single Module Test Program panel is displayed. For information about this panel, see *Test the Model Subprograms*.

Step 9: Implement the Model

After creating and testing the code frames and model components (data areas, model subprograms, maps, etc.), copy all components to the SYSLIBS library to implement the model.

To implement the model:

1. Invoke the SYSMAIN utility from the Next prompt.
2. Copy all the model components to the SYSLIBS library.

Your new model is now ready for use in the Generation subsystem.

6 CST-Clear Model

This section describes how to use the CST-Clear model to generate the clear subprogram for your model. The clear subprogram resets variables in the model PDA.

Introduction

After defining the model PDA, use the CST-Clear model to generate the clear subprogram for your model. The clear subprogram resets the #PDA-USER-AREA variables in the model PDA. If the #PDA-USER-AREA alphanumeric field is redefined into a non-alphanumeric field that does not contain data according to the specified format, an abnormal termination may occur when it is used. To avoid this, the clear subprogram can reset redefined non-alphanumeric fields. Only non-alphanumeric variables are reset. The clear subprogram can also assign initial default values for user parameters.

The CST-Clear model assumes that your model PDA has the RESET-STRUCTURE group level name. For example:

```
*
*   User defined parameter area
  2 #PDA-USER-AREA                   A  100 (1:40)
R 2 #PDA-USER-AREA                        /* REDEF. BEGIN : #PDA-USER-AREA
  3 RESET-STRUCTURE
*
```

Note: A model PDA generated by the CST-PDA model contains the RESET-STRUCTURE field.

If you do not specify a clear subprogram, the Clear Edit Buffer function on the Generation main menu sets the #PDA-USER-AREA field to blanks. The edit buffer is always cleared, regardless of whether the model uses a clear subprogram.

The nucleus invokes the clear subprogram in the following situations:

- When a user invokes the Clear Edit Buffer function on the Generation main menu.
- When a user changes the model name and the new model uses a different PDA.
- Immediately before the Read Specifications function is invoked on the Generation main menu.

Note: For an example of a generated clear subprogram, refer to CUMNC in the SYSCST library.

Parameters for the CST-Clear Model

Use the CST-Clear model to generate the clear subprogram. This model has one specification panel, Standard Parameters.

Standard Parameters Panel

```
 CUGCMA                        CST-Clear Subprogram                     CUG-MA0
 Aug 17                        Standard Parameters                       1 of 1

  Module name ........ CXMNC___
  Parameter data area  CXMNPDA_ *

  Title .............. Clear ...________________
  Description ........ Clear specification Parameters ...____________________
                       _______________________________________________________
                       _______________________________________________________
                       _______________________________________________________












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

The input fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name of the clear subprogram must be alphanumeric and no more than eight characters in length. Use the following naming convention: `CXxxC` where *xx* uniquely identifies your model.
Parameter data area	Name of the parameter data area (PDA) for your model. Natural Construct determines the name of the PDA based on the Module name specified on the Generation main menu. For example, if you entered CXMNC as the name of the clear subprogram, Natural Construct assumes the name of the PDA is CXMNPDA.

Field	Description
	Use the following naming convention: `CXxxPDA` where *xx* uniquely identifies your model.
Title	Title for the generated subprogram. The title identifies the subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the subprogram. The description is inserted in the banner at the beginning of the subprogram and is used internally for program documentation.

User Exits for the CST-Clear Model

```
CSGSAMPL                      CST-Clear Subprogram                     CSGSM0
Aug 17                             User Exits                           1 of 1

               User Exits              Exists    Sample   Required Conditional
  -------------------------------   --------  ---------- -------- ----------
  _   CHANGE-HISTORY                           Subprogram
  _   PARAMETER-DATA
  _   LOCAL-DATA
  _   PROVIDE-DEFAULT-VALUES                   Subprogram
  _   BEFORE-CHECK-ERROR                        Example
  _   ADDITIONAL-INITIALIZATIONS                Example
  _   END-OF-PROGRAM                                                          ↩
```

For information about these user exits, see ***Supplied User Exits***. For information about using the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

7 CST-Document Model

This section describes the CST-Document model, which is used to create the documentation subprogram for a model. The documentation subprogram writes information about Natural Construct-generated modules to the Predict data dictionary.

Introduction

After defining the generation and sample subprograms, you must generate the documentation subprogram to write information about Natural Construct-generated modules in the Predict data dictionary. This information includes a description of the module, as well as a description of the PF-keys and specification parameters for the module.

Note: Before you can document information about the generated modules, you must define the #PDAX-DESCS(*) field within the model PDA.

Generated using the CST-Document model, this subprogram creates a free-form description of the generated module using the specifications from the model panels. You can write this information in any language for which you have translated help text members.

The documentation subprogram writes the model description to Predict when the developer invokes the Save Specification and Source function or the Stow Specification and Source function on the Generation main menu and presses PF5 (optns). For a description of the Generation main menu, refer to *Generation Main Menu*, *Natural Construct Generation*.

Note: For an example of a generated documentation subprogram, refer to CUMND in the SYSCST library.

Parameters for the CST-Document Model

Use the CST-Document model to generate the documentation subprogram. This model has two specification panels:

- Standard Parameters Panel

- Additional Parameters Panel

Standard Parameters Panel

```
 CUGDMA                     CST-Document Subprogram                    CUGDMA0
 Apr 02                       Standard Parameters                       1 of 2

  Module name ........ CXMND___
  Model name ......... ________________________________ *
  Maps ............... ________ ________ ________ ________ ________ *
                       ________ ________ ________ ________ ________ *
  Translation LDAs ... ________ ________ ________ ________ ________ *
                       ________ ________ ________ ________ ________ *

  Title .............. Document ...____________
  Description ........ Writes Predict documentation for ...___________________
                       _______________________________________________________
                       _______________________________________________________
                       _______________________________________________________





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

The input fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name of the documentation subprogram must be alphanumeric and no more than eight characters in length. Use the following naming convention: `CXxxD` where *xx* uniquely identifies your model.
Model name	Name of the model that uses the documentation subprogram. The model must be defined.
Maps	Names of all maps (specification panels) used by the model. The documentation subprogram retrieves the specification parameters from the specified maps.
Translation LDAs	Names of the translation local data areas (LDAs) for the specified maps. You can specify the names of up to 10 translation LDAs. For information about translation LDAs, see *Step 7: Create the Translation LDAs and Maintenance Maps*.
Title	Title for the generated subprogram. The title identifies the subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.

Field	Description
Description	Brief description of the subprogram. The description is inserted in the banner at the beginning of the subprogram and is used internally for program documentation.

Additional Parameters Panel

```
 CUGDMB                     CST-Document Subprogram                     CUGDMBO
 Apr 02                      Additional Parameters                       2 of 2

   Help Text .......... Type ..... _
                        Major .... ________________________________
                        Minor .... ________________________________

        Description
   1    ______________________________________________________________________
   2    ______________________________________________________________________
   3    ______________________________________________________________________
   4    ______________________________________________________________________
   5    ______________________________________________________________________
   6    ______________________________________________________________________
   7    ______________________________________________________________________
   8    ______________________________________________________________________
   9    ______________________________________________________________________
   10   ______________________________________________________________________



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

On this panel, you do one of the following:

- Specify the Type, Major, and Minor help text components in the applicable fields.

  Natural Construct retrieves the description of all modules generated by the model from the Help Text subsystem.
- Enter a brief description of all modules generated by the model on the lines displayed in the `Description` field.

  The description is written to the Predict data dictionary.

User Exits for the CST-Document Model

```
CSGSAMPL                    Natural Construct                        CSGSM0
Apr 02                   CST-Document User Exits                     1 of 1

            User Exit                 Exists    Sample   Required Conditional
    -------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                             Subprogram
 _  LOCAL-DATA
 _  START-OF-PROGRAM
 _  ADDITIONAL-TRANSLATIONS
 _  ADDITIONAL-INITIALIZATIONS                  Example
 _  DESCRIBE-INPUTS                             Example
 _  PF-KEYS                                    Subprogram
 _  MISCELLANEOUS-VARIABLES                    Subprogram
 _  END-OF-PROGRAM                                                            ↩
```

For information about these user exits, see ***Supplied User Exits***. For information about using the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

8 CST-Frame Model

This section describes the CST-Frame model, which is used to create the sample and generation subprograms for a model. Sample subprograms provide a sample of user exit code and generation subprograms supply code frame parameters.

Sample Subprograms

Sample subprograms are invoked from a user exit and provide a starting sample to help the developer create user exit code. They can be simple or complicated, depending on the model.

Before invoking the sample subprograms, Natural Construct invokes all maintenance subprograms and the pre-generation subprogram. This ensures that the current specification parameters are valid and the conditions are set.

When creating a sample subprogram, you can include additional parameters to give the developer more control over what is generated into the user exit.

Note: To pass additional information to the subprogram, use the CU—PDA.#PDAX-FRAME-PARM variable.

To define a sample subprogram:

1. Type ".E" at the beginning of a user exit line in the Code Frame editor.
2. Press Enter.

   For more information about defining a sample subprogram, see ***Use Parameters Supplied by User Exits***.

Generation Subprograms

Generation subprograms are invoked from a code frame and supply code frame parameters. Because the lengths and contents of some parameters change based on user-supplied input values or information in Predict, these parameters must be supplied by the generation subprograms. The subprograms write statements to the Natural edit buffer, based on the user-supplied input parameters or other calculated values.

To write to the edit buffer, include a `DEFINE PRINTER(SRC=1) OUTPUT 'SOURCE'` statement in the subprogram that routes the output to the source work area. To allow models to be ported to multiple platforms, use the CU--DFPR copycode member to define the SRC printer.

All WRITE, DISPLAY, and PRINT statement output for your print file is written to the edit buffer. Use the NOTITLE option on each of these statements. If a DISPLAY statement is used in the subprogram, also use the NOHDR option. When trailing blanks should be suppressed in variable names, the PRINT statement can be a useful alternative to the WRITE statement. However, you may want to increase the line length of the edit buffer when using the PRINT statement, so variable names are not split at the hyphen (-).

Because generation logic can be highly complex, these subprograms allow ultimate flexibility. However, they are less maintainable than code frame statements because you must change Natural programs to modify the generated code.

Generation subprograms can also accept the #PDA-FRAME-PARM constant code frame parameter from the CU—PDA common parameter data area. This parameter allows a subprogram to be invoked several times within the generation process. Each time the generation subprogram is invoked, it can use the value of this parameter to determine what to generate.

To invoke a generation subprogram:

1. Specify line type "N" at the > prompt in the Code Frame editor.
2. Optionally, specify the constant parameter value at this prompt.

References

- For more information about generation subprograms, see ***Parameters Supplied by Generation Subprograms***.
- For an example of a generated generation subprogram, refer to CUMNGGL in the SYSCST library.

Parameters for the CST-Frame Model

Use the CST-Frame model to create the generation or sample subprogram. This model has one specification panel, Standard Parameters.

Standard Parameters Panel

```
 CUGFMA                       CST-Frame Subprogram                      CUG-MA0
 Mar 30                       Standard Parameters                        1 of 1

  Module name ........ CXMNGGL_
  Parameter data area  CXMNPDA_ *

  Title .............. Frame ...________________
  Description ........ This generation/sample subprogram ..___________________
                       _______________________________________________________
                       _______________________________________________________
                       _______________________________________________________

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

The input fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name of the subprogram must be alphanumeric and no more than eight characters in length. Use the following naming conventions: ■ `CXxxGyyy` where *xx* uniquely identifies your model and *yyy* identifies your generation subprogram ■ `CXxxSyyy` where *xx* uniquely identifies your model and *yyy* identifies your sample subprogram
Parameter data area	Name of the parameter data area (PDA) for your model. Natural Construct determines the PDA name based on the Module name specified on the Generation main menu. For example, if you enter "CXMNGAAA", Natural Construct assumes the PDA name is CXMNPDA. Use the following naming convention: `CXxxPDA`

Field	Description
	where *xx* uniquely identifies your model.
Title	Title for the generated subprogram. The title identifies the subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the subprogram. The description is inserted in the banner at the beginning of the subprogram and is used internally for program documentation.

User Exits for the CST-Frame Model

```
CSGSAMPL                     CST-Frame Subprogram                     CSGSM0
Mar 30                            User Exits                          1 of 1

               User Exits             Exists    Sample   Required Conditional
    -------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                             Subprogram
 _  PARAMETER-DATA
 _  LOCAL-DATA
 _  START-OF-PROGRAM
 _  GENERATE-CODE
 _  BEFORE-CHECK-ERROR                          Example
 _  ADDITIONAL-INITIALIZATIONS                  Example
 _  END-OF-PROGRAM                                                            ↩
```

For information about these user exits, see ***Supplied User Exits***. For information about using the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

9 CST-Modify and CST-Modify-332 Models

This section describes the CST-Modify and CST-Modify-332 models, which are used to create the modify (maintenance) subprograms for a model.

- CST-Modify generates specification panels that support dynamic translation.
- CST-Modify-332 generates specification panels that do not support dynamic translation; it is supplied for those who want to continue using maintenance subprograms that were generated using previous versions of Natural Construct.

Introduction

After defining the model PDA and creating the clear, read, and save subprograms; maintenance maps; and translation LDAs, you must create one or more maintenance subprograms to collect user-supplied specification parameters (#PDAX variables), perform validation checks, and set the condition codes and #PDA variables (optional).

Maintenance subprograms are executed in the same order as they appear on the Maintain Models panel. Usually, there is one maintenance subprogram for every left/right (horizontal) maintenance panel. Data edits should only be applied if the developer presses Enter or PF11 (right). Either the maintenance subprogram or the maintenance map can validate the parameters.

You should only trap PF-keys that perform specialized functions related to the panel. If you want the PF-key settings to be dependent on the default settings specified on the control record, the subprogram should not contain hardcoded PF-keys (check the PF-key values using the variables specified in CU—PDA).

The CST-Modify and CST-Modify-332 models are described in the following sections. We recommend using the CST-Modify model to create new maintenance subprograms.

Note: A maintenance subprogram can test the value of CU—PDA.#PDA-PHASE to identify the phase during which it was invoked (G for generation, M for modification, L for translation, U for sample user exits, etc.).

Example of a Maintenance Subprogram

The following example shows the first 40 lines of the CUMNMA maintenance subprogram:

```
**SAG GENERATOR: CST-MODIFY                      VERSION: 8.2.1
**SAG TITLE: Menu Model Modify Subp
**SAG SYSTEM: NATURAL-CONSTRUCT
**SAG DATA-AREA: CUMNPDA
**SAG MAP: CU--MA0
**SAG DESCS(1): This subprogram is used as modify panel 1
**SAG DESCS(2): 1 of 2
**SAG HEADER2: *0311.1,+/54
**SAG TRANSLATION-LDA(1): CU--MAL
**SAG DYNAMIC-TRANSLATION: X
```

```
****************************************************************
* Program  : CUMNMA
* System   : NATURAL-CONSTRUCT
* Title    : Menu Model Modify Subp
* Generated: Oct 21,13 at 05:33 PM by REGEN81
* Function : This subprogram is used as modify panel 1
*            1 of 2
*
*
* History
****************************************************************
DEFINE DATA
  PARAMETER USING CUMNPDA       /* Model specific data
  PARAMETER USING CU--PDA       /* Standard model parameters
  PARAMETER USING CSASTD        /* Standard message passing area
  LOCAL USING CNAMSG            /* Message retrieval passing area
  LOCAL USING CSLRCODE          /* Message return codes
  LOCAL USING CSAMARK           /* Field mark information
  LOCAL USING CSLPHASE          /* Valid generation phases
  LOCAL USING CSLSTD            /* Local message passing area
  LOCAL USING CSACURS           /* Used by CSUCURS to translate prompts
  LOCAL USING CU--MAL           /* Translation LDA
  LOCAL
    01 #PROGRAM (A8)
    01 LOCAL-TRANSLATION
      02 TEXT
        03 #HEADER2 (A54)
            INIT<'*0311.1,+/54'>
      02 REDEFINE TEXT
        03 TRANSLATION-TEXT
....
```

For an example of a maintenance subprogram subpanel generated by the CST-Modify model, refer to CUMNMBA in SYSCST.

CST-Modify Model

The CST-Modify model generates maintenance subprograms that support dynamic translation and multiple languages. To implement dynamic translation, you must also create a maintenance map and one or more translation local data areas (LDAs) for each maintenance subprogram.

The CST-Modify model generates either a main maintenance subprogram panel (defined on the Maintain Models panel) or a maintenance subprogram subpanel (invoked from the main maintenance subprogram panel using a PF-key). To reduce the amount of information on a panel, we recommend grouping similar parameters, such as windowing information, and moving that information to a subpanel.

If desired, you can use a subroutine to display a subpanel. Subroutines typically control processes that do not require a panel or subpanel to be displayed. For example, a subroutine can enable backward or forward scrolling or test a function that does not require mandatory edits for generation. Both subprograms and subroutines are invoked by PF-keys from the main maintenance subprogram panel.

All maintenance subprograms require a VALIDATE-INPUT subroutine to process mandatory edits. At generation time, the edits for the maintenance subprogram subpanel are processed first, then the edits for the main maintenance subprogram panel are processed. Therefore, any subroutine edits should also be included in the VALIDATE-INPUT subroutine.

Tip: To avoid confusion about the order of execution of the panel and subpanel subroutines, place edit checks in programs rather than in subroutines.

The CST-Modify model also allows you to override the headers and PF-keys defined on the Subprogram record.

This section covers the following topics:

Parameters for the CST-Modify Model

Use the CST-Modify model to generate a maintenance subprogram that supports dynamic translation. This model has one specification panel, Standard Parameters.

Standard Parameters Panel

```
 CUGIMA                      CST-Modify Subprogram                    CUGIMA0
 Oct 09                       Standard Parameters                      1 of 1

  Module name ........ CXMNMA__
  Parameter data area  CXMNPDA_ *

  Title ............. Modify ...______________
  Description ........ Modify server specificatn Parameters ...______________
                       ______________________________________________________
                       ______________________________________________________
                       ______________________________________________________

  Map name ........... ________ *
  Translation LDAs ... ________ ________ ________ ________ ________ *
  Cursor translation . _

  First header ....... ______________________________________________________
  Second header ...... ______________________________________________________

  Subpanel ........... _
  Window Support ..... _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        windw pfkey                   left  userX main   ↩
```

Use this panel to define standard parameters, such as the map and translation LDAs used with the maintenance subprogram and whether cursor translation is supported on the generated panel or subpanel. You can also use this panel to override the first and second headings or specify subpanel and window support.

Using PF-keys on this panel, you can change the default window settings (PF5 windw) or override the PF-key settings (PF6 pfkey).

The input fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name of the maintenance subprogram must be alphanumeric and no more than eight characters in length. Use the following naming conventions: ■ Panel: CX*xx*M*y* where *xx* uniquely identifies your model and *y* is a letter from A–J that identifies the maintenance panel (A for the first maintenance panel, B for the second, etc.) ■ Subpanel: CX*xx*M*yz*

Field	Description
	where *xx* uniquely identifies your model, *y* is a letter from A–J that identifies the maintenance panel (A for the first maintenance panel, B for the second, etc.), and *z* is a letter from A–J that identifies the subpanel.
Parameter data area	Name of the parameter data area (PDA) for your model. Natural Construct determines the PDA name based on the Module name specified on the Generation main menu. For example, if you enter "CXMNMA", Natural Construct assumes the PDA name is CXMNPDA. Use the following naming convention: `CXxxPDA` where *xx* uniquely identifies your model.
Title	Title for the generated subprogram. The title identifies the subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the subprogram. The description is inserted in the banner at the beginning of the subprogram and is used internally for program documentation.

Map name	Name of the map used for the maintenance subprogram. Natural Construct determines the name of the map based on the Module name specified on the Generation main menu. For example, if you enter CXMNMA as the subprogram name, Natural Construct assumes the map name is CXMNMA0.
	
	The specified map must exist in the current library and the map name should correspond to the maintenance subprogram name, with the addition of a zero. The zero indicates that the map has no hard-coded text and is used for dynamic translation. For example:
	
	`Program            Map`
	`CXMNMA      CXMNMA0`
	`CXMNMB       CXMNMB0`

Translation LDAs	Names of the translation local data areas (LDAs) for the maintenance subprogram. You can specify the names of up to five translation LDAs. The specified translation LDAs must exist. The LDA name should correspond to the maintenance subprogram name, with the addition of an "L". For example:
	
	`Program          Translation LDA`
	`CXMNMA      CXMNMAL`
	`CXMNMB       CXMNMBL`

Cursor translation	Indicates whether users can modify the text on this panel while in translation mode. To support cursor translation, mark this field.
First header	First heading displayed on the generated subprogram panel or the SYSERR number(s) that supplies the heading. By default, this header is automatically populated with the description specified on the model record. To override this default, specify the new header in this field.

Field	Description
	To specify the positioning of the heading, use special syntax after the text or SYSERR numbers. By default, the header is displayed at the left margin. To center *First Heading* across 50 bytes for example, type: `First Heading,+/50` The text before `,+/` indicates the heading displayed. The number after `,+/` indicates the number of bytes within which the heading is centered. For information about SYSERR message numbers, see ***Use SYSERR References*** or refer to the SYSERR utility in the Natural Utilities documentation. **Note:** Data substitution within SYSERR references is not supported in this context.
Second header	Second heading displayed on the generated panel or the SYSERR number(s) that supplies the heading. By default, this header is populated with the description specified on the subprogram record, if it exists. Unlike the model record, which populates the first header field, the subprogram record only exists if you create it. To supply a second header (if no subprogram record exists) or to override the default, specify a new header in this field. **Note:** We recommend using this field to define the second heading, instead of the description on the Maintain Subprograms panel. The Natural Construct nucleus does not reference the Subprogram record for supplied models, so the description used to populate the second header will not exist unless you create it. To specify the heading position, use special syntax after the text or SYSERR number. By default, the header is displayed at the left margin. To center *Second Heading* across 50 bytes for example, type: `Second Heading,+/50` The text before `,+/` indicates the heading displayed. The number after `,+/` indicates the number of bytes within which the heading is centered. For information about SYSERR message numbers, see ***Use SYSERR References*** or refer to the SYSERR utility in the Natural Utilities documentation.
Subpanel	Indicates whether the generated subprogram is created as a subpanel that is invoked from a main panel (such as a help selection window). To create the subprogram as a subpanel, mark this field. By default, the Natural Construct nucleus controls the help, retrn, quit, left, right, and main PF-keys (defined on the control record) for a main panel, and the help, retrn, quit, and main PF-keys for a subpanel. To define the processing for additional keys (the left and right keys, for example) on a subpanel, press PF6 (pfkey) on the Standard Parameters panel. For more information, see ***Define Non-Standard PF-Keys***.

Field	Description
Window support	Indicates whether the generated subprogram is displayed in a window. To display the generated subprogram in a window, mark this field. By default, the PF-keys and messages are displayed within the generated window, and a frame (border) is displayed around the generated window. To change the default window settings, press PF5 (windw) on the Standard Parameters panel. For more information, see *Change the Default Window Settings*.

Define Non-Standard PF-Keys

To define the processing for non-standard PF-keys:

1 Press PF6 (pfkey) on the Standard Parameters panel.

The PF-Key Parameters window is displayed. For example:

```
 CUGIMAA                     Natural Construct                     CUGIMAA0
 Oct 09                      PF-key Parameters                       1 of 1

        Subprogram           Subroutine                NAMED
        ---------- -------------------------------- ----------
   PF5  ________  ________________________________ __________
   PF6  ________  ________________________________ __________
   PF9  ________  ________________________________ __________

   PF4  ________  ________________________________ __________ test

   PF7  ________  ________________________________ __________ bkwrd
   PF8  ________  ________________________________ __________ frwrd

   PF10 ________  ________________________________ __________ left
   PF11 ________  ________________________________ __________ right
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

By default, the Natural Construct nucleus controls the help, retrn, quit, left, right, and main PF-keys for a main panel (defined on the control record), and the help, retrn, quit, and main PF-keys for a subpanel. Using this window, you can override the nucleus-controlled PF-keys displayed on a subpanel.

Note: The left and right PF-keys are available only if the maintenance subprogram is a subpanel.

2 Define the processing and name for the non-standard PF-key.

Note: You can also change the processing and/or name for a non-standard PF-key currently defined in the window.

Use the following input fields to define the non-standard PF-key:

Field	Description
Subprogram	Name of the subprogram executed when the corresponding PF-key is pressed. This subprogram is invoked during generation to process the VALIDATE-INPUT subroutine.
Subroutine	Name of the subroutine executed when the corresponding PF-key is pressed.
NAMED	Name of the PF-key (either text or a valid SYSERR message number). If this field is blank, the default key names are used. For information about SYSERR message numbers, see ***Use SYSERR References*** or refer to the SYSERR utility in the Natural Utilities documentation.

3 Press Enter.

User Exits for the CST-Modify Model

```
CSGSAMPL                    CST-Modify Subprogram                    CSGSMO
Oct 09                            User Exits                          1 of 1

               User Exits              Exists    Sample   Required Conditional
    ------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                              Subprogram
 _  PARAMETER-DATA
 _  LOCAL-DATA
 _  START-OF-PROGRAM
 _  BEFORE-CHECK-ERROR                           Example
 _  BEFORE-STANDARD-KEY-CHECK                    Example
 _  ADDITIONAL-TRANSLATIONS
 _  ADDITIONAL-INITIALIZATIONS                   Example
 _  BEFORE-INPUT
 _  INPUT-SCREEN                                 Example              X
 _  AFTER-INPUT
 _  BEFORE-INVOKE-SUBPANELS                                           X
 _  AFTER-INVOKE-SUBPANELS                                            X
 _  BEFORE-REINPUT-MESSAGE
 _  VALIDATE-DATA                               Subprogram
 _  MISCELLANEOUS-SUBROUTINES                    Example
 _  END-OF-PROGRAM                               Example
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
frwrd help  retrn quit                        bkwrd frwrd                      ↩
```

For information about these user exits, see ***Supplied User Exits***. For information about using the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

CST-Modify-332 Model

Use the CST-Modify-332 model to generate a maintenance subprogram that does not support dynamic translation. This model is provided for those who want to continue using maintenance subprograms that were generated under previous versions of Natural Construct.

This section covers the following topics:

Parameters for the CST-Modify-332 Model

Use the CST-Modify-332 model to generate the maintenance subprogram. This model has one specification panel, Standard Parameters.

Standard Parameters Panel

```
 CUGMMA                     CST-Modify-332 Subprogram                      CUGMMA0
 Oct 09                        Standard Parameters                          1 of 1

  Module name ........ CXMNMA__
  Parameter data area  CXMNPDA_ *
  Map name ........... CXMNMA1_ *

  Title .............. _________________________
  Description ........ Maintenance for specification parameters.______________
                       _______________________________________________________
                       _______________________________________________________
                       _______________________________________________________












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

The input fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name of the maintenance subprogram must be alphanumeric and no more than eight characters in length. Use the following naming convention: `CXxxMy` where *xx* uniquely identifies your model and *y* is a letter from A–J that identifies the maintenance panel (A for the first maintenance panel, B for the second, etc.).
Parameter data area	Name of the parameter data area (PDA) for your model. Natural Construct determines the PDA name based on the Module name specified on the Generation main menu. For example, if you enter "CXMNMA", Natural Construct assumes the PDA name is CXMNPDA. Use the following naming convention: `CXxxPDA` where *xx* uniquely identifies your model.

Map name	Name of the map used for the maintenance subprogram. Natural Construct determines the name of the map based on the Module name specified on the Generation main menu. For example, if you enter CXMNMA as the subprogram name, Natural Construct assumes the map name is CXMNMA1 (for English). The map must exist in the current library, and the map name should correspond to the maintenance subprogram name, with the addition of the language code. For example:
	`Program            Map`
	`CXMNMA      CXMNMA1`

Title	Title for the generated subprogram. The title identifies the subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the subprogram. The description is inserted in the banner at the beginning of the subprogram and is used internally for program documentation.

User Exits for the CST-Modify-332 Model

```
CSGSAMPL                  CST-Modify-332 Subprogram                    CSGSMO
Oct 09                            User Exits                           1 of 1

               User Exits             Exists    Sample   Required Conditional
    -------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                             Subprogram
 _  LOCAL-DATA
 _  START-OF-PROGRAM
 _  AFTER-INPUT                                 Example
 _  PROCESS-SPECIAL-KEYS                       Subprogram              X
 _  VALIDATE-DATA                              Subprogram                     ↩
```

For information about these user exits, see ***Supplied User Exits***. For information about using the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

10 CST-Panel Model

This section describes the CST-Panel model, which is used to create the client modify (maintenance) subprogram for a model. The model-generated specification panels run as part of a common wizard in the Construct Program Generation plug-in for Natural for Windows.

Introduction

To enable a Natural Construct model for the Construct Program Generation plug-in, you must first determine how many panels will be used for the wizard and what information will be presented on each panel. In most cases, the panels will follow a one-to-one relationship with the corresponding modify server panels generated by the CST-Modify model.

The CST-Panel model generates one column of GUI controls per panel. Each column contains a label and control for editing a model specification panel. For greater flexibility, you can include additional controls within user exits. Alternately, you can create your own wizard instead of using the supplied common wizard. (In this case, you will not use the CST-Panel model.)

Note: The CST-Panel model is only available in the Construct Program Generation plug-in for Natural for Windows. This model is not available in the Natural Construct character interface (NCSTG).

Parameters for the CST-Panel Model

Use the CST-Panel model to generate the client modify (maintenance) subprogram for your model.

After specifying the required parameters on one panel, select Next to proceed to the next panel. To generate the module, select Finish on the last specification panel.

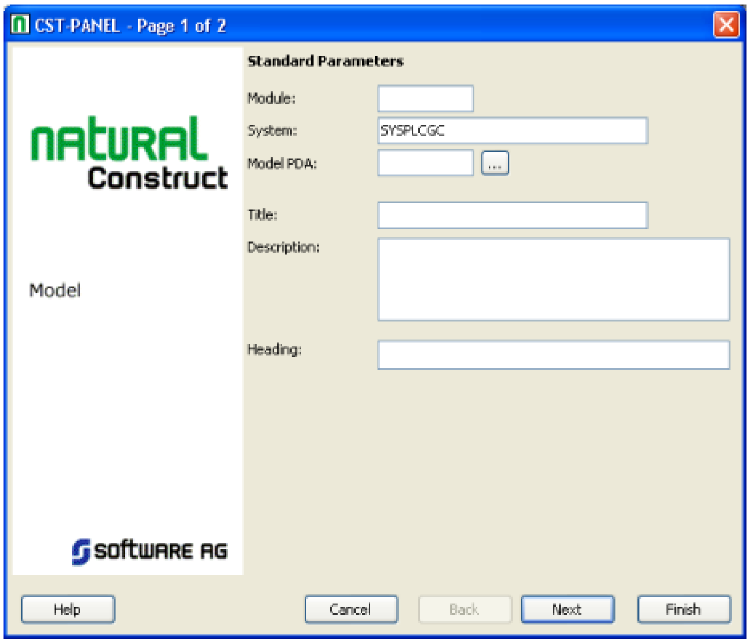
The CST-Panel model has two specification panels:

- Standard Parameters Panel

- Additional Parameters Panel

Standard Parameters Panel

The following example shows the Standard Parameters panel for CST-Panel:



Use this panel to define standard parameters for your wizard, such as the name of the generated subprogram and the heading displayed at the top of the wizard panel.

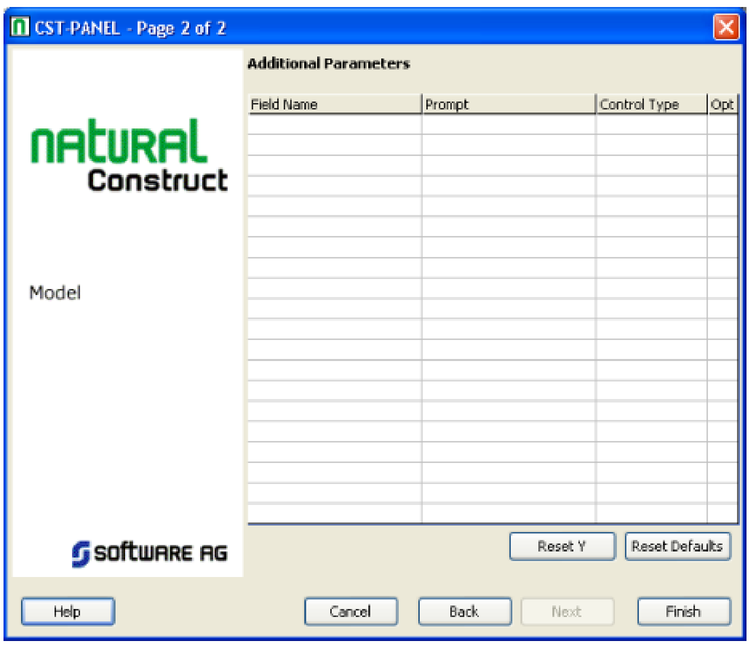
The input fields on this panel are:

Field	Description
Module	Name of the Natural subprogram (module) to generate. Module names use the following naming convention: `WCNxxMy` where *xx* is the unique identifier for your model and *y* is letter from A–J that identifies the maintenance panel (A for the first panel, B for the second, etc.). For more information, see *Naming Conventions for Model Components*.
System	Name of the system (by default, the name of the current library). 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 subprogram.)

Field	Description
Model PDA	Name of the parameter data area (PDA) for your model.
Title	Title for the generated subprogram. The title identifies the subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the subprogram. The description is inserted in the banner at the beginning of the subprogram and is used internally for program documentation.
Heading	Heading displayed at the top of the generated wizard panel. For example, **Standard Parameters**.

Additional Parameters Panel

The following example shows **Additional Parameters** for CST-Panel:



Use this panel to position the GUI controls for your wizard panel in a single column. Each row in the grid above represents one field in the column and one label for each field.

The input fields on this panel are:

Field	Description
Field Name	Name of a #PDA or #PDAX variable field in your model PDA or the name of a field in CU--PDA. To select the field from a list of fields in the model PDA, select **....** **Note:** To use an array field, specify the index value, for example #PDAX-VIEW(5), or select an EDITAREA control.
Prompt	Text used as a label for the field. A TEXTCONSTANT Natural GUI control is generated for each control. The string is based on the value entered in this field. Only the TOGGLEBUTTON (or checkbox) control does not have an extra control generated, as this control includes its own label.
Control Type	Type of Natural GUI control to use. Select one of the following: INPUTFIELD TOGGLEBUTTON EDITAREA SELECTIONBOX
Opt	To modify the size and location attributes for GUI controls, click the … button. The Optional Parameters window is displayed. For a description of this window, see *Reset Default Size and Location Variables*.

Reset Default Rectangle Y Values

You can reset and reassign the default Y-coordinate values for GUI controls. Use this option when you have customized the default Y value and want to restore the default.

To reset and reassign the default Y-Coordinate values:

- Select **Reset Y**.

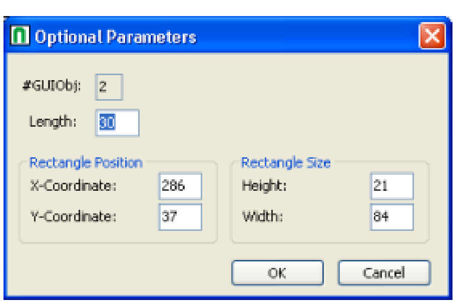
Reset Default Size and Location Variables

You can customize the size and location-related attributes for GUI controls.

To customize the size and location-related attributes:

1 Select **Reset Defaults**.

The **Optional Parameters** window is displayed. For example:



2 Change the current size and location-related attributes.

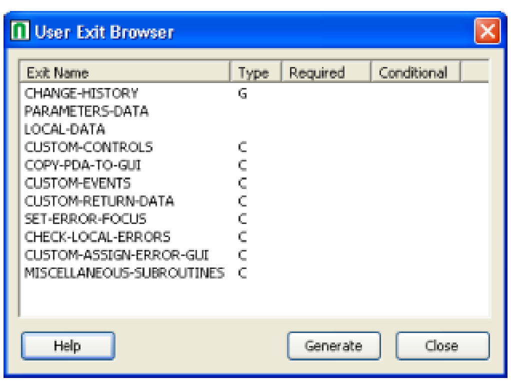
The input fields in this window are:

Field	Description
#GUIObj	Identifier for the current GUI object.
Length	Maximum number of characters that can be entered as input for the specified GUI control. This field applies to EDITAREA and INPUTFIELD controls only.
X-Coordinate	X coordinate value for the GUI control.
Y-Coordinate	Y coordinate value for the GUI control. **Note:** The generated TEXTCONTSTANT control is derived from the X-Coordinate and Y-Coordinate values. These values identify the location of the GUI control in pixels from the top, left corner of the configurable area of the panel.
Height	Height of the GUI control in pixels.
Width	Width of the GUI control in pixels.

3 Select **OK**.

User Exits for the CST-Panel Model

The following example shows the User Exit Browser panel for the CST-Panel model:



This section describes the model-specific user exits on this panel. For information about the common user exits on this panel, see ***Supplied User Exits***. For information about using the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

Note: For more information about the model-specific user exits, refer to the examples and notes in the sample code generated for each exit.

The model-specific user exits for CST-Panel are:

- CHECK-LOCAL-ERRORS
- COPY-PDA-TO-GUI
- CUSTOM-ASSIGN-ERROR-GUI
- CUSTOM-CONTROLS
- CUSTOM-EVENTS
- CUSTOM-RETURN-DATA
- SET-ERROR-FOCUS

CHECK-LOCAL-ERRORS

Use this exit to check for local validation errors on wizard panels. Local errors are detected when the user selects **Finish**, but before the model's validation subprogram is called. If a local validation error occurs, the error is displayed to the user and **Next** or **Finish** becomes inactive.

Note: Local validations do not occur when the user selects **Previous**.

COPY-PDA-TO-GUI

Use this exit to copy additional fields from the model PDA to GUI controls.

CUSTOM-ASSIGN-ERROR-GUI

Use this exit in combination with the *SET-ERROR-FOCUS* exit to set focus to custom GUI controls when a validation error is detected.

CUSTOM-CONTROLS

Use this exit to add additional GUI controls to your wizard panel. For each new control, note the number of the last GUIOBJ (GUI object) used by the last generated control and increment by 1.

CUSTOM-EVENTS

Use this exit to handle events for custom GUI controls on the wizard panel. For example, you can use this exit to add a button to the wizard panel and write code to respond to a click event. When a user selects the button, a window is invoked to display additional lookup data.

CUSTOM-RETURN-DATA

Use this exit to copy any additional GUI control values to the model PDA or CU--PDA data areas.

SET-ERROR-FOCUS

Use this exit in combination with the ***CUSTOM-ASSIGN-ERROR-GUI*** exit to set focus to custom GUI controls when a validation error is detected.

11 CST-PDA Model

All models require three external parameter data areas (PDAs): the model PDA, CU—PDA, and CSASTD. CU—PDA and CSASTD are supplied with Natural Construct. The model PDA is user-created and contains variables and conditions specific to the model. This section describes how to use the CST-PDA model to generate the model PDA.

Introduction

All models require the following external parameter data areas (PDAs):

PDA	Description
Model PDA	User-defined; contains variables and conditions specific to a model. **Note:** If you are creating a model that generates modules to run on a Natural Construct client, you must also generate a stream subprogram to convert the contents of the model PDA into a format that can be transmitted between the client and the server. For information, see ***CST-Stream Model***.
CU—PDA	Supplied with Natural Construct.
CSASTD	Supplied with Natural Construct.

The model PDA passes information between the Natural Construct nucleus and the model and generation subprograms. Before generating your model PDA, create the code frames and define your model. Natural Construct uses information in the model code frames to generate the model PDA, such as:

- substitution parameters
- condition codes

The CST-PDA model builds the model PDA by scanning the model code frames for substitution parameters and condition codes. Substitution parameters are character strings that begin with an ampersand (&) and end with a special character such as a period (.), parentheses, or an asterisk (*), but not a hyphen (-).

For each substitution parameter, the model generates a field (prefixed by #PDAX) within the re-definition of the #PDA-USER-AREA field in the model PDA. The model assigns the default format and length for alphanumeric fields (A10), which you can change as required.

For each condition code, the model generates a logical field (prefixed by #PDAC) within the re-definition of the #PDA-CONDITION-CODES field in the model PDA.

References

- For information about isolating the parameters for your model PDA, see ***Step 4: Isolate the Parameters in the Prototype***.

- For information about creating code frames and defining models, see ***Step 5: Create Code Frame(s) and Define the Model***.
- For more information about creating the model PDA, see ***Step 6: Create the Model PDA***.
- For an example of a generated model PDA, refer to CUMNPDA in the SYSCST library.
- For more information about substitution parameters, refer to the Natural documentation.

Parameters for the CST-PDA Model

Use the CST-PDA model to create the model PDA. This model has one specification panel, Standard Parameters.

Standard Parameters Panel

```
 CUPDMA                  CST-PDA Parameter Data Area                    CUPDMA1
 Feb 04                      Standard Parameters                         1 of 1

  Module name ........ CXMNPDA_
  Model name ......... ________________________________ *















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

The input fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name of the model PDA must be alphanumeric and no more than eight characters in length. Use the following naming convention: `CXxxPDA` where *xx* uniquely identifies your model.
Model name	Name of the model that uses the model PDA. **Note:** Ensure that the specified model and its corresponding code frames have been defined on the Maintain Models panel.

After specifying the required parameters and generating the model PDA, edit the generated code and assign the correct format and length for each field. All substitution parameters are generated with a default format and length of A10. You can also add any new parameters your model PDA may require.

12 CST-Postgen Model

This section describes the CST-Postgen model, which is used to create the pre-generation subprogram for a model. The post-generation subprogram supplies values for the substitution parameters in the code frames, which is the final stage of the generation process.

Introduction

After defining the pre-generation subprogram, use the CST-Postgen model to generate the post-generation subprogram. This subprogram supplies values for substitution parameters in the code frames (identified by &). It is invoked as the final stage of the generation process when the application developer enters "G" in the Function field on the Generation main menu.

The post-generation subprogram substitutes the code frame parameters with the corresponding substitution values by stacking the substitution parameters and their corresponding values. Use the `STACK TOP DATA FORMATTED` statement to stack these values. Natural Construct performs the corresponding substitutions in the edit buffer and produces the final version of the generated program.

During the generation process, code lines specified in the code frame are written to the edit buffer, as well as the output of the generation subprogram contained in the code frame. Any substitution parameters are included in the edit buffer exactly as they appear in the code frame.

Note: For an example of a generated post-generation subprogram, refer to CUMNPS in the SYSCST library.

Parameters for the CST-Postgen Model

Use the CST-Postgen model to create the post-generation subprogram. This model has one specification panel, Standard Parameters.

Standard Parameters Panel

```
 CUGOMA                     CST-Postgen Subprogram                      CUGOMA0
 May 26                       Standard Parameters                        1 of 1


  Module name ........ CXMNPS__
  Model name ......... ________________________________ *

  Title .............. Post-gen subprogram
  Description ........ Post-generation subprogram. Stack post generation______
                       changes._______________________________________________
                       _______________________________________________________
                       _______________________________________________________











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

The input fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name must be alphanumeric and no more than eight characters in length. Use the following naming convention: `CXxxPS` where *xx* uniquely identifies your model.
Model name	Name of the model that uses the post-generation subprogram. **Note:** Ensure that the specified model and its corresponding code frames have been defined on the Maintain Models panel.
Title	Title for the generated subprogram. The title identifies the subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the subprogram. The description is inserted in the banner at the beginning of the subprogram and is used internally for program documentation.

User Exits for the CST-Postgen Model

```
CSGSAMPL                   CST-Postgen Subprogram                    CSGSMO
May 26                           User Exits                          1 of 1

               User Exits              Exists    Sample   Required Conditional
    -------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                            Subprogram
 _  PARAMETER-DATA
 _  LOCAL-DATA                                Subprogram
 _  START-OF-PROGRAM                           Example
 _  ADDITIONAL-SUBSTITUTION-VALUES            Subprogram
 _  BEFORE-CHECK-ERROR                         Example
 _  ADDITIONAL-INITIALIZATIONS                 Example
 _  END-OF-PROGRAM                                                              ↩
```

For information about these user exits, see ***Supplied User Exits***. For information about using the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

13 CST-Pregen Model

This section describes the CST-Pregen model, which is used to create the pre-generation subprogram for a model. The pre-generation subprogram is invoked:

- During the generation phase after all maintenance subprograms have been executed
- Whenever the SAMPLE command is issued from the User Exit editor

Introduction

After generating the maintenance subprograms, generate the pre-generation subprogram to assign #PDAC condition values based on user-supplied parameters or other calculated values. The pre-generation subprogram also assigns the values of #PDA variables in the model PDA that are required by any subsequent generation subprograms.

Generated using the CST-Pregen model, this subprogram is invoked after all maintenance subprograms are executed during the generation phase or when the SAMPLE command is issued from the User Exit editor. It is the first user subprogram invoked.

Note: All #PDAC-prefixed condition values are reset before generation begins.

The pre-generation subprogram should also calculate the values of any #PDA variables required by subsequent generation subprograms.

For simple models that do not have code frames, this subprogram can also perform the functions of a generation subprogram. (Condition code values and derived fields can also be assigned within the maintenance subprograms.)

Note: For an example of a generated pre-generation subprogram, refer to CUMNPR in the SYSCST library.

Parameters for the CST-Pregen Model

Use the CST-Pregen model to create the pre-generation subprogram. This model has one specification panel, Standard Parameters.

Standard Parameters Panel

```
 CUGPMA                        CST-Pregen Subprogram                    CUG-MA0
 May 26                         Standard Parameters                      1 of 1

  Module name ........ CXMNPR_
  Parameter data area  CXMNPDA_ *

  Title .............. Pre-generation subprogram
  Description ........ Pre-generate subprogram.______________________________
                       Set conditions and assign shared PDA variables.________
                       _______________________________________________________
                       _______________________________________________________











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

The input fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name must be alphanumeric and no more than eight characters in length. Use the following naming convention: `CXxxPR` where *xx* uniquely identifies your model.
Parameter data area	Name of the parameter data area (PDA) for your model. Natural Construct determines the PDA name based on the Module name specified on the Generation main menu. For example, if you enter "CXMNPR", Natural Construct assumes the PDA name is CXMNPDA. Use the following naming convention: `CXxxPDA` where *xx* uniquely identifies your model.
Title	Title for the generated subprogram. The title identifies the subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the subprogram. The description is inserted in the banner at the beginning of the subprogram and is used internally for program documentation.

User Exits for the CST-Pregen Model

```
CSGSAMPL                   CST-Pregen Subprogram                      CSGSMO
May 26                           User Exits                           1 of 1

             User Exits             Exists    Sample   Required Conditional
    -------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                           Subprogram
 _  PARAMETER-DATA
 _  LOCAL-DATA                                Example
 _  ASSIGN-DERIVED-VALUES                    Subprogram
 _  SET-CONDITION-CODES                      Subprogram    X          X
 _  GENERATE-CODE
 _  BEFORE-CHECK-ERROR                        Example
 _  ADDITIONAL-INITIALIZATIONS                Example
 _  END-OF-PROGRAM                                                           ↩
```

For information about these user exits, see ***Supplied User Exits***. For information about using the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

14 CST-Proxy Model

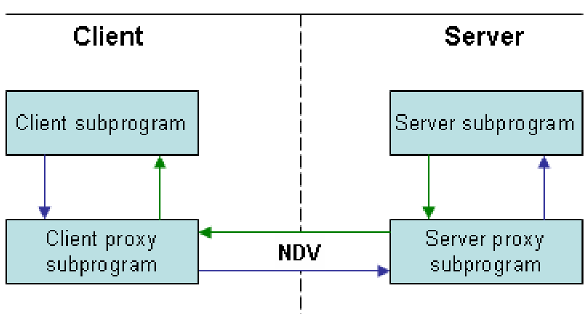
This section describes the CST-Proxy model, which is used to generate a client or server proxy to remotely access a subprogram on the server.

Introduction

The CST-Proxy model generates either a client or server proxy to access a subprogram on the server. The proxy acts as a bridge between a subprogram on the client and a subprogram on the server. When a request to the server is initiated from the client (for example, when a user requests active help for a field on a panel), the following process occurs:

1. The client subprogram issues a request, which invokes the client proxy subprogram.
2. The client proxy subprogram converts the data into the network transfer format and identifies the name of the server proxy to be invoked.
3. The data is sent to the server via NDV (Natural Development Server).
4. The server proxy subprogram converts the data to Natural data format and invokes the server subprogram.
5. The server subprogram completes the request.
6. The server proxy subprogram converts the data into network transfer format.
7. The data is sent to the client via NDV.
8. The client proxy subprogram converts the data to Natural data format and returns the information to the client subprogram.

The following diagram illustrates this process. Blue arrows indicate data sent to the server subprogram; green arrows indicate data returned to the client subprogram:



Parameters for the CST-Proxy Model

Use the CST-Proxy model to generate either a client or server proxy to access a subprogram on the server. This model has one specification panel, Standard Parameters.

Standard Parameters Panel

```
 CUGXMA                       CST-PROXY Subprogram                    CUGXMA0
 Apr 02                       Standard Parameters                      1 of 1

  Module ............. MYPROXY_
  System ............. CNDPRO__________________________

  Title .............. Proxy for...______________
  Description ........ Description of proxy that...______________________________
                       _______________________________________________________________
                       _______________________________________________________________
                       _______________________________________________________________

  Subprogram ......... ________ *
  Gen client proxy ... _
  Server proxy subp .. ________ *






Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit        1:V                                userX main   ↩
```

The input fields on the Standard Parameters panel are:

Field	Description
Module	Name of the proxy subprogram you are creating (by default, the name specified in the Module name field on the Generation main menu). The name must follow standard Natural naming conventions, must be alphanumeric, and cannot be more than eight characters in length.
System	Name of the system (by default, the name of the current library). The system name must be alphanumeric, no more than 32 characters in length, and does not have to be associated with a Natural library ID. (The combination of module and system names is used as a key to access help information.)
Title	Title for the proxy subprogram. The title identifies the subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.

Field	Description
Description	Brief description of the proxy subprogram. The description is inserted in the banner at the beginning of the subprogram and is used internally for program documentation.
Subprogram	Name of the subprogram for which you are generating the proxy.
Gen client proxy	Indicates whether the generated proxy is a client proxy or a server proxy. By default, a server proxy is generated. To generate a client proxy, mark this field and specify the name of the corresponding server proxy in Server proxy subp. **Note:** The specified server proxy subprogram must exist before you can generate the client proxy subprogram; generate the server proxy first.
Server proxy subp	Name of the server proxy subprogram.

Specify the Number of Occurrences Returned

If the proxy handles 1:V arrays, specify the maximum number of 1:V arrays that can be returned to the client for each request. A 1:V array can consist of either one-dimensional data, such as a list of repeating values, or two-dimensional data, such as a row of record data.

To specify the maximum number of occurrences to return for each request:

1 Press PF5 (1:V) on the Standard Parameters panel.

The 1:V Overrides window is displayed. For example:

```
                          1:V Overrides
    01 >>

     1 Structure .........
       Field .............
       Occurrences ........ _____ / _____ / _____

     2 Structure .........
       Field .............
       Occurrences ........ _____ / _____ / _____

     3 Structure .........
       Field .............
       Occurrences ........ _____ / _____ / _____

     4 Structure .........
       Field .............
       Occurrences ........ _____ / _____ / _____
  Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---P
        help  retrn             retrv       bkwrd  frwrd         ↩
```

Note: If no fields in the target subprogram use the 1:V notation, a message is displayed. Otherwise, the model determines these values and displays their names.

2 Specify the maximum number of occurrences that can be returned to the client with each call to the server.

   Press PF5 (retrv) to update the information from the server.

3 Press PF2 (retrn) to return to the Standard Parameters panel.

User Exits for the CST-Proxy Model

```
CSGSAMPL                    NATURAL CONSTRUCT                        CSGSM0
 Apr 02                          User Exits                             1 of 1

               User Exit              Exists    Sample   Required Conditional
     ------------------------------ -------- ---------- -------- ------------
  _  CHANGE-HISTORY                          Subprogram
  _  LOCAL-DATA                               Example
  _  ON-ERROR-MSG-NR                          Example                X
  _  START-OF-PROGRAM                         Example
  _  BEFORE-CALL-SERVER                       Example                X
  _  AFTER-CALL-SERVER                        Example                X
  _  BEFORE-CALL-OBJECT                       Example                X
  _  AFTER-CALL-OBJECT                        Example                X
  _  SET-DATA-LENGTH                          Example
  _  SET-RETURN-BLOCKS                        Example
  _  BEFORE-COMPRESS-OUTPUT                   Example
  _  AFTER-COMPRESS-OUTPUT                    Example
  _  BEFORE-EXPAND-INPUT                      Example
  _  AFTER-EXPAND-INPUT                       Example
  _  MISCELLANEOUS-SUBROUTINES                Example
  _  END-OF-PROGRAM                           Example
 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
       help  retrn quit                    bkwrd frwrd                        ↩
```

For information about these user exits, see ***Supplied User Exits***. For information about using the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

15 CST-Read Model

This section describes the CST-Read model, which is used to create the read subprogram for a model. The read subprogram reads the specifications for the model.

Introduction

After defining the model PDA and clear subprogram, you must create a subprogram to read the specifications from a previously-generated module. The generated subprogram has one INPUT statement for each #PDAX variable in the model PDA.

A read subprogram generated by the CST-Read model contains a series of INPUT statements that accept the data previously placed in the Natural stack. The read subprogram is invoked when the developer invokes the Read Specifications function on the Generation main menu.

Before the read subprogram is invoked, all **SAG parameter values are placed on the Natural stack. The read subprogram repeats a series of INPUT statements to accept the stacked parameters and assign them to the correct PDA variables. This subprogram must correspond to the save subprogram that writes the **SAG parameter lines. The read subprogram can also read common parameters from a different model.

Notes:

1. Natural Construct invokes the clear subprogram before invoking the read subprogram. It is not necessary to save null parameter values.
2. For an example of a generated read subprogram, refer to CUMNR in the SYSCST library.

Parameters for the CST-Read Model

Use the CST-Read model to create the read subprogram. This model has one specification panel, Standard Parameters.

Standard Parameters Panel

```
 CUGRMA                        CST-Read Subprogram                       CUG-MA1
 Nov 28                        Standard Parameters                        1 of 1

  Module name ........ CXMNR___
  Parameter data area  CXMNPDA_ *

  Title .............. _________________________
  Description ........ Read parameter specification.__________________________
                       _______________________________________________________
                       _______________________________________________________
                       _______________________________________________________










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

The input fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name of the read subprogram must be alphanumeric and no more than eight characters in length. Use the following naming convention: `CXxxR` where *xx* uniquely identifies your model.
Parameter data area	Name of the parameter data area (PDA) for your model. Natural Construct determines the name of the PDA based on the Module name specified on the Generation main menu. For example, if you enter "CXMNR", Natural Construct assumes the PDA name is CXMNPDA. Use the following naming convention: `CXxxPDA` where *xx* uniquely identifies your model.
Title	Title for the generated subprogram. The title identifies the subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.

Field	Description
Description	Brief description of the subprogram. The description is inserted in the banner at the beginning of the subprogram and is used internally for program documentation.

User Exits for the CST-Read Model

```
 CSGSAMPL                    CST-Read Subprogram                        CSGSM0
 Nov 28                           User Exits                            1 of 1

               User Exits             Exists    Sample    Required Conditional
     -------------------------------- -------- ---------- -------- ------------
  _  CHANGE-HISTORY                            Subprogram
  _  PARAMETER-DATA
  _  LOCAL-DATA                                 Example
  _  INPUT-ADDITIONAL-PARAMETERS               Subprogram
  _  BEFORE-CHECK-ERROR                         Example
  _  ADDITIONAL-INITIALIZATIONS
  _  END-OF-PROGRAM
```

For information about these user exits, see ***Supplied User Exits***. For information about using the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

16 CST-Save Model

This section describes the CST-Save model, which is used to generate the save subprogram for a model. The save subprogram writes the specification parameters to the source buffer.

Introduction

To read an existing program, your model must have both a save and a read subprogram. The save subprogram must contain a separate WRITE statement for each specification parameter (#PDAX variable). Use the equal sign (=) notation to include the variable contents with the name of the variables. For example:

```
WRITE(SRC) NOTITLE '=' #PDAX-variable-name
```

Note: Use a separate WRITE statement for each component of an array.

For an example of a save subprogram, refer to CUMNS in the SYSCST library.

Parameters for the CST-Save Model

Use the CST-Save model to create the save subprogram. This model has one specification panel, Standard Parameters.

Standard Parameters Panel

```
 CUGAMA                          CST-Save Subprogram                        CUG-MA1
 Feb 27                          Standard Parameters                         1 of 1

  Module name ........ CXMNS___
  Parameter data area  CXMNPDA_ *

  Title .............. Save ...__________________
  Description ........ Save parameter specification ...________________________
                       __________________________________________________________
                       __________________________________________________________
                       __________________________________________________________











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

The input fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name of the save subprogram must be alphanumeric and no more than eight characters in length. Use the following naming convention: `CXxxS` where *xx* uniquely identifies your model.
Parameter data area	Name of the parameter data area (PDA) for your model. Natural Construct determines the name of the PDA based on the Module name specified on the Generation main menu. For example, if you enter "CXMNS", Natural Construct assumes the PDA name is CXMNPDA. Use the following naming convention: `CXxxPDA` where *xx* uniquely identifies your model.
Title	Title for the generated subprogram. The title identifies the subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.

Field	Description
Description	Brief description of the subprogram. The description is inserted in the banner at the beginning of the subprogram and is used internally for program documentation.

User Exits for the CST-Save Model

```
CSGSAMPL                     CST-Save Subprogram                       CSGSM0
Feb 27                            User Exits                           1 of 1

              User Exits             Exists    Sample   Required Conditional
    -------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                             Subprogram
 _  PARAMETER-DATA
 _  LOCAL-DATA                                  Example
 _  START-OF-PROGRAM
 X  SAVE-PARAMETERS                            Subprogram    X
 _  BEFORE-CHECK-ERROR                          Example
 _  ADDITIONAL-INITIALIZATIONS                  Example
 _  END-OF-PROGRAM                                                            ↩
```

For information about these user exits, see ***Supplied User Exits***. For information about using the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

17 CST-Shell Model

Introduction

The CST-Shell model generates a template for a model subprogram; it is similar to the supplied Shell model. The main differences between the models are that the CST-Shell model:

- Supports regeneration
- Supports messaging

The CST-Shell model creates the `DEFINE DATA ... END-DEFINE` framework containing definitions for the global data area (GDA), parameter data areas (PDAs), local data areas (LDAs), or views specified on the Standard Parameters panel, as well as the required REPEAT loops and messaging subroutines. You can use this time-saving model to generate startup modules for your model subprograms.

References

- For an example of a generated shell program, refer to CUMPSLFV in the SYSCST library.
- For information about the Shell model, refer to *Shell Model*, *Natural Construct Generation*.

Parameters for the CST-Shell Model

Use the CST-Shell model to create the shell subprogram. This model has one specification panel, Standard Parameters.

Standard Parameters Panel

```
CUGSMA                        CST-Shell Program                          CUGSMA0
Jul 05                       Standard Parameters                          1 of 1

  Module name ........ CXMPSLFV
  Module type ........ _
  System name ........ NCSTDEMO________________________ *

  Title .............. CST module ...___________
  Description ........ This CST module is used for ...________________________
                       _______________________________________________________
  Messaging support .. _
  Global data area ... ________ *
  Parameter data area  ________ ________ ________ ________ ________ *
  Local data area .... ________ ________ ________ ________ ________ *
                       ________ ________ ________ ________ ________ *

  Views ........  1    ________________________________ *
                  2    ________________________________ *
                  3    ________________________________ *
                  4    ________________________________ *
                  5    ________________________________ *
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
main  help  retrn quit                                         userX main   ↩
```

The input fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name of the shell program must be alphanumeric and no more than eight characters in length.
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)
System name	Name of the system (by default, the name of the current library). The system name must be alphanumeric, no more than 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.)
Title	Title for the generated subprogram. The title identifies the subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.

Field	Description
Description	Brief description of the subprogram. The description is inserted in the banner at the beginning of the subprogram and is used internally for program documentation.
Messaging support	Indicates whether the shell program supports the dynamic translation of messages. To support dynamic translation, mark this field.
Global data area	Name of the global data area used by the generated module.
Parameter data area	Names of up to five inline parameter data areas used by the generated module. **Note:** If the Module type is P or S, you cannot specify parameter data.
Local data area	Names of up to 10 inline or external local data areas used by the generated module.
Views	Names of up to five Predict views used by the generated module.

User Exits for the CST-Shell Model

```
CSGSAMPL                      CST-Shell Program                         CSGSM0
Jul 05                           User Exits                             1 of 1

               User Exits              Exists    Sample   Required Conditional
    ------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                             Subprogram
 _  PARAMETER-DATA
 _  LOCAL-DATA                                  Example
 _  START-OF-PROGRAM
 _  GENERATE-CODE
 _  BEFORE-CHECK-ERROR                          Example
 _  ADDITIONAL-INITIALIZATIONS                  Example
 _  END-OF-PROGRAM                                                              ↩
```

For information about these user exits, see ***Supplied User Exits***. For information about using the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

18 CST-Stream Model

This section describes the CST-Stream model, which is used to create the stream subprogram for a model. The stream subprogram converts the contents of a model PDA between internal and streamed format.

Introduction

When deploying a GUI front-end for a module on a Natural Construct client, Natural Construct must be able to translate the specification data passed to the server from the client. To do this, the model requires a stream subprogram to convert the contents of the model PDA into a format that can be transmitted between the client and the server.

If your model generates modules for a Natural Construct client, generate the model PDA and then use the CST-Stream model to generate the stream subprogram. For more information about generating the model PDA, see ***CST-PDA Model***.

Parameters for the CST-Stream Model

Use the CST-Stream model to create the stream subprogram. This model has one specification panel, Standard Parameters.

Standard Parameters Panel

```
 CUGTMA                       CST-Stream Subprogram                      CUGTMA0
 Jul 05                        Standard Parameters                        1 of 1
  Module ............ ________
  System ............ C821____________________________


  Title ............. Stream Subprogram .._____
  Description ....... This Stream Subprogram will convert Models:___________
                      (...model name...)_____________________________________
                      PDA between internal and streamed formats.____________
                      ______________________________________________________


  Model PDA ......... ________ *
  Generate trace code  _






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

The input fields on the Standard Parameters panel are:

Field	Description
Module	Name specified on the Generation main menu. The name of the stream subprogram must be alphanumeric and no more than eight characters in length. Use the following naming convention: `CXxxT` where *xx* uniquely identifies your model.
System	Name of the system (by default, the name of the current library). 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 subprogram.)
Title	Title for the generated subprogram. The title identifies the subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the subprogram. The description is inserted in the banner at the beginning of the subprogram and is used internally for program documentation.
Model PDA	Name of the PDA used by the model for which you are generating the stream subprogram.

Field	Description
Generate trace code	Indicates whether extra code is generated into the stream subprogram to help trace inconsistencies between data sent by the client and data expected by the server. To generate trace code, mark this field.

User Exits for the CST-Stream Model

```
CSGSAMPL                     Natural Construct                        CSGSMO
Jul 05                          User Exits                            1 of 1
               User Exit               Exists    Sample   Required Conditional
    ------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                            Subprogram
 _  LOCAL-DATA
 _  ADDITIONAL-INITIALIZATIONS                 Example
 _  END-OF-PROGRAM
```

For information about these user exits, see ***Supplied User Exits***. For information about using the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

19 CST-Validate Model

This section describes the CST-Validate model, which is used to create the validation subprogram for a model. The validation subprogram verifies inputs for the model during the generation process.

Introduction

If you code validations within the maintenance panel modules, it is difficult to invoke the validations from batch programs or GUI clients. Instead, you can consolidate all model validation within a validation subprogram. To confirm input values for your model, use the CST-Validate model to generate a validation subprogram and then add the subprogram to the model record on the Maintain Models panel.

The following example shows how to use a validation subprogram to validate inputs for a maintenance panel:

```
**SAG DEFINE EXIT VALIDATE-DATA
  ASSIGN CSAVAL.VALIDATE-SPECIFIC-FIELD(1) = 'field1'
  ASSIGN CSAVAL.VALIDATE-SPECIFIC-FIELD(2) = 'field2'
  ASSIGN CSAVAL.VALIDATE-SPECIFIC-FIELD(3) = 'field3'
  CALLNAT  'CUBOVAL'  CSAVAL
                      CUBOPDA    /*your model PDA name
                      CU—PDA
                      CSAMARK
                      CSAERR
                      CSASTD
  PERFORM  REINPUT-MESSAGE
*
**SAG  END-EXIT
```

Parameters for the CST-Validate Model

Use the CST-Validate model to create the validation subprogram. This model has one specification panel, Standard Parameters.

Standard Parameters Panel

```
CUVAMA                    CST-Validate Subprogram                    CUVAMA0
Sep 07                      Standard Parameters                       1 of 1

  Module ............ ________
  System ............ NCSTDEMO________________________

  Title ............. Validate Subprogram ..___
  Description ....... This Validation Subprogram will validate Inputs________
                      for the model: ....________________________________
                      _______________________________________________________
                      _______________________________________________________


  Model PDA ......... ________ *









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

The input fields on the Standard Parameters panel are:

Field	Description
Module	Name specified on the Generation main menu. The name of the validation subprogram must be alphanumeric and no more than eight characters in length. Use the following naming convention: `CXxxVAL` where *xx* uniquely identifies your model.
System	Name of the system (by default, the name of the current library). 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 subprogram.)
Title	Title for the generated subprogram. The title identifies the subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the subprogram. The description is inserted in the banner at the beginning of the subprogram and is used internally for program documentation.
Model PDA	Name of the PDA used by the model for which you are generating the validation subprogram.

User Exits for the CST-Validate Model

```
CSGSAMPL                     Natural Construct                        CSGSMO
Sep 07                           User Exits                           1 of 1
               User Exit               Exists    Sample   Required Conditional
    -------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                              Subprogram
 _  LOCAL-DATA
 _  GENERATE-VALIDATIONS
 _  GENERATE-SUBROUTINES                        Subprogram
```

For information about these user exits, see ***Supplied User Exits***. For information about using the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

Code Validations

The CST-Validate model codes validations as subroutines in the GENERATE-SUBROUTINES user exit. For each #PDAX-FIELD-NAME field you want to validate, create a subroutine called V-*`field-name`* to perform the validations. Whenever a validation error is found, the V-*`field-name`* subroutine must:

- Assign `CSASTD.RETURN-CODE = 'E'`
- Assign the error message in CSASTD.MSG
- Perform an ESCAPE-ROUTINE to bypass subsequent checks

Notes:

1. To retrieve SYSERR messages, use the CU--VERR copycode.
2. For more information about coding validations, see ***GENERATE-SUBROUTINES***.

Validate Array Fields

For array fields, the V-*`field-name`* subroutine validates all occurrences for which validation is requested. These occurrences are supplied in the #INDEX.#FROM (1:3) fields (redefined into #I1, #I2 and #I3). To return multiple errors (for separate field occurrences), perform the CHECK-AFTER-EDIT subroutine when an error occurs within an array field. This will add the error to the error list but allow editing of subsequent indexes to occur.

The following example shows the validation routine for a two-dimensional array called #PDAX-PHYSICAL-KEY:

```
**********************************************************************
DEFINE SUBROUTINE V_PHYSICAL-KEY
**********************************************************************
*
  FOR #INDEX.#OCC(1) = #INDEX.#FROM(1) TO #INDEX.#THRU(1)
    FOR #INDEX.#OCC(2) = #INDEX.#FROM(2) TO #INDEX.#THRU(2)
      /*
      /* Validate #PDAX-PHYSICAL-KEY(#I1,#I2)
      ASSIGN CPAEL.FILE-NAME = CUBOPDA.#PDAX-PRIME-FILE
      ASSIGN CPAEL.FILE-CODE = CUBOPDA.#PDAX-PHYSICAL-KEY(#I1,#I2)
      ASSIGN CPAEL.DDM-PREFIX = CPAFI.DDM-PREFIX
      CALLNAT 'CPUEL' CPAEL CSASTD
      IF NOT CPAEL.#FIELD-FOUND
        ASSIGN CNAMSG.MSG-DATA(1) = CPAEL.FIELD-NAME
        ASSIGN CNAMSG.MSG-DATA(3) = CPAEL.FILE-NAME
        INCLUDE CU--VER2 '0096'
            ''':1::2:not in:3:'''
            'CUBOPDA.#PDAX-PHYSICAL-KEY(#I1,#I2)'
      END-IF
    END-FOR
  END-FOR
END-SUBROUTINE /* V_PHYSICAL-KEY
```

Tips

- If you do not want to exit the current subroutine, as with array processing, use the CU--VERZ copycode instead of CU--VERR.
- To return a warning message, rather than an error, use the CU--VWAR copycode.

20 User Exits for the Administration Models

This section describes the user exits supplied for the Natural Construct administration models. The administration models generate the model subprograms used by all models.

What are User Exits?

User exits insert customized or specialized processing into a model subprogram, which is preserved when the module is regenerated. Natural Construct provides a wide variety of user exits for the administration models. The exits vary depending on the type of subprogram generated. Some exits contain sample code or subroutines, while others generate the DEFINE EXIT…END-EXIT lines only — and you provide the code.

You can modify any user exit code generated into the edit buffer. If multiple user exits are generated with the same name, Natural Construct merges them into a single exit.

User exits are provided for the following administration models:

- CST-Clear
- CST-Read
- CST-Save
- CST-Modify and CST-Modify-332
- CST-Pregen
- CST-Postgen
- CST-Frame
- CST-Document
- CST-Validate
- CST-Stream
- CST-Shell

Reuse User Exit Code

If you specify a new model name on the Generation main menu (M function) and the source buffer contains code, you can retain the code and use it with the model you are creating. This functionality saves time and effort when creating models that use the same code.

If the source buffer contains code when you specify a new model name, the following window is displayed:

```
                          Natural Construct           CSGNEW0
                          CLEAR Source Area


           Mark if you wish to clear the source area          _ ↩
```

To retain the code in the source buffer for use with the new module:

- Press Enter.

  The first specification panel for the new model is displayed and Natural Construct retains the user exit code for use with the new module.

To clear the code in the source buffer (and not save it for the new module):

1. Select `Mark if you wish to clear the source area`.
2. Press Enter.

   The source buffer is cleared and the first specification panel for the specified model is displayed.

Invoke the User Exit Editor

You can invoke the User Exit editor from the Generation main menu or from the last specification panel for a model that supports user exits.

- To invoke the User Exit editor from the Generation main menu, refer to *User Exit Editor*, *Natural Construct Generation*.
- To invoke the User Exit editor from the model specification panels, press PF11 (userX) on the last specification panel for a model that supports user exits.

If user exits are defined for the specified module, the existing user exit code is displayed in the User Exit editor. If no exits are defined, a list of the exits available for that model is displayed.

Tip: To select additional exits, enter "SAMPLE" at the > prompt.

Note: The SAMPLE command is performed automatically when you invoke the User Exit editor and no user exits are defined for the specified module.

The User Exits panel is similar for all models. The following example shows the User Exits panel for the CST-Clear model:

```
CSGSAMPL                     CST-Clear Subprogram                      CSGSM0
Aug 17                            User Exits                           1 of 1

               User Exits              Exists    Sample   Required Conditional
 -------------------------------    --------  ---------- -------- ----------
 _   CHANGE-HISTORY                           Subprogram
 _   PARAMETER-DATA
 _   LOCAL-DATA
 _   PROVIDE-DEFAULT-VALUES                   Subprogram
 _   BEFORE-CHECK-ERROR                        Example
 _   ADDITIONAL-INITIALIZATIONS                Example
 _   END-OF-PROGRAM                                                          ↩
```

The fields on this panel are:

Field	Description
User Exits	Names of the user exits available for this model. If a user exit is required and not conditional (its existence is not based on condition codes in the code frames), it is marked by default.
Exists	Indicates whether the corresponding user exit is defined. ■ If the exit exists, this field is marked. ■ If the exit does not exist, this field is blank.
Sample	Indicates the contents of the user exit. ■ If the exit is empty (contains DEFINE EXIT … END-EXIT lines), this field is blank. ■ If the exit contains a subprogram, "Subprogram" is displayed. ■ If the exit contains sample code, "Example" is displayed.
Required	Indicates whether the user exit is required. ■ If the exit must be specified, "X" is displayed. ■ If the exit is optional, this field is blank.
Conditional	Indicates whether the user exit is conditional (its existence is based on condition codes in the code frames). ■ If the exit is conditional, "X" is displayed. ■ If the exit is optional, this field is blank.

To select a user exit displayed on the User Exits panel:

1 Type "X" in the input field to the left of each user exit you want to use.

2 Press Enter.

The selected user exits are displayed in the User Exit editor.

Note: Fully qualify all references to database fields with the file name.

Tip: You can also define user exits in the User Exit editor without using the SAMPLE command.

Define User Exits

The code specified within a user exit depends on the type of module generated and the user exit used. However, all Natural Construct user exits have the following format:

```
0010 DEFINE EXIT user-exit-name
0020   user exit code
0030 END-EXIT user-exit-name
```

Note: Do not insert comments or Natural code on the DEFINE EXIT and END-EXIT lines.

Supplied User Exits

This section describes the user exits supplied for the Natural Construct administration models. The user exits are listed in alphabetical order. For many exits, one or more examples are included.

The supplied user exits are:

- ADDITIONAL-INITIALIZATIONS
- ADDITIONAL-SUBSTITUTION-VALUES
- ADDITIONAL-TRANSLATIONS
- AFTER-INPUT
- AFTER-INVOKE-SUBPANELS
- ASSIGN-DERIVED-VALUES
- BEFORE-CHECK-ERROR
- BEFORE-INPUT
- BEFORE-INVOKE-SUBPANELS
- BEFORE-REINPUT-MESSAGE
- BEFORE-STANDARD-KEY-CHECK
- CHANGE-HISTORY
- DESCRIBE-INPUTS
- END-OF-PROGRAM
- GENERATE-CODE
- GENERATE-SUBROUTINES
- GENERATE-VALIDATIONS
- INPUT-ADDITIONAL-PARAMETERS
- INPUT-SCREEN
- LOCAL-DATA
- MISCELLANEOUS-SUBROUTINES

ADDITIONAL-INITIALIZATIONS

This user exit generates the framework for any additional initializations performed in the INITIALIZATIONS subroutine.

Example of Code

```
** 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-SUBSTITUTION-VALUES

This user exit is used in combination with the LOCAL-DATA user exit. It generates STACK statements for code frame parameters that do not have a corresponding variable in the model PDA.

Example of Code

```
DEFINE EXIT ADDITIONAL-SUBSTITUTION-VALUES
*
* Substitution for frame parameters not defined in model PDA
STACK TOP DATA FORMATTED '&CENTERED-HEADER1'
                          #CENTERED-HEADER1
STACK TOP DATA FORMATTED '&CENTERED-HEADER2'
                          #CENTERED-HEADER2
STACK TOP DATA FORMATTED '&DATE-EM'
                          #DATE-EM
STACK TOP DATA FORMATTED '&EOD-TABT'
                          #EOD-TABT
STACK TOP DATA FORMATTED '&EXPORT-DELIMITER'
                          #EXPORT-DELIMITER
```

```
STACK TOP DATA FORMATTED '&GT-LT'
                           #GT-LT
STACK TOP DATA FORMATTED '&HEAD1-LEN'
                           #HEAD1-LEN
STACK TOP DATA FORMATTED '&HEAD2-LEN'
                           #HEAD2-LEN
STACK TOP DATA FORMATTED '&INPUT-LINES'
                           #INPUT-LINES
STACK TOP DATA FORMATTED '&KEY-PREFIX'
                           #KEY-PREFIX
STACK TOP DATA FORMATTED '&LT-GT'
                           #LT-GT
STACK TOP DATA FORMATTED '&PARM-NAT-FORMAT'
                           #PARM-NAT-FORMAT
STACK TOP DATA FORMATTED '&PREFIX-NAT-FORMAT'
                           #PREFIX-NAT-FORMAT
STACK TOP DATA FORMATTED '&SEL-TBL-SIZE'
                           #SEL-TBL-SIZE
STACK TOP DATA FORMATTED '&TIME-EM'
                           #TIME-EM
STACK TOP DATA FORMATTED '&UQ'
                           #UQ
STACK TOP DATA FORMATTED '&UQ-FOUND'
                           #UQ-FOUND
STACK TOP DATA FORMATTED '&VALUE-UQ'
                           #VALUE-UQ
STACK TOP DATA FORMATTED '&VAR-UQ'
                           #VAR-UQ
STACK TOP DATA FORMATTED '&VIEW-LDA'
                           #VIEW-LDA
STACK TOP DATA FORMATTED '&WINDOW-WIDTH'
                           #WINDOW-WIDTH
STACK TOP DATA FORMATTED '&WITH-BLOCK'
                           #WITH-BLOCK
END-EXIT ADDITIONAL-SUBSTITUTION-VALUES
```

ADDITIONAL-TRANSLATIONS

This user exit generates the framework for additional translations performed in the GET-PROMPT-TEXT subroutine.

Example of Code

```
3070 **SAG DEFINE EXIT ADDITIONAL-TRANSLATIONS
3080 *
3090  IF #FIRST-TRANSLATION OR CU--PDA.#PDA-PHASE = CSLPHASE.#TRANSLATE
3100     THEN
3110     PERFORM SET-MODIFY-HEADER3
3120     /*
3130     /* Set completed message
3140     RESET CNAMSG.INPUT-OUTPUTS
```

```
    ASSIGN CNAMSG.MSG-DATA(1) = #PDA-FRAME-PARM
    ASSIGN CNAMSG.MSG = CUBASRPL.#RETURN-MESSAGE
    PERFORM GET-MESSAGE-TEXT
    ASSIGN CUBASRPL.#RETURN-MESSAGE = CNAMSG.MSG
    RESET CNAMSG.INPUT-OUTPUTS
    /*
    /* Assign available keys
    ASSIGN CU--PDA.#PDA-AVAILABLE1-NAME = #AVAILABLE1-NAME
    ASSIGN CU--PDA.#PDA-AVAILABLE2-NAME = #AVAILABLE2-NAME
    ASSIGN CU--PDA.#PDA-AVAILABLE3-NAME = #AVAILABLE3-NAME
    RESET #FIRST-TRANSLATION
    /*
    /* Override pfkey settings
    RESET #LOCAL-PFKEYS-REQUIRED
    /*
    /* Set all PF-keys named off
    INCLUDE CU--SOFF
    /*
    /* Set Help and Return keys
    SET KEY DYNAMIC CU--PDA.#PDA-PF-HELP = HELP
            NAMED CU--PDA.#PDA-HELP-NAME
    SET KEY DYNAMIC CU--PDA.#PDA-PF-RETURN
            NAMED CU--PDA.#PDA-RETURN-NAME
    END-IF
    **SAG END-EXIT
```

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
- Add non-standard PF-key processing to a module

For example, when you add a non-standard PF-key, you can 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. If you do not include the PERFORM NEW-SCREEN code and continue with execution after processing this exit, an Invalid PF-key message is displayed.

Example of Code

```
**SAG DEFINE EXIT AFTER-INPUT
  IF #FORMAT-HELP
    RESET #FORMAT-HELP
    ASSIGN  CU--FHL.#TEXT-REQUIRED = TRUE
    PERFORM GET-PROMPT-TEXT
    INPUT WINDOW = 'FRMT' USING MAP 'CU--FHO'
    ASSIGN CSAMARK.ERROR-POS = POS(#PDAX-VARIABLE-FORMAT)
    ESCAPE BOTTOM (NEW-SCREEN.)
```

```
  END-IF
*
**SAG END-EXIT
```

AFTER-INVOKE-SUBPANELS

This user exit generates the framework for any processing performed after subpanels are invoked.

Example of Code

```
DEFINE EXIT AFTER-INVOKE-SUBPANELS
  PERFORM SET-MORE-INDICATORS
END-EXIT
```

ASSIGN-DERIVED-VALUES

This user exit generates initialization statements for all #PDA variables in the model PDA. The variables are assigned null default values. You can modify the generated code as desired.

Tip: If you add specification parameters to the model PDA, you can get sample statements for the new parameters by regenerating this exit. Regeneration adds the new variables, but does not modify code from the previous generation.

Example of Code

```
DEFINE EXIT ASSIGN-DERIVED-VALUES
*
* Initialize '#PDA-' parameters in PDA.
  ASSIGN #PDA-FIELD-TYPE = ' '
  ASSIGN #PDA-FIELD-REDEFINED = FALSE
  ASSIGN #PDA-LEVEL-NUMBER = 0
  ASSIGN #PDA-FIELD-FORMAT = ' '
  ASSIGN #PDA-FIELD-LENGTH = 0
  ASSIGN #PDA-UNITS = 0
  ASSIGN #PDA-DECIMALS = 0
  ASSIGN #PDA-FROM-INDEX(*) = 0
  ASSIGN #PDA-THRU-INDEX(*) = 0
  ASSIGN #PDA-FIELD-RANK = 0
  ASSIGN #PDA-FILE-CODE = 0
  ASSIGN #PDA-MAX-LINES = 0
  ASSIGN #PDA-WFRAME = ' '
  ASSIGN #PDA-WLENGTH = ' '
  ASSIGN #PDA-WCOLUMN = ' '
  ASSIGN #PDA-WBASE = ' '
END-EXIT ASSIGN-DERIVED-VALUES
```

BEFORE-CHECK-ERROR

This user exit generates the framework for any processing performed before a standard error check. When an error condition occurs, the END-OF-PROGRAM user exit is bypassed. If a model subprogram requires processing before leaving the program, use this user exit to specify the processing.

Example of Code

```
**SAG DEFINE EXIT BEFORE-CHECK-ERROR
*
* Use this user exit for specific error checking
  IF CSASTD.RETURN-CODE = CSLRCODE.#INTERRUPT(*)
    ASSIGN C--PDA.#PDA-PHASE = #SAVE-PHASE
  END-IF
 **SAG END-EXIT
```

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:

- Look up a code table (to display a description, as well as a code value)
- Issue SET CONTROL statements
- Capture or default map variables prior to displaying each panel

Example of Code

```
DEFINE EXIT BEFORE-INPUT
*
* Assign external value
  FOR #I = 1 TO 7
    IF #PDAX-BACKGROUND-COLOUR = #CD(#I) THEN
      ASSIGN #REVERSED-CD(#I) = TRUE
      ESCAPE BOTTOM
    END-IF
  END-FOR
END-EXIT
```

BEFORE-INVOKE-SUBPANELS

This user exit generates the framework for any processing performed before subpanels are invoked.

Example of Code

```
DEFINE EXIT BEFORE-INVOKE-SUBPANELS
  IF CU--PDA.#PDA-PHASE NE CSLPHASE.#TRANSLATE THEN
    PERFORM VALIDATE-FILE-INFO
  END-IF
END-EXIT
```

BEFORE-REINPUT-MESSAGE

The code in this user exit allows you to interrogate the message codes and override the display logic for the generated messages. For example, if the logic specifies that a message is ignored, you can display the message. If the logic specifies that the program is interrupted, you can terminate the program.

Example of Code

```
END-SUBROUTINE /* INPUT-SCREEN
*
* DEFINE SUBROUTINE REINPUT-MESSAGE
*
**SAG DEFINE EXIT BEFORE-REINPUT-MESSAGE
   IF CSASTD.RETURN-CODE = CSLRCODE.#COMMUNICATION THEN
      ESCAPE BOTTOM(PROG.) IMMEDIATE
   END-IF
**SAG END-EXIT
   DECIDE FOR FIRST CONDITION
      WHEN CSASTD.RETURN-CODE = CSLRCODE.#CONTINUE(*)
             IGNORE
      WHEN CSASTD.RETURN-CODE = CSLRCODE.#INTERRUPT(*)
                  ESCAPE BOTTOM(NEW-SCREEN)
       WHEN NONE
             IGNORE
END-DECIDE
```

BEFORE-STANDARD-KEY-CHECK

The code in this user exit checks any additional PF-keys defined for a maintenance subprogram or prepares for standard PF-key validations.

Example of Code

```
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
```

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

```
DEFINE EXIT CHANGE-HISTORY
* Changed on Aug 17,07 by SAG for release ____
* >
* >
* >
END-EXIT CHANGE-HISTORY
```

DESCRIBE-INPUTS

This user exit contains statements that document specification parameter values (#PDAX variables) in the model PDA. For example, if you are documenting a menu program, this user exit contains the menu function codes and descriptions.

Example of Code

```
DEFINE EXIT DESCRIBE-INPUTS
*
* Enter other model parameters to be documented.
* Use WRITE statements of the following format:
*     WRITE(SRC) NOTITLE LDA.#Variable-name #PDAX-variable-name
END-EXIT DESCRIBE-INPUTS
```

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) before exiting the module. You can also use this exit to assign the current key value to a global variable so it is carried into other modules that use the same key.

> **Note:** 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.

Example of Code

```
**SAG DEFINE EXIT END-OF-PROGRAM
*
* Actions to be performed before program exit.
  IF #PDAX-GDA NE ' ' AND #PDA-PHASE = 'M' THEN
    ASSIGN CNAEXIST.#OBJECT-SOURCE = 'O'
    ASSIGN CNAEXIST.#LIBRARY-NAME = *LIBRARY-ID
    ASSIGN CNAEXIST.#INCLUDE-STEPLIB-SEARCH = TRUE
    ASSIGN CNAEXIST.#OBJECT-NAME   = #PDAX-GDA
    CALLNAT 'CNUEXIST' CNAEXIST
                       CSASTD
    PERFORM CHECK-ERROR
    IF NOT CNAEXIST.#OBJECT-EXISTS THEN
      ASSIGN CNAMSG.RETURN-CODE = CSLRCODE.#WARNING
      ASSIGN CNAMSG.MSG-DATA(1) = CU--MAL.#GDA
      ASSIGN CNAMSG.MSG-DATA(2) = #PDAX-GDA
      INCLUDE CU--GMSG '2128'
        ''':1::2::3:not in current library or STEPLIBs'''
    END-IF
  END-IF
**SAG END-EXIT
```

GENERATE-CODE

This user exit generates the framework for any code generated by a model subprogram.

Example of Code

```
DEFINE EXIT GENERATE-CODE
*
  RESET CSASELFV CSASELFV.GENERAL-INFORMATION
                 CSASELFV.FIELD-SPECIFICATION(*)
  MOVE CUMPPDA.#PDAX-VIEW-LPDA-STRUCT-NAME(*) TO
                                   CSASELFV.#VIEW-LPDA-STRUCT-NAME(*)
  MOVE CUMPPDA.#PDAX-FIELD-NAME(*) TO CSASELFV.FIELD-NAME(*)
  MOVE CUMPPDA.#PDAX-FIELD-FORMAT(*) TO CSASELFV.FIELD-FORMAT(*)
  MOVE CUMPPDA.#PDAX-FIELD-LENGTH(*) TO CSASELFV.FIELD-LENGTH(*)
  FOR #I = 1 TO #MAX-FLDS
```

```
    MOVE CUMPPDA.#PDAX-MAX-OCCURS(#I) TO
                                    CSASELFV.FIELD-OCCURRENCES(#I,1)
  END-FOR
  MOVE CUMPPDA.#PDAX-STRUCTURE-NUMBER(*) TO
                                    CSASELFV.#STRUCTURE-NUMBER(*)
  MOVE CUMPPDA.#PDAX-FIELD-PROMPT-OR-TEXT(*) TO
                                    CSASELFV.FIELD-HEADINGS(*)
  ASSIGN CSASELFV.#ARRAY-RANK-SELECTED = 1
  CALLNAT 'CSUSELFV' CSASELFV
                     CU--PDA
                     CSASTD
  ASSIGN CSASTD.ERROR-FIELD-INDEX1 = CSASELFV.#ERROR-FIELD-INDEX
  PERFORM CHECK-ERROR
  RESET CSASTD.ERROR-FIELD-INDEX1
  MOVE CSASELFV.FIELD-NAME(*) TO CUMPPDA.#PDAX-FIELD-NAME(*)
  MOVE CSASELFV.FIELD-FORMAT(*) TO CUMPPDA.#PDAX-FIELD-FORMAT(*)
  MOVE CSASELFV.FIELD-LENGTH(*) TO CUMPPDA.#PDAX-FIELD-LENGTH(*)
  MOVE CSASELFV.#STRUCTURE-NUMBER(*) TO
                               CUMPPDA.#PDAX-STRUCTURE-NUMBER(*)
  MOVE CSASELFV.FIELD-HEADINGS(*) TO
                               CUMPPDA.#PDAX-FIELD-PROMPT-OR-TEXT(*)
  MOVE CSASELFV.#VIEW-LPDA-STRUCT-NAME(*) TO
                               CUMPPDA.#PDAX-VIEW-LPDA-STRUCT-NAME(*)
  FOR #I = 1 TO #MAX-FLDS
    MOVE CSASELFV.FIELD-OCCURRENCES(#I,1)
      TO CUMPPDA.#PDAX-MAX-OCCURS(#I)
    EXAMINE CUMPPDA.#PDAX-FIELD-PROMPT-OR-TEXT(#I) FOR '/'
      REPLACE WITH ' '
  END-FOR
END-EXIT GENERATE-CODE
```

GENERATE-SUBROUTINES

This user exit generates the framework for validations performed by the model validation subprogram. It is used in conjunction with the GENERATE-VALIDATIONS user exit and is available for modules generated using the CST-Validate model.

Code validations as subroutines in this user exit. For each #PDAX-FIELD-NAME field you want to validate, create a subroutine called V-*field-name* to perform the validations. Whenever a validation error is found, the V-*field-name* subroutine must:

- Assign `CSASTD.RETURN-CODE = ‘E’`
- Assign the error message in CSASTD.MSG
- Perform an ESCAPE-ROUTINE to bypass subsequent checks

Tip: To retrieve SYSERR messages, use the CU--VERR copycode.

Tip: To return a warning message, rather than an error, use the CU--VWAR copycode.

References

- For more information about coding validations, see ***CST-Validate Model***.
- For information about validating array fields, see ***Validate Array Fields***.

GENERATE-VALIDATIONS

This user exit generates the framework for validations performed by the model validation subprogram. It is used in conjunction with the GENERATE-SUBROUTINES user exit and is available for modules generated using the CST-Validate model.

Note: For more information, see ***CST-Validate Model***.

INPUT-ADDITIONAL-PARAMETERS

This user exit contains an INPUT statement to read parameters that are not automatically included in a read subprogram.

Example of Code

```
DEFINE EXIT INPUT-ADDITIONAL-PARAMETERS
*
* Input all other parameters..
*
*   /* Input parameter SAMPLE
*   WHEN #LINE = 'SAMPLE:'
*     INPUT CXMYPDA.#PDAX-SAMPLE
END-EXIT INPUT-ADDITIONAL-PARAMETERS
```

INPUT-SCREEN

This user exit generates code to input screens (maps) for a maintenance subprogram.

Example of Code

```
DEFINE EXIT INPUT-SCREEN
IF CSASTD.RETURN-CODE = CSLERROR.#OK OR = CSLERROR.#WARNING
  INPUT WITH TEXT CSASTD.MSG
    MARK POSITION CSAMARK.ERROR-COLUMN IN CSAMARK.ERROR-POS
    USING MAP 'map'
ELSE
  INPUT WITH TEXT CSASTD.MSG
    MARK POSITION CSAMARK.ERROR-COLUMN IN CSAMARK.ERROR-POS
    ALARM
    USING MAP 'map'
END-IF
END-EXIT INPUT-SCREEN
```

LOCAL-DATA

The code in this exit defines additional local variables used in conjunction with other user exits.

Example of Code

```
**SAG DEFINE EXIT LOCAL-DATA
    01 #HELPR(A8) INIT<'CD-HELPR'>
  LOCAL USING CNAEXIST
**SAG END-EXIT
```

MISCELLANEOUS-SUBROUTINES

This user exit generates the framework for any additional subroutines used by a maintenance subprogram.

Example of Code

```
DEFINE EXIT MISCELLANEOUS-SUBROUTINES
**
**********************************************************************
DEFINE SUBROUTINE subroutine-name
**********************************************************************
**
  ESCAPE ROUTINE IMMEDIATE
END-SUBROUTINE /* subroutine-name
END-EXIT MISCELLANEOUS-SUBROUTINES
```

MISCELLANEOUS-VARIABLES

This user exit generates code to write the field and prompt values to Predict. To generate the correct code, translation LDAs must adhere to the following naming standards:

Field	Prompt
#PDA-GEN-PROGRAM	CUMNMAL.#GEN-PROGRAM
#PDAX-TITLE	CUMNMAL.#TITLE

Example of Code

```
DEFINE EXIT MISCELLANEOUS-VARIABLES
*****************************************************************
DEFINE SUBROUTINE MISCELLANEOUS
*****************************************************************
*
  WRITE(SRC) NOTITLE 20T CU--DOCL.#MISC-SPECIFICATIONS
  WRITE(SRC) NOTITLE     CU--PDA.#PDA-UNDERSCORE-LINE (AL=70)
  WRITE(SRC) NOTITLE ' '
```

```
END-SUBROUTINE /* MISCELLANEOUS
END-EXIT
```

PARAMETER-DATA

This user exit generates the framework to process any additional parameters used in conjunction with other programs.

Example of Code

```
DEFINE EXIT PARAMETER-DATA
** PARAMETER USING PDAname
** PARAMETER
**  01 #Additional-parameter1
**  01 #Additional-parameter2
END-EXIT PARAMETER-DATA
```

PF-KEYS

This user exit documents information about PF-keys supported by a generated subprogram to the Predict data dictionary.

To document information about PF-keys:

1 Select the PF-KEYS user exit.

2 Press Enter.

A window is displayed, in which you can specify the supported PF-keys. Descriptions of the keys are added to Predict.

Example of Code

```
* Translate pfkey functions
  PERFORM GET-CDKEYFL-TEXT
*
* Write pfkey names and functions
  PRINT(SRC) NOTITLE / 20T CU--DOCL.#PFKEY-SUPPORT
    / ' '
    / 3T CU--DOCL.#PFKEY 14T CU--DOCL.#FUNCTION
    / 3T CU--PDA.#PDA-UNDERSCORE-LINE (AL=10)
         CU--PDA.#PDA-UNDERSCORE-LINE (AL=60)
    / 3T CDKEYLDA.#KEY-NAME(2)
     14T CDKEYFL.#KEY-FUNCTION(2)
END-SUBROUTINE /* PF-KEYS
END-EXIT
DEFINE EXIT PF-KEYS
*****************************************************************
DEFINE SUBROUTINE PF-KEYS
*****************************************************************
```

```
*
* Translate pfkey names
  INCLUDE CU--DOC
*
* Translate pfkey functions
  PERFORM GET-CDKEYFL-TEXT
*
* Write pfkey names and functions
  PRINT(SRC) NOTITLE / 20T CU--DOCL.#PFKEY-SUPPORT
    / ' '
    / 3T CU--DOCL.#PFKEY 14T CU--DOCL.#FUNCTION
    / 3T CU--PDA.#PDA-UNDERSCORE-LINE (AL=10)
         CU--PDA.#PDA-UNDERSCORE-LINE (AL=60)
    / 3T CDKEYLDA.#KEY-NAME(3)
     14T CDKEYFL.#KEY-FUNCTION(3)
END-SUBROUTINE /* PF-KEYS
END-EXIT
```

PROCESS-SPECIAL-KEYS

This user exit is required for the CST-Modify-332 model if the maintenance subprogram supports special PF-keys (all keys other than Enter and help, return, quit, right, and left PF-keys).

Note: Define the special PF-keys on the Maintain Subprograms panel. For information, see *Maintain Subprograms Function*.

After defining the keys and generating the model, this exit contains code you can use as a starting point for processing the keys.

Example of Code

```
DEFINE EXIT PROCESS-SPECIAL-KEYS
  ASSIGN #PF-KEY = *PF-KEY
  DECIDE ON FIRST VALUE OF *PF-KEY
    VALUE #PF-*0039
      /*
      /* Perform *0039 processing
      ASSIGN CSASTD.MSG = '*0039 processing completed successfully'
      ESCAPE TOP
    NONE VALUE
      IF *PF-KEY NE 'ENTR'
        REINPUT 'Invalid key:1:entered',#PF-KEY
      END-IF
  END-DECIDE
END-EXIT PROCESS-SPECIAL-KEYS
```

PROVIDE-DEFAULT-VALUES

This user exit provides a list of default values for model parameters. If desired, it can also supply values for other parameters you want to initialize. Natural Construct provides default values for the #PDAX variables in the model PDA.

Tip: To specify default values for additional parameters in a model PDA, regenerate this user exit. This adds the new variables but does not modify the code from the previous generation.

Example of Code

```
DEFINE EXIT PROVIDE-DEFAULT-VALUES
  ASSIGN CXMNPDA.#PDAX-DESCS(*) = ' '
  ASSIGN CXMNPDA.#PDAX-USE-MSG-NR = FALSE
  ASSIGN CXMNPDA.#PDAX-PDA = ' '
  ASSIGN CXMNPDA.#PDAX-FILE-NAME = ' '
  ASSIGN CXMNPDA.#PDAX-FIELD-NAME = ' '
  ASSIGN CXMNPDA.#PDAX-MAP-NAME = ' '
  ASSIGN CXMNPDA.#PDAX-LINES-PER-SCREEN = 0
  ASSIGN CXMNPDA.#PDAX-WINDOW-BASE = ' '
  ASSIGN CXMNPDA.#PDAX-WINDOW-BASE-LINE = 0
  ASSIGN CXMNPDA.#PDAX-WINDOW-BASE-COLUMN = 0
  ASSIGN CXMNPDA.#PDAX-WINDOW-SIZE = ' '
  ASSIGN CXMNPDA.#PDAX-WINDOW-LINE-LENGTH = 0
  ASSIGN CXMNPDA.#PDAX-WINDOW-COLUMN-LENGTH = 0
  ASSIGN CXMNPDA.#PDAX-WINDOW-FRAME = FALSE
END-EXIT PROVIDE-DEFAULT-VALUES
```

SAVE-PARAMETERS

This user exit is required for the CST-Save model. It generates a WRITE statement for each specification parameter (#PDAX variable) in the model PDA. Elements of array variables are written individually, including the number of array occurrences. The WRITE statement has the following format:

```
WRITE(SRC) NOTITLE '=' #PDAX-variable-name
```

Natural Construct transforms these lines as follows:

```
**SAG variable name: variable contents
```

and writes them at the beginning of Natural Construct-generated modules.

Tip: If you add specification parameters to a model PDA, regenerate this user exit to generate the WRITE statements for the new parameters. Regeneration adds the new variables, but does not modify code from the previous generation.

Example of Code

```
DEFINE EXIT SAVE-PARAMETERS
FOR #I = 1 TO 4
  IF #PDAX-DESCS(#I) NE ' ' THEN
    COMPRESS '#PDAX-DESCS(' #I '):' INTO #TEXT
    LEAVING NO
    PRINT(SRC) NOTITLE #TEXT #PDAX-DESCS(#I)
  END-IF
END-FOR
WRITE(SRC) NOTITLE '=' #PDAX-USE-MSG-NR
  / '=' #PDAX-PDA
  / '=' #PDAX-FILE-NAME
  / '=' #PDAX-FIELD-NAME
  / '=' #PDAX-MAP-NAME
  / '=' #PDAX-LINES-PER-SCREEN
  / '=' #PDAX-WINDOW-BASE
  / '=' #PDAX-WINDOW-BASE-LINE
  / '=' #PDAX-WINDOW-BASE-COLUMN
  / '=' #PDAX-WINDOW-SIZE
  / '=' #PDAX-WINDOW-LINE-LENGTH
  / '=' #PDAX-WINDOW-COLUMN-LENGTH
  / '=' #PDAX-WINDOW-FRAME
END-EXIT SAVE-PARAMETERS
```

SET-CONDITION-CODES

This user exit is required for the CST-Pregen model. It generates initialization statements for all conditions (#PDAC variables) in the model PDA. You can modify the generated code as desired.

A condition is set to True when a variable corresponding to the condition exists in the model PDA and has a non-null value. The variables and conditions are linked through their names; the #PDAX-name variable corresponds to the #PDAC-`name` or #PDAC-`name`-SPECIFIED condition.

For example, if the model PDA contains:

- #PDAX-USE-MSG-NR(L) variable
- #PDAC-USE-MSG-NR(L) condition

This user exit generates the following code:

```
WHEN #PDAX-USE-MSG-NR NE FALSE
   #PDAC-USE-MSG-NR = TRUE
```

If the model PDA contains:

- #PDAX-GDA(A8) variable
- #PDAC-GDA-SPECIFIED(L) condition

This user exit generates the following code:

```
WHEN #PDAX-GDA NE ' '
   #PDAC-GDA-SPECIFIED = TRUE
```

The WHEN clause is blank for all conditions that have no corresponding variable in the model PDA.

Code for the conditions currently existing in this user exit is not generated. When you regenerate this user exit, only the code for new conditions (that were added to the model PDA since the previous generation) is added.

Example of Code

```
DEFINE EXIT SET-CONDITION-CODES
*
* Set conditions in PDA.
  DECIDE FOR EVERY CONDITION
    WHEN #PDAX-USE-MSG-NR NE FALSE
      ASSIGN #PDAC-USE-MSG-NR = TRUE
    WHEN #PDAX-FILE-NAME NE ' '
      ASSIGN #PDAC-FILE-NAME-SPECIFIED = TRUE
    WHEN #PDAX-FIELD-NAME NE ' '
      ASSIGN #PDAC-FIELD-NAME-SPECIFIED = TRUE
    WHEN #PDAX-PDA NE ' '
      ASSIGN #PDAC-PDA-SPECIFIED = TRUE
    WHEN NONE
       IGNORE
  END-DECIDE
END-EXIT
```

START-OF-PROGRAM

The code in this user exit is executed once at the beginning of the generated subprogram after all standard initial values are assigned. You can use this exit to do any initial setup required. For example:

- Initialize input values from globals
- Set window or page sizes
- Capture security information for a restricted data area

SUBSTITUTION-VALUES

This user exit is used by the CST-Postgen model, which generates the post-generation subprogram for a model. The post-generation subprogram generates STACK statements for substitution variables in the model PDA. To generate STACK statements for any substitution variables that are not in the model PDA, select the SUBSTITUTION-VALUES or *ADDITIONAL-SUBSTITUTION-VALUES* user exit (see below for a comparison).

If you select the SUBSTITUTION-VALUES user exit, STACK statements for all substitution variables are generated in the exit — those in the model PDA, as well as any additional variables. You can modify these variables as desired.

Which user exit you select depends on whether you want the model to stack substitution parameters in the code frame or in a user exit, thereby overriding the default substitution parameter handling.

- If you use the SUBSTITUTION-VALUES user exit, you must code all substitution values in the exit since default code will not be generated.
- If you use the ADDITIONAL-SUBSTITUTION-VALUES user exit (or no user exit), the model automatically stacks any model PDA variables that match the &SUBSTITUTION values in the code frame. For example:

  ```
  STACK TOP DATA FORMATTED '&PRIME-FILE' #PDAX-PRIME-FILE
  ```

Note: Use either the SUBSTITUTION-VALUES user exit or the ADDITIONAL-SUBSTITUTION-VALUES user exit, but not both.

VALIDATE-DATA

The code in this user exit performs edit checks on each parameter on a maintenance map (specified in the Map name field on the Standard Parameters panel). This section contains examples of user exit code for the CST-Modify and CST-Modify-332 model. The CST-Modify model supports dynamic multilingual specification panels and messages using SYSERR references and substitution variables. The code generated in this exit contains SYSERR numbers and substitution values.

Example of Code for CST-Modify Model

```
DEFINE EXIT VALIDATE-DATA
  DECIDE FOR EVERY CONDITION
    WHEN #HEADER1 = ' '
      ASSIGN CNAMSG.MSG-DATA(1) = #HEADER1
      INCLUDE CU--RMSG '2001'
      ''':1::2::3:is required'''
      '#HEADER1'
    WHEN #HEADER2 = ' '
      ASSIGN CNAMSG.MSG-DATA(1) = #HEADER2
      INCLUDE CU--RMSG '2001'
      ''':1::2::3:is required'''
```

```
      '#HEADER2'
    WHEN #PDA-GEN-PROGRAM = ' '
      ASSIGN CNAMSG.MSG-DATA(1) = #GEN-PROGRAM
      INCLUDE CU--RMSG '2001'
      ''':1::2::3:is required'''
      '#PDA-GEN-PROGRAM'
    WHEN #PDA-SYSTEM = ' '
      ASSIGN CNAMSG.MSG-DATA(1) = #SYSTEM
      INCLUDE CU--RMSG '2001'
      ''':1::2::3:is required'''
      '#PDA-SYSTEM'
    WHEN #PDA-TITLE = ' '
      ASSIGN CNAMSG.MSG-DATA(1) = #TITLE
      INCLUDE CU--RMSG '2001'
      ''':1::2::3:is required'''
      '#PDA-TITLE'
    WHEN CUBAPDA.#PDAX-DESCS = ' '
      ASSIGN CNAMSG.MSG-DATA(1) = #DESCS
      INCLUDE CU--RMSG '2001'
      ''':1::2::3:is required'''
      'CUBAPDA.#PDAX-DESCS'
    WHEN CUBAPDA.#PDAX-GDA = ' '
      ASSIGN CNAMSG.MSG-DATA(1) = #GDA
      INCLUDE CU--RMSG '2001'
      ''':1::2::3:is required'''
      'CUBAPDA.#PDAX-GDA'
    WHEN CUBAPDA.#PDAX-GDA-BLOCK = ' '
      ASSIGN CNAMSG.MSG-DATA(1) = #GDA-BLOCK
      INCLUDE CU--RMSG '2001'
      ''':1::2::3:is required'''
      'CUBAPDA.#PDAX-GDA-BLOCK'
    WHEN CUBAMAL.#DESCRIPTION = ' '
      ASSIGN CNAMSG.MSG-DATA(1) = #DESCRIPTION
      INCLUDE CU--RMSG '2001'
      ''':1::2::3:is required'''
      'CUBAMAL.#DESCRIPTION'
    WHEN CUBAMAL.#GDA = ' '
      ASSIGN CNAMSG.MSG-DATA(1) = #GDA
      INCLUDE CU--RMSG '2001'
      ''':1::2::3:is required'''
      'CUBAMAL.#GDA'
    WHEN CUBAMAL.#GDA-BLOCK = ' '
      ASSIGN CNAMSG.MSG-DATA(1) = #GDA-BLOCK
      INCLUDE CU--RMSG '2001'
      ''':1::2::3:is required'''
      'CUBAMAL.#GDA-BLOCK'
    WHEN CUBAMAL.#GEN-PROGRAM = ' '
      ASSIGN CNAMSG.MSG-DATA(1) = #GEN-PROGRAM
      INCLUDE CU--RMSG '2001'
      ''':1::2::3:is required'''
      'CUBAMAL.#GEN-PROGRAM'
    WHEN CUBAMAL.#SYSTEM = ' '
```

```
      ASSIGN CNAMSG.MSG-DATA(1) = #SYSTEM
      INCLUDE CU--RMSG '2001'
      ''':1::2::3:is required'''
      'CUBAMAL.#SYSTEM'
    WHEN CUBAMAL.#TITLE = ' '
      ASSIGN CNAMSG.MSG-DATA(1) = #TITLE
      INCLUDE CU--RMSG '2001'
      ''':1::2::3:is required'''
      'CUBAMAL.#TITLE'
END-EXIT ↩
```

Example of Code for CST-Modify-332 Model

```
DEFINE EXIT VALIDATE-DATA
*
* Edit checks on map parameters.
  DECIDE FOR EVERY CONDITION
    WHEN #HEADER1 = ' '
      REINPUT 'Header1 is required'
      MARK *#HEADER1 ALARM
    WHEN #HEADER2 = ' '
      REINPUT 'Header2 is required'
      MARK *#HEADER2 ALARM
    WHEN CDDIALDA.#PROGRAM = ' '
      REINPUT 'Program is required'
      MARK *CDDIALDA.#PROGRAM ALARM
    WHEN CDGETDCA.#DIRECT-COMMAND = ' '
      REINPUT 'Direct Command is required'
      MARK *CDGETDCA.#DIRECT-COMMAND ALARM
    WHEN NONE IGNORE
  END-DECIDE
END-EXIT VALIDATE-DATA
```

The basic structure of this user exit is supplied in the above format. You can edit the supplied code as required.

Tip: If you add specification parameters to the model PDA, you can generate sample statements for the new parameters by regenerating this user exit. Regeneration adds the new variables, but does not modify code from the previous generation.

21 Modifying the Supplied Models

This section describes how to modify the models supplied by Natural Construct. In most cases, the existing model can be customized by modifying the code frames associated with the model or the copycode members used in the generated modules. In some cases, the generated code may need to be modified by the subprograms in the model code frames (identified by the CU prefix).

Introduction

The source code for all CU-prefixed subprograms is supplied with Natural Construct. To reduce dependencies between Predict and the Natural Construct models, all models use external subprograms to access the Predict data dictionary (they do not access Predict directly).

Do not modify the supplied model subprograms, as changes to these subprograms may have to be reapplied with each new release of Natural Construct. If you want to modify supplied subprograms, copy the supplied subprogram and use a CX prefix (rather than the CU prefix) to name it.

Additionally, do not modify the supplied code frames. All supplied code frames end with a suffix value of 9 (for example, CMNA9) and updated Natural Construct code frames end with a suffix of 8. To create a custom code frame, copy and rename the supplied code frame with a lower suffix value (for example, CMNA7) and modify the new code frame. Natural Construct searches for and uses the code frame with the lowest suffix value when the program is generated. Document all changes so they can be reapplied to subsequent versions of Natural Construct. For more information, see ***Maintain Models Function***.

Note: If changes are confined to model subprograms or copycode members used in modules generated by the model, use the multiple steplib feature to customize the model. For information, see ***Use Steplibs to Modify Models***.

Change the Supplied Models

Typically, the Natural Construct administrator makes changes to the generation models. Before a modified model is available for general use, it should be thoroughly tested. The following sections explain how to modify the supplied model code frames, subprograms, and copycode, as well as how to modify the external data areas and subprograms used by the generation models:

- Modify Code Frames
- Modify the Model Subprograms

- Modify Copycode (CC*) and External Data Areas and Subprograms (CD*)

Modify Code Frames

Do not modify the supplied code frames. Instead, make a copy the code frames you want to customize and modify the copy. Keep the original code frames so they can be referred to if problems arise. Changes to code frames take effect immediately after the code frame is saved.

Note: Document all modifications to the code frames so changes can be reapplied to new versions of Natural Construct.

To modify a code frame:

1 Copy the code frame and use an X prefix to name the copy. For example, the CFEXAM9 code frame becomes XFEXAM9.

> **Tip:** Rather than copying and renaming individual model components, you can create standard, development, and production versions of all system files. Use the CSFUNLD and CSFLOAD utilities to move code frames between files. For information, see *Import and Export Utilities*.

2 Copy the model that uses the modified code frame and give the copy a different name.

For example, the Menu model becomes Menu2.

3 Invoke the model copy to test changes to your code frame.

For example, you can invoke Menu2 model to test the modified code frame without interrupting the use of the Menu model.

4 Change the X prefix back to a C and change the 9 in the last position of the code frame name to a lesser number (from 1 to 7).

For example, the XFEXAM9 code frame becomes CFEXAM7. Natural Construct always uses the code frame ending with the lesser number.

> **Note:** Do not use the number 8 in the last position of the code frame name. Number 8 is reserved for future changes to the supplied code frames (should they be issued). For more information about modifying code frames, see *Step 5: Create Code Frame(s) and Define the Model*.

Modify the Model Subprograms

Because the production copies of the model subprograms are invoked from the SYSLIBS library, you can modify and test the model subprograms within the SYSCST library without affecting existing users of the model. To invoke Natural Construct from the SYSCST library (instead of the SYSTEM library), use the CSTG command (not NCSTG).

To invoke Natural Construct from the SYSCST library (instead of the SYSTEM library):

- Enter "CSTG" at the Natural prompt (instead of NCSTG).

To modify a supplied model subprogram (prefixed by CU):

1. Copy the subprogram and change the CU prefix to CX.
2. Copy the corresponding model and refer the copy to the new CX subprogram.

   Note: Use the CSUTEST utility to test the model subprograms individually. For more information, see *Test the Model Subprograms*.

3. After testing the model subprograms in the SYSCST library, copy the modified modules to the SYSLIBS library in the FNAT system file.

   Tip: If you change the condition codes in the model PDA, copy the object code for the model PDA into the SYSLIBS library as well.

   Note: If Natural Construct is invoked from a steplib, you do not have to rename the supplied subprograms during modification and testing. Instead, copy the subprogram to a test library or other higher level steplib. Once tested, you can copy the modules to the steplib reserved by all development libraries for modifying the supplied modules.

Modify Copycode (CC*) and External Data Areas and Subprograms (CD*)

If you modify any of the CC or CD-prefixed supplied modules and want to apply the changes:

- To programs generated in all libraries, copy the modified modules to the SYSTEM library.
- To one application, copy the modified modules to the corresponding application library.

If you modify the CC or CD-prefixed modules and assign a new name to the modified modules, reference the new name in the Natural Construct standard models. For example, if you modify CCSTDKEY and name the new module MYSTDKEY, refer the Natural Construct standard models to MYSTDKEY instead of CCSTDKEY.

The supplied CSXCNAME user exit subprogram (in the SYSCSTX library) allows users to substitute their own symbols or names for the default values generated into a Natural Construct object (CC* copycode and CD* routines, for example). If this subprogram exists in the SYSLIBS library, it is invoked immediately before the post-generation subprogram for the current model.

The main function of the CSXCNAME subprogram is to place a list of substitution symbols and values on the Natural stack. For example, if you enter the following code in CSXCNAME:

```
STACK TOP DATA FORMATTED 'CCSTDKEY' 'MYSTDKEY'
```

Natural Construct scans for CCSTDKEY and replaces it with MYSTDKEY.

Example of Modifying a Model

This section describes how to modify the maintenance model (Maint). The modifications include the option to generate depth scrolling capabilities, in addition to the current up-down and left-right scrolling. This capability allows a user to scroll a three-dimensional array using the PF4 and PF5 keys. Additionally, the user can name these keys on the second specification panel.

To modify a model:

1. Determine what modifications are required by manually applying the changes to a maintenance program generated by the model.

   The modified program is the prototype. To identify which code frames, PDA, and subprograms to modify, invoke the Maintain Models panel and display information for the Maint model. For information, see *Maintain Models Function*.

2. Modify the parameter data area (PDA) as follows:

   - Copy the PDA and change the CU prefix to CX.
   - Add a #PDAC-DEPTH-KEYS logical variable to the end of the redefinition of #PDA-CONDITION-CODES.
   - Add a #PDAX-DEPTH-KEYS logical variable to the end of the redefinition of #PDA-USER-AREA.
   - Add two A5 fields (#PDAX-DEPTH-IN and #PDAX-DEPTH-OUT, for example).
   - Stow the modified PDA in the SYSCST library.

   Note: If you are executing the steplib version of Natural Construct, move the model PDA to a lower level steplib and make the changes without renaming the object.

3. Modify the second maintenance map and subprogram as follows:

   Tip: The subprogram name is displayed in the top left corner of the panel; the map name is displayed in the top right corner of the panel.

- Copy the current versions and change the CU prefix to CX.
- Add the #PDAX-DEPTH-KEYS, #PDAX-DEPTH-IN, and #PDAX-DEPTH-OUT fields to the new map. For example:

```
Include Depth Keys: _ (Named: _____ and _____)
```

- Stow the new map and subprogram.

Note: Validation edits (ensuring the keys are named if they are included, for example) can be initiated on the map or within the invoking subprogram.

4 Modify the code frames as follows:

- Identify the code frames to modify.

  The easiest way to do this is by selecting the Options field when generating a program using the Maint model. When the Status window is displayed, select the Embedded statements option. The generated program will then contain comments showing where each code block originated.
- Copy the code frames and change the C prefixes to X.
- Modify the X code frames in the DEPTH-KEYS condition. You can name the keys using substitution parameters assigned in the post-generation subprogram. For example:

```
DEPTH-KEYS                                                      1
SET KEY CDKEYLDA.#DEPTH-IN-KEY NAMED "&DEPTH-IN'                "
SET KEY CDKEYLDA.#DEPTH-OUT-KEY NAMED "&DEPTH-OUT'              "
```

- Save the code frame.
- Make a copy of the model and have the copied model refer to the X copies.

Note: Add the new PF-keys to CDKEYLDA. For information, refer to *Adding a New PF-Key*, *Natural Construct Generation*.

5 Modify the model subprograms as follows:

- Make copies of the subprograms and name the copies using an X prefix (or use a steplib).
- Modify the clear subprogram to initialize the new parameters. For example:

```
RESET #PDAX-DEPTH-KEYS
ASSIGN #PDAX-DEPTH-IN = 'front'
ASSIGN #PDAX-DEPTH-OUT = 'back'
```

- Modify the pre-generation subprogram to assign the #PDAC-DEPTH-KEYS logical condition variable to True if the user marks the #PDAX-DEPTH-KEYS field.
- Modify the post-generation subprogram to assign the names of the depth keys. For example:

```
IF #PDAC-DEPTH-KEYS THEN
  STACK TOP DATA FORMATTED '&DEPTH-IN' #PDAX-DEPTH-IN
  STACK TOP DATA FORMATTED '&DEPTH-OUT' #PDAX-DEPTH-OUT
END-IF
```

- Modify the save subprogram to write the new parameters. For example:

```
IF #PDAC-DEPTH-KEYS THEN
  WRITE(SRC) NOTITLE '=' #PDAX-DEPTH-KEYS
  WRITE(SRC) NOTITLE '=' #PDAX-DEPTH-IN
  WRITE(SRC) NOTITLE '=' #PDAX-DEPTH-OUT
END-IF
```

- Modify the read subprogram to accept the new parameters. For example:

```
WHEN #LINE = 'DEPTH-KEYS:'
  INPUT #PDAX-DEPTH-KEYS
WHEN #LINE = 'DEPTH-IN:'
  INPUT #PDAX-DEPTH-IN
WHEN #LINE = 'DEPTH-OUT:'
  INPUT #PDAX-DEPTH-OUT
```

6 Test the modified model in the SYSCST library (using the CSTG command).

  You can also test individual components of the model using the CSUTEST program or debug the model using the trace options available through the Generation main menu. For more information, see ***Test the Model Subprograms***.

7 Migrate the modified model as follows:

  - Copy the modules for the modified subprograms and PDA from the SYSCST library to the SYSLIBS library.
  - Modify the model definition record (Maintain Models panel) to refer to the modified code frame. For information, see ***Maintain Models Function***.

8 Document all modifications to the model.

Use Steplibs to Modify Models

Using Natural Security, you can define up to eight steplibs for each Natural Construct library. The searching order is the current library (*LIBRARY), the first steplib (if present), the second steplib (if present), …, the eighth steplib (if present), and then the SYSTEM library.

If you store the executing Natural Construct modules in a steplib, you can store your modified model subprograms or copycode in a higher level steplib, effectively overriding any supplied Natural Construct modules with the same names and types. In this way, users access your modified models and the supplied models remain untouched.

When you invoke Natural Construct from a steplib, use the CSTG command (as in the SYSCST library) — not the NCSTG command. The NCSTG command always invokes the copy of Natural Construct that is stored in the SYSLIBS library and bypasses the steplibs.

Tip: To use the NCSTG command, you can write an NCSTG program to fetch CSTG in the application library.

Because SYSCST is available in a steplib, this method can regulate access to the Administration subsystem. As the Natural Construct administrator, you can use the security routines in the SYSCSTX library to control access to this subsystem.

The following example describes how to use the steplib method to eliminate direct command processing in Natural Construct-generated programs. Direct command processing is triggered by the #PDAX-DIRECT-COMMAND-PROCESS variable on the CU—MA0 map. You can remove the field that contains this variable from the CU—MA0 map and move the modified map into a steplib at a higher level than the SYSCST library.

To use steplibs to modify a model (assuming that APPL is the application library):

1. Define the steplibs to APPL in the following order: NODIRECT, SYSCST, and SYSTEM from Natural Security.

   NODIRECT is a new library and SYSCST and SYSTEM are steplibs of this new library.
2. Copy the CU—MA0 map from the SYSCST library to the NODIRECT library.
3. Edit the CU—MA0 map in the NODIRECT library.

   Delete the text `Mark to include Direct Command Processing` and define the field containing the #PDAX-DIRECT-COMMAND-PROCESS variable as non-display.
4. Stow the modified CU—MA0 map.
5. If you deleted the field that contains the #PDAX-DIRECT-COMMAND-PROCESS variable, copy all the modules that use the CU—MA0 map in the SYSCST library to the NODIRECT library and catalog them.

Because SYSCST and SYSTEM are steplibs of NODIRECT, these modules can be cataloged in the NODIRECT library.

Note: If you use the steplib version of Natural Construct for batch regeneration, use the CSTBGEN command instead of the NCSTBGEN command.

Invoke Natural Construct From a Steplib

To invoke Natural Construct from a steplib:

1. Define the SYSCST and SYSLIBS libraries as steplibs of all development libraries requiring Natural Construct.
2. Define a higher level steplib where modules can be stored that override the supplied objects.
3. Add a module called NCSTG to the new steplib and code it as follows:

```
FETCH 'CSTG'
END
```

Tip: If extensive code frame changes are required, consider installing a second copy of the Natural Construct system file. You can then make changes to code frames directly, without having to make a copy of individual frames and/or modules. You can use the compare facilities supplied with Natural Construct to compare modified models and code frames with the originals. For information about the compare facilities, see *Compare Menu Function*.

22 External Objects

This section describes the supplied programs, subprograms, and helproutines that help simplify and standardize the model creation process. These utilities can be invoked by the supplied models or by user-written models.

Note: The source code for external objects is not supplied.

Introduction

All model subprograms use external parameter data areas (PDAs) stored in the SYSCST library. The source for the PDAs is provided and contains details about each parameter. For example, some of the listings for the CPAEL PDA are:

```
Parameter CPAEL     Library SAG                                DBID  18 FNR   4
Command                                                                     > +
I T L Name                            F Leng Index/Init/EM/Name/Comment
Top - -------------------------------- - ---- ---------------------------------
    1 CPAEL
    2 INPUTS
    3 FILE-NAME                       A   32 /* File Name.
    3 FIELD-NAME                      A   32 /* Field name to be found in the
    3 #SIMPLE-OUTPUTS-ONLY            L      /* True if interested in
  *                                          /* #FIELD-FOUND only
  *                                          /* given file
    2 INPUT-OUTPUTS
    3 FILE-CODE                       P    8 /* If this code is known,
  *                                          /* NSC checks are avoided.
    3 DDM-PREFIX                      A   16 /* Field prefix on DDM,
  *                                          /* this will be set if correct
  *                                          /* FILE-CODE is not provided.
    2 SIMPLE-OUTPUTS
    3 #FIELD-FOUND                    L      /* True if field found on file
    3 FIELD-IS-REDEFINED              L      /* The field is redefined.
------------------------------------------------------------------ S 70   L 1   ↩
```

CPAEL contains a level 1 structure called CPAEL. Depending on the type of parameter, the remaining parameters are grouped into the following structures: INPUTS, INPUT-OUTPUTS, and OUTPUTS. This layout is the same for all PDAs used by the supplied subprograms.

Note: Be careful when modifying fields in the INPUT-OUTPUTS structure; these fields may retain information across multiple calls.

You can define the PDAs as local data areas (LDAs) within the model subprograms that invoke the utilities. CPAEL is the PDA corresponding to the CPUEL subprogram utility, which returns information about a field in Predict.

The following example shows a model subprogram that requires field information from Predict:

```
DEFINE DATA PARAMETER
 PARAMETER
  .
  .
  .
LOCAL USING CPAEL
LOCAL USING CSASTD
  .
  .
  .
END-DEFINE
  .
  .
  .
ASSIGN CPAEL.FILE-NAME = #PDAX-FILE-NAME
ASSIGN CPAEL.FIELD-NAME = #PDAX-FIELD-NAME
CALLNAT 'CPUEL' CPAEL CSASTD
*
*Check outputs of CPUEL
  .
  .
  .
END
```

This section provides a brief description of the supplied program, subprogram, and helproutine utilities. For examples of how to invoke the utilities, refer to the source code for the supplied model subprograms in the SYSCST library (prefixed by CU).

Note: Driver programs for many of the supplied model programs and subprograms are included on the Natural Construct tape (prefixed by CTE). These driver programs are also available through the Drivers menu option on the Administration main menu. If a driver program is available, its location is listed in the *Drivers Menu Option* section for the program or subprogram. For information about invoking the driver programs, see ***Drivers Menu Function***.

This section covers the following topics:

- Object Categories
- Error Processing
- Passing of Structure Names
- Restricted Data Areas
- Callback Functions

- Subprogram Chaining

Object Categories

The supplied objects are divided into three categories, based on the type of information they access. Each category is identified by its prefix as follows:

Prefix	Object Categories
CN*	Identifies objects that return or generate data based on information in the Natural system files.
CP*	Identifies objects that return or generate data based on information in Predict.
CS*	Identifies objects that are miscellaneous validation, calculation, or translation routines. Most of these routines do not access system file information, but some access Natural Construct system files.

Whenever possible, use the supplied programs, subprograms, and helproutines instead of accessing the system file information directly. This helps protect your programs from unwanted changes to the internal structure. Natural Construct maintains the upward compatibility of the supplied programs, subprograms, and helproutines.

Error Processing

Many of the supplied subprograms return information through the CSASTD parameter data area (PDA). The value in the RETURN-CODE field should be checked after each call. If it is not blank, it should be passed back to the generation nucleus so the user is aware of the problem.

The following example shows a model subprogram that invokes the CPUEL utility:

```
DEFINE DATA
 PARAMETER USING CUMYPDA
 PARAMETER USING CU--PDA
 PARAMETER USING CSASTD
 LOCAL USING CPAEL
 .
 .
 .
END-DEFINE
 .
 .
 .
CALLNAT 'CPUEL' CPAEL CSASTD
IF CSASTD.RETURN-CODE NE ' ' THEN
   ESCAPE ROUTINE IMMEDIATE
END-IF
```

Passing of Structure Names

To invoke the supplied subprograms, pass only the level 1 structures in the PDA. This way, if new parameters are added to the utilities in future versions of Natural Construct, you need only recatalog your model subprograms to incorporate the changes.

Restricted Data Areas

Some subprograms have restricted data areas to retain information across multiple calls. The restricted data areas are identified by an R in the third position of the data area name (CPRELNX, for example).

You do not need to be concerned with the contents of these data areas. Define them as local data areas within the invoking subprograms and pass them to the subprogram that is invoked.

Tip: As with all PDAs, the name of the structure passed to the subprogram always matches the name of the data area itself.

Callback Functions

Many of the Natural Construct utility subprograms iterate through system data and, for each record found, call a user-supplied routine. For example, CPURLRD is used to retrieve all relationships related to a particular file. Rather than returning these relationships to the caller of CPURLRD, the caller must supply the name of a subprogram CPURLRD should call for each relationship found.

These routines accept an A1 array to allow the caller of the utility to communicate information to and from the subprogram called by the utility. This data area is represented by CSAPASS. It is accepted by the utility as a 1:v array so that the actual size of the data area can be determined by the requirements of the caller.

Subprogram Chaining

When a subprogram performs read logical processing and returns a series of records, it is sometimes difficult or inefficient for the subprogram to *remember* where it left off in a previous call. Also, this type of processing can be awkward to code in the invoking object because it must define looping logic and issue iterative CALLNATs until a certain end condition is reached.

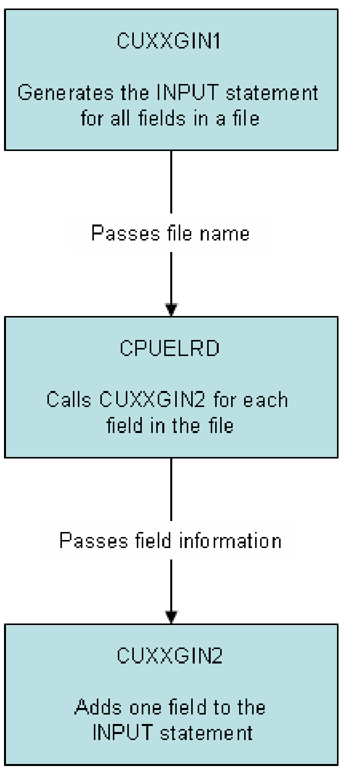
To avoid these problems, some subprograms do not return the information to the calling object. Instead, the calling object passes the name of a subprogram that is invoked for each record encountered. To generate an INPUT statement containing all fields in a file, for example, you can use the CPUELNX and CPUELRD subprograms. This section describes these subprograms.

Without Subprogram Chaining (CPUELNX)

The CPUELNX subprogram can be called iteratively to continually return the next field in the file until an end-of-file condition is reached. The model subprogram that generates the INPUT statement must define the looping logic and make iterative CALLNATs to include each field in the INPUT statement.

With Subprogram Chaining (CPUELRD)

The CPUELRD subprogram can be invoked once by the model subprogram (CUXXGIN1, for example). This subprogram receives the name of a file and a subprogram to CALLNAT (CUXXGIN2, for example). It traverses the file and CALLNATs the subprogram for each field. That subprogram adds the current field to the INPUT statement generated. For example:



To allow CPUELRD to remember information across iterative calls, a 1K area is passed to CUXXGIN2. This area can be redefined into individual fields, such as current status information, that are required by CUXXGIN2 across multiple calls. It can also pass additional information between CUXXGIN1 and CUXXGIN2.

Note: For an example of how subprogram chaining is used, refer to the CUFMGIN1 and CUFMGIN2 programs in the SYSCST library.

Natural-Related Subprograms (CNU*)

The subprograms described in this section retrieve information from the Natural system files to assist in the generation process. For subprograms that return information about Natural objects (programs, data areas, etc.), the specified data area object must exist in the current library or one of its steplibs.

Tip: Driver programs for many of the supplied model programs and subprograms are included on the Natural Construct tape (prefixed by CTE). These driver programs are also available through the Drivers menu option on the Administration main menu. If a driver program is available, its location is listed in the *Drivers Menu Option* section for the program or subprogram. For information about invoking the driver programs, see ***Drivers Menu Function***.

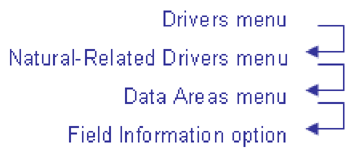
This section describes the following subprograms:

CNUEL Subprogram

CNUEL	Description
What it does	Retrieves information about a field in a local data area (LDA) or parameter data area (PDA). This subprogram receives the name of a field and data area (CNAEL.INPUTS) and returns information about the field (CNAEL.OUTPUTS), such as the structure to which the field belongs, the field format and type, the level number, and the starting and ending index for arrays.
PDAs used	■ CNAEL ■ CSASTD
Files accessed	■ SYSTEM-FUSER

CNUEL	Description
	■ SYSTEM-FNAT

Drivers Menu Option




```
 CTEELN            ***** Natural Related Subprograms *****         CTEELN1
 Feb 25                - Driver for subprogram CNUEL -             03:30 PM

*Data Area Name : ________
 Field Name.....: ________________________________
 Structure Name : ________________________________

 View Of Name...:

 Field Found....:     Field Format:       Lvl Number....:
 Constant Field :     Field Length:
 Field Redefined:     Rank........:
 Lvl Type Trail :
 From Index  Thru Index  1:V   Field Occurrences
 ----------  ----------  ---   -----------------



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                mai ↩
```

CNUELNX Subprogram

CNUELNX	Description
What it does	Returns information about the next field in a data area. This subprogram receives the name of an external data area and returns information about the next field in that data area. On the first call to this subprogram, the specified field is returned. On subsequent calls, the next fields are returned. CNRELNX (PDA containing reserved variables) keeps track of the current position of the data area and must not be modified by the calling program. **Note:** For information about INPUT/OUTPUT parameters, refer to the CNAELNX data area in the SYSCST library.

CNUELNX	Description
PDAs used	■ CNAELNX ■ CNRELNX ■ CSASTD
Files accessed	■ SYSTEM-FNAT

CNUELNX On Unix Platforms

On Unix platforms, it is necessary to explicitly close any open cursors. CNUELNX does this automatically whenever a data area is read in its entirety. However, if you want the calling program to only read a portion of the data area, you must insert additional code to close the open cursor. For example:

```
/* close the object
IF CNRELNX.NATA1500-END-OF-FILE
  IGNORE
ELSE
  CNAELNX.#CLOSE-OBJECT := TRUE
  CALLNAT 'CNUELNX' CNAELNX CNRELNX CSASTD
END-IF
```

Drivers Menu Option

Drivers menu
Natural-Related Drivers menu
Data Areas menu
Get Next Field Information option

```
 CTENLNX           ***** Natural Related subprograms *****         CTENLNX1
 Feb 25               - Driver for subprogram CNUELNX -            03:15 PM

*Data Area Name...: ________      Field Count:       Constant Field :
 First Time.......: X             End Of File:       Dynamic Field..:
 Structure Name...:                                  Field Redefined:
 Field Name.......:                                  Field Format...:
 Field Length.....:                 Units:           Decimals.......:
 View Of Name.....:
 Level Number.....:        Basic Occurrences: _      Rank...........:
 Level Type Trail.:
 Occurrences Found:
 Starting At: 1_

                           From Index     Thru index        1:V Field Occur
 Object location          ------------   ------------       --- -----------
   Library: ________
   DBID...: _____
   FNR....: _____
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

Tip: As this subprogram can have up to 99 field levels, enter a level number in the Starting At field to display the specified level (as well as the next nine levels).

CNUERMSG Subprogram

CNUERMSG	Description
What it does	Receives a Natural error message number and returns the error message text. This subprogram receives a Natural error message number (CSASTD.MSG-NR) and returns the corresponding error message text (CSASTD.MSG). For example, the message text for Natural message number 0888 is `Storage Overflow During Compilation or Execution`.
PDAs used	■ CSASTD
Files accessed	■ SYSTEM-FNAT

Note: This subprogram returns system error messages, rather than application error messages. For information about application error messages, see *CNUMSG Utility*.

Drivers Menu Option

Drivers menu
Natural-Related Drivers menu
Retrieve System Error Msg option

```
 CTEERMSG              N a t u r a l   C o n s t r u c t              CTERMSG1
 Aug 14                 Driver for subprogram CNUERMSG                  1 of 1


  Msg Nr...: ____   Error Fld:
  Ret Code :

  Msg:

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

CNUEXIST Subprogram

CNUEXIST	Description
What it does	Checks for the existence of a Natural module. This subprogram receives the name of a Natural module and determines whether its source, compiled object, or both exist. If the source and/or compiled object exist, the subprogram returns the module type (P for program) and library name(s) in which the source and/or compiled object(s) were found. If the module is not found in the current library, you can request a search of all steplibs. In this case, the name of the first library in which the module was found is returned.
PDAs used	■ CNAEXIST ■ CSASTD
Files accessed	■ SYSTEM-FUSER ■ SYSTEM-FNAT

Drivers Menu Option

Drivers menu
Natural-Related Drivers menu
Verify Source/Object Existence option

```
 CTEEXIST          ***** Natural Related Subprograms *****          CTEXIST
 Feb 09              - Driver for subprogram CNUEXIST -             05:31 PM

*Object/Source Name......: ________                  Source     Object
 Object/Source or Both...: _                        --------   --------
 Search type...                      Exists.:
   Library + Steplib Search: _       Type...:
   or                                Library:
   Specific library search           DBID...:
     Library Name........: ________  FNR....:
     DBID................: _____     User...:
     FNR.................: _____     Date...:
     (Blank implies current library) Time...:
```

CNUGDABL Subprogram

CNUGDABL	Description
What it does	Builds a full path name for a global data area (GDA) block. This subprogram receives a GDA name and the name of a GDA block. It returns the full path name from the master block to the specified block. For example, if BLOCK11 is a sub-block of BLOCK1, which is a sub-block of MASTER-BLOCK, the following full path name is returned: `MASTER-BLOCK.BLOCK1.BLOCK11`
PDAs used	■ CNAGDABL ■ CSASTD
Files accessed	■ SYSTEM-FUSER ■ SYSTEM-FNAT

Drivers Menu Option

Drivers menu
Natural-Related Drivers menu
Build Path Name for GDA Block option

```
CTEGDABL              N a t u r a l   C o n s t r u c t              CTEGDAB1
Aug 14                 Driver for subprogram CNUGDABL                    1 of 1

*GDA Name......: ________
 Block Name....: ________________________________

 Full Path Name:


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                  mai ↩
```

CNUGDAEL Subprogram

CNUGDAEL	Description
What it does	Verifies that a field is contained in a global data area (GDA). This subprogram receives the name of a GDA and the name of a field. If the field exists in the GDA, this subprogram returns a confirmation flag.
PDAs used	■ CNAGDAEL
Files accessed	■ SYSTEM-FNAT ■ SYSTEM-FUSER

Drivers Menu Option

Drivers menu
Natural-Related Drivers menu
Verify GDA Field Existence option

```
CTEGDAEL              N a t u r a l   C o n s t r u c t             CTEGDAE1
Aug 14                 Driver for subprogram CNUGDAEL                   1 of 1

*GDA Name...: ________
 Field Name : ________________________________
 Field Found:



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                  mai ↩
```

CNUGENDA Subprogram

CNUGENDA	Description
What it does	Adds a field to a data area. This subprogram receives the definition of a field (field type, level number, field name, field format and length, and the number of occurrences, for example) to be added to a data area and generates the field definition at the end of the current edit buffer. For information about INPUT/OUTPUT parameters, refer to the CNAGENDA data area in the SYSCST library. **Note:** Before this subprogram is invoked, the calling program must set the Natural editor to a data area type of A, L, or G.
PDAs used	■ CNAGENDA ■ CNRGENDA ■ CSASTD
Files accessed	■ None

To use this utility internally, issue a CALLNAT to the following subprogram immediately after calling CNUGENDA:

```
CALLNAT 'CNUGENDU'
```

There are no parameters for this subprogram.

Drivers Menu Option


```
  CTEGENDA          ***** Natural Related Subprograms *****         CTEGEND1
  Feb 25              - Driver for subprogram CNUGENDA -            03:19 PM

 Field Name: ________________________________

 Field Type: _ Format: _  FIELD-LENGTH-A FIELD-LENGTH-A 00000000000
 Level.....: _ Length: ____________ Occurrences: _______________
 Comment....: ________________________________

 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
       help  retrn quit                                                  mai ↩
```

CNUMPPRF Subprogram

CNUMPPRF	Description
What it does	Reads a map profile from a Natural system file. This subprogram receives the name of the map profile in the CSAMPSET.#PROFILE field. It reads the profile from the Natural system file (FNAT) and returns the map settings. For information about the OUTPUT parameters, refer to the CSAMPSET data area in the SYSCST library.
PDAs used	■ CSAMPSET ■ CSASTD
Files accessed	■ SYSTEM-FNAT

Note: This routine is not available on all platforms.

Drivers Menu Option

Drivers menu
Natural-Related Drivers menu
Map Settings Information option

```
 CTEMPPRF             N a t u r a l   C o n s t r u c t             CTEMPRF1
 Aug 14                Driver for subprogram CNUMPPRF                  1 of 1

Map Profile....: ________     Layout......:              Map Type.....:
Map Version....:              Map Name....:              Std  Keys....:

        +---------------------------------------------+
    1__ | Delimiter Class   AD   CD   Delimiter Char |
DC:     | ---------------   --   --   -------------- |   Col Shift....:
PS:     |                                             |   Case Deflt...:
LS:     |                                             |   Cursor Skip..:
ZP:     |                                             |   PM...........:
        +---------------------------------------------+

Write Statement:              CV..........:              Justification:
Input Statement:              Error Code..:              Enforce Attr :
Auto Rule Rank :              Hlp Fld Dflt:
Fill Character :              Help........:
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                    bkwrd frwrd                   mai ↩
```

CNUMSG Subprogram

CNUMSG	Description
What it does	Returns application message text from the SYSERR message file. **Note:** For more information, see ***CNUMSG Utility***.
PDAs used	■ CNAMSG ■ CSASTD
Files accessed	■ SYSTEM-FUSER

Drivers Menu Option

Drivers menu
Natural-Related Drivers menu
Retrieve Application Error Message
Retrieve Single Message option

```
CTEMSG            ***** Natural Related subprograms *****         CTEMSG1
Oct 16               - Driver for subprogram CNUMSG -             08:53 AM

Message Number.: 0008  *Message Library: CSTMSG__
Message Text (Input)
  ______________________________________________________________________

Retrieval Method: R ('R' for Retrieve, 'S' for Substitute, 'B' for Both)

Message Substitution
  Data(1): ________________________________  *Message Library: CSTLDA__
  Data(2): ________________________________  *Message Library: CSTLDA__
  Data(3): ________________________________  *Message Library: CSTLDA__

Default Languages
  *LANGUAGE: 1  1) 1_ 2) 1_ 3) 1_ 4) 1_ 5) 1_ 6) 1_ 7) 1_ 8) 1_

Response Code: 0     ( 9 - unsuccessful )

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                mai ↩
```

CNUNPDA Subprogram

CNUNPDA	Description
What it does	Determines what the parameters are in a given subprogram and then passes the parameters to another subprogram to process.
Parameters/PDAs used	■ CNANPDA ■ CU__PDA ■ #PASS (*) ■ CSASTD If you are not using method 0, these are the only parameters used. If you are using method 0 and creating your own subprogram to process individual fields, your new subprogram requires the following parameter data areas:

CNUNPDA	Description

	`PARAMETER USING CNANPDA`
	`PARAMETER USING CNANPDA2`
	`PARAMETER 01 DATA-PTR(A1/1:V)  (can be replaced with #PASS from ↩ above)`
	`PARAMETER USING CU__PDA   (can be a model PDA)`
	`PARAMETER USING CSASTD      ↩`

Drivers Menu Option

Drivers menu
Natural-Related Drivers menu
Process Parameter Data option

```
CTENPDA            ***** Natural Related Subprograms *****         CTENPDA1
Aug 15               - Driver for subprogram CNUNPDA -             11:35 AM

Inputs
  subprogram name ....... ________ *     callnat ............... CTENPDA2
  method ................ ___ *          Include step Library .. _
  Detail ext. data Areas  2__ *          level 1 fields only ... _
  Include Redefine Base . ___ *          Include Redefines ..... 2__
  Basic occurrences ..... _              Only process 1:V fields _
  Specific 1:V override . ____ ____ ____ ____ ____ ____ ____ ____ ____ ____
                          ____ ____ ____ ____ ____ ____ ____ ____ ____ ____
                          ____ ____ ____ ____ ____ ____ ____ ____ ____ ____
  default 1:V ........... _____
  Skip data areas ....... ________ ________ ________ ________ ________
                          ________ ________ ________ ________ ________
                          ________ ________ ________ ________ ________
                          ________ ________ ________ ________ ________
                          ________ ________ ________ ________ ________
Outputs
  Library ............... ________
  number of parms ....... _____          1:V parms found ....... _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                                                  main
```

Note: In addition to these fields, CNUNPDA also has access to the PDA-NAME input parameter in CNANPDA. When PDA-NAME contains the name of an external data area, the parameters in PDA-NAME are processed and the parameters in SUBPROGRAM-NAME are ignored.

When using the CTENPDA driver program to call CNUNPDA, the program:

- Receives the name of a Natural subprogram in the Subprogram name field.
- Reads the parameters for the subprogram.
- Determines which fields are available in the PDA (either internally or externally defined).
- Passes each field to the processing subprogram specified in the Callnat field. When method 0 is used, each PDA in the subprogram is processed by the subprogram specified in the Callnat field.

  Note: When the method is 0 for CTENPDA, whatever is in the Callnat field is executed. By default this is the CTENPDA2 subprogram. Regardless of which subprogram is specified in the Callnat field, the subprogram is executed once for each of the parameters found in that subprogram. CTENPDA2 uses a map to display the attributes for each parameter in the specified subprogram.

CTENPDA returns the name of the library containing the Natural subprogram, the number of fields in the PDA, and the number of 1:V fields. The map for CTENPDA uses the variables in CNANPDA (which include the input of the subprogram name and the output of the number of parameters processed). The available methods are:

- 0 - Normal call
- 1 - Check Existence of 1:V Array
- 2 - Generate As Local
- 3 - Generate As Arguments
- 4 - Generate As Input Fields
- 5 - Generate As Helproutine Parms
- 6 - Generate As Parameter
- 7 - Generate As External Parameter
- 8 - Generate As Parameter String
- 9 - Generate As Non Specified
- 10 - Generate As Restricted Input
- 11 - Generate As Unicode Parameter String

For common processes, such as generating an LDA based on the PDAs of a specified subprogram, a non-0 method can be used. This means the value in the Callnat field is ignored and a Construct internal subprogram is called instead of CTENPDA2.

The generated PDA information can be useful when generating:

- A CALLNAT statement for the specified subprogram, since a CALLNAT statement requires all level 1 fields in the PDA to be passed to the subprogram (method 3).
- An INPUT statement to test the specified subprogram. In this case, all the fields required for the subprogram can be easily defined in the INPUT statement (method 4).
- Fields for a DEFINE DATA statement. This is helpful when creating a module that will issue a CALLNAT to the subprogram that is being processed (method 2 or method 6).

How the CST internal subprogram uses the extracted PDA information is determined by the specified method and other input values (such as, only process 1:V fields).

Using the internal subprogram is similar to using method 0 with a CALLNAT to CTNEPDA2, except the method is different and the name of the subprogram called to process each PDA field is not required. The key difference is that information is written to the editor, instead of displayed on the screen (as with CTENPDA2). When not using method 0, therefore, you must quit out of CTENPDA to see the processing results.

Once you know what you want to generate, you can use parameters shown on the CTENPDA screen by placing them in the PDA for CNUNPDA (CNANPDA) and invoking the following CALLNAT statement:

```
CNANPDA.METHOD := CNLNPDA.GENERATE-AS-LOCAL
CNANPDA.INCLUDE-STEPLIB-SEARCH := TRUE
CNANPDA.INCLUDE-REDEFINE-BASE :=  CNLNPDA.INCLUDE-BASE-AND-REDEF-LEVEL
CNANPDA.INCLUDE-REDEFINES :=   CNLNPDA.INCLUDE-ALL-REDEFINES
CALLNAT 'CNUNPDA' CNANPDA
                  CUDRPDA  (or CU__PDA, or any other model PDA)
                  #PASS(*)
                  CSASTD
```

Example of Using CTENPDA

This section provides an example of using the Driver for subprogram CNUNPDA program, CTENPDA. CTENPDA is useful when determining the effect of the input parameters on the output parameters. For example, you can enter "CALC" in Subprogram name, "CTENPDA2" in Callnat, "2" in Detail ext. data Areas, and "2" in Include redefines to determine the output in the CNANPDA2 PDA (which is placed on a map in CTENPDA2). These details help identify valuable attributes about fields in the data area, such as:

- Level number (which allows you to reject all fields that are not level 1, for example).
- Field name
- Field format
- Field length

For this example, the supplied CALC subprogram (available in the SYSCSTDE library) is used. The PDA for CALC is:

```
DEFINE DATA PARAMETER
1 INPUT-DATA
   2 #FUNCTION (A30)
   2 #FIRST-NUM (N5.2)
   2 #SECOND-NUM (N5.2)
   2 #SUCCESS-CRITERIA (N5)
1 OUTPUT-DATA
   2 #RESULT (N11.2)
   2 #TIME (T)
   2 #SUCCESS (L)
END-DEFINE      ↩
```

In the following example, CALC is used as input to the subprogram name field:

```
CTENPDA           ***** Natural Related Subprograms **           CTENPDA1
Sep 12              - Driver for subprogram CNUNPDA -            05:14 PM

Inputs
  subprogram name ....... CALC____ *     callnat ............... CTENPDA2
  method ................ ___ *          Include step Library .. _
  Detail ext. data Areas  2__ *          level 1 fields only ... _
  Include Redefine Base . ___ *          Include Redefines ..... 2__
  Basic occurrences ..... _              Only process 1:V fields _
  Specific 1:V override . ____ ____ ____ ____ ____ ____ ____ ____ ____ ____
                          ____ ____ ____ ____ ____ ____ ____ ____ ____ ____
                          ____ ____ ____ ____ ____ ____ ____ ____ ____ ____
  default 1:V ........... _____
  Skip data areas ....... ________ ________ ________ ________ ________
                          ________ ________ ________ ________ ________
                          ________ ________ ________ ________ ________
                          ________ ________ ________ ________ ________
                          ________ ________ ________ ________ ________
Outputs
  Library ............... ________
  number of parms ....... _____          1:V parms found ....... _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                                                main   ↩
```

Press Enter to display details about the first level 1 field in the data area for CALC:

```
Return to caller ........
Current line number ..... 2               Current Attribute number  1
External data area ......
Structure name ..........
Field name .............. INPUT-DATA
Redef start ............             Field redefined .........
Level number ............ 1          Field format ............
Rank ....................            Field length ............
Found-1-V ..............             Dynamic .................
From index .............
Thru index .............
Field occurrences .......
Start Occurences at .....  1
Occurrences.......


Level type trail . S
From (1)
From (2)
From (3)
To   (1)
To   (2)                                                                     ↩
```

This screen displays details about the INPUT-DATA field and is for display only (i.e., you cannot change the values on the screen). Press Enter again to display the first level 2 field in INPUT-DATA:

```
Return to caller ........
Current line number ..... 3               Current Attribute number  2
External data area ......
Structure name .......... INPUT-DATA
Field name .............. #FUNCTION
Redef start .............             Field redefined .........
Level number ............ 2           Field format ............  A
Rank ....................             Field length ............  30.0
Found-1-V ...............             Dynamic .................
From index ..............
Thru index ..............
Field occurrences .......
Start Occurences at .....  1
Occurrences.......


Level type trail . S          F
From (1)
From (2)
From (3)
To   (1)
To   (2)
```

This screen displays details about the #FUNCTION field. Continue pressing Enter to display every level 1 and under field in the data area. After the last field is displayed, the input screen is redisplayed.

Notes:

1. To return to the input screen at any time, select `Return to caller` and press Enter.
2. To display additional occurrences, specify the occurrence number in the `Start occurrences at` field. Up to 99 occurrences are displayed.

The parameters for CTENPDA2 are:

```
PARAMETER USING CNANPDA
PARAMETER USING CNANPDA2
PARAMETER 01 DATA-PTR(A1/1:V)
PARAMETER USING CU__PDA
PARAMETER USING CSASTD
```

The important inputs when using method 0 are in CNUNPDA. These are:

- CNANPDA.SUBPROGRAM-NAME /* the name of the subprogram from which the parameters will be retrieved

- CNUNPDA.CALLNAT /* the new subprogram mentioned above
- CNUNPDA.METHOD /* should be left as 0

If the non-zero methods do not provide what you need, you can create your own version of CTENPDA2 and do whatever you want with the available variables (the same PDA variables used in CTENPDA2). For example, if you add INCLUDE CU--DFPR to your program, you can use SRC (the editor) as a printer output destination:

```
INCLUDE CU--DFPR
WRITE (SRC) 'Hello' /* this writes Hello to the editor
END
```

For an example of using the CNUNPDA subprogram, refer to CUDRGEN2 in the SYSCST library.

Note: To see how CUDRGEN2 works, run the Natural Debugger and place a break in CUDRGEN2. Then use the Driver model to generate a program.

CNUPEXST Subprogram

CNUPEXST	Description
What it does	Checks for the existence of a map profile. This subprogram receives the name of a map profile and verifies that it exists in the Natural FNAT system file.
PDAs used	■ CNAPEXST
Files accessed	■ SYSTEM-FNAT

Note: This subprogram is not available on all platforms.

Drivers Menu Option

Drivers menu
Natural-Related Drivers menu
Verify Map Profile Existence option

```
 CTEPEXST             N a t u r a l  C o n s t r u c t            CTEPXST1
 Aug 14                Driver for subprogram CNUPEXST                 1 of 1

  Map Profile Name..: ________
  Map Profile Exists:




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CNUSEL Subprogram

CNUSEL	Description
What it does	Selects fields from data areas (local or parameter). This subprogram receives the name of a local (LDA) or parameter data area (PDA) and browses fields in the data area. To select a field, mark it. If more than one field is marked, only the first field is selected. You can enter "X" to terminate the display or "T" to position the list to the top.
PDAs used	■ CNASEL ■ CSASTD
Files accessed	■ None

Drivers Menu Option

```
 CTESEL          ***** Construct Related Subprograms *****        CTESEL1
 Oct 09              - Driver for subprogram CNUSEL -            01:52 PM

 *Data Area Name..: ________  Fld Name:

                                                     Field Occurrences
  Structure Number:           Field Format:          -----------------
  Type Of Field...:           Field Length:
  Level Number....:           Units.......:
  Total Fields Cnt: 0         Decimals....:









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CNUSRCNX Subprogram

CNUSRCNX	Description
What it does	Receives the name of the Natural object and returns the next source line. The first call to the subprogram returns the first source line. Subsequent calls return the next lines.
PDAs used	■ CNASRCNX ■ CNRSRCNX ■ CSASTD **Note:** The CNRSRCNX data area (containing reserved variables) keeps track of the current position of the object source and must not be modified by the calling program.
Files accessed	■ SYSTEM-FUSER ■ SYSTEM-FNAT

Drivers Menu Option

Drivers menu
Natural-Related Drivers menu
Get Next Source Line option

```
 CTESRCNX             N a t u r a l   C o n s t r u c t            CTESRCN1
 Aug 14                Driver for subprogram CNUSRCNX                1 of 1

 *Object Name: CTELRDSM                  Version:
  First Time : X                         Include Comments: _

  Src Line...:      Userid:              Date...:     -  -        Type:
  End Of Src :      Level :              Time...:    .  .  .      SM..:

  Src Code...:



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                mai ↩
```

CNUSRCRD Subprogram

CNUSRCRD	Description
What it does	Reads source text and performs specified processing. This subprogram receives the name of a Natural object (in the CNASRCRD.#OBJECT-NAME field) and the name of the subprogram invoked to process each source line (in the CNASRCRD.#CALLNAT field). It passes the fields it receives to the subprogram it invokes. CU--PDA, which contains the model parameters, is also passed to CNUSRCRD, as well as CSAPASS (redefined as required). It *remembers* information between calls to the subprogram that processes each source line.
PDAs used	■ CNASRCRD ■ CU--PDA (model PDA) ■ CSAPASS (redefined as required) ■ CSASTD
Files accessed	■ SYSTEM-FUSER ■ SYSTEM-FNAT

Drivers Menu Option

Drivers menu
Natural-Related Drivers menu
CALLNAT For Each Source Line option

```
 CTESRCRD             N a t u r a l   C o n s t r u c t            CTESRCR1
 Aug 14                Driver for subprogram CNUSRCRD                1 of 1

 *Object Name: ________          Finished:
 CALLNAT.....: CTESRCSM          Include Comments: _

  Object Information
  ------------------
  Type.......:      Version:       Userid:           Time:    .  .  .
  SM.........:      Level..:                         Date:      -  -

  Src Line...:
  Source Code:
             :

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

Note: If you change the name of the subprogram in the CALLNAT field, the specified subprogram must have the same parameters as those in the PDAs used by CNUSRCRD.

Natural-Related Helproutines (CNH*)

You can attach the helproutines in this section to fields that require the input of Natural information (such as object names, message numbers, etc.). They are active helproutines that populate the field to which they are attached.

CNHMDL Helproutine

CNHMDL	Description
What it does	Browses all the Natural Construct models for selection. Valid restriction parameters are: ■ S (display statement models only) ■ M (display program models only) ■ B (display all models)
Attached to	Input of a Natural Construct model name.

CNHMDL	Description
Parameters used	■ #PDA-RESTRICTION(A1) ■ #PDA-KEY(A32) (model name)
Files accessed	■ NCST-MODEL

CNHMSG Helproutine

CNHMSG	Description
What it does	Browses for and displays the application error message text. You can add new messages to the application by pressing the Add PF-key (the new message number is always adjusted to the next available number).
Attached to	Input of a message number field.
Parameters used	■ #PDA-MESSAGE(A65) ■ #PDA-MESSAGE-LIBRARY(A8) ■ #PDA-KEY(N4)
Files accessed	■ SYSTEM-FUSER

CNHOBJ Helproutine

CNHOBJ	Description
What it does	Browses all objects of a specified type in the current library. This helproutine receives an object type and browses all the objects with that type that exist in the current library. Valid object types are: ■ P (program) ■ N (subprogram) ■ S (subroutine) ■ M (map) ■ H (helproutine) ■ C (copycode) ■ A (parameter) ■ G (global) ■ L (local) ■ T (text) ■ * (all) ■ 2 (subprogram/helproutine)

CNHOBJ	Description
	■ 3 (subprogram/helproutine/subroutine) ■ 4 (program/subprogram/helproutine/subroutine) ■ 5 (command processor) ■ D (data area)
Attached to	Input of a Natural object name field.
Parameters used	■ #PDA-TYPE(A1) ■ #PDA-KEY(A8) /* Start/Return key
Files accessed	■ SYSTEM-FUSER

Natural Construct Generation Utility Subprograms (CSU*)

The subprograms in this section perform specialized functions to assist in the generation process.

Note: Driver programs for many of the supplied programs/subprograms are available through the Drivers menu option on the Administration main menu. If a driver program is available, its location is listed in the *Drivers Menu Option* section in the program/subprogram description. For more information about the supplied driver programs, see ***Drivers Menu Function***.

These subprograms are:

- CSU-VAR Subprogram
- CSUBANN Subprogram
- CSUBLDRP Subprogram
- CSUBMIT Subprogram (Mainframe)
- CSUBYTES Subprogram
- CSUCASE Subprogram
- CSUCCMD Subprogram
- CSUCENTR Subprogram
- CSUCOMPR Subprogram
- CSUCTRL Subprogram
- CSUCURS Subprogram
- CSUCURS1 Subprogram
- CSUDB2SP Subprogram
- CSUDELFF Subprogram
- CSUDEFLT Subprogram
- CSUDYNAT Subprogram
- CSUEMLEN Subprogram

- CSUENDX Subprogram
- CSUFDEF Subprogram
- CSUFRVAR Subprogram
- CSUGEN Subprogram
- CSUHEADS Subprogram
- CSUINCL Subprogram
- CSUIS Subprogram
- CSULABEL Subprogram
- CSULENGT Subprogram
- CSULPS Subprogram
- CSUMAX Subprogram
- CSUMIMAX Subprogram
- CSUMODEL Subprogram
- CSUMORE Subprogram
- CSUMPBOX Subprogram
- CSUMPCPR Subprogram
- CSUMPDUP Subprogram
- CSUMPLAY Subprogram
- CSUMPMMS Subprogram
- CSUMPOVL Subprogram
- CSUMPREG Subprogram
- CSUMPTAB Subprogram
- CSUMPTST Subprogram
- CSUNATFM Subprogram
- CSUNEWX Subprogram
- CSUOG Subprogram
- CSUPARMS Subprogram
- CSUPARTY Subprogram
- CSUPPER Program
- CSUREADS Subprogram
- CSUREF Subprogram
- CSUSCAN Subprogram
- CSUSELFV Subprogram
- CSUSETKY Subprogram
- CSUSETW Subprogram
- CSUSORT Program
- CSUSPLIT Program
- CSUSUB Program (Mainframe)
- CSUSUBP Subprogram
- CSUTEST Program
- CSUTLATE Subprogram
- CSUTRANS Subprogram
- CSUXCHK Subprogram

- CSU2LONG Subprogram

CSU-VAR Subprogram

CSU-VAR	Description
What it does	Validates a specified variable name. This subprogram receives a string and checks for a valid Natural naming convention. Use it whenever a name used as a Natural variable is entered. If the name is invalid, the subprogram returns a message containing the reason. **Note:** The variable name can be fully qualified (contain a prefix).
Parameters used	■ #PDA-STRING(A65) /*INPUT ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Validation Subroutines menu
Validate a Variable Name option

```
 CTE-VAR          ***** Construct Related Subprograms *****        CTE-VAR1
 Oct 09                - Driver for subprogram CSU-VAR -           02:58 PM

String: _________________________________________________________________

Msg...:



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CSUBANN Subprogram

CSUBANN	Description
What it does	Generates the standard banner into the source buffer. Use this subprogram to generate Natural or Visual Basic comments.
PDAs used	■ CSABANN ■ CSASTD
Files accessed	■ None

CSUBLDRP Subprogram

CSUBLDRP	Description

What it does	Builds a report layout. This subprogram builds a report layout for the Batch, Browse, and Browse-Select models. It can be invoked from a sample subprogram within a user exit. The invoking subprogram must issue an initial RESET statement to clear the structures in CSASELFV. For example:
	`RESET CSASELFV`
	`CSASELFV.GENERAL-INFORMATION`
	`CSASELFV.FIELD-SPECIFICATION(*)`
	The sample subprogram must also contain a SET KEY ALL statement.

CSUBLDRP	Description
	For an example of how to invoke the CSUBLDRP utility, refer to the CUSCSRP subprogram in the SYSCST library.
PDAs used	■ CSABLDRP ■ CSASELFV ■ CSASTD
Files accessed	■ None

CSUBMIT Subprogram (Mainframe)

CSUBMIT	Description
What it does	Submits a job for execution. The JCL for the job must be in the source buffer. **Note:** This subprogram is used in conjunction with the CSUSUB command. For more information, refer to *JCL Submit Utility (Mainframe)*, *Natural Construct Generation*.
PDAs used	■ CSASTD
Files accessed	■ None

CSUBYTES Subprogram

CSUBYTES	Description
What it does	Calculates the required bytes for a field, based on the field's Natural format and length. This subprogram receives the length and format of a field and returns the number of bytes occupied by the field.
PDAs used	■ CSABYTES ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
General Utility Subprograms menu
Storage Required for a Variable option

```
 CTEBYTES         ***** Construct Related Subprograms *****        CTEBYTE1
 Feb 25              - Driver for subprogram CSUBYTES -            03:49 PM

Field Format: _        Bytes.......:
Field Length: ____________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CSUCASE Subprogram

CSUCASE	Description

What it does	Converts a string to upper/lower/mixed case. This subprogram receives a string and a desired function. It converts and returns the string as follows:
	■ If the function is "U", this subprogram converts all alpha characters in the string to upper case.
	■ If the function is "L", it converts all alpha characters to lower case.
	■ If the function is "M", it converts the alpha characters as follows:
	  ■ Removes leading hash (#) or plus (+) characters
	  ■ Replaces all dashes (-) and underscores (_) with blanks
	  ■ Converts the first character, as well as all characters following a dash or underscore, to upper case

PDAs used	■ CSACASE ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
General Utility Subprograms menu
Convert Text to Upper/Lower/Mix option

```
 CTECASE               N a t u r a l   C o n s t r u c t           CTECASE1
 Aug 14                 Driver for subprogram CSUCASE                1 of 1

Function: _    U=Upper, L=Lower, M=Mixed Case
String..: _________________________________________________________________

          ___________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai
```

CSUCCMD Subprogram

CSUCCMD	Description
What it does	Generates command block delimiters into the Natural source buffer for super models (generate multiple modules). This subprogram receives a command type, an eight-character module name, a module type, and, optionally, a model name. Natural Construct evaluates the contents of these command blocks after it processes the pre-generation subprogram for the super model. Before continuing the generation, Natural Construct either creates the child model specification or saves, stows, and catalogs the contents of the command block. CSUCCMD must always be called twice — first to initialize the command block and then to close it after generating the contents of the command block. **Note:** 1. See the CSLCCMD local data area for valid command values. 2. You cannot use nested command blocks.
PDAs used	■ CSACCMD ■ CSASTD
Files accessed	■ None

CSUCENTR Subprogram

CSUCENTR	Description
What it does	Centers a text string. This subprogram centers text, such as headings, within a variable. The length passed to this subprogram should be one of the following: ■ the length of the variable that stores the heading ■ the length of the AL parameter that displays the variable that stores the heading
PDAs used	■ CSACENTR ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Text Related Subprograms menu
Center a Text String option

```
 CTECENTR             N a t u r a l   C o n s t r u c t            CTECNTR1
 Aug 14                Driver for subprogram CSUCENTR                1 of 1

Length: ___

String: _________________________________________________________________
        ___________________________________

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

CSUCOMPR Subprogram

CSUCOMPR	Description
What it does	Generates an IF clause for two structures. The subprogram receives two structure names and a list of underlying components to compare. It generates the IF clause according to the criteria requested (LT, LE, GT, GE). **Note:** DB2 users should use the CSUDB2SP subprogram to compare key values (see *CSUDB2SP Subprogram* for a description).
PDAs used	■ CSACOMPR CSASTD ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
DB2 Related Subprograms menu
Generate an IF for a Superdescriptor option

```
 CTECOMPR             N a t u r a l   C o n s t r u c t              CTECOMP1
 Aug 14                Driver for subprogram CSUCOMPR                  1 of 1

Comparison Operator.: __     Lhs Structure: ________________________________
Tab.................: ___    Rhs Structure: ________________________________
No. Of Components...: ___
                               Component Fld Name
                      +-----------------------------------+
                  1__ | ________________________________ |
                      | ________________________________ |
                      | ________________________________ |
                      | ________________________________ |
                      | ________________________________ |
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                    bkwrd frwrd                   mai
```

CSUCTRL Subprogram

CSUCTRL	Description
What it does	Retrieves information from the Natural Construct control record and sets the PF-keys, help indicator, underscore characters, position indicators, disable indicator, scroll indicator, "of" right prompt, and dynamic attributes for Natural Construct.
PDAs used	■ CU--PDA ■ CSASTD
Files accessed	■ NCST-CONTROL

CSUCURS Subprogram

CSUCURS	Description
What it does	Determines the position of the field in which the cursor is placed. This subprogram is invoked when runtime translation is requested. It determines the message numbers and positions associated with fields in a translation LDA and invokes the CSUTLATE subprogram to perform runtime translation. For more information, see ***CSUTLATE Subprogram***.
Parameters/PDAs used	■ #TRANSLATION-DATA(A1/1:V) ■ #SYSERR-APPL(A8) ■ #DATA-AREA-NAME(A8) ■ #TEXT-REQUIRED(L) ■ #LENGTH-OVERRIDE(I4) ■ CSACURS ■ CSASTD
Files accessed	■ None

CSUCURS1 Subprogram

CSUCURS1	Description
What it does	Determines the position of a single field in which the cursor is placed. This subprogram is invoked whenever runtime translation of a single field is requested. It determines the message number and position associated with the field and invokes the CSUTLATE subprogram to perform runtime translation. For more information, see ***CSUTLATE Subprogram***.
Parameters/PDAs used	■ #TRANSLATION-DATA(A1/1:V) ■ #SYSERR-APPL(A8) ■ CSASTD
Files accessed	■ None

CSUDB2SP Subprogram

CSUDB2SP	Description

What it does	Generates a FIND statement for a superdescriptor. This statement retrieves DB2 records based on a complex key definition. If a complex key is composed of 5 fields (Field1, Field2, Field3, Field4, and Field5), for example, the generated FIND/WHERE clause is:
	
	`Field1 GE #INPUT.Field1`
	`SORTED BY Field1`
	`          Field2`
	`          Field3`
	`          Field4`
	`          Field5`
	`WHERE Field2 GE #INPUT.Field2`
	`AND Field3 GE #INPUT.Field3`
	`AND Field4 GE #INPUT.Field4`
	`AND Field5 GE #INPUT.Field5`
	`OR Field1 GE #INPUT.Field1`
	`AND Field2 GE #INPUT.Field2`
	`AND Field3 GE #INPUT.Field3`
	`AND Field4 GT #INPUT.Field4`
	`OR Field1 GE #INPUT.Field1`
	`AND Field2 GE #INPUT.Field2`
	`AND Field3 GT #INPUT.Field3`
	`OR Field1 GE #INPUT.Field1`
	`AND Field2 GT #INPUT.Field2`
	`OR Field1 GT #INPUT.Field1`
	
	Note: #INPUT is the qualifier for the RHS fields of the in equations.

PDAs used	■ CSADB2SP ■ CU--PDA ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
DB2 Related Subprograms menu
Generate a FIND for a Superdescriptor option

```
 CTEDB2SP             N a t u r a l   C o n s t r u c t             CTEDB2S1
 Aug 14                Driver for subprogram CSUDB2SP                 1 of 1

*File Name.........: ________________________________   Find Next Record: _
*Field Name........: ________________________________
 Function..........: ________________________________

 LHS Structure.....: ________________________________
 LHS Index.........: _______________
 RHS Structure.....: ________________________________
 RHS Index.........: ________________________________

 Prefix Length.....: ___
 Low Key Structure : ________________________________
 High Key Structure: ________________________________

 Tab...............:
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai
```

CSUDELFF Subprogram

CSUDELFF	Description
What it does	Deletes the lines containing */ in the edit buffer. This subprogram searches for and deletes the lines containing */ in the edit buffer. These lines are written by WRITE/PRINT statements when the DEFINE PRINTER OUTPUT 'SOURCE' statement is used.
PDAs used	■ None
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Edit Buffer Related Subprograms menu
Delete Lines Containing */ option

```
 CTEDELFF             N a t u r a l   C o n s t r u c t             CTEMAP1
 Aug 14                Driver for subprogram CSUDELFF                 1 of 1

                  +-------------------------------------+
                  |                                     |
                  |       PRESS ENTER TO EXECUTE.       |
                  |                                     |
                  +-------------------------------------+

                      Read in New Source: _
                     *New Source Name...: ________
                      New Source Library: DEVEX___

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

CSUDEFLT Subprogram

CSUDEFLT	Description
What it does	Provides default specification values for Natural Construct models. This subprogram provides an interface between generated applications and the user-maintained CSXDEFLT sample exit subprogram. To override the default settings, modify CSXDEFLT. The CCDEFLTA, CCDEFLTL, and CCDEFLTN copycode members return defaults for alphanumeric, logical, and numeric values, respectively.
PDAs used	■ CSADEFLT ■ CSASTD
Files accessed	■ None

CSUDYNAT Subprogram

<table>
<tr><th>CSUDYNAT</th><th>Description</th></tr>
<tr><td>What it does</td><td>Builds parameters containing dynamic attributes. This subprogram receives a set of dynamic attribute characters in the CSADYNA.#ATTRIBUTE-CHARS(A11/1:13) field and builds the definition for the DY= parameter. The positioning within this array indicates the type of dynamic attribute assigned. The positions and attributes are:

1 (normal intensity)
2 (intensified)
3 (blinking)
4 (cursive/italic)
5 (underlined)
6 (reversed video)
7 (blue)
8 (green)
9 (neutral/white)
10 (pink)
11 (red)
12 (turquoise)
13 (yellow)

For example, if you enter:
<pre>#ATTRIBUTE-CHARS(1) = '}'
#ATTRIBUTE-CHARS(2) = '{'</pre>
This subprogram returns:
<pre>#DY-PARAMETER = DY={I</pre>
If the caller’s attributes are printable special characters, they are represented literally. Otherwise, they are represented using the HH syntax.
Note:

The dynamic attribute character specified in position 1, which corresponds to normal intensity, is always coded at the end of the DY= parameter.
Programs containing those represented in hex may not be portable.
</td></tr>
<tr><td>PDAs used</td><td>CSADYNAT
CSASTD</td></tr>
<tr><td>Files accessed</td><td>None</td></tr>
</table>

Drivers Menu Option

Drivers menu
Additional Drivers menu
Text Related Subprograms menu
Build Dynamic Attribute option

```
 CTEDYNAT             N a t u r a l   C o n s t r u c t             CTEDYNT1
 Aug 14                Driver for subprogram CSUDYNAT                  1 of 1

                            Attribute Characters
                            --------------------

    (1) Normal Intensity..: _                 (8) Green............: _
    (2) Intensified.......: _                 (9) Neutral (white)...: _
    (3) Blinking..........: _                (10) Pink.............: _
    (4) Cursive/Italic....: _                (11) Red..............: _
    (5) Underlined........: _                (12) Turquoise........: _
    (6) Reversed Video....: _                (13) Yellow...........: _
    (7) Blue..............: _

    Dynamic Attribute Parameter:

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CSUEMLEN Subprogram

CSUEMLEN	Description
What it does	Determines the number of characters (bytes) required to display an edit mask. This subprogram receives the name of an edit mask and the format of the field to which the edit mask is applied. It returns the number of characters (bytes) required to display the edit mask.
PDAs used	■ CSAEMLEN ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Text Related Subprograms menu
Calculate Bytes to Display Emask option

```
 CTEEMLEN             N a t u r a l   C o n s t r u c t            CTEMLEN1
 Aug 14                Driver for subprogram CSUEMLEN                1 of 1

Edit Mask.....: ______________________________________________________
Field Format..: __

Display Length:


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

CSUENDX Subprogram

CSUENDX	Description
What it does	Generates the end of a user exit prompt. This subprogram is used by sample subprograms that generate multiple user exits. Call this subprogram after each user exit is generated. **Note:** You do not need to call this subprogram after the last user exit.
PDAs used	■ None
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
User Exit Related Subprograms menu
Generate End to User Exit option

```
 CTEENDX              N a t u r a l   C o n s t r u c t              CTEMAP1
 Aug 14                  Driver for subprogram CSUENDX                 1 of 1

                    +-------------------------------------+
                    |                                     |
                    |       PRESS ENTER TO EXECUTE.       |
                    |                                     |
                    +-------------------------------------+

                        Read in New Source: _
                       *New Source Name...: ________
                        New Source Library: DEVEX___

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CSUFDEF Subprogram

CSUFDEF	Description
What it does	Validates a field definition. This subprogram receives the Natural format and length of a field and a list of invalid field formats to disallow. To disallow control variables as input variables, for example, list the invalid formats in the CSAFDEF.#INVALID FORMATS field. If the field definition is valid, nothing is returned in CSUFDEF. If the field definition is invalid, CSASTD.MSG and CSASTD.ERROR-FIELD contain an error message and the invalid component of the field (FIELD-FORMAT, DECIMALS, or UNIT).
PDAs used	■ CSAFDEF ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Validation Subroutines menu
Validate Field Definition option

```
 CTEFDEF               N a t u r a l   C o n s t r u c t            CTEFDEF1
 Aug 14                 Driver for subprogram CSUFDEF                 1 of 1

Field Format...: _           Invalid Formats: __________
Field Length...: _____


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

CSUFRVAR Subprogram

CSUFRVAR	Description
What it does	Returns the parameters and conditions from the model code frames. This subprogram receives a model name and traverses its code frames. It returns the code frame parameters and conditions.
PDAs used	■ CSAFRVAR ■ CSASTD
Files accessed	■ NCST-FRAME-LINES ■ NCST-MODEL

Drivers Menu Option

Drivers menu
Additional Drivers menu
NCST Model/Frame Related Subprograms menu
Model Code Frame Information option

```
   CTEFRVAR              N a t u r a l   C o n s t r u c t              CTEFRVR1
   Aug 14                 Driver for subprogram CSUFRVAR                  1 of 1

   *Model Name: ________________________________

   No. Of Conditions : 0
   No. Of Frame Parms: 0
   +----------------------------------------+---------------------------------+
   |  1__        Conditions                 |      1__   Frame Parameters     |
   |  --------------------------------      |     --------------------------  |
   |                                        |                                 |
   |                                        |                                 |
   |                                        |                                 |
   |                                        |                                 |
   |                                        |                                 |
   +----------------------------------------+---------------------------------+


  Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
        help  retrn quit                        bkwrd frwrd                  mai
```

CSUGEN Subprogram

CSUGEN	Description
What it does	Issues a CALLNAT to the Natural Construct Generate function for a specified module. This subprogram receives the names of a model PDA and a model information PDA (CSAMODEL, which must contain the name of the model) and uses the inputs to generate the module code into the Natural source buffer. When the CALLNAT is made to the module, the code is appended to the contents of the source buffer. The source buffer name or type does not change. **Note:** 1. The specified model PDA must contain the model parameters required for generation. 2. This subprogram requires a NATPARM SSIZE of 55 or greater.
PDAs used	■ CSAGEN ■ CSAMODEL

CSUGEN	Description
	■ CU--PDA ■ CSASTD
Files accessed	■ NCST-ADA

CSUHEADS Subprogram

CSUHEADS	Description
What it does	Separates a line of headings into separate headings. This subprogram receives a line of headings and returns three separate headings (each with the length of longest heading).
PDAs used	■ CSAHEADS ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Text Related Subprograms menu
Separate a Line of Headings option

```
 CTEHEADS             N a t u r a l   C o n s t r u c t              CTEHEAD1
 Aug 14                Driver for subprogram CSUHEADS                  1 of 1

Headings: ______________________________   Field Headings Stacked
                                           ----------------------
Field Heading Width: 0



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CSUINCL Subprogram

CSUINCL	Description
What it does	Inserts the source for all copycode (currently in the edit buffer) into the edit buffer.
PDAs used	■ None
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Edit Buffer Related Subprograms menu
Expand All Copy Codes option

```
 CTEINCL              N a t u r a l   C o n s t r u c t              CTEMAP1
 Aug 14                  Driver for program CSUINCL                   1 of 1

                    +-------------------------------------+
                    |                                     |
                    |        PRESS ENTER TO EXECUTE.      |
                    |                                     |
                    +-------------------------------------+

                         Read in New Source: _
                        *New Source Name...: ________
                         New Source Library: DEVEX___

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                mai ↩
```

CSUIS Subprogram

CSUIS	Description
What it does	Verifies whether the contents of an alphanumeric field can be converted to a specified format and length. If the format and length are invalid Natural formats, CSASTD.MSG contains an error message when this subprogram is invoked. If the format and length are valid, CSASTD.MSG is blank. In some cases, a user must specify a value using a certain (variable) format and length. For example, the minimum/maximum key values should be valid values corresponding to the format and length of the key. You cannot use the Natural IS function because the format is not known until runtime.
PDAs used	■ CSAIS ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Validation Subroutines menu
Validate Format for Input Value option

```
 CTEIS                  N a t u r a l   C o n s t r u c t            CTEIS1
 Aug 14                   Driver for subprogram CSUIS                1 of 1

  Field Value.: __________________________________________________
  Field Format: _
  Field Length: ___

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                mai ↩
```

CSULABEL Subprogram

CSULABEL	Description
What it does	Verifies a Natural looping label. This subprogram receives a string of characters and validates it against the Natural label naming convention. ; if the label is not valid, CSASTD.MSG contains an error message. ■ If the label is valid, CSASTD.MSG is blank ■ If the label is not valid, CSASTD.MSG contains an error message
Parameters/PDAs used	■ #PDA-LABEL(A32) ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Validation Subroutines menu
Validate a Label option

```
 CTELABEL              N a t u r a l   C o n s t r u c t              CTELABL1
 Aug 14                 Driver for subprogram CSULABEL                  1 of 1

  Label: ________________________________

  Msg..: __________________________________________________________________


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                   mai ↩
```

CSULENGT Subprogram

CSULENGT	Description
What it does	Builds an input prompt and calculates the length of the heading. This subprogram receives a field name, format, and length. It builds the input prompt from the field headings (if no heading was given, the field name is converted to mixed case) and calculates the length from the format, length, and edit mask. It also returns the heading length and sign option (based on the field format and edit mask).
PDAs used	■ CSALENGT ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Text Related Subprograms menu
Calculate Length of a Heading option

```
 CTELENGT             N a t u r a l   C o n s t r u c t           CTELNGT1
 Aug 14                Driver for subprogram CSULENGT                1 of 1

Field Name....: ________________________________   Field Length....: ______
Field Headings: ______________________________     Field Format....: _
              : ______________________________     Sign............: _
              : ______________________________

Edit Mask.....: ______________________________________________________

Input Prompt..:                                    Heading Length..:
Sg Option.....:                                    Fld Displ Length:

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CSULPS Subprogram

CSULPS	Description
What it does	Changes the display language (*Language value) and sets the translation required flag to True. This subprogram displays a list of all available languages supported by Natural. When a new language is selected, it switches the user's session to that language and sets the translation required flag to True.
Parameter/PDAs used	■ #PDA-TRANSLATION-REQUIRED (L) ■ CSASTD
Files accessed	■ SYSDIC-FI

CSUMAX Subprogram

CSUMAX	Description
What it does	Generates the assignment of a maximum value for a field. This subprogram receives the name, format, and length of a variable and generates the assignment of the maximum value for the field into the edit buffer. It is used when reading a file for all values with a specified prefix, where the suffix extends from the lowest to the highest value.
PDAs used	■ CSAMAX ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Edit Buffer Related Subprograms menu
Generate Assign of Max Field Val option

```
   CTEMAX               N a t u r a l   C o n s t r u c t           CTEMAX1
   Aug 14                 Driver for subprogram CSUMAX                1 of 1

  Field : _________________________________________________________________
  Format: _
  Length: _____
  Tab...: __

  Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
        help  retrn quit                                                 mai ↩
```

CSUMIMAX Subprogram

CSUMAX	Description
What it does	Generates the assignment of a minimum value for a field. This subprogram receives the name of a variable and its format and length. It generates the assignment of the minimum/maximum values for the field into the edit buffer.
PDAs used	■ CSAMIMAX ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Edit Buffer Related Subprograms menu
Generate Assign of Min Field Val option

```
    CTEMIMAX             N a t u r a l   C o n s t r u c t            CTEMIMX1
  Aug 14               Driver for subprogram CSUMIMAX                   1 of 1

 Field : _________________________________________________________________

 Format: __        Minimum Value: _     Non Negative Min/Max: _     Tab: __
 Length: _____     Descending...: _     DB2 Date/Time Stamp : _


 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
       help  retrn quit                                                 mai ↩
```

CSUMODEL Subprogram

CSUMORE	Description
What it does	Returns information about a Natural Construct model. This subprogram receives the name of a model and returns the model description, generator mode and type, and the names of the model PDA, subprograms, and code frames.
PDAs used	■ CSAMODEL ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
NCST Model/Frame Related Subprograms menu
NCST Models Information option

```
 CTEMODEL             N a t u r a l   C o n s t r u c t             CTEMODL1
 Aug 14                Driver for subprogram CSUMODEL                 1 of 1

 *Model Name.......: ________________________________
  Model Description:

  No. Modify Subps:      Modify Subps  Code Frames  Clear Subp...:
  No. Code Frames :      ------------  -----------  Read Subp....:
  Generator Mode..:                                 Save Subp....:
  Generator Type..:                                 Pre-Gen Subp.:
  Display Window..:                                 Post-Gen Subp:
  Start Comment...:                                 Doc Subp.....:
  End Comment.....:                                 Pda Name.....:




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CSUMORE Subprogram

CSUMORE	Description

What it does	Builds the initial assignment for the LEFT-MORE/RIGHT-MORE array. This subprogram receives a function (L for the LEFT-MORE array, R for the RIGHT-MORE) and the number of panels used by a program. These arrays contain the prompts displayed at the top left or right corner of the panels. The prompts indicate the number of panels located to the left or right of the current panel.
	
	For example, to generate the initial value for the LEFT-MORE-PROMPT array for a program with two panels, enter:
	
	`CSAMORE.#LEFT-RIGHT = 'L'`
	`CSAMORE.#MAX-WINDOW = 2`
	
	The subprogram writes the following to the source buffer:

CSUMORE	Description

	`INIT < ' ','<1 more' >`
	
	To generate the initial value for the RIGHT-MORE-PROMPT array for a program with two panels, enter:
	
	`CSAMORE.#LEFT-RIGHT = 'R'`
	
	The subprogram writes the following to the source buffer:
	
	`INIT < '1 more >','' >`
	
	Note: If the value of *Language is not 1 during generation, the word "more" is not included in the initial values.
	
	Tip: Use a scalar field rather than an occurrence of this array. Before the map is displayed, assign the array occurrence to the scalar field. Using arrays on maps makes them difficult to maintain and less suitable to use as standard layouts.

PDAs used	■ CSAMORE ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Edit Buffer Related Subprograms menu
Build Left/Right/More Prompt option

```
 CTEMORE              N a t u r a l   C o n s t r u c t              CTEMORE1
 Aug 14                 Driver for subprogram CSUMORE                  1 of 1

Left/Right:  _ (L or R)
Max Windows: __


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                mai ↩
```

Note: For more information on changing the size of the left or right prompt, see ***Use CSXDEFLT Overrides***.

CSUMPBOX Subprogram

CSUMPBOX	Description
What it does	Handles the map edit buffer. This subprogram receives a function and parameters (in CSAMPBOX). It initializes the map edit buffer or generates variable, array, and text control blocks into the edit buffer.
PDAs used	■ CSAMPBOX ■ CSASTD
Files accessed	■ None

CSUMPCPR Subprogram

CSUMPCPR	Description
What it does	Replaces the map settings in the edit buffer with values from the CSAMPSET parameter data area.
PDAs used	■ CSAMPSET ■ CSASTD
Files accessed	■ None

CSUMPDUP Subprogram

CSUMPDUP	Description
What it does	Checks for the duplication of fields on a map. This subprogram determines whether there are any fields duplicated in the CSAMPFLD.FIELD-INFO(*) structure. If there are duplicate fields, CSASTD.MSG contains an error message when this subprogram is invoked.
PDAs used	■ CSAMPFLD ■ CSASTD
Files accessed	■ None

CSUMPLAY Subprogram

CSUMPLAY	Description
What it does	Loads the map layout into the edit buffer and returns the map settings. This subprogram receives the name, layout, and type of map and loads the specified map into the edit buffer. It returns the map settings.
PDAs used	■ CSAMPSET ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Map Related Subprograms menu
Load Map Layout and Settings option

```
 CTEMPLAY              N a t u r a l   C o n s t r u c t             CTEMPLY1
 Aug 14                 Driver for subprogram CSUMPLAY                 1 of 1

*Layout..: ________        Error Code :            Dc:     Zp............:
                           Map Version:            Ps:     Pm............:
                           Profile....:            Ls:     Cursor Skip...:

 Delimiter Class..:                                        Std Keys......:
 Ad...............:                                        Justification :
 Delimiter Char...:                                        Col Shift.....:
 Cd...............:                                        Case Deflt....:

 Write Statement..:        CV.........:                    Auto Rule Rank:
 Input Statement..:        Filler Char:                    Enforce Attr..:

 Help.............:
 Help-As-Fld-Deflt:

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                mai ↩
```

CSUMPMMS Subprogram

CSUMPMMS	Description
What it does	Merges the settings for two maps. This subprogram merges the map settings from CSAMPSET and CSAMPOUT. The settings in CSAMPSET override the settings in CSAMPOUT and the result is stored in CSAMPOUT.
PDAs used	■ CSAMPSET ■ CSAMPOUT
Files accessed	■ None

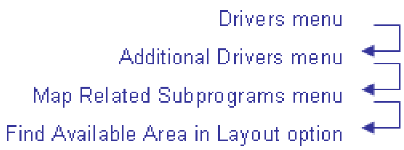
CSUMPOVL Subprogram

CSUMPOVL	Description
What it does	Checks the boundary on a map and determines if there are overlapping fields. This subprogram checks whether the fields specified in CSAMPFLD exceed the line size or page size of the available map panel. The available map panel is a block of consecutive lines on the panel. This block is determined by the specified page and line size, excluding the map layout and any PF-keys. The fields on the map cannot overlay the layout or PF-keys.
PDAs used	■ CSAMPFLD ■ CSASTD
Files accessed	■ None

CSUMPREG Subprogram

CSUMPREG	Description
What it does	Determines the available map area in a map layout. This subprogram determines the first and last line on a map that is available for editing in a specified map layout.
PDAs used	■ CSAMPREG ■ CSASTD
Files accessed	■ None

Drivers Menu Option




```
 CTEMPREG             N a t u r a l   C o n s t r u c t            CTEMPRG1
 Aug 14                Driver for subprogram CSUMPREG                1 of 1


 *Layout: ________   First Available Line:         Layout Page Size:
                     Last  Available Line:         Layout Line Size:




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CSUMPTAB Subprogram

CSUMPTAB	Description
What it does	Calculates the absolute field coordinates on a map and creates the field prompts. This subprogram receives field information from CSAMPFLD and returns the absolute field positions and prompts in CSAMPX-Y. Dots are added to each field prompt in a region to extend its length to that of the longest prompt in that region (... for ISA format and . . . for SAA format). **Note:** For more information about the data returned, refer to the CSAMPX-Y data area in the SYSCST library.
PDAs used	■ CSAMPFLD ■ CSAMPX-Y ■ CSASTD
Files accessed	■ None

CSUMPTST Subprogram

CSUMPTST	Description
What it does	Tests the specifications for the map currently in the edit buffer.
PDAs used	■ CSAMPTST ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Map Related Subprograms menu
Test Map Specifications option

```
 CTEMPTST             N a t u r a l   C o n s t r u c t             CTEMTST1
 Aug 14                Driver for subprogram CSUMPTST                   1 of 1

   Read in New Map: _            Page Size: 23_
  *Map Name.......: ________     Line Size: 80_
   Map Library....: DEVEX___

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                mai ↩
```

CSUNATFM Subprogram

CSUNATFM	Description

What it does	Builds a valid Natural format definition from the formats and lengths specified. This subprogram receives the format and length values and combines these to build a valid Natural format string. For example, if you enter:
	`CSANATFM.FIELD-LENGTH = 9.0`
	`CSANATFM.FIELD-FORMAT = 'P'`
	CSUNATFM produces the following output:
	`CSANATFM.#Natural-FORMAT = P9`

PDAs used	■ CSANATFM ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
General Utility Subprograms menu
Build Natural Format option

```
 CTENATFM         ***** Construct Related subprograms *****        CTENTFM
 Feb 25               - Driver for subprogram CSUNATFM -           03:50 PM

  Field Format: _          NATURAL Format:
  Field Length: ____________


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CSUNEWX Subprogram

CSUNEWX	Description
What it does	Generates a new user exit prompt. This subprogram receives the name of a user exit and generates a starting point (DEFINE EXIT *exit-name*, for example) for the user exit. It initiates a new user exit for sample subprograms that are capable of generating more than one exit.
PDAs used	■ CSANEWX
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
User Exit Related Subprograms menu
Generate New User Exit Prompt option

```
CTENEWX          N a t u r a l   C o n s t r u c t       CTENEWX1
Aug 14             Driver for subprogram CSUNEWX           1 of 1

                User Exit Name: ________________________________

  Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--
        help  retrn quit                                             ↩
```

CSUOG Subprogram

CSUOG	Description

What it does	Comments out all code within a specified user exit. This subprogram receives the name of a user exit and inserts comment indicators at the beginning of each line of code within the specified exit.
	
	Specify the name of the user exit in the #USER-EXIT (A65) variable. For example, to comment out all code within the MOVE-TO user exit, specify the following:
	
	`0040   01 #USER-EXIT (A65)`
	`.`
	`.`
	`.`
	`3800     #USER-EXIT := 'MOVE-TO'`
	`3810     CALLNAT 'CSUOG' #USER-EXIT`

PDAs used	■ CSAOG
Files accessed	■ None

CSUPARMS Subprogram

CSUPARMS	Description
What it does	Returns the value of a NATPARM parameter. This subprogram receives a NATPARM parameter and returns its corresponding value. Valid NATPARM parameters are: ■ CF ■ DC ■ IA ■ ID ■ KD ■ ML ■ TB ■ UL **Note:** For information about INPUT/OUTPUT parameters, refer to the CSAPARMS data area in the SYSCST library.
PDAs used	■ CDUPARMA ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
General Utility Subprograms menu
Get a NATPARM Parameter Value option

```
 CTEPARMS             N a t u r a l   C o n s t r u c t             CTEPARM1
 Aug 14                Driver for subprogram CSUPARMS                  1 of 1

  Parameter....: __  (ID,CF,UL,TB,IA,DC,KD,ML)
  Alpha Value..:
  Numeric Value:





Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CSUPARTY Subprogram

CSUPARTY	Description
What it does	Determines Natural data types and returns the byte length. This subprogram receives the format and length for a data type and indicates whether it is a valid Natural data type. If it is, this subprogram returns the byte length.
PDAs used	■ CSAPARTY ■ CSASTD
Files accessed	■ None

CSUPPER Program

CSUPPER	Description
What it does	Converts the contents of the source buffer into upper case. This program reads through the source buffer and converts specified lower case characters into upper case.
PDAs used	■ None
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Edit Buffer Related Subprograms menu
Convert Source to Upper Case option

```
 CTEPPER              N a t u r a l   C o n s t r u c t            CTEMAP1
 Aug 14                  Driver for program CSUPPER                 1 of 1

                    +-------------------------------------+
                    |                                     |
                    |        PRESS ENTER TO EXECUTE.      |
                    |                                     |
                    +-------------------------------------+

                         Read in New Source: _
                        *New Source Name...: ________
                         New Source Library: DEVEX___

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                  mai ↩
```

CSUREADS Subprogram

CSUREADS	Description
What it does	Reads the specification parameters for a module. This subprogram receives the name of a source module. If the module was generated using Natural Construct, the subprogram reads the source code and returns the model parameter data area (PDA) containing the parameters used to generate the module. You can use the passed model PDA to call the model subprograms for the model used to generate the module. This subprogram also returns a data area describing the model and listing the names of the model subprograms. **Note:** This subprogram requires a NATPARM SSIZE of 55 or greater.
Parameters/PDAs used	■ #READ-THIS-MODULE(A8) ■ CSAMODEL ■ CU--PDA ■ CSASTD
Files accessed	■ NCST-ADA ■ SYSTEM-FUSER

Tip: If you know the name of the model used to generate the specified module, you can pass its model PDA to CSUREADS rather than CU--PDA. After the call to CSUREADS, the model PDA is populated with the parameters used to generate that module.

CSUREF Subprogram

CSUREF	Description
What it does	Generates referential integrity checks against foreign files. This subprogram is typically called three times: once to generate the data structures (DATA) required by the generated code, once to generate the update edits (UPDATE), and once to generate the delete edits (DELETE). Set the value of CSAREF.FUNCTION-CODE to either DATA, UPDATE, or DELETE. After the first call, this subprogram returns the number of update and delete edits found. This avoids unnecessary subsequent calls.
PDAs used	■ CSAREF ■ CU--PDA ■ CSASTD
Files accessed	■ SYSDIC-RL ■ SYSDIC-FI

CSUSCAN Subprogram

CSUSCAN	Description
What it does	Scans for the existence of a string in the edit buffer. This subprogram receives a string and scans for (not absolute) the existence of the string in the edit buffer.
PDAs used	■ CSASCAN
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Edit Buffer Related Subprograms menu
Scan for Existence of a String option

```
 CTESCAN               N a t u r a l   C o n s t r u c t          CTESCAN1
 Aug 14                  Driver for subprogram CSUSCAN                1 of 1

  String..: ________________________________________________________________
  Absolute: _ (Mark if scan string need not be delimited by special chars)
  Found...: _

  Read in New Source: _
 *New Source Name...: ________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                   mai ↩
```

CSUSELFV Subprogram

<table>
<tr><th>CSUSELFV</th><th>Description</th></tr>
<tr><td>What it does</td><td>Selects fields/variables from views, LDAs, or PDAs. This subprogram selects up to 40 fields/variables from up to 6 different views, LDAs, or PDAs and appends the selected fields/variables to CSASELFV. Existing fields/variables in CSASELFV cannot be re-selected.

When selecting from data areas, you cannot select the following:

■ constants
■ more than one structure

If you specify the select all option, then the first structure in the data area is selected.

The invoking subprogram should issue an initial RESET statement to clear the structures in CSASELFV, such as:

RESET CSASELFV
CSASELFV.GENERAL-INFORMATION
CSASELFV.FIELD-SPECIFICATION(*)</td></tr>
<tr><td>PDAs used</td><td>■ CSASELFV
■ CSASTD</td></tr>
<tr><td>Files accessed</td><td>■ None</td></tr>
</table>

CSUSETKY Subprogram

CSUSETKY	Description
What it does	Returns PF-key definitions from the control record to support variable PF-keys in Natural Construct. The PF-key names are returned in the CSASETKY.#PF-NAME(*) array. The index for each array element corresponds to the PF-key number. The following example indicates that PF1 is named "help": `#PF-NAME(1) = 'help'`
PDAs used	■ CSASETKY ■ CSASTD
Files accessed	■ NCST-CONTROL

Drivers Menu Option

Drivers menu
Additional Drivers menu
General Utility Subprograms menu
Find PF Key Related Information option

```
   CTESETKY              N a t u r a l   C o n s t r u c t            CTESETK1
   Sep 07                 Driver for subprogram CSUSETKY                  1 of 1

            Pf Name             Pf Number                   Pf Key
            -------             ---------                   ------
             main             Main......: 12             Pf Main......: PF12
             retrn            Return....: 2              Pf Return....: PF2
             quit             Quit......: 3              Pf Quit......: PF3
             test             Test......: 4              Pf Test......: PF4
             bkwrd            Backward..: 7              Pf Backward..: PF7
             frwrd            Forward...: 8              Pf Forward...: PF8
             left             Left......: 10             Pf Left......: PF10
             right            Right.....: 11             Pf Right.....: PF11
             help             Help......: 1              Pf Help......: PF1
                              Available1: 5              Pf Available1: PF5
                              Available2: 6              Pf Available2: PF6
                              Available3: 9              Pf Available3: PF9

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

CSUSETW Subprogram

CSUSETW	Description

What it does	Returns the SET CONTROL parameters to define a window. This subprogram receives the parameters for a window (such as frame, line size, column size, base line, and base column). It returns the SET CONTROL parameters to define the window. For example, if the parameters are:
	`CSASETW.FRAME=TRUE`
	`CSASETW.LINE-SIZE=70`
	`CSASETW.COLUMN-SIZE=5`
	This subprogram returns:

CSUSETW	Description
	`CSASETW.SET-CONTROL.PARM='WBFL70C5'`
PDAs used	■ CSASETW ■ CSASTD
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Text Related Subprograms menu
Build Window Settings option

```
 CTESETW              N a t u r a l   C o n s t r u c t            CTESETW1
 Aug 14                 Driver for subprogram CSUSETW                 1 of 1

  Frame......: _ Line Size..: ___  Base Line..: ___  Required Width : ___
                 Column Size: ___  Base Column: ___  Required Height: ___

  Set Control Parm:


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                mai ↩
```

CSUSORT Program

CSUSORT	Description
What it does	Sorts a 2-dimensional array based on specified column positions. This subprogram receives a 2-dimensional array and sorts the array based on the desired column positions. A Natural SORTSIZE is not required because the sort uses an internal bubble sort algorithm. **Note:** For an example of how to call this subprogram, refer to the CSASORT data area.
Parameters/PDAs used	■ CSASORT ■ #SORT-DATA(A1/1:V,1:V) ■ CSASTD
Files accessed	■ None

CSUSPLIT Program

CSUSPLIT	Description
What it does	Splits lines in the source buffer that are longer than 72 characters. Only lines with code extending beyond column 72 are split; lines with comments extending beyond column 72, but not code, are ignored. If a text string (enclosed within quotes) extends beyond column 72, the entire string is moved to the next line.
PDAs used	■ None
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
Edit Buffer Related Subprograms menu
Split Long Lines in Edit Buffer option

```
 CTESPLIT             N a t u r a l   C o n s t r u c t          CTEMAP1
 Aug 14                 Driver for program CSUSPLIT                1 of 1

                     +-------------------------------------+
                     |                                     |
                     |       PRESS ENTER TO EXECUTE.       |
                     |                                     |
                     +-------------------------------------+

                          Read in New Source: _
                         *New Source Name...: ________
                          New Source Library: DEVEX___

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

CSUSUB Program (Mainframe)

CSUSUB	Description
What it does	Submits a job for execution. The JCL for the job must be in the source buffer. This subprogram is used in conjunction with the CSUSUB command. For information, refer to *JCL Submit Utility (Mainframe)*, *Natural Construct Generation*.
PDAs used	■ None
Files accessed	■ None

CSUSUBP Subprogram

CSUSUBP	Description
What it does	Returns information about a Natural Construct model subprogram, such as the PF-key settings and the window sizes. This subprogram receives the name of a model subprogram and returns information about that subprogram. The information corresponds to the data accessed through the Maintain Subprograms function. **Note:** For more information, see ***Maintain Subprograms Function***.
PDAs used	■ CSASUBP

CSUSUBP	Description
	■ CSASTD
Files accessed	■ NCST-SUBPROGRAM

Drivers Menu Option

Drivers menu
Additional Drivers menu
NCST Model/Frame Related Subprograms menu
NCST Subprograms Information option

```
 CTESUBP              N a t u r a l   C o n s t r u c t            CTESUBP1
 Aug 15                 Driver for subprogram CSUSUBP                 1 of 1

 Subprogram Name: ________
 Description....:

 Backward Forward Flag:    Window Length :      Key Name   No. Other Keys: _
 Left Right Flag......:    Window Columns:      --------
 Test Key Flag........:











Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CSUTEST Program

CSUTEST	Description
What it does	Tests the subprograms for Natural Construct-generated models. This program tests the individual subprograms for Natural Construct-generated models. For information, see *Test the Model Subprograms*.
PDAs used	■ None
Files accessed	■ NCST-SUBPROGRAM ■ NCST-CONTROL

Drivers Menu Option

Drivers menu
Additional Drivers menu
NCST Model/Frame Related Subprograms menu
Test NCST Generated Models option

```
CSUTEST                 N a t u r a l  C o n s t r u c t              CSUTESM1
Aug 14                     Single Module Test Program                 04:54 PM

Code Function              *Model: ________________________________
---- -------------------    Number all subprograms to be executed
 R   Release Variables      |                     |
 *   Execute All Subp.      V                     |
1-9  Execute One Subp.      _  Clear : ________   V
 E   Edit source            _  Mod  1: ________   _ Mod  6: ________
 C   Clear Edit Buffer      _  Mod  2: ________   _ Mod  7: ________
 ?   Help                   _  Mod  3: ________   _ Mod  8: ________
 .   Terminate              _  Mod  4: ________   _ Mod  9: ________
---- -------------------    _  Mod  5: ________   _ Mod 10: ________
 _                          _  Pregen: ________   _ Save  : ________
                Source      _  Documt: ________   _ Postgn: ________
                Lines
          Total:  133                            Frame Parameter or Exit Name
                            _  Other : ________ ________________________________
                            _  Other : ________ ________________________________
                            _  Other : ________ ________________________________
                            _  Other : ________ ________________________________
  Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
        help  retrn quit                                                   mai ↩
```

CSUTLATE Subprogram

CSUTLATE	Description
What it does	Translates message text at runtime. This subprogram receives a message number and position value and retrieves the appropriate text. If the message text contains multiple items delimited by a slash (/), the position value identifies which text is translated. This subprogram is invoked from the CSUCURS and CSUCURS1 subprograms.
PDAs used	■ CSATLATE ■ CSASTD
Files accessed	■ SYSTEM-FUSER

CSUTRANS Subprogram

CSUTRANS	Description
What it does	Translates screen prompts before they are displayed. This subprogram receives a defined data structure (typically a translation LDA) containing SYSERR message numbers and translates them into the appropriate text. **Note:** 1. For more information, see ***CSUTRANS Utility***. 2. For information about SYSERR message numbers, see ***Use SYSERR References***. 3. For information about formatting the message text, see ***Format SYSERR Message Text***.
PDAs used	■ CSATRANS ■ CSASTD
Files accessed	■ SYSTEM-FUSER

Drivers Menu Option

Drivers menu
Natural-Related Drivers menu
Retrieve Application Error Messages
Retrieve Block Messages option

```
CTETRANS          ***** Natural Related subprograms *****
Oct 21               - Driver for subprogram CSUTRANS -                1 of 1

Translation LDA .... CTE-MAL

Input Parameters ... #GEN-PROGRAM *2000.1,.___________
                     #SYSTEM *2000.2,+___________
                     #GDA *2000.3,>___________
                     #TITLE *2001.1,+/16________
                     #DESCS *2001.2,.___________
                     #GDA-BLOCK *2001.3,>___________
                     #MAP-HEADER1 *2049.1,./18________
                     #MAP-HEADER2 *2049.2,>/18________
                     #USE-MSG-NR *2050.1,.___________
                     #PASSWORD-CHECK *2050.2,./20________
                     #MESSAGE-LIBRARY CSTLDA__
                     #LDA-NAME CTE-MAL_
                     #TEXT-REQUIRED X
                     #LENGTH-OVERRIDE __________0
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help        quit        reset       bkwrd frwrd       right left     ↩
```

Note: This driver program is provided as a sample only. Because the screen prompts translated by CSUTRANS vary depending on the application under development, the driver program must be tailored to the application.

CSUXCHK Subprogram

CSUXCHK	Description
What it does	Scans for the existence of a user exit in the edit buffer. This subprogram receives the name of a user exit and scans the edit buffer for that name.
PDAs used	■ CSAXCHK
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
User Exit Related Subprograms menu
Validate Format for Input Value option

```
 CTEXCHK               N a t u r a l   C o n s t r u c t            CTEXCHK1
 Aug 14                  Driver for subprogram CSUXCHK                1 of 1

   User Exit Name....: ________________________________
   Found............:

   Read in New Source: _
  *New Source Name...: ________
   New Source Library: DEVEX___


Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CSU2LONG Subprogram

CSU2LONG	Description
What it does	Converts a long variable name to an abbreviation. This subprogram receives a long character string (32 characters) and a desired length, and returns the truncated string (abbreviation). The sixth position of the string is the first position truncated. If no length is given, the default is 30. If the long string is not longer than the desired length, the string is still truncated. For example, if the long string is "THIS-IS-A-LONG-VARIABLE" and the desired length is 20, the short string is "THIS-A-LONG-VARIABLE". **Tip:** Use this subprogram when you add characters to a file or field name that is already 32 characters long.
PDAs used	■ CSA2LONG
Files accessed	■ None

Drivers Menu Option

Drivers menu
Additional Drivers menu
General Utility Subprograms menu
Shorten a Long Variable Name option

```
 CTE2LONG             N a t u r a l   C o n s t r u c t             CTE2LNG1
 Aug 14                Driver for subprogram CSU2LONG                 1 of 1

  Long Name.....: ________________________________
  Maximum Length: ___

  Short Name....:


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

Predict-Related Subprograms (CPU*)

The subprograms described in this section retrieve information from the Predict data dictionary. While some of these subprograms generate code, most supply information to the calling subprogram and the calling subprogram generates the code.

Notes:

1. If you use Software AG's Entire Net-work, the Predict data can reside on a platform other than the platform on which Natural Construct is running.
2. Driver programs for many of the supplied programs/subprograms are available through the Drivers menu option on the Administration main menu. If a driver program is available, its location is listed in the *Drivers Menu Option* section in the program/subprogram's description. For more information about the supplied driver programs, see ***Drivers Menu Function***.

This section covers the following topics:

- With Natural Security Installed
- CPU-OBJ Subprogram
- CPU-OBJ2 Subprogram
- CPU-OREL Subprogram
- CPU-VIEW Subprogram
- CPUEL Subprogram
- CPUELDE Subprogram

With Natural Security Installed

If Natural Security is installed, the Predict-related subprograms restrict access to file and field information. Users can only retrieve information for files linked to the current application.

While generating a program, the program may access information about the same file many times. To avoid security checks each time, the access subprograms use the FILE-CODE field. This INPUT/OUTPUT field accesses file information and acts as a cipher code to avoid multiple security checks on the same file; it is available for all supplied subprograms.

If you are developing under Natural Security, include the FILE-CODE field in the model PDA for each file used multiple times during generation. The FILE-CODE field is passed in the PDA of the access subprogram and reassigned back to the model PDA after each call.

To avoid security checks for each access, the model subprogram that invokes CPUEL contains the following statements:

```
ASSIGN CPAEL.FILE-CODE = #PDA-FILE-CODE
CALLNAT 'CPUEL' CPAEL CSASTD
ASSIGN #PDA-FILE-CODE = CPAEL.FILE-CODE
```

Note: For an example of using these subprograms to restrict access to file and field information, refer to the CUSCGPR program in the SYSCST library.

CPU-OBJ Subprogram

CPU-OBJ	Description
What it does	Generates an external data area based on a Predict file view. This subprogram receives the view name and a set of logical variables that define the generation options. It generates an external data area structure to match the view. It can also generate the C# variables for each C* variable that corresponds to an MU or PE and/or includes the corresponding REDEFINE fields for redefined fields or superdescriptors. **Note:** For information about INPUT/OUTPUT parameters, refer to the CPA-OBJ data area in the SYSCST library.
PDAs used	■ CPA-OBJ ■ CSASTD
Files accessed	■ SYSDIC-EL ■ SYSDIC-FI

Drivers Menu Option

Drivers menu
Predict-Related Drivers menu
Generate Data Areas menu
External option

```
 CTE-OBJ              N a t u r a l   C o n s t r u c t             CTE-OBJ1
 May 12                 Driver for subprogram CPU-OBJ                   1 of 1

*File: ________________________________


Build Redefines..: _    Structure Level: _
SuperDe Redefines: _    Joined Fld Name: ________________________________
Cstars...........: _    Joined Length..: ___
Use Cutoff.......: _    Cutoff Value...: ________

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CPU-OBJ2 Subprogram

CPU-OBJ2	Description
What it does	Issues CALLNAT to the #CALLNAT subprogram and passes information about elements that make up an object. This subprogram receives: ■ a view name ■ a key name ■ a set of options ■ the name of a passed subprogram to CALLNAT An object is derived from view and key names. The view and key names are based on intra-object relationships defined in Predict (for example, ORDER-HEADER-HAS-ORDER-LINES). The elements of an object are the individual fields in the files that make up the object. This subprogram traverses the object tree and checks each element. For each element, it CALLNATs the #CALLNAT subprogram and passes it information about the element (for example, the format, length, and type). You can set options to limit or extend the number of elements to check (for example, whether to include all field redefinitions or just the lowest levels). **Note:** This subprogram replaces the CPU-OBJ subprogram; for all new development, use CPU-OBJ2.

CPU-OBJ2	Description
Parameters/PDAs used	■ CPA-OBJ2 ■ CPA-ODAT ■ CU--PDA ■ #PASS(A1/1:V) ■ CSASTD
Files accessed	■ SYSDIC

CPU-OREL Subprogram

CPU-OREL	Description
What it does	Adds entity information to a table. This subprogram receives the name of an object and information about each entity belonging to the object. It adds this information to a table. Optionally, it can display tracing information. **Note:** For more information, refer to CPA-OREL.ENTITY(*).
PDAs used	■ CPA-OREL ■ CU__PDA ■ CSASTD
Files accessed	■ SYSDIC-RL ■ SYSDIC-FI ■ SYSDIC-EL

CPU-VIEW Subprogram

CPU-VIEW	Description
What it does	Generates field definitions based on the contents of a Predict view. This subprogram receives the name of a Predict view and a set of logical parameters defining the options to be generated. It generates the view definition as it should appear in the `DEFINE DATA . . . END-DEFINE` block of a Natural program, subprogram, or helproutine. This subprogram can also generate the C# variables for each C* variable that corresponds to an MU (multiple-valued) or PE (periodic group), and/or includes the corresponding REDEFINE fields for redefined fields or superdescriptors. You can use this subprogram to define a structure based on a view in Predict. The format and length for each field is generated. **Note:**

CPU-VIEW	Description
	1. This subprogram differs from CPU-OBJ in that it generates internal rather than external data structures. 2. For information about INPUT/OUTPUT parameters, refer to the CPA-VIEW data area in the SYSCST library.
PDAs used	■ CPA-VIEW ■ CSASTD
Files accessed	■ SYSDIC-EL ■ SYSDIC-FI

Drivers Menu Option

Drivers menu
Predict-Related Drivers menu
Generate Data Areas menu
Internal option

```
 CTE-VIEW             N a t u r a l   C o n s t r u c t            CTE-VEW1
 May 12                Driver for subprogram CPU-VIEW                1 of 1

 *File....: ________________________________
  View....: ________________________________        Gen 01 Level......: _
  Omit Fld: ________________________________

  Variable Indexes : _    Include Hyper DE...: _    Include MU Counter: _
  Build Redefines..: _    Include Phonetic DE: _    Include PE Counter: _
  SuperDe Redefines: _    Include Sub DE.....: _    Include MU Hyper..: _
  Specify Formats..: _    Include Super DE...: _    Include PE Hyper..: _
  Cstars...........: _    Redefine Cstars....: _

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CPUEL Subprogram

CPUEL	Description
What it does	Returns Predict information about a field in a file. This subprogram finds a field in a Predict file and returns information about the field. **Note:** CPUEL supports dynamic field lengths and AVs with Adabas.
PDAs used	■ CPAEL ■ CSASTD
Files accessed	■ SYSDIC-EL

Drivers Menu Option

Drivers menu
Predict-Related Drivers menu
Field Information menu
Single Field option

```
 CTEEL             ***** PREDICT Related Subprograms *****         CTEEL11
 Feb 25,13              - Driver for subprogram CPUEL -            1 more >

*File Name..: ________________________________ DDM Prefix: ________________
*Field Name : ________________________________
 Simple Outputs: _

Fld Found...: X ADABAS Fld Name: AA          Fld Format....: A   Field Uq : U
Ver Found...:   Fld Length.....: 8.00        Dynamic.......:     De Type..: D
Lvl Number..: 1 Sign...........:             Suppression...:     Gr Struct:
Occurrences.:       Fld Type.......:         A/Descend.....: A   Pe Ind...:
                    Fld Redefined  :         Rank..........:

Edit Mask...:                                      Field Headings:
DDM Fld Name: PERSONNEL-ID                   PERSONNEL
Index Name..:                                ID
Fld Sequence: 100
NAT Length..:          PRD Length..: 8.00        PREDICT Format: A
IMS Fld Name:          IMS Root Key:             Access Lvl:
IMS Fld Len :          IMS Offset..:             Update Lvl:
Character Set:
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                                  left  right main   ↩
```

Press Enter to display the second panel. For example:

```
 CTEEL            ***** PREDICT Related Subprograms *****         CTEEL21
 < 1 more              - Driver for subprogram CPUEL -            03:26 PM

 File Name..: CST-EMPLOYEES
 Field Name : PERSONNEL-ID

 LEVEL
 -----
 DDM Field Name                    Field Type    Is Redefined







 Keyword Mask                     Keywords Returned
 ________________________________
 ________________________________



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

CPUELDE Subprogram

CPUELDE	Description
What it does	Returns a description attribute from a specified file. This subprogram receives the name of a file and finds a description attribute. It returns the names of all fields that have the DESCRIPTION keyword.
PDAs used	■ CPAELDE ■ CSASTD
Files accessed	■ SYSDIC-FI ■ SYSDIC-EL ■ SYSDIC-KY

CPUELKY Subprogram

CPUELKY	Description
What it does	Returns keywords linked to a field in a specified file. This subprogram receives the name of a file and field; it returns keywords linked to the field.
PDAs used	■ CPAELKY ■ CSASTD
Files accessed	■ SYSDIC-FI ■ SYSDIC-EL ■ SYSDIC-KY

CPU-FREL Subprogram

CPU-FREL	Description
What it does	Retrieves information about a foreign relationship and CALLNATs a pass-through subprogram. This subprogram passes CPA-FREL, CU--PDA, and CSASTD to the pass-through subprogram.
PDAs used	■ CPARLRD ■ CU--PDA ■ CPA-FREL ■ CSASTD
Files accessed	■ SYSDIC-FI ■ SYSDIC-EL

CPUELNX Subprogram

CPUELNX	Description
What it does	Returns field-by-field information if it is called repeatedly. This subprogram receives the name of a Predict file, the CPAELNX data area (contains options for field types), and the CPRELNX data area (contains information about current processing), and logically reads through the fields in the file. **Note:** 1. CPRELNX contains reserved variables that keep track of the current position; it must not be modified by the calling program. 2. For information about INPUT/OUTPUT parameters, refer to the CPAELNX data area in the SYSCST library.

CPUELNX	Description
PDAs used	■ CPAELNX ■ CPRELNX ■ CSASTD
Files accessed	■ SYSDIC-EL ■ SYSDIC-FI

Drivers Menu Option

Drivers menu
Predict-Related Drivers menu
Field Information menu
Get Next Field option

```
 CTEELNX           ***** PREDICT Related Subprograms *****        CTEENX11
 Feb 25               - Driver for subprogram CPUELNX -           1 more >

*File Name....: _______________________________ First Time : X EOF.....:
 DDM Prefix...: ________________

Redef Base Fld: _  Super Subs: _  Mus.......: _  Nulls Only : _ Counters: _
First Redefine: _  Phonetics : _  Pe Groups : _  Seq Only...: _ Groups..: _
All Redefines : _  Hypers....: _  Pes.......: _  Uq Only....: _ Fillers : _
Max Rede Rank : _  Derived...: _  Mus in Pes: _  VE Only....: _ REDE STR: _

Fld Name......:                                  Fld Type...:
Fld Format....:                                  Length.....:
PREDICT Format:                                  Sign.......:

ADABAS Name...:    Fld Def..:   De Type..:   Fld Count..:     Rank..:
Level Number..:    Fld Uq...:   Pe Ind...:   Occurrences:

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

Press Enter to display the second panel. For example:

```
 CTEELNX           ***** PREDICT Related Subprograms *****         CTEENX21
 < 1 more             - Driver for subprogram CPUELNX -            03:38 PM

       Field Headings
------------------------------
 PERSONNEL                       IMS Offset....:          Access Lvl:
 ID                              IMS Fld Name..:          Update Lvl:
                                 IMS Fld Length:

Index Name..:
DDM Fld Name: PERSONNEL-ID

Edit Mask...:
Level Type Trail:    ->    ->    ->    ->    ->    ->    ->
Redefine Trail..:    ->    ->    ->    ->    ->    ->    ->

Fld is Redefined:    Redefine Cnt:

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

CPUELRD Subprogram

CPUELRD	Description
What it does	Reads through the fields in a Predict file, issues a CALLNAT for the specified subprogram for each field, and passes information about the field to the subprogram. It receives: ■ the name of a file ■ the name of a subprogram to CALLNAT ■ the selection criteria (in CPAELRD.INPUTS) The subprogram traverses the specified file. For each selected field, it CALLNATs the passed subprogram to process the current field.
PDAs used	■ CPAELRD ■ CU--PDA (model PDA) ■ CSAPASS (can be redefined as required and used to store additional information that must be preserved between CALLNATs) ■ CSASTD
Files accessed	■ SYSDIC-EL

Drivers Menu Option

Drivers menu
Predict-Related Drivers menu
Field Information menu
CALLNAT for Each Field in a File option

```
 CTEELRD             N a t u r a l   C o n s t r u c t             CTEELRD1
 Aug 14                Driver for subprogram CPUELRD                   1 of 1

*File Name.......: ________________________________   Fld Count......:
*Key Name........: ________________________________   Level..........:
CALLNAT..........: CTELRDSM                          Max Rede Rank..: _

ReDe Base Fld: _ SPs/SBs..: _ Pes...: _ Pe Group: _ Only VE..: _ Fillers: _
First ReDe...: _ Phonetics: _ Mus...: _ Mu in Pe: _ Only UQ..: _ Derived: _
All ReDe.....: _ Hypers...: _ Groups: _ Counters: _ Only Null: _ Rede St: _

Fld Name  :                            Format :    PRD Format :
DDM Field :                            Fld UQ :    Length.....:
Index.... :                            Type...:    Adabas Name:
Headings  :                            De Type:    Occurrences:
                                       Pe Type:               :
Edit Mask :                            Rank...:               :
Type Trail:                            Redef..:    ReDe Count :
ReDe Trail:
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                    bkwrd frwrd                   mai ↩
```

Note: If you change the name of the subprogram in the CALLNAT field, the specified subprogram must have the same parameters as those in the PDAs used by CPUELRD.

CPUELVE Subprogram

CPUELVE	Description
What it does	Returns the verification rule names for a field in a file. This subprogram finds a field in Predict and returns the names of the verification rules of type N (Natural Construct).
PDAs used	■ CPAELVE ■ CSASTD
Files accessed	■ SYSDIC-EL

Drivers Menu Option

Drivers menu
Predict-Related Drivers menu
Field Information menu
Get Verification Rule Names option

```
 CTEELVE                N a t u r a l   C o n s t r u c t           CTEELVE1
 Aug 14                  Driver for subprogram CPUELVE                1 of 1

*File Name : ________________________________     Field Found.........:
*Field Name: ________________________________     Num of Verifications:

                  +-----------------------------------+
                  |  1__   VERIFICATION NAME           |
                  | ------------------------------     |
                  |                                    |
                  |                                    |
                  |                                    |
                  |                                    |
                  |                                    |
                  +-----------------------------------+



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                        bkwrd frwrd                mai ↩
```

CPUEXIST Subprogram

CPUEXIST	Description
What it does	Verifies the existence of a specified Predict object. This subprogram receives the name and type of an object and verifies its existence in Predict.
PDAs used	■ CPAEXIST ■ CSASTD
Files accessed	■ SYSDIC-SY ■ SYSDIC-PR ■ SYSDIC-KY ■ SYSDIC-DB ■ SYSDIC-FI ■ SYSDIC-RL ■ SYSDIC-VE

Drivers Menu Option

Drivers menu
Predict-Related Drivers menu
Verify Object Existence option

```
 CTEXIST              N a t u r a l   C o n s t r u c t             CTEXST1
 Aug 14                Driver for subprogram CPUEXIST                1 of 1

  Object Name: ________________________________    Object Exists:
  Object Type: __ (SY,PR,KY,FI,DB,RL,VE)




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CPUFI Subprogram

CPUFI	Description
What it does	Returns Predict information about a file. This subprogram receives the name of a file and returns Predict information about that file.
PDAs used	■ CPAFI ■ CSASTD
Files accessed	■ SYSDIC-FI

Drivers Menu Option

Drivers menu
Predict-Related Drivers menu
File Information option

```
 CTEFI                 N a t u r a l   C o n s t r u c t           CTEFI1
 Aug 14                  Driver for subprogram CPUFI               1 of 1

*File Name: ________________________________            Ripp File Nr..:
 File Type:                                             Ext File  Nr..:

 Master File Name..:
 Primary Seq Field :

 DDM Prefix........:                                    IMS DB Number.: 00
 DDM File Name.....:                                    IMS File Level:
 IMS Parent File...:                                    IMS File Nr...: 00
 IMS Root File Name:                                    IMS Seg Type..:
 IMS DBD Name......:                                    IMS DDM Suffix:
 IMS Seg Name......:                                    DDM Matches...:
 IMS Root Seg Name :


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

CPUHOLD Subprogram

CPUHOLD	Description
What it does	Determines the hold field for a file. This subprogram receives the name of a file and determines the hold field for the file. To define a hold field, attach the HOLD-FIELD keyword to the field in Predict.
PDAs used	■ CPAHOLD ■ CSASTD
Files accessed	■ SYSDIC-FI ■ SYSDIC-EL

CPUKY Subprogram

CPUKY	Description
What it does	Retrieves information related to a Predict keyword. You can use the keyword comments to store attribute values that can be returned by this subprogram.
PDAs used	■ CPAKY ■ CSASTD
Files accessed	■ SYSDIC-KY ■ SYSDIC-EL

CPUREDEF Subprogram

CPUREDEF	Description
What it does	Generates redefinitions for compound keys, superdescriptors, or redefined fields in Predict. This subprogram invokes the CPUXPAND subprogram, which retrieves the components of the field to be redefined. Redefinitions can be generated in either inline or external data area format.
PDAs used	■ CPAREDEF ■ CSASTD
Files accessed	■ SYSDIC-EL

Drivers Menu Option

Drivers menu
Predict-Related Drivers menu
Field Information menu
Generate Field Redefinition option

```
 CTEREDEF             N a t u r a l   C o n s t r u c t            CTERDEF1
 Aug 14                Driver for subprogram CPUREDEF                1 of 1

 *File : ________________________________          Redef Level.........: _
 *Field: ________________________________          Change Format N to A: _

  Super Options
  -------------
  Include Deriv Level: _   Inside Histogram: _
  Include Redef Level: _   Omit Format.....: _

  Resets Required:




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CPURL Subprogram

CPURL	Description
What it does	Returns information about a relationship in Predict. This subprogram receives a Predict relationship name and returns information about the relationship.
PDAs used	■ CPARL ■ CSASTD
Files accessed	■ SYSDIC-RL

Drivers Menu Option


```
CTERL                  N a t u r a l   C o n s t r u c t             CTERL1
Aug 14                   Driver for subprogram CPURL                 1 of 1

*Relationship Name: ________________________________ Relationship Found:
                                                     Relationship Type :

       Relationship File                  Relationship Field          Card
 -------------------------------- --------------------------------  ----



     Ddm Relationship Field        Minimum    Average     Maximum
 -------------------------------- -------    --------    -------



Constraint Type Upd:               Db2 Constraint Name:
Constraint Type Del:
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                mai
```

CPURLRD Subprogram

CPURLRD	Description
What it does	Retrieves the Predict relationships for a specified file, and optionally a specified type. This subprogram receives: ■ the name of a file ■ a relationship type (optional) ■ the name of a subprogram (in CPARLRD.INPUTS) It finds relationships for the specified file, issues a CALLNAT to the specified subprogram, and passes the information about the relationship to the subprogram for processing.
PDAs used	■ CPARLRD ■ CU--SYSLIBSPDA (model PDA) ■ CSAPASS (can be redefined as required and used to store additional information that must be preserved between CALLNATs) ■ CSASTD

CPURLRD	Description
Files accessed	■ SYSDIC-FI ■ SYSDIC-KL

Drivers Menu Option

Drivers menu
Predict-Related Drivers menu
CALLNAT for Each Relationship option

```
        CTERLRD         N a t u r a l   C o n s t r u c t         CTERLRD1
        Aug 14            Driver for subprogram CPURLRD             1 of 1

          *File Name.............: ________________________________
           Relationship Type.....: _
           CALLNAT...............: CTELRDSM
           Relationship Count....:
           Relationship Name.....:
           Relationship File ....:
           Relationship Field....:
           DDM Relationship Field:
           Cardinality...........:
           Minimum...............:
           Average...............:
           Maximum...............:
           DB2 Constraint Name...:
           Constraint Type Upd...:
           Constraint Type Del...:
       Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10-
             help  retrn quit                                              ↩
```

CPUSUPER Subprogram

CPUSUPER	Description
What it does	Returns the definition for a super/subdescriptor (or DB2 compound key). This subprogram receives the name of a superdescriptor or subdescriptor (or DB2 compound key) and the name of the Predict file or table to which it belongs. It returns information about the derived fields.
PDAs used	■ CPASUPER ■ CSASTD
Files accessed	■ SYSDIC-EL

Drivers Menu Option

Drivers menu
Predict-Related Drivers menu
Field Information menu
Expand Superde or Redefine option

```
CTESUPER          ***** Predict Related Subprograms *****        CTESUPR1
Oct 09              - Driver for subprogram CPUSUPER -           03:08 PM

*File Name : ________________________________     Superde Length....:
*Field Name: ________________________________     Superde Format....:

 Contains Repeating Fields:                       C#Derivation Group:
+-------------------------------------------------------------------------
|  1__                           Start End   A/ Fld Sup PE   Dimension
|       Source Field Name        Char  Char  D  Typ Opt Ind  1   2   3
|  ------------------------------- ----- ----- -  --- --- --- --- --- ---
|
|
|
|
|
+-------------------------------------------------------------------------

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                    bkwrd frwrd                 mai ↩
```

CPUUNIQ Subprogram

CPUUNIQ	Description
What it does	Determines the unique description field (primary key). This subprogram receives the name of a file and determines the unique description field (primary key) for the file.
PDAs used	■ CPAUNIQ ■ CSASTD
Files accessed	■ SYSDIC-FI ■ SYSDIC-EL

CPUVE Subprogram

CPUVE	Description
What it does	Prints verification rules to the source buffer. This subprogram prints either the code or the data definition for a type N (Natural Construct) verification rule to the source buffer.
PDAs used	■ CPAVE ■ CSASTD
Files accessed	■ SYSDIC-VE-ACT

Drivers Menu Option

Drivers menu
Predict-Related Drivers menu
Field Information menu
Generate Verification Rules option

```
 CTEVE                 N a t u r a l   C o n s t r u c t           CTEVE1
 Aug 14                  Driver for subprogram CPUVE               1 of 1

 Verification Name: ________________________________ Verification Found:
*User View Name...: ________________________________ Rule Generated....:
*DDM Field Name...: ________________________________

 Generate Data....: _
 Occurrences......: _______________









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai ↩
```

CPUVERUL Subprogram

CPUVERUL	Description
What it does	Returns information about Predict verification rules.
PDAs used	■ CPAVERUL ■ CSASTD
Files accessed	■ SYSDIC-VE

CPUXPAND Subprogram

CPUXPAND	Description
What it does	Expands a super/subdescriptor or redefined field. This subprogram receives: ■ the name of a super/subdescriptor (or DB2 compound key) ■ the name of the Predict file (or table) to which the key belongs ■ the expansion options ■ the name of a subprogram to CALLNAT (in CPAXPAND.INPUTS) ■ the parameters in the model PDA (CU--PDA) ■ an additional A1/1:v parameter (CSAPASS) It expands the specified super/subdescriptor (or DB2 compound key) into its underlying components. For each component, it CALLNATs the specified subprogram. **Note:** When this subprogram expands a superdescriptor, redefinitions of the derived fields are included.
PDAs used	■ CPAXPAND ■ CU--PDA ■ CSAPASS ■ CSASTD
Files accessed	■ SYSDIC-EL

Drivers Menu Option

Drivers menu
Predict-Related Drivers menu
Field Information menu
CALLNAT for Sup/Subde Components option


```
   CTEXPAND             N a t u r a l  C o n s t r u c t           CTEXPN11
   Aug 14                Driver for subprogram CPUXPAND               1 of 3

   *File Name......: ________________________________  Phantom Bytes: _
   *Base Field Name: ________________________________  Fillers......: _
   CALLNAT.........: CTELRDSM     P


    Base Field Information                          Field Headings
    ----------------------                   --------------------------------
    Sequence:        Adabas Field Name:     :
    Format..:        Field Definition :     :
    Length..:        Field Type.......:     :

    Edit Mask......:
    DDM Field Name :


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

Note: If you change the name of the subprogram in the CALLNAT field, the specified subprogram must have the same parameters as those in the PDAs used by CPUXPAND.

Press Enter to display the second panel. For example:

```
 CTEXPAND             N a t u r a l  C o n s t r u c t            CTEXPN21
 Aug 14                Driver for subprogram CPUXPAND                2 of 3


Derived Field Information                        Field Headings
-------------------------                ----------------------------------
First Showing.:                         :
Field Count...:                         :
Whole Field...:                         :

Sequence......:        Adabas Field Name:      Start Character:
Format........:        Field Definition :      End   Character:
Length........:        Field Type.......:

Edit Mask.....:
Field Name....:
DDM Field Name:

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                      left  right mai
Scrolling performed                                                          ↩
```

Press Enter to display the third panel. For example:

```
 CTEXPAND             N a t u r a l  C o n s t r u c t            CTEXPN31
 Aug 14                Driver for subprogram CPUXPAND                3 of 3

Ascending/Descending
Expanded Field Information                       Field Headings
--------------------------               ----------------------------------
Field  Count..:                         :
Offset Start..:                         :
Offset End....:                         :

Sequence......:        Predict Format...:      Special characteristic:
Format........:        Field Definition :
Length........:

Edit Mask.....:
Field Name....:
DDM Field Name:

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                      left  right mai
Scrolling performed                                                          ↩
```

Predict-Related Helproutines (CPH*)

You can attach the following helproutines to fields that require the input of Predict information. They are active helproutines that fill the field to which they are attached.

Note: Some of the following routines provide help information, although they are coded as subprograms and not as helproutines. This provides greater flexibility to access help information.

This section covers the following topics:

CPHEL Subprogram

CPHEL	Description
What it does	Browses the fields in a file for selection. This subprogram receives the name of a Predict file. (If no file name is specified, it provides file selection.) It browses all the fields in the specified file and returns the selected field.
Attached to	Input of a Predict field name.
PDAs used	▪ CPAHEL ▪ CSASTD
Files accessed	▪ SYSDIC-FI

CPHELB Subprogram

CPHELB	Description
What it does	Browses the fields in a file for selection. This subprogram receives the name of a file and browses all the fields in the file for selection. Optionally, this subprogram can browse only the descriptor fields. **Note:** For information about INPUT/OUTPUT parameters, refer to the CPHELBA data area in the SYSCST library.
PDAs used	▪ CPAHEL

CPHELB	Description
	■ CSASTD
Files accessed	■ SYSDIC-EL

CPHFI Helproutine

CPHFI	Description
What it does	Browses Predict views/files for selection. This helproutine browses all the views and files in Predict for selection.
Attached to	Input of a Predict file name.
Parameters used	■ #PDA-FILE(A32)
Files accessed	■ SYSDIC-FI

CPHFIB Subprogram

CPHFIB	Description
What it does	Browses Predict views and files for selection.
Parameters/PDAs used	■ #PDA-KEY(A32) ■ CSASTD
Files accessed	■ SYSDIC-FI

CPHPRED Helproutine

CPHPRED	Description
What it does	Browses Predict objects (by object type) for selection. This helproutine receives an object type and browses the Predict objects of that type for selection. Valid object types are: ■ S (system) ■ P (program) ■ K (keyword) ■ M (module) ■ R (report)
Attached to	Input of a Predict object type.
Parameters used	■ #PDA-TYPE(A1)

CPHPRED	Description
	■ #PDA-KEY(A32)
Files accessed	■ SYSDIC-SY ■ SYSDIC-PR ■ SYSDIC-KY ■ SYSDIC-RE ■ SYSDIC-MO

CPHRL Helproutine

CPHRL	Description
What it does	Browses the names of Predict relationships for selection. This helproutine receives the names of a Predict relationship and a file and returns the selected relationship. If a file name is specified, the helproutine browses only the Predict relationships for that file. If no file name is specified, it browses all existing relationships.
Attached to	Input of a Predict relationship name.
Parameters used	■ #PDA-FILE(A32) ■ #PDA-RELATIONSHIP-NAME(A32)
Files accessed	■ SYSDIC-FI ■ SYSDIC-RL

CPHSET Helproutine

CPHSET	Description
What it does	Sets a flag to indicate that help was requested for a field. This helproutine receives the name of a parameter and sets a flag to indicate help was requested. The parameter should be checked after the INPUT statement. If a flag is set, for example, reset the flag and issue CALLNATs to do the help processing. This technique allows the helproutine to access all data entered in a single panel transaction. When you generate a browse subprogram, for example, you can type the file name (without pressing Enter) on the Additional Parameters panel and request help for a field.
Attached to	Any input field.
Parameters used	■ #PDA-SET-HELP(L)
Files accessed	■ SYSDIC-FI ■ SYSDIC-RL

General Purpose Generation Subprograms (CU--*)

The subprograms described in this section are general purpose generation subprograms. These subprograms are identified by a CU-- prefix.

CU--EM Subprogram

CU--EM	Description
What it does	Returns edit masks used by the generated programs for displaying date and time fields. This subprogram can be changed to suit your standards. Changes to this routine should be made in a higher level steplib to minimize maintenance. Unless you modify your models, the date and time field edit masks should not be longer than nine characters.
PDAs used	■ CU--EMA

CU--LRP Subprogram

CU--LRP	Description
What it does	Returns the left and right prompt displayed on the Natural Construct panels. The left prompt displays the current month and day in *DATX (EM=LLL''DD), which can be language sensitive. The right prompt displays the "1 of 1" or "1 of 3" panel indicators, depending on the number of panels. This prompt uses several control record fields to build the prompt position indicators, which are compressed on both sides of the "of" indicator.
Parameters/PDAs used	■ #PDA-LEFT-PROMPT(A9) ■ #PDA-LEFT-INDICATOR(A4) ■ #PDA-RIGHT-PROMPT-OF(A4) ■ #PDA-RIGHT-INDICATOR(A4) ■ #PDA-RIGHT-PROMPT(A9) ■ CSASTD

CU--MSG Subprogram

<table>
<tr><th>CU--MSG</th><th>Description</th></tr>
<tr><td>What it does</td><td>Returns the text for an application error message. It receives a message number in #PDA-FRAME-PARM. After ensuring this literal is numeric, it retrieves the short message for the SYSTEM application and the *Language variable.

The error message is written (left-justified and enclosed within single quotes) to the source buffer, thus substituting for the frame parameter. The usual search criteria and defaults (English) apply. The following example shows a code frame:
<pre>USE-MSG-NR   1
ASSIGN MSG-INFO.##MSG-NR = 8123   "
ELSE   1
ASSIGN MSG-INFO.##MSG =   "
SUBPROGRAM:CU--MSG PARAM: 8123   N "</pre></td></tr>
<tr><td>PDAs used</td><td>■ CU--PDA
■ CSASTD</td></tr>
<tr><td>Files accessed</td><td>■ Application error message file</td></tr>
</table>

CU--UL Subprogram

CU--UL	Description
What it does	Returns the underscore line used on Natural Construct panels. This subprogram receives an underscore character set and creates the underscore line. The character(s) specified on the control record (A4) is duplicated to fill the A80 length.
Parameters/PDAs used	■ #PDA-UNDERSCORE(A4) ■ #PDA-UNDERSCORE-LINE(A80) ■ CSASTD

23 Supplied Administration Utilities

Introduction

This section describes the utilities supplied with Natural Construct for all supported platforms. To invoke a utility, enter its name at the Next prompt (Direct Command box for Unix).

Note: When a description refers to "your print file" for mainframe users, it refers to Print File 1. When a description refers to "your print file" for Unix users, it refers to DEVICE LPT1.

Import and Export Utilities

This section explains how to transfer data across dissimilar platforms (for example, from Unix to mainframe).

Natural Construct import and export utilities read and write their data from and to work file 1. This is true for each of the following utilities:

Utility	Described in
CSFLOAD CSFUNLD	*Multiple Code Frame Import Utility* *Multiple Code Frame Export Utility*
CSHLOAD CSHUNLD	*CSHLOAD Load Utility* *CSHUNLD Unload Utility*
CSMLOAD CSMUNLD	*Natural Construct Generation*

A work file written on one platform (such as Unix) can be read by 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	Define work file 1 as a PC file and activate PC Connection before running the utility. (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 Code Frame Import Utility

The CSFLOAD frame import utility imports selected code frames from work file 1 to the code frame file. A report of the imported code frames is written to your print file.

CSFLOAD accepts up to 100 frame names and replace options in the form:

```
Code frame . ________  Replace .... _
```

The following example shows the CSFLOAD window:

```
CSFLOAD              ***** Natural Construct *****               CSFLOAD0
Nov 18                 Multiple Code frame Import                  1 of 1

 Code frame . ________  Replace .... _

                          Selected
 ----------------------------------------------------------------------
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--P
      help  retrn
```

Note: To replace the existing code frames with code frames with the same names in work file 1, mark the Replace field. If you do not want to replace the existing code frames, leave the Replace field blank.

Examples of Input Values

Values entered	Result
Code frame: *	Imports all code frames from work file 1. If a code frame with the same name exists in the code frame file, it is not replaced.
Code frame: MENU	Imports the MENU code frame from work file 1. If the MENU code frame exists in the code frame file, it is not replaced.
Code frame: MENU Replace: X	Imports the MENU code frame from work file 1. If the MENU code frame exists in the code frame file, it is replaced.
Code frame: FM*	Imports all code frames beginning with "FM" from work file 1. If a code frame with the same name exists in the code frame file, it is not replaced.
Code frame: . (period)	Terminates the CSFLOAD utility. **Note:** When running in batch mode, the CSFLOAD 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 frame input combination (for example: FM*,,Y).

Multiple Code Frame Export Utility

The CSFUNLD frame export utility exports selected code frames from the code frame file to work file 1. A report of the exported code frames is written to your print file.

Note: You can export a maximum of 1000 code frames at one time.

Enter each code frame name one name at a time. As you enter the names, they are automatically displayed on the panel.

For each exported code frame, you can specify whether to export its recursive (nested) code frames — if any exist. To export recursive code frames, mark the Include recursive code frames field. If you do not want to export recursive code frames, leave the field blank.

Examples of Input Values

Values entered	Result
*	Exports all code frames to work file 1.
MENU X	Exports the MENU code frame including any recursive (nested) code frames to work file 1.
FM*	Exports all code frames beginning with FM to work file 1.

Enter a period (.) to terminate the input.

Frame Hardcopy Utility

The CSFHCOPY frame hardcopy utility allows you to print a hardcopy list of your code frames, regardless of your teleprocessing monitor. All output is routed to your print file.

Enter each code frame name one name at a time. As you enter the names, they are automatically displayed on the panel.

Examples of Input Values

Values entered	Result
*	Routes all code frames to your print file.
MENU	Routes the MENU code frame to your print file.
FM*	Routes all code frames beginning with "FM" to your print file.

Enter a period (.) to terminate the input.

Comparison Utilities

This section describes utilities you can use to compare two Natural source modules and to compare a range of models in different libraries.

CSGCMPS Utility

This program compares two Natural source modules. You can compare the contents of two saved modules or you can compare the contents of the module currently in the source buffer to the contents of a saved module.

Specify the library ID, module name, database ID, and file number for each module you want to compare. In addition, you can specify the following options:

- ignore comment lines
- ignore trailing comments
- ignore leading spaces
- provide summary only

When you invoke the CSGCMPS utility online, the following window is displayed:

```
                       Compare Criteria
                Library  Object   Database File or Source Area
                ======== ======== ======== ====     ===========
    Old version ===> CST531M_ CSGCMPS_   017    029          _
    New version ===> CST821M_ CSGCMPS_   017    029          _

    Options...
       Ignore comment lines...... _
       Ignore trailing comments.. _
       Ignore leading spaces..... _
       Summary only.............. _
    Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF1
                     quit                                          ↩
```

CSGCMPL Utility

This program compares a range of modules in one library to the same modules in another library.

Specify the library ID, database ID, file number, and range value for the modules you want to compare. In addition, you can specify the following options:

- summary only
- only report if different
- ignore comment lines
- ignore trailing comments
- ignore leading spaces
- only compare object types

Online

When you invoke the CSGCMPL utility online, the following window is displayed:

```
                       Source Range Compare Facility
                                   Library  Database File Dominant
                                   ======== ======== ==== ========
        Old library.............. ________   ___17  ___29     X
        New Library.............. ________   ___17  ___29     _
        Program range............ ________ thru ________
        Summary only............. _
        Only report if different.. _
        Ignore comment lines...... _
        Ignore trailing comments.. _
        Ignore leading spaces..... _
        Only compare object types  __________ (ACGHLMNPST)         ↩
```

The Dominant column indicates the range of modules to be compared. Only modules that exist in the dominant library and in the other specified library are included in the compare results. Modules that only exist in the non-dominant library are not included.

The Only compare object types field limits the comparison to modules of a specified object type. Valid object types are:

Object Type	Description
A	Parameter
C	Copycode
G	Global data area
H	Helproutine
L	Local data area

Object Type	Description
M	Map
N	Subprogram
P	Parameter data area
S	Subroutine
T	Text

In Batch

Batch mode is the most efficient method of comparing many modules. The following SYSIN shows an example of using this utility in batch:

```
LOGON CST421M
CSGCMPL OLD-LIB,001,002,X,NEW-LIB,003,004, ,BEGIN,END,S,D,C,T,L,NPH
FIN
```

where:

OLD-LIB	Indicates the name of a library containing modules to be compared.
001	Indicates the database ID for OLD-LIB.
002	Indicates the system file number for OLD-LIB.
X	Indicates that the OLD-LIB is dominant (all modules in the dominant library are compared to matching modules in the other specified library).
NEW-LIB	Indicates the name of a library containing modules to be compared.
003	Indicates the database ID for NEW-LIB.
004	Indicates the system file number for NEW-LIB.
blank	If blank, indicates that OLD-LIB is dominant. If X, indicates that NEW-LIB is dominant.
BEGIN	Indicates the name of the first module in the range compared.
END	Indicates the name of the last module in the range compared.
S	Indicates a summary report (does not display detailed differences).GThi◎ option displays the names of the modules and whether the module contents are the same in both libraries.
D	Indicates that only modules that are different are included on the output report. Modules that are identical in both libraries are not included.
C	Indicates that Natural comment lines (lines beginning with "*" or "/*") are not compared.
T	Indicates that trailing comments (comments beginning with "/*") are not compared.
L	Indicates that leading spaces are not compared (changes in alignment will not show up as differences).
NPH	Indicates the list of Natural object types compared within the specified range of modules.

Upper Case Translation Utility

If you are developing applications in a language that does not support lower case Latin characters, use the supplied CVUPPERC utility to convert the Natural Construct components to upper case. This utility converts all Natural Construct-installed SYSERR message text and source code, as well as the contents of the Natural Construct system file, to upper case.

Notes:

1. Since this conversion requires a significant amount of processing, only run this utility in a batch environment.
2. Before running this utility, ensure that the batch job defines the correct Natural Construct logical file, FUSER system file, and FNAT system file.

Use the following SYSIN to invoke the CVUPPERC utility:

```
LOGON SYSCST
CVUPPERC
FIN
```

After converting the components to upper case, this utility issues a CATALL in the SYSCST library. To reflect the changes in your production environment, manually transfer all modules from the SYSCST library to the SYSLIBS library after the modules have been cataloged.

Additional Utilities

The utilities in this section generate cross-reference information for all subprograms referenced by a code frame or model. You can use these utilities either online (recommended) or in batch mode to determine which subprograms are invoked. If subprograms are missing, the utility will write a report to the screen.

Determine Which Subprograms Are Referenced by Code Frames

To determine which subprograms are referenced by code frames, run the CVUVALF utility either online or in batch mode. The CVUVALF utility:

- Verifies that all subprograms referenced by all code frames exist in the current library.
- Generates CALLNAT statements for all subprograms used in the code frames. Online, these statements are generated into the program editor and you can view the parameter list.

Online

To invoke the CVUVALF utility online:

1. Logon to the SYSCST library.
2. Run the CVUVALF program.

In Batch

To invoke the CVUVALF utility in batch mode:

- Use the following SYSIN:

  ```
  LOGON SYSCST
  CVUVALF
  FIN
  ```

  In this example, the utility provides a list of all subprograms that are referenced by all code frames and do not exist in the SYSCST library.

Determine Which Subprograms Are Referenced by Models

To determine which subprograms are referenced by models, run the CVUVALM utility either online or in batch mode. The CVUVALM utility:

- Verifies that all subprograms referenced by all models exist in the current library.
- Generates CALLNAT statements for all subprograms used in the models. Online, these statements are generated into the program editor and you can view the parameter list.

Online

To invoke the CVUVALM utility online:

1. Logon to the SYSCST library.
2. Run the CVUVALM program.

In Batch

To invoke the CVUVALM utility in batch mode:

- Use the following SYSIN:

```
LOGON SYSCST
CVUVALM
FIN
```

  In this example, the utility provides a list of all subprograms that are referenced by all models and do not exist in the SYSCST library.

24 Using SYSERR for Multilingual Support

This section describes how Natural Construct uses the Natural SYSERR utility to dynamically translate text and messages. SYSERR contains reference numbers that reference text strings in one or more languages.

Introduction

Natural Construct supports the dynamic translation of text and messages on many specification panels. Instead of typing text for panel headings, field prompts, error messages, etc., you can use a SYSERR reference number. At runtime, the reference number is replaced with its corresponding SYSERR text.

Maintenance

Using SYSERR references reduces your maintenance efforts. To modify a field prompt used on many panels, for example, you can change the text in SYSERR and all fields that use that reference number display the new name at runtime. It also helps maintain consistency throughout your generated applications, by ensuring that the same text is displayed in multiple locations.

Translation

For each SYSERR reference number, you can define message text in other languages. At runtime, text for the currently-selected language (the current value of the *Language system variable) is retrieved.

The text on all Natural Construct panels can be dynamically translated into any Natural-supported language.

Note: If you only require one language, this feature can be disabled during installation. For more information, see ***Static (One-Language) Mode***.

The default language for Natural Construct is English (*Language 1), which is always supported. Check with your local Software AG office to ensure that your language is supported.

Define SYSERR References

Each SYSERR reference number can have up to 15 distinct text entries — each one separated by a (/) slash delimiter. For information about setting up reference numbers, refer to the SYSERR utility in the Natural Utilities documentation.

To use SYSERR reference numbers in Natural Construct, the reference must follow a pattern where the first character is an * (asterisk) and the next four digits represent a valid SYSERR reference number. For example:

```
*nnnn
```

where * indicates the currently specified SYSERR message library and *nnnn* represents a valid reference number. To identify one of the 15 possible positions within a SYSERR reference number, use the following notation:

```
*nnnn.A
```

where *A* is a number from 1–9 or a letter from A–F. The numbers 1–9 represent the first nine positions and the letters A–F represent the 10th to 15th positions. For example, to reference the fifteenth position within a reference number, specify:

```
*nnnn.F
```

Note: We recommend that you always specify a position value, even if there is only one occupied position in the reference number. This eliminates the need to modify SYSERR references if additional positions are occupied in the future.

Use SYSERR References

You can use SYSERR reference numbers in several ways, such as:

- On Maps (Screen Prompts)
- For Panel Headings and PF-Key Names
- In Messages
- For Text Translation
- With Substitution Values

All text members, excluding the help text members, reside within the SYSERR utility. Each text member is identified by a two-part key — a SYSERR library name and a four-digit number.

- For more information about SYSERR, refer to the Natural Utilities documentation.
- For information about using SYSERR references in help text, see *Use Message Numbers*.

On Maps (Screen Prompts)

To display panels in many languages, Natural Construct uses a single map approach. Variables for all screen prompts are defined and initialized in a translation local data area (LDA) associated with each map.

Translation LDAs initialize the screen prompts with SYSERR references for the dynamic translation version or constants for the static version. All supplied LDAs use SYSERR references by default, but you can change this if desired. For more information about dynamic and static installations, refer to the installation documentation.

The one-to-one association between a map and its translation LDA is an effective method for naming and tracking panels and their prompts. Each supplied map and its translation LDA have identical names — except for the last character. The last character in a map name is "0" (zero) and the last character in a translation LDA name is "L". For example, the second specification panel for the Menu model is CUMNMB0 and the translation LDA is CUMNMBl.

Screen prompts are typically translated prior to displaying a panel, and panels usually have more than one prompt. For this reason, Natural Construct uses the CSUTRANS utility to receive a block of text and translate all references numbers. The CSTLDA library in SYSERR is dedicated to Natural Construct and contains all language-independent prompt text.

For more information, see ***CSUTRANS Utility***.

For Panel Headings and PF-Key Names

You define and maintain panel headings and PF-key names in the Administration subsystem: the first heading for a model specification panel on the Maintain Models panel and the PF-key settings on the Natural Construct control record.

Note: When we refer to panel headings and PF-key names, we are referring to the Natural Construct panels and PF-keys and not those used by the generated applications.

You define and maintain panel headings and PF-key names in the Administration subsystem: the first heading for a model specification panel on the Maintain Models panel and the PF-key settings on the Natural Construct control record.

If desired, you can use the CST-Modify model to generate a maintenance subprogram for the model that can override these defaults. Maintenance subprograms reference the #HEADER1 and #HEADER2 internal variables to display panel headings. If these headings are not overridden by the maintenance subprograms, Natural Construct automatically uses the defaults supplied by the nucleus (in the CU—PDA.#HEADER1 and CU—PDA.#HEADER2 variables). For more information about overriding panel headings, see ***Standard Parameters Panel***.

All Natural Construct panel headings and PF-key names support text or SYSERR references (the *nnnn.A* notations). For example, to name a PF-key "main" on the control record, enter one of the following:

- "*0033.5" (which corresponds to "main" in SYSERR)
- "main" (which disables the dynamic translation feature)

All heading and PF-key text is saved in the same SYSERR library as the prompt text (CSTLDA).

In Messages

All Natural Construct messages also support dynamic translation. Messages have action properties (verbs), whereas screen prompts have descriptive properties (adjectives). For this reason, the message and prompt text is stored in separate SYSERR libraries and use separate translation utilities:

- Messages are stored and maintained in the CSTMSG library and are accessed via the CNUMSG single message utility.
- Screen prompts are stored and maintained in the CSTLDA library and are accessed via the CSUTRANS utility.

If you change the supplied screen prompt text, ensure that the screen prompt and message text are consistent. If the message text references a different SYSERR number than the screen prompt, the message may be confusing.

With modules for which source is not supplied, Natural Construct uses the text substitution feature supported by the CNUMSG utility (where `:1::2::3:` are place holders for potential substitution values). For example, if the screen prompt is "Module name" and the message is "`:1::2::3:`is required", the message is displayed as: Module name is required.

This message substitution feature provides many benefits, including:

- Consistent use of panel and message text
- Reuse of common messages, such as "is required"
- Reduced volume of message translation
- Consistent wording between modules
- Support for a cleaner and crisper look

The following example shows a typical message and how it is coded:

```
ASSIGN CNAMSG.MSG-DATA(1) = CU—MAL.#GEN-PROGRAM
INCLUDE CU—RMSG '2001'
        ''':1::2::3:is required'''
        '#PDA-PROGRAM-NAME'
```

This assignment transfers the contents of the corresponding prompt variable into the first (of a possible three) substitution data member: `CNAMSG.MSG-DATA(1)`. The members are then transferred into an INCLUDE member that calls the CNUMSG utility.

In the preceding code example, CU—MAL is the translation LDA for the CU—MA0 map and CU—MAL. #GEN-PROGRAM is the prompt variable containing the initialized text (either "Module" or the SYSERR number that references "Module"). The 2001 on the INCLUDE line represents the SYSERR reference number that points to the message: "`:1::2::3:`is required". The "`:1::2::3:`is required" text below the INCLUDE code is used as an internal default should the text not be found.

You can use the Natural Construct messaging infrastructure to override the message lookup and force the CNUMSG utility to disregard the SYSERR reference number and use the text (":1::2::3:is required") instead. This feature is useful during model development because you can enter message text in the source code or test the code without calling the SYSERR utility. To do this for a single module, add a single line before the previous code example as follows:

```
ASSIGN CNAMSG.INSTALL-LANGUAGE = *LANGUAGE
```

To do this for an application, change the initial value for the CNAMSG.INSTALL-LANGUAGE variable and recompile all the Natural Construct model subprograms.

The following INCLUDE code members all retrieve message text, but process the text in different ways:

INCLUDE Code Member	Description
INCLUDE CU—RMSG	Retrieves and displays messages on current panel.
INCLUDE CU—SERR	Retrieves and sets error code messages and then exits current module.
INCLUDE CU—GMSG	Retrieves messages and continues processing (typically used for warning messages).
INCLUDE CU—GTXT	Retrieves messages and continues processing, but does not transfer the text to the CSASTD structure (typically used to perform initializations without corrupting the messaging data in CSASTD).

For Text Translation

You can translate text in one of two ways: mass translation from within the SYSERR utility or context translation from within Natural Construct, which uses the SYSERR utility to store text for all supported *Language values. English is the default language; it is always supplied and supported.

Since translation is typically performed once shortly after installation (or not at all if the product is delivered with the text translated), Natural Construct provides a special translation mode that is invoked via a command you can secure. This command, menut, accesses the Administration subsystem in translation mode with all translatable prompts and headings highlighted for easy identification.

Mass Translation

All Natural Construct text is available in SYSERR. The combination of the SYSERR library name and a four-digit number is the unique key or pointer to a particular text member. For example, the "*:1::2::3:*is required" message is stored in the CSTMSG library and its four-digit number is 2001; the "Module" screen prompt is stored in the CSTLDA library and its four-digit number is 1000.

In SYSERR, you can translate many messages one after the other (mass translation). This mechanism is fine for messages where the context is not critical. For example, the "*:1::2::3:*is required" message is universal and used frequently by all types of modules.

Screen prompts are more context sensitive; they may belong in a particular group or depend on a heading for meaning. To translate screen prompts, it is a good idea to perform a mass translation first and then check each panel individually for context. This is the most efficient way to translate a large number of text members, as this translation can be accomplished by less experienced Natural Construct users or a translation service.

Context Translation

Natural Construct's context (cursor-sensitive) translation provides a simple but effective method to check or change the results of a mass translation. It allows you to display a panel, place your cursor on highlighted text, press Enter, and be presented with a window in which you can change or translate the text. For example:

```
CSUTLATE                    Natural Construct
Jul 04                   Translate Short Message                   1 of 1

Language Short Message ( CSTLDA2101 )
———— ....+....1....+....2....+....3....+....4....+....5....+....6....+

English  Module/Model/Maps                                            /+20  ↩
```

This feature is even more convenient on a PC using Entire Connection, in which case you can double-click any prompt to perform the translation.

Notes:

1. You can also use the context translation mechanism to perform the original translation (instead of mass translation).
2. Because messages are displayed one at a time, they do not require context translation.

Since translation is typically performed once shortly after installation (or not at all if the product is delivered with the text translated), Natural Construct provides a translation mode command that you can secure. This command, menut, accesses the Administration subsystem in translation mode with all translatable prompts and headings highlighted for easy identification.

Unlike messages, which all use the same byte length, screen prompts vary in length depending on panel design and available space. For performance and space considerations, multiple screen prompts may share the same SYSERR location. For example, SYSERR number 2000 corresponds to the following text:

```
CSTLDA2000 Module/System/Global data area                    /+20
```

where `CSTLDA2000` indicates the SYSERR library and the four-digit number that identifies the values: Module, System, and Global data area (delimited by a "/"). Decimal numbers indicate which text is retrieved (for example, 2000.1 for Module, 2000.2 for System, and 2000.3 for Global data area). Since prompts can be different lengths, the /+20 notation indicates that each of these prompts can occupy up to 20 bytes on any panel they are used.

With Substitution Values

Substitution values are additional data that can be displayed with message text at runtime. For example, you can specify that Menu (the substitution value) be displayed with Main (the message text). The actual substitution value can be either text or another reference number. Most areas in Natural Construct that support reference numbers also support data substitution. For information about supported areas, see ***Supported Areas in Natural Construct***.

To use substitution values with a reference number, the reference number must be defined in the SYSERR utility with the *`:1::2::3:`* place holders. For more information, refer to *REINPUT Statement*, *Natural Statements* documentation.

To specify substitution values for a reference number that contains place holders, type the reference number (**`nnnn.A`* format), followed by a comma (,) delimiter, and up to three substitution values. For example, if you enter:

```
0200.1,Menu,Model
```

where `0200.1` corresponds to the message text `:1::2::3:Program`, and Menu and Model are the substitution values. At runtime, the following text is displayed:

```
Menu Model Program
```

In this example, Menu replaced the first place holder and Model replaced the second.

Note: If no substitution values are defined, the place holders are ignored.

You can enter text, or reference numbers, or both as substitution values. For example, if you enter:

```
0200.1,Menu,0502.4
```

where `Menu` is the first substitution value and `0502.4` is the second substitution value (which corresponds to the message text "Model"). At runtime, the following message is displayed:

```
Menu Model Program
```

Format SYSERR Message Text

In some areas where SYSERR references are used, you can specify how the retrieved message text is formatted at runtime. The following table describes the formatting characters:

Character	Description
,	Separates the **nnnn.A* notation from the format characters.
.	Fills the remaining blanks.
+	Centers the retrieved text.
<	Left-justifies the retrieved text. Typically, you will not use this character because retrieved text is left-justified by default.
>	Right-justifies the retrieved text.
/	Indicates the end of format characters and the beginning of the field length override. For example, "+/30" indicates that the first 30 characters of returned text are centered. Any additional characters are truncated. This character is used with alignment characters (such as +, <, or >).
NN	Indicates the field length override value. Using the example above (+/30), the field length override is 30 characters.

The following examples show different methods of formatting the text for SYSERR reference number 0210.1 (which references the text "Field Help"):

<table>
<tr><th>Format Specified</th><th>Result</th></tr>
<tr><td>*0210.1,+/24</td><td>Centers text in 24 bytes. At runtime, text is displayed as:<pre>| Field Help |</pre></td></tr>
<tr><td>*0210.1,>/24</td><td>Right-justifies text. At runtime, text is displayed as:<pre>| Field Help|</pre></td></tr>
<tr><td>*0210.1,/24 or *0210.1,</24</td><td>Left-justifies text (the default). At runtime, text is displayed as:<pre>|Field Help |</pre></td></tr>
<tr><td>*0210.1,./24</td><td>Left-justifies text and fills the remaining blank spaces with periods. At runtime, text is displayed as:<pre>|Field Help..................................|</pre></td></tr>
</table>

Supported Areas in Natural Construct

The following table lists the areas where you can use SYSERR references. The Substitutions column indicates whether substitution values are supported for the corresponding panel; the Formatting column indicates whether formatting is supported.

Location	Panel Element	Substitutions	Formatting
Maintain Control Record panel	PF-key names	No	No
	Panel indicators	No	No
Maintain Models panel	Description	Yes	No
Maintain Subprogram panel	Description	Yes	No
	PF-key names	No	No
Standard Parameters panel (CST-Modify model)	Header 1	Yes	Text centering only
	Header 2	Yes	Text centering only
	PF-key names	No	No
Translation local data areas (LDAs)	CNUMSG utility	Yes	Partial support
	CSUTRANS utility	Yes	Yes
Help Text editor	Header 1	Yes	No
	Header 2	Yes	No
	Hotlinks	Yes	No
	Body of help text	Yes	Yes

- For information on substitution values, see ***With Substitution Values***.
- For information on formatting, see ***Format SYSERR Message Text***.

The following table lists sections where you can find more information about each of the Natural Construct functions and utilities in which SYSERR reference numbers are supported:

To Learn More About	Refer To
Maintain Control Record panel	***Maintain Control Record Function***
Maintain Models panel	***Maintain Models Function***
Maintain Subprogram panel	***Maintain Subprograms Function***
CST-Modify model Standard Parameters panel	***Parameters for the CST-Modify Model***
Translation LDA utilities (CNUMSG and CSUTRANS)	■ ***CNUMSG Utility*** ■ ***CSUTRANS Utility***
Help Text editor	*Editing Help Text*

CSUTRANS Utility

Natural Construct translates screen prompts before they are displayed. As most panels have multiple prompts, Natural Construct incorporates the CSUTRANS utility to receive a block of text and translate all references to SYSERR numbers into the appropriate *Language text.

CSUTRANS translates 1:V data structures and is used extensively for dynamic translation. The utility reads through a supplied local data area, looking for one of two patterns: *`nnnn` or *`nnnn.A`.

The *`nnnn` pattern returns all text for that SYSERR number, whereas the *`nnnn.A` pattern returns only the text in the specified position (delimited by a /, such as *`nnnn`.1 for the first position, *`nnnn`.2 for the second, *`nnnn`.A for the 10th, etc.). The extension in the *`nnnn.A` pattern is alphanumeric; valid values range from 1–9 and A–F, for a total of 15 possible positions.

To locate the text corresponding to a message number, specify the library in which the SYSERR message numbers and text reside. By default, CSUTRANS checks the SYSERR message CSTLDA library. In most cases, you will create your own SYSERR message library. When you do, enter the library name in the #MESSAGE-LIBRARY field.

In addition to retrieving the appropriate language message text, CSUTRANS searches for any formatting characters and formats the text as appropriate.

You can also use SYSERR numbers to assign the INIT values for fields in the translation LDAs. These LDAs are passed through the CSUTRANS utility, which expects a certain data structure. The following example illustrates this structure for the Standard Parameters panel for the Batch model:

```
***SAG TRANSLATION LDA
***used by map CUBAMA0.
   1 CUBAMAL
   2 TEXT                            /* Corresponds to SYSERR message
   3 #GEN-PROGRAM             A   20 INIT<'*2000.1,.'>
   3 #SYSTEM                  A   20 INIT<'*2000.3,.'>
   3 #GDA                     A   20 INIT<'*2000.2,.'>
   3 #TITLE                   A   20 INIT<'*2001.3,.'>
   3 #DESCRIPTION             A   20 INIT<'*2001.2,.'>
   3 #GDA-BLOCK               A   20 INIT<'*2001.1,.'>
 R 2 TEXT
   3 TRANSLATION-TEXT
   4 TEXT-ARRAY               A    1 (1:120)
   2 ADDITIONAL-PARMS
   3 #MESSAGE-LIBRARY         A    8 INIT<'CSTLDA'>
   3 #LDA-NAME                A    8 INIT<'CUBAMAL'>
   3 #TEXT-REQUIRED           L      INIT<TRUE>
   3 #LENGTH-OVERRIDE         N    10 /* Explicit length to translate
```

Tip: To change the library name, use the #MESSAGE-LIBRARY variable.

Some of the important structural elements in this LDA are:

- The first comment line (`**SAG TRANSLATION LDA`) indicates that this is a translation LDA. During a Static install, Natural Construct scans for this comment line and replaces the SYSERR numbers with the appropriate text.
- The `CUBAMAL` level 1 structure name is typically the LDA name. You should use this qualifier to reference the variables.
- The level 3 variables (`#GEN-PROGRAM`, `#SYSTEM`, `#GDA-BLOCK`, etc.) are the screen prompts, which are initialized with a SYSERR number. All SYSERR numbers use the *`nnnn.A` notation and are listed in sequential order (so that CSUTRANS does not retrieve SYSERR *2000, then *2001, and then *2000 again).

  Note: The sequence order does not apply to the *`nnnn.A` notation extensions (.A). For example, you can list *2000.2 before *2001.1.

- The `TEXT-ARRAY` value must match the total number of bytes in all screen prompt variables to be translated.
- The `#MESSAGE-LIBRARY` value indicates the SYSERR library name used to retrieve text.
- The `#TEXT-REQUIRED` logical variable indicates whether translation is required for Natural Construct modules. If translation is required, `#TEXT-REQUIRED` ensures that translation is only performed once.

The SYSERR INIT values have the following format:

Position	Format
Byte 1	Must be an asterisk (*).
Bytes 2–5	Must be numeric and represent a valid SYSERR number. The first five bytes are mandatory. These values are used to retrieve the text associated with the corresponding SYSERR number and the current value of *Language. If the text for the current language is not available, CSUTRANS follows a modifiable hierarchy of *Language values until text is retrieved (you define this hierarchy in the DEFAULT-LANGUAGE field within the CNAMSG local data area). As the original development language, English (*Language 1) should always be available. **Note:** CSUTRANS does not perform substitutions (using `:1::2::3:`). To perform substitutions, call the CNUMSG subprogram. For information, see ***CNUMSG Utility***.
Byte 6	Can be a period (.), which indicates that the next byte is a position value.
Byte 7	Can be a position value. Valid values are 1–9, A (byte 10), B (byte 11), C (byte 12), D (byte 13), E (byte 14), F (byte 15), and G (byte 16). For example, *2000.2 identifies the text for SYSERR number 2000, position 2 (as delimited by a / in SYSERR). If the message for SYSERR number 2000 is Module/System/Global data area, only System is retrieved.

Position	Format
	If you reference the same SYSERR number more than once in a translation LDA, define the INIT values on consecutive lines to reduce the number of calls to SYSERR. (The position values for a SYSERR number can be referenced in any order.) **Tip:** To minimize confusion, we recommend that you use the `.A` extension even when there is only one position defined for the SYSERR number.
Byte 8	Can be a comma (,), which indicates that the next byte or bytes contain special format characters. Values specified before the comma (,) indicate what text to retrieve; values specified after the comma indicate how the text is displayed. **Note:** Although you can use a comma in byte 6 (instead of a period), use the `.A` extension in bytes 6 and 7.
Byte 9	After the comma, can be one of the following: ■ . (period) Indicates that the first position after the field name is blank and the remainder of the field prompt is filled with periods (`Module ..........:`, for example). ■ + Indicates that the text is centered using the specified field length override (see description of Byte 10). If you do not specify the override length, Natural Construct uses the actual field length. ■ < Indicates that the text is left-justified (this is the default). ■ > Indicates that the text is right-justified. ■ / Indicates that a length override value follows. This character is placed after the alignment character (+,< or >). For example, /+20 indicates that the text is centered within 20 bytes.
Bytes 10–16	After the / (override length indicator), indicates the override length in bytes.

If you want to use the override length notation (*0200.4,+/6, for example) and the LDA field is too small (A6, for example), define a larger field, redefine it using a shorter display value, and then use the override length notation. For example:

```
01  #FIELD-NAME                   A 12 INIT<'*0200.4+/6'>
01  Redefine #FIELD-NAME
  02  #SHORT-FIELD-NAME           A  6
```

CNUMSG Utility

Unlike CSUTRANS, the CNUMSG utility only retrieves text for one message at a time. It is typically used to retrieve warning or error messages, and sometimes to retrieve text for initialization. CNUMSG can also substitute values in the text it retrieves (up to a maximum of three substitution values). CNUMSG retrieves the message from SYSERR and checks to see whether the message has any substitution place holders. If it does, then the substitution text data members (CNAMSG.MSG-DATA(*)) are substituted into the appropriate place holder. If the data member is another SYSERR reference, it is retrieved and substituted. All unused substitution place holders are removed. By default, CNUMSG uses the CSTMSG SYSERR library for messages and the CSTLDA SYSERR library for substitution data fields.

This subprogram receives the following input:

- message number
- message library (CSTMSG by default)
- message text
- substitution data members
- message libraries for data members (CSTLDA by default)
- retrieval method
- default languages (used if message number is not located using *Language)

It processes message text based on one of the following retrieval methods:

- R

  Performs text retrieval based on message numbers. This method retrieves the SYSERR message *as is* without any text substitution. This method works well for cases where substitutions are not desirable and the `:1::2::3:` place holders should be left intact (for example, when generating a call to CNUMSG itself). A message number can be entered in either the Message Number or Message Text (Input) field. If a message number is entered in the Message Number field, the corresponding text is retrieved from the message library (CSTMSG by default) and displayed at runtime. If the Message Number field is blank, the subprogram scans the Message Text (Input) field for a message number. If one is located, it is replaced with its corresponding text from the message library.

  For example, assume message number *2309 corresponds to the message text "`:1::2::3:`does not exist". If this message number is located in either the Message Number or Message Text (Input) fields, the subsystem will retrieve the message text "`:1::2::3:`does not exist".

- S

Performs text substitutions in the Message Text (Input) field. If placeholders are found in the message text, this method substitutes the data into the `:1::2::3:` place holders without retrieving the main message text. For example, you can use this method to apply substitutions to a text string that is created programmatically. This method only substitutes the available (passed) data into the place holders. Unused place holders are removed. Placeholders are replaced at runtime with a value entered in one of the Message Substitution Data fields (`:1::2::3:`). Placeholders are entered in the following format: ":*N*:", where *N* identifies one of the three Message Substitution Data fields.

For example, if you enter the following message text: "`:1::2::3:`does not exist", and the Message Substitution Data field 1 is "File", and the Message Substitution Data field 2 is "NCST-CUSTOMER", the message text "File NCST-CUSTOMER does not exist" is returned.

- B or blank

  Performs text retrieval using methods R and S. This method retrieves the message text and performs the substitutions. It is the most commonly used method and is the default setting when the method is blank. This method also supports inline retrieval and substitution; that is, typing the message number and substitution values directly in the Message Text (Input) field.

  For example, if you enter the following entry in the Message Text (Input) field: "*2309,*2075.1,NCST-CUSTOMER", the subprogram assigns 2309 as the message number and retrieves the message "`:1::2::3:`does not exist". The first substitution value is retrieved from message 2075.1, which is "File". The second substitution value is the text "NCST-CUSTOMER". At runtime, "File NCST-CUSTOMER does not exist" is displayed.

All other method settings will return a fatal error without performing any actions.

If you are using message numbers, you can specify up to eight default languages. If the message text for the message number is not found using the currently selected language (*Language), the subprogram will search for the message in each of the specified default languages.

The search begins with the *Language code specified in the first Default Language field through to the last Default Language field in which a code is specified. If the message is still not located, the subprogram will search the message text for the default system *Language code of 1 (English).

Note: You can center text entered in the Message Text (Input) field using the ",+/*NN*" notation, where *NN* is the number of characters to be centered. For more information about message numbers and placeholders, see ***Use SYSERR References***.

Examples of Using the CNUMSG Utility

For the following examples, assume you want to create the message: `ADD Action Description is required` and the available SYSERR numbers and text are:

SYSERR Reference Number	SYSERR Library	SYSERR Text
*2001	CSTMSG	*:1::2::3:*is required
*1116.1	CSTLDA	Action/Subprogram
*1117.1	CSTLDA	Description

Example 1: Typical Text Retrieval

```
ASSIGN #DESCRIPTION ="*1117.1"...          /* Variable with a SYSERR reference
ASSIGN CNAMSG.MSG-DATA(1) = "ADD"  ..    ../* Hardcoded text
ASSIGN CNAMSG.MSG-DATA(2) = "*1116.1"      /* SYSERR Reference
ASSIGN CNAMSG.MSG-DATA(3) = #DESCRIPTION   /* Variable reference
INCLUDE CU—GMSG "2001"
       """:1::2::3:is required"
       """ """
```

Example 2: Text Retrieval Using a Comma as the Delimiter

```
ASSIGN CNAMSG.MSG = "*2001,ADD,*1116.1,*1117.1"
INCLUDE CU—GMSG " "
       """:1::2::3:is required"
       """ """
```

Both of these examples build the same message. Example 1 is the preferred method because it is much more explicit. The method in Example 2 is useful when only the message text is available and the input must be entered in one field, such as the `Description`, `Header`, or `Title` fields.

Note: Example 2 also supports centering. If you specify `+/`*NN* in your message text, CNUMSG uses the *NN* value as the centering length and removes the remainder of the text (the `,+/NN` pattern).

Static (One-Language) Mode

By default, Natural Construct is installed in dynamic (multilingual) mode, which allows users to display Natural Construct in any available language. If you intend to operate Natural Construct in one language only and do not require dynamic translation, you can replace all SYSERR references with text when Natural Construct is installed. During installation, Natural Construct provides a Static option that retrieves and replaces the **nnnn* references with the appropriate *Language text.

Notes:

1. Before using the Static option, check with your local Software AG office to ensure that your language is supported. If you are installing a static version in any language except English, which is always supported, review all messages in the CSTLDA library in SYSERR to ensure they are translated into the desired language.
2. Installing in static mode does not limit your ability to generate multilingual applications; static mode applies to the interface only.

The Static option does not replace every SYSERR reference with text; it only replaces SYSERR references in the most frequently used modules. The following table describes the areas affected and the replacements made:

Area	Replacements
Screen prompts	In all translation LDAs for which source is supplied (CU prefix), the Static option replaces references with text. To identify a translation LDA, Natural Construct checks the first comment line for `**SAG TRANSLATION LDA`.
Translation LDAs	For the most frequently used translation LDAs for which source is not supplied, you can generate static text LDAs and subprograms. For information, see *Create Performance LDAs and Subprograms*.
Headings and PF-key names	For all panel headings and PF-key names (which are installed with SYSERR references), you have the option of replacing the references with text.
Messages	Dynamically translated at runtime (since messages are only displayed during an error or warning condition).
Help text	Dynamically displayed at runtime (displayed on request).

Note: Natural Construct can also use the English text supplied with each INCLUDE code member and bypass the SYSERR retrieval process (see *In Messages*).

There are two options for installing in static mode:

- *Install Natural Construct in Static Mode*
- *Create Performance LDAs and Subprograms*

You can specify either or both options.

Note: If you are installing a static version in any language except English, review all messages in the CSTLDA library in SYSERR to ensure they are translated into the desired language.

Install Natural Construct in Static Mode

To install Natural Construct in static mode:

1. Log onto the SYSCST library.
2. Enter "NCSTI" (Natural Construct Install) at the Natural prompt.

   The Natural Construct Installation main menu is displayed. For example:

```
NCSTI          ***** N A T U R A L   C O N S T R U C T *****          ↩
Feb 27                  - Installation Main Menu -              9:52 AM
                                                                      ↩
               Code Function                                          ↩
               ---- ----------------------------------------          ↩
                S   Static Install (one language)                     ↩
                L   Create Performance LDAs                           ↩
                I   Create Performance Subps                          ↩
                ?   Help                                              ↩
                .   Terminate                                         ↩
               ---- ----------------------------------------          ↩
          Code: _                                                     ↩
                                                                      ↩
                                                                      ↩
                                                                      ↩
                                                                      ↩
                                                                      ↩
                                                                      ↩
                                                                      ↩
                                                                      ↩
Direct command...: ______________________________________________________
Enter--PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12-
       help  retrn quit        flip                                  main ↩
↩
```

3 Enter "S" in `Code`.

The Static Install (one language) window is displayed. For example:

```
INSTALL         ***** N A T U R A L   C O N S T R U C T *****

Feb 27               - Static Install (one language) -           9:57 AM


Enter the language in which you would like Natural Construct

installed (Any PF-key to quit): 1_

```

4 Enter the number for the language in which you want to install Natural Construct (for example, "2" for German, "3" for French).

Natural Construct recreates all the LDAs for the model specification panels and replaces the SYSERR references to field prompts with the text for the language specified. The following window is displayed:

```
INSTALL         ***** N A T U R A L   C O N S T R U C T *****

Feb 27               - Static Install (one language) -          10:37 AM


All data areas have been populated with text appropriate to

language 1  .

In order to complete this process, please recompile all

Natural modules in the SYSCST library beginning with CU, CG

and copy the object code for these modules to SYSLIBS.

```

Note: Set the Natural RUNTIME parameter to 40.

5 Perform a CATALL on modules beginning with "CU" or "CG" in the SYSCST library.

You need only select the subprogram and local data area (LDA) modules. In addition, mark the `Catalog ALL Source-programs` option to catalog all source modules. You may want to do this step in batch mode, because many modules are affected. You can use the following input:

```
LOGON SYSCST
CATALL CU*,,X,,,X,,,,X,,,,,
CATALL CG*,,X,,,X,,,,X,,,,,
FIN
```

6 Copy the object code from these modules into the SYSLIBS library.

If you prefer to do this in batch mode, the SYSMAIN input commands are supplied below. Ensure that the IM=D parameter is set in your NATPARM. Use the following batch input:

```
LOGON SYSTEM
SYSMAIN
MENU C,C,CU*,TYPE,N,FM,SYSCST,DBID,xxx,FNR,yyy,TO,SYSLIBS,DBID,xxx,%
FNR,yyy,REP
SYSMAIN
MENU C,C,CG*,TYPE,N,FM,SYSCST,DBID,xxx,FNR,yyy,TO,SYSLIBS,DBID,xxx,%
FNR,yyy,REP
FIN
```

Create Performance LDAs and Subprograms

Regardless of whether you choose the Static Install function or not, this option will enhance performance by creating several subprograms that eliminate calls to SYSERR to build many of the frequently used screens (such as the Generation main menu). Because these programs are not supplied in source form, use the Create Performance LDAs function to create LDAs containing the text appropriate to the desired language and then use the Create Performance Subps function to create the performance subprograms. You can repeat these two steps as many times as desired, depending on how many languages you want to make available.

Note: Natural Construct supplies the performance subprograms for English. If you are running Natural Construct in a language for which these subprograms have not been created, the English subprograms will be invoked.

To create performance LDAs and subprograms for the Natural Construct nucleus:

1. Copy the contents (source and object) of the SYSCST00 library into the SYSCST*nn* library (where *nn* is the language code for the language you want to support, such as 1 for English, 2 for German, 3 for French).
2. Log onto the SYSCST*nn* library.
3. Enter "NCSTI" (Natural Construct Install) at the Natural prompt.

   The Natural Construct Installation main menu is displayed.

   Note: When running NCSTI to create these LDAs and subprograms, the DC and ID characters must be set to the default (DC=. and ID=,).

4. Enter "L" in `Code`.

   The Create Performance LDAs window is displayed. For example:

```
INSTALL2        ***** N A T U R A L   C O N S T R U C T *****
Feb 27                   - Create Performance LDAs -                10:18 AM

NOTE: You must be in library SYSCSTnn (where nn represents
      the language number) in order to execute this function.
      This step may be repeated for as many languages
      as desired.

You are currently in library: SYSCST01
About to create performance LDAs for language: 1
Press ENTER to continue - any PF-key to stop.
```

5 Press Enter.

A confirmation window is displayed. For example:

```
INSTALL2        ***** N A T U R A L   C O N S T R U C T *****
Feb 27                   - Create Performance LDAs -                10:21 AM

All data areas have been populated with text appropriate to
language 1   .  Please CATALL this library ( SYSCST01 )
before creating the Performance Subprograms.
```

Note: You must be logged onto the SYSCST*nn* library corresponding to the language for which you are creating the LDAs. This allows multiple languages to be supported, since the LDAs are created in different libraries.

6 Press Enter.

7 Perform a CATALL on this library, ensuring that all 10 LDAs are cataloged successfully.

8 Log onto the SYSCST library.

9 Enter "NCSTI" at the Natural prompt.

The Natural Construct Installation main menu is displayed.

10 Enter "I" in `Code`.

The Create Performance Subps window is displayed. For example:

```
CSTTRANS        ***** N A T U R A L   C O N S T R U C T *****                ↩

Feb 27                 - Create Performance Subps -                10:24 AM
                                                                            ↩

NOTE:  This function must be executed from library SYSCST                   ↩

                                                                            ↩

Enter the language number for which you would like                          ↩

performance subprograms generated: __                                       ↩

(Press any PF-key to stop)                                                  ↩
 ↩
```

11 Enter the number of the language for which you have created performance LDAs.

Natural Construct creates object-only performance subprograms for the specified language.

12 Copy the performance subprograms from the SYSCST library to the SYSLIBS library.

These modules begin with "CZ" and end with the *Language value for the language in which you are installing (for example, CZHOBJ2 for German).

13 Log onto the SYSCSTX library and edit the CSXDEFLT subprogram as follows:

- Set the PERFORMANCE default to TRUE (must be in uppercase). For example:

```
**SAG DEFINE EXIT GENERATE-CODE
*
* Your code to implement defaulting for your CST models.
DECIDE ON FIRST VALUE CSADEFLT.PARM-NAME
  VALUE 'PERFORMANCE'
    ASSIGN CSADEFLT.PARM-VALUE = 'TRUE'
  NONE
    IGNORE
END-DECIDE
**SAG END-EXIT
```

- Save the CSXDEFLT subprogram in the SYSCSTX library.
- Use the Natural SYSMAIN utility to copy CSXDEFLT to the SYSCST library.
- Catalog CSXDEFLT in the SYSCST library.
- Use the SYSMAIN utility to copy the CSXDEFLT object code to the SYSLIBS library.

A Appendix A: Glossary of Terms

The following terms are used throughout this documentation:

Term	Definition
Browse program	Program that retrieves records from a specified file and allows users to select a record for processing. Sometimes referred to as a query program.
Browse a file	View the records in a specified file.
Code frame	Block of code that performs a specified function. A code frame is the basic element of a model; it is a skeleton outline of the code generated by the model.
Constant	Value that is always the same.
Copycode	Static code that is provided to copy and use in INCLUDE statements.
Cursor-sensitive or Cursor sensitivity	Ability to move the cursor to an item on the screen and press Enter to select the item. If you are using a PC connection to access Natural Construct, you can double-click with the mouse to select.
Data area	Natural module in which data is stored. For example, a parameter data area (PDA) stores parameters that are passed between subprograms, and a global data area (GDA) stores data that is used by all programs within an application.
Enter	Type a value in a field and press Enter (or Return).
Execute	Start or display a program, menu, panel, editor, utility, etc. Also referred to as "invoke".
Field	Area in a window or on a panel that either displays information or requires the user to specify information (for example, type or select information).
Function	Menu option, for example, the Maintain Models function on the Administration main menu.
Helproutine	Natural module that displays a help panel.
Invoke	See Execute.
Mark a field	Type any non-blank character in the field. **Note:** You may also be required to press the Enter key.

Term	Definition
Model	Natural Construct template used to record specifications and generate source code into a Natural buffer.
Module	Any object that is generated by Natural Construct or created in Natural.
Object	Any entity that represents a business function and is used by Natural Construct.
Optional field	Field for which information is optional rather than required.
Panel	Screen or map on which parameters may be specified.
Parameter	Value for a field.
PF-key	Program function key. To perform the associated function, press that key. For example, pressing PF1 (help) displays help information.
Program	Block of code that performs a function, such as a subprogram, subroutine, helproutine, etc. Also referred to as a module.
Query program	See Browse program.
Required field	Field for which input is required.
Return code	Code entered on a menu to return to the previous panel. The return code on Natural Construct menus is a period (.)
Scroll	Move forward (down), backward (up), left, or right through the information displayed on a panel or in a window.
Specify	Supply a value for an input field (for example, by typing a value in the field and pressing the Enter key or by marking the field).
Subprogram	Self-contained block of code that is called via parameters by a program to perform a function.
Subroutine	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 have the same format and different values at generation time.
Terminate	End your Natural Construct session.
User exit	Area in the program code that is reserved for user-defined functions. In these areas, users can change the functionality of the generated functions to suit their own requirements. User exit code is preserved when the program is regenerated.
Utility	Supplied program/subprogram that performs a specific function (for example, the model load utility).
Variable	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	Separate, self-contained area displayed on a panel (for example, a help window).