Natural Construct

Administration and Modeling

software AG

Manual Order Number: CST451-022IBU

This document applies to Natural Construct Version 4.5 and to all subsequent releases.

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

Readers' comments are welcomed. Comments may be addressed to the Documentation Department at the address on the back cover or to the following e-mail address:

Documentation@softwareag.com

© Copyright Software AG 2003
All rights reserved
Printed in the Federal Republic of Germany

The name Software AG and/or all Software AG product names are either trademarks or registered trademarks of Software AG. Other company and product names mentioned herein may be trademarks of their respective owners.

TABLE OF CONTENTS

PREFACE

Mainframe and Unix Platforms 18
Purpose and Structure of this Documentation 19
Document Conventions 21
Other Resources 22
Related Documentation 22
User Documentation 22
Installation Documentation 22
Other Documentation 22
Related Courses 23

1. INTRODUCTION TO NATURAL CONSTRUCT

What is Natural Construct? 26
Natural Construct Subsystems 27
Invoking Natural Construct 28
Natural Construct Libraries 29
SYSLIBS Library 30
SYSTEM (FNAT) Library 30
SYSCST Library 30
SYSCSTX Library 30
NCSTDEMO, NCSTDEM2, NCSTDEMV, and NCSTDEMS Libraries 31
USERLIB Library 31
Executing Generation Facilities from a Steplib with Natural Security Installed 31
Using PF-Keys 32
Help and Return Codes on Menus 33
Accessing Online Help 33
Panel-Level Help 33
Field-Level Help 35
Passive 35
Active 35
Automatic Upper Case Translation 36
Natural Construct Messages 37
Storing Saved Modules 37
Using Direct Commands 37

2. USING THE ADMINISTRATION SUBSYSTEM

Administration Main Menu 40
Maintain Models Function 42
PF4 (frame) 46
Code Frame Menu Function 49
Edit Code Frame Function. 50
Save Code Frame Function 50
List Code Frames Function 51
Purge Code Frame Function 52
Clear Edit Buffer Function 52
Print Saved Code Frame Function. 52
Maintain Subprograms Function 52
Maintain Control Record Function 53
Compare Menu Function. 56
Compare Models Function 57
Comparing the Components of Two Models in Different Files 58
Comparing the Components for Two Models in the Same File 59
Compare a Range of Models with the Same Name in Two Files 60
Compare Frames Function. 61
Compare Two Code Frames 61
Comparing a Range of Frames with the Same Name in Two Files. 63
Comparing All the Frames Used by Two Models. 65
Drivers Menu Function 66
Multilingual Support for Natural Construct 67
PF12 (lang) 68
Administration Main Menu in Translation Mode 69
Using Translation Mode 70
Editing Text in the Current Language 71
Translate Text to Another Language 72
User Exit Subprograms to Implement Security. 74
Defining Default Specifications 75
Using the CSXDEFLT Subprogram 76
Using Library-Specific Defaulting 77
Using CSXCNAME Overrides 78
Assigning Corporate Defaults 78
Using Predict Keywords 79
Defining a Default Primary Key 79
Defining a Default Logical Hold Field 80
Defining a Default Object Description 80

3. USING THE CODE FRAME EDITOR

Invoking the Code Frame Editor 82
Using the Code Frame Editor 84
Order of Command Execution. 87
Line Commands. 88
Edit Commands 90
Positional Edit Commands 92
Maintain Current PF-key Profile Window 92
Edit Recovery 94
GUI Sample Subprogram 94

4. CREATING NEW MODELS

Components of a Natural Construct Model. 96
How Natural Construct Executes a Model 98
Building a New Model 99
Step 1: Define the Scope of the Model 99
Is the Scope Too Broad? 100
Is the Scope Too Narrow? 100
What to Generate and Why 100
Step 2: Create the Prototype 101
Step 3: Scrutinize the Prototype 101
Step 4: Isolate the Parameters in the Prototype 101
Which Elements Need to be Parameterized?. 101
Remove Redundant Parameters. 102
Compile Time Versus Execution Time 102
Step 5: Create Code Frame(s) and Define the Model 103
Creating the Code Frames 103
Substitution Parameters 104
Parameters Supplied by Generation Subprograms 104
Parameters Supplied by Nested Code Frames. 105
Parameters Supplied by User Exits 106
Code Frame Conditions 109
Defining the Model 112
Naming Conventions for Model Components. 113
Step 6: Create the Model PDA 115
Model PDA 118
CU—PDA 120
CSASTD PDA. 130
Step 7: Create the Translation LDAs and Maintenance Maps 131
Format of the Translation LDAs 131
Maintenance Maps. 134

Step 8: Create the Model Subprograms 134
Modify Subprograms 134
When are Modify Subprograms Invoked? 135
Pre-generation Subprogram 136
Generation Subprograms 136
Post-generation Subprogram 137
Stack Order of Substitution Parameters 139
Blanks versus Nulls 139
Clear Subprogram 139
When are Clear Subprograms Invoked? 139
Save Subprogram 140
Read Subprogram 141
Sample User Exit Subprograms 142
Document Subprogram 142
Testing the Model Subprograms 143
Debugging a Model 146
Miscellaneous Tips and Precautions 148
Implementing Your Model 149
Statement Models 149
Code Alignment of Generated Statement Models 150
Utility Subprograms and Helproutines 151

5. NEW MODEL EXAMPLE

Procedure to Build the Example Model 154
Step 1: Define the Scope of the Model 154
Step 2: Create the Prototype 154
Step 3: Scrutinize the Prototype 155
Step 4: Isolate the Parameters in the Prototype 155
Parameters for the Program Header 155
Parameters for the Program Body 156
Step 5: Create the Code Frame and Define the Model 156
Creating the Code Frame 156
Defining the Model 157
Step 6: Create the Model PDA 159
Step 7: Create Translation LDAs and Maintenance Maps 160
Creating the Translation LDAs 160
Creating the Maintenance Maps 161
Step 8: Create the Model Subprograms 163
Creating the Modify Subprograms 163
Creating the Pre-generation Subprogram 166
Creating the Post-generation Subprogram 167
Creating the Clear Subprogram 168

Creating the Save Subprogram 169
Creating the Read Subprogram 170
Creating the Generation Subprogram 171
Creating the Document Subprogram 172
Testing the Model Subprograms 173
Step 9: Implement the Model 173

6. CST-PDA MODEL

Introduction 176
Parameters for the CST-PDA Model. 177
Standard Parameters Panel 177
Layout of the Generated Model PDA 178

7. CST-CLEAR MODEL

Introduction 180
Parameters for the CST-Clear Model 181
Standard Parameters Panel 181
User Exits for the CST-Clear Model 182

8. CST-READ MODEL

Introduction 184
Parameters for the CST-Read Model 185
Standard Parameters Panel 185
User Exits for the CST-Read Model 186

9. CST-SAVE MODEL

Introduction 188
Parameters for the CST-Save Model. 189
Standard Parameters Panel 189
User Exits for the CST-Save Model 190

10. CST-MODIFY AND CST-MODIFY-332 MODELS

Introduction 192
CST-Modify Model. 193
Parameters for the CST-Modify Model. 194
Standard Parameters Panel 194
PF5 (windw) 199
PF6 (pfkey) 200
User Exits for the CST-Modify Model 202
CST-Modify-332 Model 203
Example of a Model Modify Subprogram. 204

Parameters for the CST-Modify-332 Model 205
Standard Parameters Panel 205
User Exits for the CST-Modify-332 Model 207

11. CST-PREGEN MODEL
Introduction 210
Parameters for the CST-Pregen Model 211
Standard Parameters Panel 211
User Exits for the CST-Pregen Model. 212

12. CST-POSTGEN MODEL
Introduction 214
Parameters for the CST-Postgen Model 215
Standard Parameters Panel 215
User Exits for the CST-Postgen Model. 216

13. CST-FRAME MODEL
Sample Subprograms. 218
Generation Subprograms. 219
Parameters for the CST-Frame Model 220
Standard Parameters Panel 220
User Exits for the CST-Frame Model 221

14. CST-DOCUMENT MODEL
Introduction 224
Parameters for the CST-Document Model 225
Standard Parameters Panel 225
Additional Parameters Panel 226
User Exits for the CST-Document Model. 227

15. CST-VALIDATE MODEL
Introduction 230
Parameters for the CST-Validate Model. 231
Standard Parameters Panel 231
User Exits for the CST-Validate Model 233
Coding Validations 233
Validating Array Fields. 234

16. CST-STREAM MODEL

Introduction 236
Parameters for the CST-Stream Model 237
Standard Parameters Panel 237
User Exits for the CST-Stream Model 238

17. CST-SHELL MODEL

Introduction 240
Parameters for the CST-Shell Model 241
Standard Parameters Panel 241
User Exits for the CST-Shell Model 243

18. USER EXITS FOR THE ADMINISTRATION MODELS

What are User Exits? 246
Reusing User Exit Code 246
Invoking the User Exit Editor 247
Defining User Exits 249
Supplied User Exits 249
ADDITIONAL-INITIALIZATIONS 249
ADDITIONAL-SUBSTITUTION-VALUES 250
ADDITIONAL-TRANSLATIONS 251
AFTER-INPUT 252
AFTER-INVOKE-SUBPANELS 252
ASSIGN-DERIVED-VALUES 253
BEFORE-CHECK-ERROR 254
BEFORE-INPUT 254
BEFORE-INVOKE-SUBPANELS 255
BEFORE-REINPUT-MESSAGE 255
BEFORE-STANDARD-KEY-CHECK 256
CHANGE-HISTORY 256
DESCRIBE-INPUTS 256
END-OF-PROGRAM 257
GENERATE-CODE 258
GENERATE-SUBROUTINES 259
GENERATE-VALIDATIONS 259
INPUT-ADDITIONAL-PARAMETERS 259
INPUT-SCREEN 260
LOCAL-DATA 260
MISCELLANEOUS-SUBROUTINES 260
MISCELLANEOUS-VARIABLES 261
PARAMETER-DATA 261
PF-KEYS 262

PROCESS-SPECIAL-KEYS 263
PROVIDE-DEFAULT-VALUES 263
SAVE-PARAMETERS 264
SET-CONDITION-CODES 265
START-OF-PROGRAM 266
SUBSTITUTION-VALUES 266
VALIDATE-DATA 266

19. MODIFYING THE SUPPLIED MODELS

Introduction 270
Modify the Supplied Models 271
Modifying Code Frames 271
Modifying the Model Subprograms 272
Modifying Copycode (CC*) and External Data Areas and Subprograms (CD*) 273
Example of Modifying a Model 274
Using Steplibs to Modify Models 277
Invoking Natural Construct From a Steplib 278

20. EXTERNAL OBJECTS

Introduction 280
Object Categories 282
Processing Errors 282
Passing Structure Names 283
Restricted Data Areas 283
Callback Functions 283
Subprogram Chaining 284
No Subprogram Chaining (CPUELNX) 284
Using Subprogram Chaining (CPUELRD) 285
Natural-Related Subprograms (CNU*) 286
CNUEL Subprogram 287
Drivers Menu Option 287
CNUELNX Subprogram 288
CNUELNX On Unix Platforms 288
Drivers Menu Option 289
CNUERMSG Subprogram 290
Drivers Menu Option 290
CNUEXIST Subprogram 291
Drivers Menu Option 291
CNUGDABL Subprogram 292
Drivers Menu Option 292
CNUGDAEL Subprogram 293
Drivers Menu Option 293

CNUGENDA Subprogram 294
Drivers Menu Option. 294
CNUMPPRF Subprogram 295
Drivers Menu Option. 295
CNUMSG Subprogram 296
Drivers Menu Option. 298
CNUPEXST Subprogram 299
Drivers Menu Option. 299
CNUSEL Subprogram. 300
Drivers Menu Option. 300
CNUSRCNX Subprogram. 301
Drivers Menu Option. 301
CNUSRCRD Subprogram. 302
Drivers Menu Option. 302
Natural-Related Helproutines (CNH*) 304
CNHMDL Helproutine 304
CNHMSG Helproutine 305
CNHOBJ Helproutine 305
Natural Construct Generation Utility Subprograms (CSU*). 307
CSU-VAR Subprogram. 308
Drivers Menu Option. 308
CSUBANN Subprogram 309
CSUBLDRP Subprogram 309
CSUBMIT Subprogram (Mainframe). 310
CSUBYTES Subprogram 310
Drivers Menu Option. 310
CSUCASE Subprogram 311
Drivers Menu Option. 311
CSUCCMD Subprogram. 312
CSUCENTR Subprogram 313
Drivers Menu Option. 313
CSUCOMPR Subprogram. 314
Drivers Menu Option. 314
CSUCTRL Subprogram 315
CSUCURS Subprogram 315
CSUCURS1 Subprogram 316
CSUDB2SP Subprogram. 316
Drivers Menu Option 317
CSUDELFF Subprogram 318
Drivers Menu Option. 318
CSUDEFLT Subprogram 319

CSUDYNAT Subprogram 320
Drivers Menu Option 321
CSUEMLEN Subprogram 322
Drivers Menu Option 322
CSUENDX Subprogram 323
Drivers Menu Option 323
CSUFDEF Subprogram 324
Drivers Menu Option 324
CSUFRVAR Subprogram 325
Drivers Menu Option 325
CSUGEN Subprogram 326
CSUHEADS Subprogram 326
Drivers Menu Option 327
CSUINCL Subprogram 327
Drivers Menu Option 327
CSUIS Subprogram 328
Drivers Menu Option 328
CSULABEL Subprogram 329
Drivers Menu Option 329
CSULENGT Subprogram 330
Drivers Menu Option 330
CSULPS Subprogram 331
CSUMAX Subprogram 331
Drivers Menu Option 331
CSUMIMAX Subprogram 332
Drivers Menu Option 332
CSUMODEL Subprogram 333
Drivers Menu Option 333
CSUMORE Subprogram 334
Drivers Menu Option 335
CSUMPBOX Subprogram 335
CSUMPCPR Subprogram 336
CSUMPDUP Subprogram 336
CSUMPLAY Subprogram 337
Drivers Menu Option 337
CSUMPMMS Subprogram 338
CSUMPOVL Subprogram 338
CSUMPREG Subprogram 339
Drivers Menu Option 339
CSUMPTAB Subprogram 340
CSUMPTST Subprogram 340
Drivers Menu Option 340

CSUNATFM Subprogram 341
Drivers Menu Option 341
CSUNEWX Subprogram 342
Drivers Menu Option 342
CSUPARMS Subprogram 343
Drivers Menu Option 343
CSUPARTY Subprogram 344
CSUPPER Program 344
Drivers Menu Option 344
CSUREADS Subprogram 345
CSUREF Subprogram 346
CSUSCAN Subprogram 347
Drivers Menu Option 347
CSUSELFV Subprogram 348
CSUSETKY Subprogram 349
Drivers Menu Option 349
CSUSETW Subprogram 350
Drivers Menu Option 350
CSUSORT Program 351
CSUSPLIT Program 351
Drivers Menu Option 352
CSUSUB Program (Mainframe) 352
CSUSUBP Subprogram 353
Drivers Menu Option 353
CSUTEST Program 354
Drivers Menu Option 354
CSUTLATE Subprogram 355
CSUTRANS Subprogram 356
Drivers Menu Option 358
CSUXCHK Subprogram 359
Drivers Menu Option 359
CSU2LONG Subprogram 360
Drivers Menu Option 360
Predict-Related Subprograms (CPU*) 361
With Natural Security Installed 361
CPU-OBJ Subprogram 362
Drivers Menu Option 362
CPU-OBJ2 Subprogram 363
CPU-OREL Subprogram 364
CPU-VIEW Subprogram 364
Drivers Menu Option 365

CPUEL Subprogram 366
Drivers Menu Option. 366
CPUELDE Subprogram 367
CPUELKY Subprogram 368
CPU-FREL Subprogram 368
CPUELNX Subprogram 369
Drivers Menu Option. 369
CPUELRD Subprogram 370
Drivers Menu Option. 371
CPUELVE Subprogram 372
Drivers Menu Option. 372
CPUEXIST Subprogram 373
Drivers Menu Option. 373
CPUFI Subprogram 374
Drivers Menu Option. 374
CPUHOLD Subprogram 375
CPUKY Subprogram 375
CPUREDEF Subprogram 375
Drivers Menu Option. 376
CPURL Subprogram 376
Drivers Menu Option. 377
CPURLRD Subprogram 377
Drivers Menu Option. 378
CPUSUPER Subprogram 378
Drivers Menu Option. 379
CPUUNIQ Subprogram. 379
CPUVE Subprogram 380
Drivers Menu Option. 380
CPUVERUL Subprogram 381
CPUXPAND Subprogram. 381
Drivers Menu Option. 382
Predict-Related Helproutines (CPH*). 384
CPHEL Subprogram 384
CPHELB Subprogram 384
CPHFI Helproutine 385
CPHFIB Subprogram 385
CPHPRED Helproutine 385
CPHRL Helproutine 386
CPHSET Helproutine 387
General Purpose Generation Subprograms (CU--*) 388
CU--EM Subprogram 388
CU--LRP Subprogram 388

CU--MSG Subprogram 389
CU--UL Subprogram 389

21. SUPPLIED ADMINISTRATION UTILITIES
Introduction 392
Import and Export Utilities 393
Multiple Code Frame Export Utility 394
Multiple Code Frame Import Utility 395
Frame Hardcopy Utility 396
Comparison Utilities 397
CSGCMPS Utility 397
CSGCMPL Utility 398
Using CSGCMPL Online 398
Using CSGCMPL in Batch 399
Upper Case Translation Utility 401
Additional Utilities 402

22. USING SYSERR FOR MULTILINGUAL SUPPORT
Introduction 404
Maintenance 404
Translation 404
Defining SYSERR References 405
Using SYSERR References 406
On Maps (Screen Prompts) 406
For Panel Headings and PF-Key Names 407
In Messages 407
For Text Translation 409
Mass Translation 409
Context Translation 410
With Substitution Values 411
Formatting SYSERR Message Text 412
Supported Areas in Natural Construct 413
CSUTRANS Utility 415
CNUMSG Utility 419
One Language (Static) Mode 421

APPENDIX A — GLOSSARY OF TERMS 423

INDEX 427

PREFACE

This preface explains how information is presented for different platforms, as well as the purpose and structure of the *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.

The following topics are covered:

- **Mainframe and Unix Platforms**, page 18
- **Purpose and Structure of this Documentation**, page 19
- **Document Conventions**, page 21
- **Other Resources**, page 22

Mainframe and Unix Platforms

The majority of the information in this documention applies to all supported platforms. When differences in operation exist for different platforms, the following methods are used to explain them:

- When a description applies to only one supported platform, the platform is indicated in parentheses. For example: (mainframe) or (Unix).
- When a minor difference exists, it is explained in parentheses. For example: “Enter CSTG at the Next prompt (in the Direct command box on Unix).”
- When a more significant difference exists, a note explains the difference. For example:

Unix Note:
Enter ...

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

Purpose and Structure of this Documentation

This documentation explains how to invoke and use the Administration subsystem of Natural Construct. Its purpose is to help Natural Construct administrators:

- 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.

The following table describes the information contained in each chapter:'

Chapter	Title	Topics
1	**Introduction to Natural Construct**, page 25	Contains a general description of Natural Construct and the basic information you need to use the Administration subsystem.
2	**Using the Administration Subsystem**, page 39	Describes the features and functions of the Administration main menu, in both standard and translation mode, as well as the implementation of security precautions.
3	**Using the Code Frame Editor**, page 81	Describes the fields and features of the Code Frame editor, as well as the line and edit commands you can use in the editor.
4	**Creating New Models**, page 95	Describes the procedure for creating a new Natural Construct model.
5	**New Model Example**, page 153	Contains an example of how to create a new model using the steps described in Chapter 4.
6	**CST-PDA Model**, page 175	Describes the model that generates parameter data areas (PDAs) for your models.
7	**CST-Clear Model**, page 179	Describes the model that generates clear subprograms for your models.
8	**CST-Read Model**, page 183	Describes the model that generates read subprograms for your models.
9	**CST-Save Model**, page 187	Describes the model that generates save subprograms for your models.
10	**CST-Modify and CST-Modify-332 Models**, page 191	Describes the model that generates modify subprograms for your models.

Chapter	Title	Topics (continued)
11	**CST-Pregen Model**, page 209	Describes the model that generates pre-generation subprograms for your models.
12	**CST-Postgen Model**, page 213	Describes the model that generates post-generation subprograms for your models.
13	**CST-Frame Model**, page 217	Describes the model that generates frame subprograms for your models.
14	**CST-Document Model**, page 223	Describes the model that generates document subprograms for your models.
15	**CST-Validate Model**, page 229	Describes the model that generates validate subprograms for your models.
16	**CST-Stream Model**, page 235	Describes the model that generates stream subprograms for your models.
17	**CST-Shell Model**, page 239	Describes the model that generates a template for a model subprogram.
18	**User Exits for the Administration Models**, page 245	Describes the user exits supplied for the Natural Construct administration models.
19	**Modifying the Supplied Models**, page 269	Describes how to modify models supplied with Natural Construct.
20	**External Objects**, page 279	Describes the supplied subprograms and helproutines.
21	**Supplied Administration Utilities**, page 391	Describes the supplied utilities for all supported platforms.
22	**Using SYSERR for Multilingual Support**, page 403	Describes how to use the SYSERR utility to provide multilingual support.
Appendix	**Appendix A — Glossary of Terms**, page 423	Contains a glossary of terms used throughout this documentation.

Document Conventions

Throughout this documentation, the following conventions apply:

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

Other Resources

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

Related Documentation

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

User Documentation

For information about using Natural Construct, see:

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

Installation Documentation

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

Other Documentation

This section lists documents published by WH&O International:

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

Related Courses

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

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

INTRODUCTION TO NATURAL CONSTRUCT

This chapter introduces you to Natural Construct. It describes how to invoke 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.

The following topics are covered:

- **What is Natural Construct?**, page 26
- **Invoking Natural Construct**, page 28
- **Using PF-Keys**, page 32
- **Accessing Online Help**, page 33
- **Automatic Upper Case Translation**, page 36
- **Natural Construct Messages**, page 37
- **Storing Saved Modules**, page 37
- **Using Direct Commands**, page 37

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

Advantages	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.
Productivity	The 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 contains 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 • 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 documentors or developers to create and maintain help text at the map and/or input field level. For more information about this subsystem, refer to *Natural Construct Help Text*.

Invoking Natural Construct

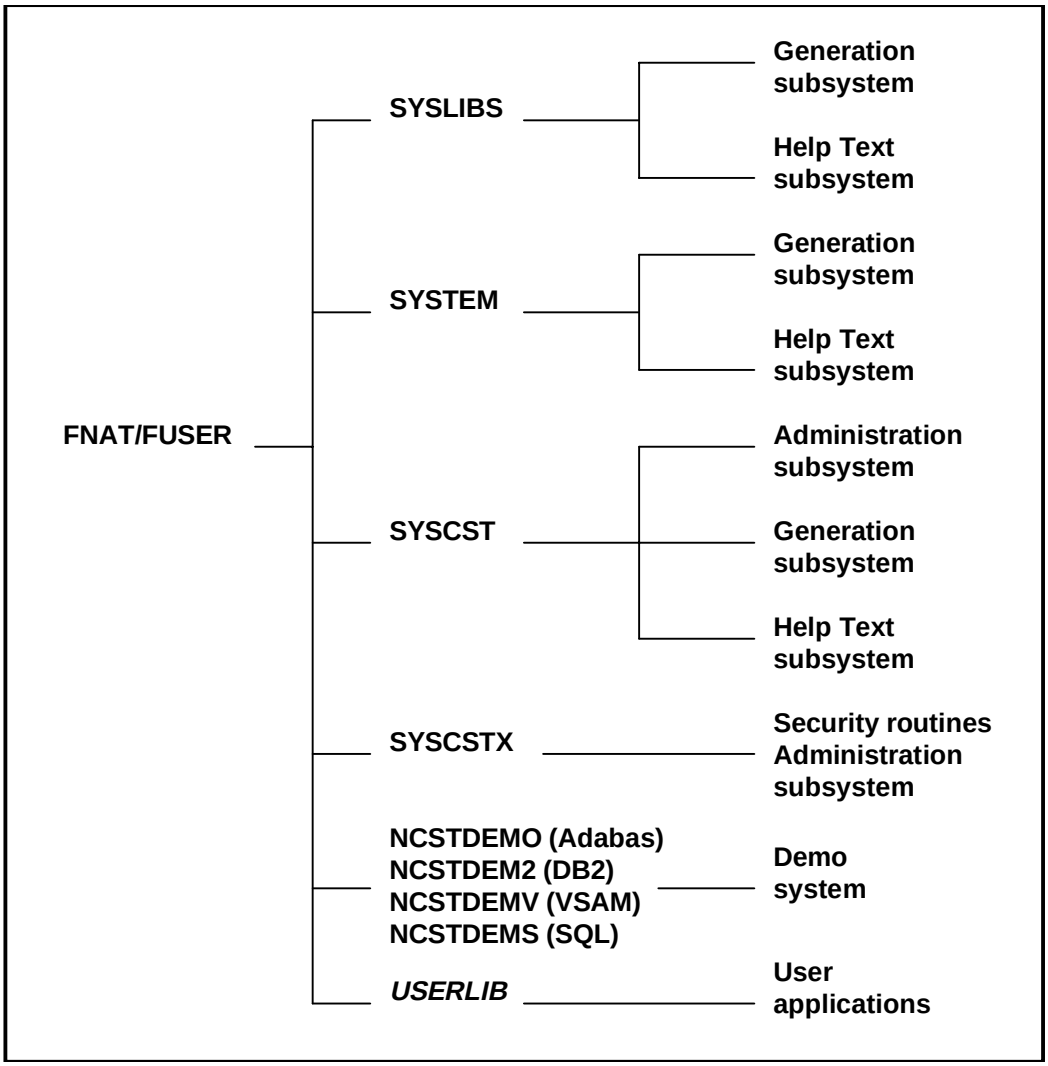
You can invoke the Administration subsystem in standard or translation mode. Translation mode allows you to create multilingual specification panels for developers, as well as dynamically maintain the panel components.

The following sections describe how to invoke each Natural Construct subsystem, how to invoke the Administration subsystem in standard and translation mode, and how to invoke 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 Generation main menu. This method ensures proper cleanup of the environment.

Natural Construct Libraries

Copies of Natural Construct are stored in the following libraries:



Natural Construct Libraries

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

SYSLIBS Library

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

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

SYSTEM (FNAT) Library

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

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

SYSCST Library

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

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

SYSCSTX Library

The SYSCSTX library contains sample security routines provided with Natural Construct. It is used by administrators.

The security routines can be used as is or modified as desired. To make the routines active, they must be moved to the SYSCST library.

NCSTDEMO, NCSTDEM2, NCSTDEMV, and NCSTDEMS 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.

Executing Generation Facilities from a Steplib with Natural Security Installed

With Natural Security installed, you can invoke 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 you invoke Natural Construct from a steplib, the highest level steplib should contain a replacement for the NCSTG program, such as:

```
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, see USR1025P in the SYSEXT library for an example of how to set up your steplibs.

Using PF-Keys

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

PF-keys 13 to 24 are equivalent to PF-keys 1 to 12, respectively. However, only PF1 to PF12 are displayed.

Note: You can change the function and/or description associated with each key (for more information, see **Administration Main Menu**, page 40). Within this documentation, we refer to the standard default values.

By default, 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 (*), pressing PF1 displays a window from which you can select a valid value for the field. For information, see **Field-Level Help**, page 35. When the cursor is not in a field followed by an asterisk (*), pressing PF1 displays a table of contents from which you can select a topic. For information, see **Panel-Level Help**, page 33. **Note:** An asterisk is the default help indicator for Natural Construct. The help indicator for your organization may be different.
PF2	retrn	Displays the previous panel. Pressing PF2 is equivalent to entering a period (.) in the Code field on a menu.
PF3	quit	Terminates the Natural Construct session. In most cases, a confirmation window is displayed when you press PF3. Press PF3 again to complete the termination process and return to Natural.
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.

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

Accessing 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). Natural Construct online help is described in the following sections.

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 in an input field when you request help, Natural Construct displays help information for that field. For more information, see **Field-Level Help**, page 35.

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
```

Panel-Level Help for the Administration Main Menu — Page 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, either:

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

2 Press PF1 (help).

or

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

2 Press Enter (mainframe).

Active

➢ To display active field-level help, either:

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

2 Press PF1 (help).

or

1 Type “?” in the first-character position of a field followed by an *.

2 Press Enter (mainframe).

The following example shows the 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
```

Active Field-Level Help Window

➢ 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.

Automatic Upper Case Translation

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 you exit, the case setting is restored to the same value as when Natural Construct was invoked.

Mainframe Note:
You must specify your teleprocessing (TP) monitor's command for lower case. In ComPlete, for example, issue the LOW command.

Natural Construct Messages

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

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

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

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

For all REINPUT and INPUT message numbers, you can also use the SYSERR utility to add other languages. Generation and CDUTRANS messages are stored in the CSTAPPL library. For information about defining references, see **Defining SYSERR References**, page 405.

Storing 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.

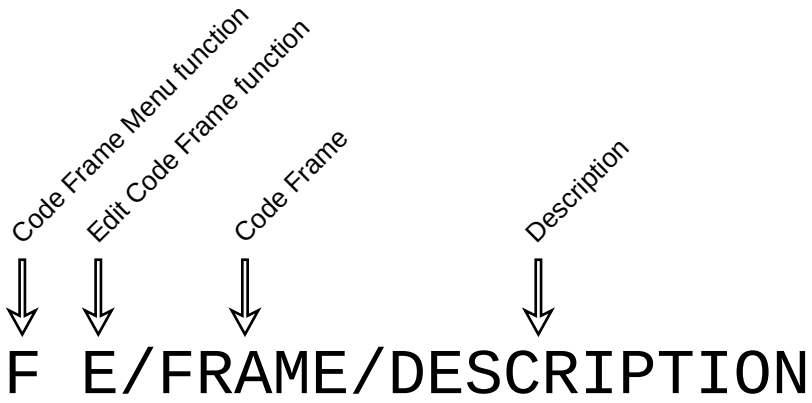
Using Direct Commands

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:



Sample Direct Command

This direct command invokes 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.

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

When you enter 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 invoke the Administration subsystem (MENU) and edit the code frame, DESCRIPTION:

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

USING THE ADMINISTRATION SUBSYSTEM

This chapter describes how to use the Administration subsystem of Natural Construct. It includes information on multilingual support and using the Administration main menu in translation mode, as well as the user exit subprograms you can use to implement security.

The following topics are covered:

- **Administration Main Menu**, page 40
- **Multilingual Support for Natural Construct**, page 67
- **Administration Main Menu in Translation Mode**, page 69
- **User Exit Subprograms to Implement Security**, page 74

For information about invoking the Administration subsystem, see **Invoking Natural Construct**, page 28.

Administration Main Menu

When you invoke the Administration subsystem, the Administration main menu is displayed:

```
CSDMAIN              N a t u r a l   C o n s t r u c t              CSDMNM0
Oct 04                    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
```

Administration Main Menu

➢ To perform a function listed on this menu:

1 Type the corresponding one-character function code in the Function field.

2 Press Enter.

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.
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.
S	Maintain Subprograms	Displays the Maintain Subprograms panel, where you can maintain the modify specification subprograms used by the generation models.

Code	Function	Description (continued)
	Note:	To ensure backward compatibility, this option supports models written prior to V3.4.1. that generate subprogram records. Models generated by the CST-Modify model define windows and PF-keys without using the subprogram records.
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.).
C	Compare Menu	Displays the Compare menu. Using the functions available through this menu, you can compare code frames used by the models.
D	Drivers Menu	Displays the Drivers menu. Using the driver programs available through this menu, you can invoke many of the utility subprograms supplied with Natural Construct. (The source code for these subprograms is not supplied.)

These functions are described in the following sections. For a description of the Help and Return functions, see **Help and Return Codes on Menus**, page 33.

Maintain Models Function

When you invoke the Maintain Models function, the Maintain Models panel is displayed:

```
CSDFM                    N a t u r a l   C o n s t r u c t              CSDFM0
Aug 07                          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
```

Maintain Models Panel for the Browse Model

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

The fields on the Maintain Models panel are:

Field	Description
Model	Name of the model you are 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**, page 404.
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**, page 99.

Status window	Code that indicates whether the Status window is displayed when a module is generated.
	
	If the code is Y, 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”.
	
	If this field is blank, it defaults to N.

Field	Description (continued)
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. 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. You can select a code frame and invoke the Code Frame editor from this panel. In the editor, you can also define nested code frames. For information, see **PF4 (frame)**, page 46.
Note:	Code frames used to generate maps and data areas can only have subprogram and comment lines.

Field	Description (continued)
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.

Field	Description (continued)
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.

PF4 (frame)

Press PF4 (frame) on the Maintain Models panel to select a code frame for editing. 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 07                          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
```

Maintain Models Panel for the Browse Model

➢ To select a code frame for editing:

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

2 Press PF4.
The specified code frame is displayed in the Code Frame editor.

Notice that the code frame names listed in the Code frame(s) field end with a question mark (?). 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.

When code frames are referenced in code (nested code frames), their names also end with the question mark character. The question mark indicates a hierarchy in which the code frame with the lowest number at the end of its name is used.

The code frame naming convention is as follows:

- 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.

Example of a code frame containing a nested code frame

The following example shows the CSLBA9 code frame, which is used by the Browse-Select model. It 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
    /*                                                                    "
    /* repostion 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
```

Example of a Code Frame Containing a Nested Code Frame

For more information about modifying the supplied code frames, see **Step 5: Create the Code Frame and Define the Model**, page 156.

Code Frame Menu Function

When you invoke the Code Frame Menu function, the Code Frame menu is displayed:

```
CSMMAIN                N a t u r a l   C o n s t r u c t              CSMMNM0
Aug 07                         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
```

Code Frame Menu

➢ To perform a function listed on this menu:

1 Type the corresponding one-character function code in the Function field.

2 Press Enter.

The functions available through this menu are described in the following sections.

Edit Code Frame Function

Use the Code Frame editor to create or modify a code frame. If you do not specify a code frame name in the Code Frame field on the Code Frame menu, the Code Frame editor is empty when it is displayed (see example above). If you enter the name of an existing code frame, Natural Construct reads it into the editor.

```
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
```

Code Frame Editor

To return to the Code Frame menu, enter "." (period) at the > prompt.

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**, page 84.

For more information about modifying the supplied code frames, see **Step 5: Create the Code Frame and Define the Model**, page 156.

Save Code Frame Function

This function saves the code frame currently in the edit buffer to the Natural Construct Code Frame file. If the specified code frame name already exists, the message "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 Function

The following example shows the Select Frames window:

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

Frame    Description                                User     Date      Time
-------- ----------------------------------------- -------- --------- ------
C--BAN9  Standard banner                            SAG      Sep 30,01 09:55
CBAA9    Batch define data area                     SAG      Sep 30,01 09:55
CBAB9    Batch initial setup                        SAG      Sep 30,01 09:55
CBAC9    Batch main body                            SAG      Sep 30,01 09:55
CBOA9    Object Browse Subp define data area        SAG      Sep 30,01 09:55
CBOB9    Object Browse Subp main body               SAG      Sep 30,01 09:55
CBRA9    Object Browse Static main body             SAG      Sep 30,01 09:55
CCNA9    Callnat main body                          SAG      Sep 30,01 09:55
CDRA9    Driver main body                           SAG      Sep 30,01 09:55
CETA9    Extendable Input main body                 SAG      Sep 30,01 09:55
CFMA9    Maint define data area                     SAG      Sep 30,01 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
```

Select Frames Window

The fields in this window are:

Field	Description
Frame	Code frame names in alphabetical order.
Description	Brief description of the corresponding code frame.
User	User ID for the user who last saved the corresponding code frame.
Date	Date the corresponding code frame was last saved.
Time	Time the corresponding code frame was last saved.
Frame	To select a code frame, enter the code frame name in this field. (If you enter the name of a code frame that is not currently displayed, the list is repositioned.)
Scan for	If you marked the Detail field, you can also specify a value to scan for. Detail lines are displayed for code frames containing the scanned value only.

Purge Code Frame Function

This function permanently removes a code frame from the Code Frame file.

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

Clear Edit Buffer Function

This function clears the current values from the Code Frame editor.

Print Saved Code Frame Function

This function prints a hardcopy of a code frame that has been saved.

Mainframe Note:
To use this function, you must have Com-Plete, CMS, TSO, or CICS with Natural/AF or Com-Pose. For more information, see **Frame Hardcopy Utility**, page 396.

Maintain Subprograms Function

When you invoke the Maintain Subprograms function, the Maintain Subprograms panel is displayed:

```
CSDFSP                N a t u r a l   C o n s t r u c t              CSDFSP0
Aug 07                      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
```

Maintain Subprograms Panel

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

Maintain Control Record Function

When you invoke the Maintain Control Record function, the Maintain Control Record panel is displayed:

```
CSCTRL                N a t u r a l   C o n s t r u c t              CSCTRL0
Aug 08                     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
```

Maintain Control Record Panel

Use this panel 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.

The fields on the Maintain Control Record panel are:

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. The PF-key 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 (previous panel)
Move right	Scrolls to panel on the right of current panel (next panel)
Help	Invokes help for current panel
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.

Field	Description (continued)
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 follows: ---------- 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).
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.

Field	Description (continued)
Special Hardware	Options available for terminals with special hardware:
Blinking	Support for blinking option.
Italic	Support for italic option.
Underline	Support for underline option.
Reverse video	Support for reverse video option.
Note:	Because of hardware restrictions, you may not be able to use all the options listed. For more information, see the DY Session Parameter section in the Natural Parameter Reference documentation.
Default return	Character used to return to normal (default) display; the default is >. A character must be specified in this field.

Compare Menu Function

When you invoke the Compare Menu function, the Compare menu is displayed:

```
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
```

Compare Menu

Use the functions available on this menu to compare the definitions for two models or two series of models (using the Compare Models function), or compare two code frames or two series of code frames (using the Compare Frames function).

➢ To select a function from this menu:

1 Type the corresponding one-character function code in the Function field.

2 Press Enter.

The functions available through this menu are described in the following sections. For a description of the Help and Return functions, see **Help and Return Codes on Menus**, page 33.

Compare Models Function

The following example compares the components of two models or series of models:

```
CSDCMP                N a t u r a l   C o n s t r u c t              CSDCMP10
Aug 08                          Compare Models                         1 of 1



             Old                              New
Model ...... ________________________________ ________________________________
Database ... 18_                              18_
File ....... 121                              121
Version ....                                  3.4.1



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

Compare Models Panel

Use this panel to compare code frames, model subprograms, etc., for two models or a series of models. The models can reside in different system files.

Comparing the Components of Two Models in Different Files

➢ To compare the components for two models:

1. Type the name of one model in the first Model field.
2. Type the name of the other model in the second Model field.
3. Press Enter.

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

The following example compares two models.

```
CSDCMP                N a t u r a l   C o n s t r u c t            CSDCMP10
Apr 22                         Compare Models                       1 of 1
             Old                                New
Model ...... BROWSE___________________________  BROWSE___________________________
Database ... 18___                              18___
File ....... 121__                              147__
Version .... 3.4.1                              4.2.1
Command ............ ________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      help  retrn quit                                                  main
```

Compare Models Panel

The fields on the Compare Models panel are:

Field	Description
Model	Names of the models you want to compare.
Database	Database identification (DBID) numbers for the Natural Construct system file for the specified models (18 in the example above).
File	Natural Construct file numbers for the specified models (121 in the example).
Version	Natural Construct version numbers for the specified models (3.4.1 in the example).

Comparing the Components for Two Models in the Same File

➢ To compare the components for two models in the same file:

1. Type the name of one model in the first Model field.
2. Type the name of the other model in the second Model field.
3. Type the DBID number for the Natural Construct system file in the first Database field.
4. Type the Natural Construct file number in the first File field.
5. Press Enter.
   The Show Model Differences window is displayed.

Example of comparing two models in the same file

The following example compares Browse and Browse-Select models in the same file:

```
CSDCMPD                  Natural Construct
Aug 08                 Show Model Differences

Old ........ 3.4.1                            New ........ 4.2.1
                    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 31,2001                   Jul 31,2001
Time .............  10:09.510                     10:09.510
User .............  SAG                           SAG
```

Example of Comparing Two Models in the Same File

This window displays the following information:

- Natural Construct version numbers
- Names of the models that are being compared
- Names of the model subprograms used by the models
- Names of the code frames used by the models
- Date the models were created or last saved
- Time the models were created or last saved
- User IDs of the users who created or last saved the models

Compare a Range of Models with the Same Name in Two Files

➢ To compare a range of models with the same name in two different files:

1 Type the starting value for the range in the first Model field.
The starting value can be the name of a model or the first few characters in the name of a model. 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, see **Wildcard Selection**, page 91, *Natural Construct Generation*.

2 Type the DBID number for the first range of models in the first Database field.

3 Type the DBID number for the second range in the second Database field.

4 Type the Natural Construct file number for the first range of models in the first File field.

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

6 Press Enter.
The Show Model Differences window is displayed.

Example of comparing models with the same name in two files

The following example compares the Browse model in file 116 (3.3.2 version) to the Browse model in file 120 (4.2.1 version):

```
CSDCMPD                  Natural Construct
Aug 08                 Show Model Differences

Old ........ 3.3.2                            New ........ 4.2.1
                    BROWSE                       BROWSE
------------------- ---------------------------- ------------------------
Description ......  BROWSE Program (BR)          *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
```

Example of Comparing Models with the Same Name in Two Files

Compare Frames Function

The following example shows the Compare Frames panel:

```
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 ... 18_                              18_
File ....... 121                              121
Version ....                                  3.4.1



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

Compare Frames Panel

Use this panel to compare code frames. The models containing the code frames can reside in different system files. You can also use this panel to compare all code frames and nested code frames for a model. Results are presented code frame by code frame.

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

➢ To compare two code frames:

1. Type a code frame name in the first Frame field.
2. Type a code frame name in the second Frame field.
3. Type the DBID for the first code frame in the first Database field.
4. Type the DBID for the second code frame in the second Database field.
5. Type the Natural Construct file number for the first code frame in the first File field.
6. Type the file number for the second code frame in the second File field.
7. Press Enter.
   The Summary Report window is displayed.

Example of comparing two code frames

The following example compares the CUBADA9 code frame in file 116 (4.4.1 version) to the CBAA9 code frame in file 121 (4.5.1 version):

```
CSDCMPFD                    Natural Construct                        CSDCMP
                              Summary Report


Old version .... 4.4.1               New version .... 4.5.1
      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
```

Example of Comparing Two Code Frames

The code frames compared can be different code frames in the same file, the same code frames in different files, or different code frames in different files. For information about comparing code frames in batch mode, see **Additional Utilities**, page 402.

The Summary Report window displays the following information:

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

For a line-by-line comparison, press Enter:

```
Oct 07                        Natural Construct                      04:15 PM
                                Compare Frames                       PAGE: 1
  Old version .... 4.4.1                          New version .... 4.5.1

                                  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
```

Code Frame Compare Utility Panel

The lines in the code frames that match are marked with an equal sign (=). Lines that are in the first code frame, but not the second, are marked with a minus sign (-). Lines that are in the second code frame, but not 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 about printing a hardcopy of a code frame, see **Print Saved Code Frame Function**, page 52.

Comparing a Range of Frames with the Same Name in Two Files

➢ To compare a range of code frames with the same name in two different files:

1 Type the starting value for the range in the first Frame field.
The starting value can be the name of a code frame or the first few characters in the name of a code frame. 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, see **Wildcard Selection**, page 91, *Natural Construct Generation*.

2 Type the DBID number for the first range of code frames in the first Database field.

3 Type the DBID number for the second range of code frames in the second Database field.

4 Type the Natural Construct file number for the first range of code frames in the first File field.

5 Type the Natural Construct file number for the second range of code frames in the second File field.

6 Press Enter.
The Frame Compare Facility panel is displayed.

Example of comparing a range of frames with the same name in two files

The following example compares the code frames in file 116 to those in file 121:

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

  Frame             Old                  New
  ----------------- -------------------- --------------------
_ CGMA9             DATE: 01-10-03 09:46 DATE: 01-09-27 15:03
_ CGOA9             DATE: 01-09-30 09:55 DATE: 01-09-27 15:03
_ CGPA9             DATE: 01-09-30 09:55 DATE: 01-09-27 15:03
_ CGRA9             DATE: 01-09-30 09:55 DATE: 01-09-27 15:03
_ CGSA9             DATE: 01-09-30 09:55 DATE: 01-09-27 15:03
_ CHDA9             DATE: 01-09-30 09:55 DATE: 01-09-27 15:03
  CMDA9             Does not Exist       DATE: 01-09-27 15:03
_ CMNA9             DATE: 01-09-30 09:55 DATE: 01-09-27 15:03
_ CN-BAN9           DATE: 01-09-30 09:55 DATE: 01-09-27 15:03
_ CNDA9             DATE: 01-09-30 09:55 DATE: 01-09-27 15:03
_ CNOA9             DATE: 01-09-30 09:55 DATE: 01-09-27 15:03
_ COBA9             DATE: 01-10-07 11:12 DATE: 01-09-27 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
```

Frame Compare Facility Panel

This panel is similar to the Model Compare Facility panel and functions in the same manner. For a description of this panel, see **Compare a Range of Models with the Same Name in Two Files**, page 60.

To display the comparison data for any code frame, enter C in the Selection Column field for that code frame. The Show Model Differences window and then the Code Frame Compare Utility panel are displayed. For a description of the Show Model Differences window and the Code Frame Compare Utility panel, see **Compare Two Code Frames**, page 61.

Comparing All the Frames Used by Two Models

➢ To compare all the code frames used by two models:

1. Type the name of the first model in the first Model field.
2. Type the name of the second model in the second Model field.
3. Type the DBID number for the first model in the first Database field.
4. Type the DBID number for the second model in the second Database field.
5. Type the file number for the first model in the first File field.
6. Type the file number for the second model in the second File field.
7. Press Enter.
   The Show Model Differences window is displayed.

Example of comparing all the frames used by two models

The following example compares the code frames in the Browse model with the code frames in the Browse-Select model:

```
CSDCMPFD                    Natural Construct                     CSDCMPF0
Aug 09                        Summary Report                        1 of 1

Old version .... 4.4.1               New version .... 4.4.1
     Model .... BROWSE                    Model .... BROWSE-SELECT

 Old    New    Matched    Deleted    Inserted             Comments
------ ------ ---------- ---------- ---------- ----------------------------
  576     772     396        180        376    Frames do not match
Press ENTR to continue or any PF-key to retrn
```

Summary Report Window

To display a line-by-line comparison, press Enter. For more information, see **Compare Two Code Frames**, page 61.

Drivers Menu Function

When you invoke the Drivers Menu function, the Drivers menu is displayed:

```
CTEMENU            N a t u r a l   C o n s t r u c t   4.4.1          CTEMNM0
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
```

Drivers Menu

Use this menu to access various utility subprograms supplied with Natural Construct. The drivers used to invoke the utilities are grouped according to what kind of subprogram they access.

For a description of each menu function and the subprogram it invokes, refer to the applicable Drivers Menu Option section in **External Objects**, page 279.

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 add translations for the supplied text or change the supplied text to suit your organization's standards.

Note: For information about installing Natural Construct in static or dynamic language mode, see the installation documentation.

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

- CSTLDA, for all panel and window text
- CSTMSG, for all message text

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

➢ To add text for another language or modify the supplied text, use one of the following methods:

- 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.
- 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 the SYSERR utility, see the Natural Utilities documentation. For information about Translation mode, see **PF12 (lang)**, page 68.

Note: To define the text for another language, you must first change the *Language value in the Language Preference window.

PF12 (lang)

To change languages, press PF12 (lang) on the Administration main menu. The Language Preference window is displayed:

```
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
```

Language Preference Window

➢ To select a language currently displayed in this window:

1 Move the cursor to the line containing the desired language.

2 Press Enter.
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 what 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 more information about setting up the language hierarchy, see the Natural Construct installation documentation.

Administration Main Menu in Translation Mode

To help you maintain the text for Natural Construct panels, windows, and messages, the Administration subsystem is also available in translation mode. In translation mode, you can change the text for all Natural Construct screens. For example, you can create multilingual panels by translating the text to another language or you can modify the text to reflect your organization's standards.

For information about invoking the Administration main menu in translation mode, see **Invoking Natural Construct**, page 28.

The following example shows the Administration main menu in translation mode:

```
CSDMAIN              N a t u r a l   C o n s t r u c t              CSDMNM0
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
```

Administration Main Menu in Translation Mode

Notice that functions are 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.

The following sections describe how to use translation mode to modify the text on panels, windows, and messages.

Using 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), contained in the CSTLDA library in SYSERR
- Message text, contained 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.

Within SYSERR, you can provide text in different languages for each SYSERR number. For even greater reusability, you can use a variable (such as :1:) with the text (for more information, see REINPUT in the Natural Statements documentation). 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 invoked the panel.

Translation mode allows you to change the text in SYSERR 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. To change screen text in the Generation or Help Text subsystems, invoke the Generation main menu (G function) or Help Text main menu (H function) functions from the Administration main menu in translation mode.

Depending on the current value of *Language, you can either edit the existing text or add the translations for another language. The following sections describe how to perform these tasks.

Editing 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.

The following example shows the Maintain Models panel in translation mode:

```
CSDFM                    N a t u r a l   C o n s t r u c t              CSDFM0
Aug 08                          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
```

Maintain Models Panel in Translation Mode

➢ To edit text on the Maintain Models panel:

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

2 Press Enter.
The Translate Short Message window is displayed:.

```
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
```

Translate Short Message Window

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, so be careful when changing the text for SYSERR numbers that are used on other panels.

Note: The "/+26" value in this window indicates there are up to 26 characters available for each text segment that is to be translated.

To edit text in help or select in windows, invoke the window and edit as described.

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 the Administration main menu in translation mode. For more information, see **Invoking Natural Construct**, page 28.
2. Press PF12 (lang).
   The Language Preference window is displayed. For a description of this window, see **PF12 (lang)**, page 68.
3. Move the cursor to the line containing the word, "French", and press Enter.
   The Administration main menu is displayed.
4. Display the panel you want to translate (in this example, the Maintain Models panel).
5. Put your cursor over the prompt text you want to change (not a blank input line).
6. Press Enter.
   The Translate Short Message window is displayed:

```
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
```

Translate Short Message Window on the Maintain Models Panel

7. Type the French equivalent under the English text.

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

8 Press Enter.
You are returned to the Maintain Models panel and the translated text is displayed.

9 Repeat steps 5 through 8 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 (steps 5 through 8).

➢ To translate text on the Browse specification panels. For example, perform steps 1 through 3 on the previous page, and then:

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

2 Press Enter.
The Generation main menu is displayed in translation mode.

3 Type “M” in the Function field.

4 Type “Browse” in the Model field.

5 Press Enter.
The first specification panel for the Browse model is displayed.

6 Perform the translation procedure (repeat steps 5 through 8).

User Exit Subprograms to Implement Security

Natural Construct supplies several user exit subprograms for the Administration and Help Text subsystems. Use these subprograms to implement security or restrict access to various Natural Construct modules (models, code frames, model subprograms, help text members).

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

If these subprograms exist in a library within the specified subsystem, Natural Construct invokes the applicable subprogram to enforce security when a user selects a module and action. The subprogram grants or denies access based on the specified user ID privileges. The supplied user exit subprograms are:

Function	Subprogram	Library
Model alias name support	CSXAUEXT	SYSLIBS
Generation main menu (before the post-generation subprogram is invoked)	CSXCNAME	SYSLIBS
User-defined default values for generation	CSXDEFLT	SYSLIBS
Administration main menu	CSXDUEXT	SYSCST
Code Frame menu	CSXFUEXT	SYSCST
Help Text main menu	CSXHUEXT	SYSLIBS
Maintain Model function	CSXMUEXT	SYSCST
Generation main menu (after all substitution values are generated into the program)	CSXPSCHG	SYSLIBS
Maintain Subprograms function	CSXSUEXT	SYSCST

The Natural Construct installation tape contains samples of these user exit subprograms. The sample subprograms are initially loaded into the SYSCSTX library, which is created during installation. To use a user exit subprogram, modify it for your installation and then copy it to the library indicated above.

Defining Default Specifications

You can define default specifications for individual models on an application (library) level — and use these defaults to further automate the generation process.

Note: This functionality does not apply to the statement models or the Object-Maint-PDA and Object-Maint-PDA-R models.

➢ To define default specifications for a model:

1. Type “M” in the Function field on the Generation main menu.
2. Type “DEFAULT” in the Module field.
3. Type the name of the model (for which you want to define defaults) in the Model field.
4. Press Enter to display the first specification panel and define your default specifications.
   Continue defining defaults on the remaining specification panels (if there is more than one) until you are returned to the Generation main menu.
5. Type “S” in the Function field.
6. Press Enter to save your defaults.

➢ To modify previously-defined default specifications for a model:

1. Type “R” in the Function field on the Generation main menu.
2. Type “DEFAULT” in the Module field.
3. Type the name of the model (for which you are modifying defaults) in the Model field.
4. Press Enter.
   Natural Construct reads the specifications for the model.
5. Type “M” in the Function field.
6. Press Enter to display the first specification panel and modify the default specifications.
   Continue modifying defaults on the remaining specification panels (if there is more than one) until you are returned to the Generation main menu.
7. Type “S” in the Function field.
8. Press Enter to save your modifications.

Note: While defining the default parameters, specification edits are not invoked.

Natural Construct reads the model defaults into the editor when the clear subprogram is invoked for a model. This occurs whenever you invoke the Clear Specification and Editor function while a model name is specified, or invoke the Modify Specifications function for a new model name.

The names of the modules that store the default specifications are derived from the names of the model clear subprograms. For example, if the clear subprogram for the Maint model is CUFMC, the default specifications module is C@FMC.

To maintain unique default specification modules for each model, the supplied Natural Construct models have their own unique clear subprograms.

Note: If you have custom models that share common clear subprograms, you should make copies of the models, ensure that each model has a unique name, and provide a corresponding clear subprogram with a unique name for each one.

Defaults specified at the application level override defaults specified in the PROVIDE-DEFAULT-VALUES user exit for the clear subprogram. That is, if a default specification module exists in the current library for a model, Natural Construct uses these defaults rather than the defaults specified in the user exit. If no default module exists in the current library, Natural Construct uses the defaults specified in the user exit.

On the Generation main menu, the Read Specifications, Modify Specifications, Save Generated Source, List Generated Modules, and Clear Edit Buffer functions support specification defaults. The Invoke User Exit Editor, Generate Source, Test Generated Source, Edit Generated Source, and Stow Generated Source functions do not support default specifications.

You can modify the DEFAULT keyword by changing the value of the DEFAULT-SPECIFICATION-KEYWORD parameter in the CSXDEFLT subprogram. After modifying this value, re-catalog CSXDEFLT and use the SYSMAIN utility to copy the object code into the SYSLIBS library.

Using the CSXDEFLT Subprogram

This method provides default values for model parameters that can be overridden on the model specification panels, as well as internal model parameters the developer cannot change.

The supplied models retrieve many of the default parameter values by issuing a CALLNAT to the CSUDEFLT 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 use depends on the format of the field you are providing defaults for:

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 second parameter defines the variable to which the default value is assigned. This 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'
```

During installation, CSXDEFLT is installed in the SYSCSTX library. For testing purposes, copy CSXDEFLT into the SYSCST library. For production purposes, copy CSXDEFLT into the SYSLIBS library.

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

Using Library-Specific Defaulting

This method provides default values for model parameters that appear as input fields on the specification panels for a model.

You can provide default parameter values for a model by assigning the default settings within a specified library. This method only applies to parameters supplied through input fields on the specification panels for the model. For information about providing library-specific defaulting, see **Defining Default Specifications**, page 75.

Using CSXCNAME Overrides

This method provides default values for model parameters that can be overridden by changing one source code string to another.

You can globally override defaults for all modules generated by a model by supplying the CSXCNAME user exit subprogram. If present, this subprogram is invoked as part of the post-generation process to apply any CHANGE commands to the generated source code.

To use this method, ensure that the stacked CHANGE string uniquely identifies a string (since the change will apply even if the CHANGE string is embedded in a longer string). To set the first parameter to null, stack the third parameter.

Example of using the CSXCNAME subprogram override

```
STACK TOP DATA FORMATTED 'CCSETKEY' 'MYSETKEY'
STACK TOP DATA FORMATTED '(EM=LLL MM DD)' ' ' 'X' /* Set to null
END
```

Assigning Corporate 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, see the following code example. The example illustrates how a work file number and column delimiter values are defaulted.

Example of assigning corporate defaults

```
** 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.
```

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

Note: 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.

Using 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 invoked. This reduces the number of specifications developers must provide when using the model.

Defining 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.

Defining 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. (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:
   - HOLDFIELD
   - HOLD-FIELD
   - HOLD_FIELD
   - TIMESTAMP
   - TIME-STAMP
   - TIME_STAMP
   - LOGCOUNTER
   - LOG-COUNTER
   - LOG_COUNTER
3. If the field meets the format criteria for a hold field, Natural Construct uses this field as the logical hold field.

Defining 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".

USING THE CODE FRAME EDITOR

This chapter describes the purpose and functions of 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.

This chapter describes how to invoke the Code Frame editor and the command execution order. It also describes applicable line and edit commands and how to recover edits if your session is interrupted.

The following topics are covered:

- **Invoking the Code Frame Editor**, page 82
- **Using the Code Frame Editor**, page 84

Invoking the Code Frame Editor

➢ To invoke the Code Frame editor from the Administration main menu:

1 Type “F” (Code Frame Menu function) in the Function field.

2 Press Enter.
The Code Frame menu is displayed:

```
CSMMAIN              N a t u r a l   C o n s t r u c t              CSMMNM0
Jun 20                        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
```

Code Frame Menu

3 Type “E” in the Function field on the Code Frame menu.

4 Press Enter.
The Code Frame editor is displayed:

```
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
```

Code Frame Editor

You can also invoke the Code Frame editor from the:

- Command line, by entering “E” and the code frame name separated by a slash (/)
- Maintain Models panel, by selecting a code frame and pressing PF4

To edit an existing code frame, type the name of the code frame in the Code Frame field before invoking the Code Frame editor. If you do not specify the name of a code frame, the editor is empty when displayed.

Note: For information about the functions available through this menu, see **Code Frame Menu Function**, page 49. For information about modifying the supplied code frames, see **Edit Code Frame Function**, page 50.

Using the Code Frame Editor

The following example shows a 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 maximun 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
```

Example of a Code Frame

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 the Code Frame field 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**, page 90)

Field	Description (continued)
+	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.
ABS	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).
X-Y	X and Y delimiters. To confine SCAN and CHANGE commands to code within an X-Y delimited range, mark this field. Text outside the X-Y range is not affected.
S	Total number of lines of code currently in the editor.
L	Number of the first line currently displayed in the editor.

Field	Description (continued)
T	Editor line type. Valid line types are: • N Indicates 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: `Subprogram: _____ Parameter: _____ N` • Type the name of the subprogram in the Subprogram field. If the subprogram is invoked more than once or in multiple code frames, you can specify a constant (which is placed in the #PDA-FRAME-PARM field of the CU—PDA parameter data area) in the Parameter field. 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 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 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.

Field	Description (continued)
	• 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 the C field. Whenever a code frame is updated, Natural Construct compresses blank lines and lines marked with B.
C	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 **Code Frame Conditions**, page 109. • " 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.

Order of Command Execution

The Code Frame editor executes commands in the following order:

1. Modifies text.
2. Executes line commands. Specify in the text area of the editor and precede each command with a period (.E, for example).
3. Executes edit commands. Specify at the > prompt (ADD, for example).

The line and edit commands for the Code Frame editor are described in the following sections.

Line Commands

You can issue line commands in the Code Frame editor that copy, move, and delete lines of code. Line commands must be entered in the edit area (not at the > prompt), begin with a period (.), and start in the first column position of a line. Except for the .L command, you should only use line commands on modified code after you press Enter.

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

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 Generation subsystem of Natural Construct.
.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.

Command	Function (continued)
.IF (*code frame*)	Inserts the specified code frame on the line below the line on which the command is specified. The direction indicator has no effect on this command.
.I(*member*,*startline* ,*number of lines*)	Places a *member* from the current library on to 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, but it applies to one line only.)
.MX	If the direction indicator is a plus sign (+), this command moves the line marked X to the line after the one on which .MX is specified. If the indicator is a minus sign (-), this command moves the line marked X to the line above the one on which .MX is specified.
.MY	If the direction indicator is a plus sign (+), this command moves the line marked with Y to the line after the one on which .MY is specified. If the direction indicator is a minus sign (-), this command moves the line marked Y to the line above the one on which .MY is specified.
.MX-Y	Moves the block of lines delimited by the X and Y markers. If the direction indicator is a plus sign (+), this command moves the block to the line after the one on which .MX-Y is specified. If the direction indicator is a minus sign (-), this command moves the block to the line above the one on which .MX-Y is specified.
.N	Marks the line for the POINT edit command (for more information, see the POINT command in **Positional Edit Commands**, page 92).
.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.
.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

Specify the following edit commands at the command prompt (>):

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. 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 **Maintain Current PF-key Profile Window**, page 92).
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.

Command	Function (continued)

SCAN	Scans for data in the edit area in the following ways:
	`SCAN 'scanvalue`
	Scans for text within the delimiters.
	`SCAN scan value`
	Scans for the entire text after the SCAN keyword, including spaces.
	You must use delimiters for scan values that begin with a non-alphanumeric character.
	If the direction indicator is a plus sign (+), the scan begins at the first line displayed on the panel and continues to the end of the text. If the indicator is a minus sign (-), 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).
	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 **Using the Code Frame Editor**, page 84.

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, use the following edit commands to scroll through the information:

To scroll	Then
Forward or backward *nnnn* lines.	Enter +*nnnn* or -*nnnn* at the > prompt.
Forward or backward half a panel.	Enter +H or -H at the > prompt.
Forward or backward one panel.	Enter +P or -P at the > prompt.
Forward one panel (if text was not changed).	Press Enter.
Forward to end of code frame.	Enter BOTTOM or ++ at the > prompt.
Line on which .N line command is specified to top of panel.	Enter POINT at the > prompt.
Backward to top of panel.	Enter TOP or -- at the > prompt.
To the line marked X or Y.	Enter X or Y at the > prompt.
To line *nnnn*.	Enter *nnnn* at the > prompt.

Maintain Current PF-key Profile Window

The Maintain Current PF-key Profile window allows you to change (for the current session only) the PF- and PA-key settings, the number of updates before an automatic save, and the name of the recovery member.

➢ To display the Maintain Current PF-key Profile window:

1 Enter PROFILE at the > prompt in the Code Frame editor.
The following window is displayed:

```
 CS-PROF                    Natural Construct                    CS-PRFM0
 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
```

Maintain Current PF-Key Profile Window

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 uses those defaults.

The fields in the Maintain Current PF-Key Profile window are:

Field	Description
PF1= etc.	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.

Note: 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.

Edit Recovery

The Natural Construct editors can automatically save work in the edit buffer after a certain number of updates. The number specified in the Auto save numbers field in the Maintain Current PF-Key Profile window determines how often the work is saved. If the Auto save numbers field is blank, Natural Construct does not automatically save work.

In the Maintain Current PF-Key Profile window, you can also specify the name of the recovery member where you want your work saved.

➢ To retrieve lost code:

1. Invoke the Code Frame editor.
2. Read EDITWORK (or whatever name you specified as your recovery member name in the In member field) into the edit buffer.
3. Re-specify the description, as it is not saved in the recovery member.

To recover edits, the value in the Auto save numbers field must not be blank or 0 (zero) and the value in the In member field must be specified. For information, see **Maintain Current PF-key Profile Window**, page 92.

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.

GUI Sample Subprogram

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.

CREATING NEW MODELS

This chapter 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. A section at the end of this chapter provides information about the utility subprograms and helproutines supplied with Natural Construct. These utilities can help you create your new model.

The following topics are covered:

- **Components of a Natural Construct Model**, page 96
- **How Natural Construct Executes a Model**, page 98
- **Building a New Model**, page 99
- **Testing the Model Subprograms**, page 143
- **Implementing Your Model**, page 149
- **Statement Models**, page 149
- **Utility Subprograms and Helproutines**, page 151

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 help generate each component (if applicable):

Component	Model Used To Generate
Code frames	None (create manually or copy and modify existing).
Model PDA	CST-PDA model described in **Parameters for the CST-PDA Model**, page 177.
Translation LDAs for dynamic translation)	None (create manually or copy and modify existing).
Maintenance maps	Map model (described in *Natural Construct Generation*).
Modify subprogram(s)	CST-Modify or CST-Modify-332 model described in **CST-Modify Model**, page 193.
Pre-generation subprogram	CST-Pregen model described in **Parameters for the CST-Pregen Model**, page 211.
Generation subprograms	CST-Frame model described in **Parameters for the CST-Frame Model**, page 220.
Post-generation subprogram	CST-Postgen model described in **Parameters for the CST-Postgen Model**, page 215.
Clear subprogram	CST-Clear model described in **Parameters for the CST-Clear Model**, page 181.
Save subprogram	CST-Save model described in **Parameters for the CST-Save Model**, page 189.
Read subprogram	CST-Read model described in **Parameters for the CST-Read Model**, page 185.
Sample subprogram(s)	CST-Frame model described in **Parameters for the CST-Frame Model**, page 220.
Document subprogram	CST-Document model described in **Parameters for the CST-Document Model**, page 225.

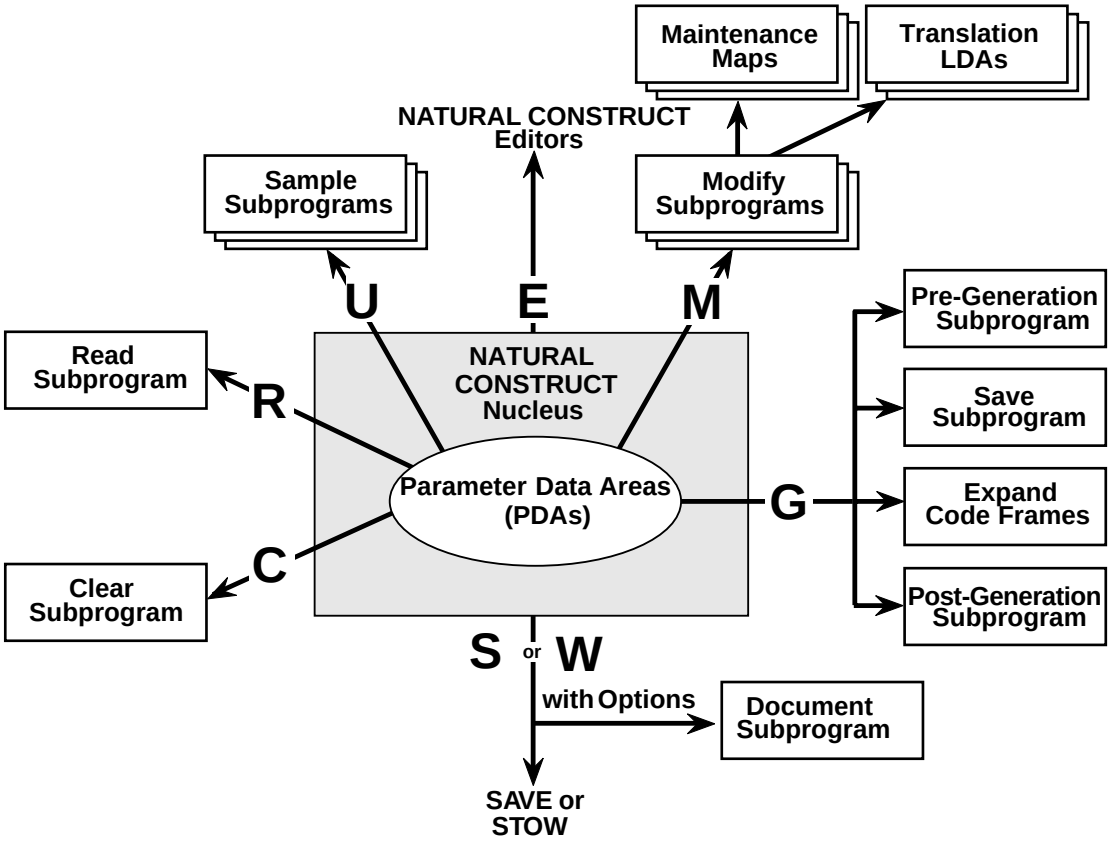
Component	Model Used To Generate (continued)
Stream subprogram	CST-Stream model described in **Parameters for the CST-Stream Model**, page 237.
Validate subprogram	CST-Validate model described in **Parameters for the CST-Validate Model**, page 231.

How Natural Construct 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 correspond to the function codes a user enters on the Generation main menu to invoke the corresponding subprogram(s):



Interaction of the Model Components

Building a New Model

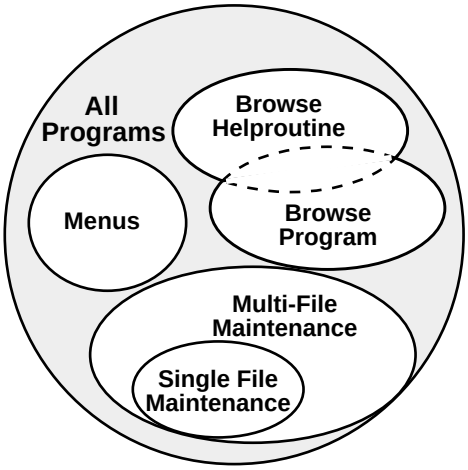
➢ To build a new Natural Construct model:

- ❑ **Step 1: Define the Scope of the Model**, page 99
- ❑ **Step 2: Create the Prototype**, page 101
- ❑ **Step 3: Scrutinize the Prototype**, page 101
- ❑ **Step 4: Isolate the Parameters in the Prototype**, page 101
- ❑ **Step 5: Create Code Frame(s) and Define the Model**, page 103
- ❑ **Step 6: Create the Model PDA**, page 115
- ❑ **Step 7: Create the Translation LDAs and Maintenance Maps**, page 131
- ❑ **Step 8: Create the Model Subprograms**, page 134

These steps are described in detail in the following sections.

Step 1: Define the Scope of the Model

Before you begin, 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:



Scope and Functionality of Your Model

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 programmers/analysts 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 you build a model containing 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 check 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.

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:

Redundant Parameters	Single Parameter

`DEFINE DATA LOCAL`
`01 #A(A1/1:50`
`.`
`.`
`END-DEFINE`
`.`
`.`
`IF #A(#CUR:50) NE ' ' THEN`
`FOR #I = #CUR TO 50`
`etc.`
	`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.`

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.

Compile Time Versus Execution Time

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

Unnecessary Constant	Determine at Execution Time

`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`
	`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`

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 execution time. 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

After creating and saving the prototype, create the code frame(s) and define the model. These procedures are described in the following sections.

Creating 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 in your prototype

3 Determine the parameters 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(c) 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
```

Example of a Code Frame in the Code Frame Editor

For a description of the Code Frame editor, see **Using the Code Frame Editor**, page 84. For information about edit commands, see **Edit Commands**, page 90.

The code frame example above shows different methods of supplying parameters for a code frame. The following sections describe each of these methods.

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 on the previous page 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.

For more information about substitution during the post-generation phase, see **Post-generation Subprogram**, page 137.

Substitution parameters cannot span multiple lines and 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**, page 118.

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 to 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.

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**, page 42) 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 substitution parameters and logicals are always available within the model PDAs.

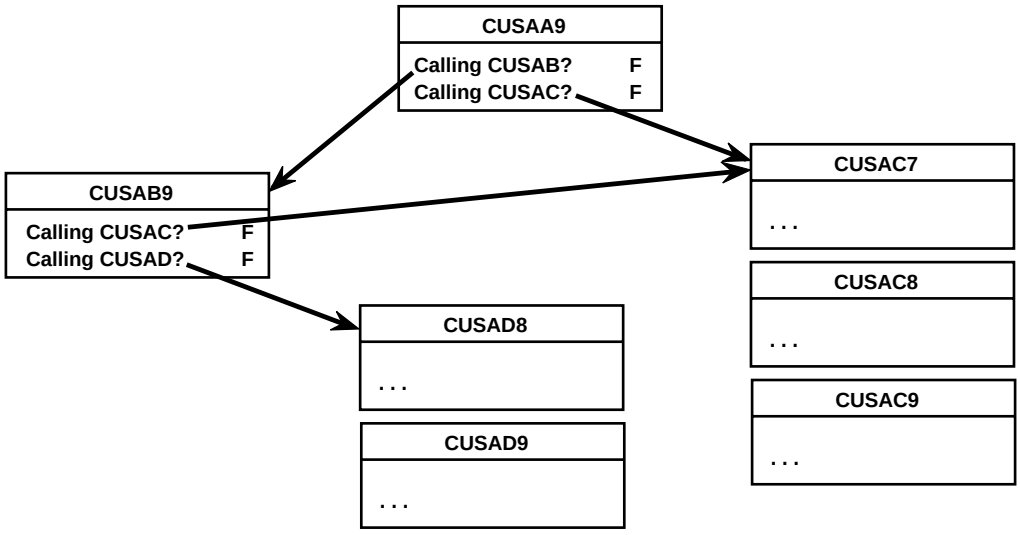
To indicate that another code frame is called on a Code Frame editor line, enter F in the corresponding T (Type) field.

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:



Example of Calling Nested Code Frames

Note: 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.

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:

```
  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
```

Example of Adding User Exit Points in a Code Frame

4 Press Enter.
The Maintain User Exit window is displayed:

```
 CSMUSEX                      Natural Construct
 Jul 28                       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
```

Maintain User Exit Window

Use this window to specify information about the user exit.

The fields in the Maintain User Exit 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 **Code Frame Conditions**, page 109). 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**, page 115. **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 has none 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 to use the defaults.

Field	Description (continued)
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 the sample subprogram code.

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) 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                                                       "
```

Example of a Condition in a Code Frame

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 a double quotation (") 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**, page 115.) 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 within the model PDA.

- When Natural Construct generates a module, it checks the condition code values to determine whether the condition is true. It resets the conditions before invoking the modify subprograms. Condition codes should be selectively set to TRUE by either the pre-generation subprogram or one of the modify 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.

Example of 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 on the previous page are inserted into the generated module when the following Boolean conditions are met:

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

Notes	Boolean Condition (continued)
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.

Defining 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 the Function field.
The Maintain Models panel is displayed:

```
CSDFM                    N a t u r a l   C o n s t r u c t            CSDFM0
Aug 09                         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
```

Maintain Models Panel

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**, page 42.

Naming Conventions for Model Components

Standardizing the names of the various components of a model makes it easier to write and debug models. All supplied model subprograms, maps, and data areas are named CU*xx*, where *xx* uniquely identifies each model. When naming the model components, we recommend you use 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 modify subprogram.

Name	Model Component (continued)
CU*xx*MA*n*	Map associated with the first modify subprogram. To support dynamic translation, use a zero (0) in the last position of the map name. 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.
CU*xx*MAL	Translation local data area (LDA) associated with the first modify 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 modify subprogram.
CU*xx*MB*n*	Map associated with the second modify subprogram.
CU*xx*MBL	Translation LDA associated with the second modify subprogram.
CU*xx*S*yyy*	Sample user exit code subprograms, where *yyy* is a 1- to 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- to 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	Document subprogram.

To modify the supplied Natural Construct models, copy the subprograms and change the prefix from CU to CX (do not modify the supplied subprogram). 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. You create the model PDA for each 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**, page 177).
CU—PDA	Supplied with Natural Construct.
CSASTD	Supplied with Natural Construct.

These PDAs must contain the following fields:

PDA	Required Fields	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-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 (continued)	Required Fields	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-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

The following sections describe the layout of these PDAs.

Note: The CSASTD PDA is used by every model. It passes messages between subprograms and is typically used for error handling.

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 reseting 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 modify subprograms are invoked. Along with the pre-generation subprogram, the modify 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 substitution parameters and logicals 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 substitution parameters and logicals 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.
*
```

CU—PDA contains the fields described in the following sections.

#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. You 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 CS-LPHASE 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: Modify 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 modify 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 modify 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 modify 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 the previous section) 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.
#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, you can 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. You can 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. You can 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 (an asterisk (*), for example).

#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-CV*n*

These fields (C) 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

Attribute	Index Occurrence (continued)
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**, page 320.

#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

CSASTD PDA contains the fields described in the following sections.

Note: The CSASTD PDA is used by every model. It passes messages between subprograms and is typically used for error handling.

MSG

This field (A79) is used with the RETURN-CODE field (see **RETURN-CODE**, page 130) 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**, page 130). 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-INDEX1/2/3

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, if needed create the 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

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                    I    4 /* Explicit length to translate
```

CUBAMAL Translation LDA for the Batch Model

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 position. For example, the CUBAMA modify 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 on the previous page. 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, you must call the CNUMSG subprogram.

Position	Format (continued)
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 a / 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 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).
>	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.

Note: For more information, see **Using SYSERR References**, page 406.

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
```

Maintenance Maps

Normally, each modify 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 chapter of *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 chapter in this documentation. The model generation models are described in the order they are implemented during the generation process.

Modify 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. These subprograms can also set condition codes and assign derived PDA variables.

Modify subprograms are executed in the same order as they appear on the Maintain Models panel. Usually, there is one modify subprogram for every left/right (horizontal) maintenance panel. Data edits should only be applied if the developer presses Enter or PF11 (right). Either the modify 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 modify subprogram (see **Maintain Subprograms Function**, page 52).

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

For an example of a generated modify subprogram, refer to CUMNMA and CUMNMB in the SYSCST library.

For information about the CST-Modify model, see **CST-Modify Model**, page 193.

When are Modify Subprograms Invoked?

The Natural Construct nucleus invokes the modify subprograms in the following situations:

Generation Main Menu

Function: M Module: TEST Panel: 2

Invokes the second maintenance panel, and:

- If the developer presses Enter, invokes the Generation main menu.
- If the developer presses PF11 (right), invokes the third panel (if there is one).
- If the developer presses PF10 (left), invokes the first panel and displays the message: Beginning of specification panels.

Function: M Module: TEST Panel:

Invokes the first maintenance panel, and:

- If the developer presses Enter or PF11 (right), invokes the second panel (if there is one).
- If the developer presses PF10 (left), invokes the first panel and displays the message: Beginning of specification panels.

Function: G Module: TEST Panel:

Invokes all maintenance panels to ensure that all parameters have been edited before generation. The input panels are not displayed unless an error is encountered.

User Exit Editor

> SAMPLE

- Invokes all maintenance panels so you can 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 modify subprograms are executed during the generation phase or after the SAMPLE command is issued from the User Exit editor. It is the first user subprogram invoked. It assigns all true condition values (see the following example), based on user-supplied input parameters or other calculated values. (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 modify subprograms.)

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**, page 211.

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. 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.

You can invoke the generation subprograms by specifying 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
```

Example of a Generation Subprogram in a Code Frame

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 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..
  0010 DEFINE DATA LOCAL
  0020   01 #MAX-LINES(P3) CONST<&MAX-SELECTIONS>
  0030   01 #LINE-NR(P3/1:#MAX-LINES)
  0040   01 #I(P3)
  0050 END-DEFINE
  0060 FOR #I = 1 TO #MAX-LINES
  0070   ASSIGN #LINE-NR(#I) = #I
  0080 END-FOR
  0090 .
  0100 .
  0110
  0120
  0130
  0140
  0150
  0160
  0170
  0180
  0190
  0200
       ....+....1....+....2....+....3....+....4....+....5....+... S 10   L 1
```

Example of Edit Buffer After the Generate Object Phase

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 (see the example on the following page).

Example of a post-generation subprogram

```
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.

Example of the STACK TOP option

```
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.

If you do not specify a clear subprogram, the Clear function on the Generation main menu sets #PDA-USER-AREA to blanks. 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.

Example of 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 component of an array.

Example of 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 this subprogram to:

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

For example, Natural Construct changes:

```
#PDAX-MAP-NAME: MYMAP
```

to:

```
**SAG MAP-NAME: MYMAP
```

These lines are placed at the beginning of the generated module.

Read Subprogram

Generated using the CST-Read model, this subprogram reads the specification parameters from 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.

Note: Natural Construct invokes the clear subprogram before invoking the read subprogram. It is not necessary to save null parameter values.

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. They 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, you can use the CU—PDA.#PDA-FRAME-PARM variable.

All modify subprograms and the pre-generation subprogram are automatically invoked before the sample subprograms are executed. 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 **Parameters Supplied by User Exits**, page 106. For an example of a sample subprogram, refer to CUFMSRIN in the SYSCST library.

Document 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 document 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**, page 403.

The document 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*.

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

Testing 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 you define the model PDA, maintenance maps, and model subprograms, you can test the individual components of the model by issuing the CSUTEST command from the SYSCST library. This supplied utility invokes the Single Module Test Program panel:

```
CSUTEST                 ***** Natural Construct *****                 CSUTESM1
Aug 01                  - 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
```

Single Module Test Program Panel

A typical test invokes the clear subprogram, one or more modify 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.

The fields on the Single Module Test Program panel are:

Field	Description
Code Function	Functions available through this panel and the codes that invoke each function. Enter the codes in the unnamed input field displayed below the Code field. Valid codes are:
R	Resets the parameter data area (PDA) passed to all model subprograms.

Field	Description (continued)
*	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).

Field	Description (continued)
Model	To display the names of the subprograms associated with a model, enter the name of the model in this field. The following information is displayed:

```
CSUTEST                  ***** Natural Construct *****                CSUTESM1
Aug 01                   - 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.
```

Single Module Test Program Panel — After Entering a Model Name

Enter a number beside each subprogram you want to execute and then enter the same number in the Code field.

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

Frame Parameter or Exit Name	Names of up to four generation subprograms and the names of the corresponding code frame parameters or user exit that is passed to each subprogram when it is executed.
Source Lines Total	Total number of lines in the source buffer.

Debugging a Model

After you create 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.

2 Press PF5 (optns).
The Optional Parameters window is displayed:

```
CSGOPTS             Natural Construct             CSGOPTS0
Oct 26             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
```

Optional Parameters Window

The fields in the Optional Parameters window are:

Field	Description
Status window	If this field is marked, the Status window is displayed during generation. Messages in this window indicate which module is executing at each stage of the generation process. The default for this field is determined by the value specified for the Status field on the Maintain Models panel (see **Maintain Models Function**, page 42).
Step	If this field is marked, you can “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	If this field is marked, messages are displayed as text (for example, “starting _” and “ending _”). If this field is not marked, messages are displayed with arrows “---> _” (starting) and “<--- _” (ending).

Field	Description (continued)
Embedded statements	If this field is marked, embedded statements are written 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	If this field is marked, the Condition Codes window displays the values of the condition codes after the pre-generation subprogram executes.
Post-generation modifications	If this field is marked, the Post-Generation Modifications window displays the values of the code frame substitution parameters identified by an ampersand (&) during generation. The window is displayed after the post-generation subprogram stacks the substitution values in the code frame.
Specifications only	If this field is marked, only the current specifications and user exit code are saved. 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	If this field is marked, the saved generated module (program, data area, etc.) is documented in the Predict data dictionary.

3 Type “G” in the Function field on the Generation main menu.
The following example shows the Status window without the Text field:

```
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 |
  | Apr 15             Optional Parameters              1 of 1 |
 +-------------------------------------------------------------------------+
 | CSGENPGF                    Natural Construct                           |
 | Apr 15                        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
```

Status Window

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.
- You can use an edit mask to write data without embedded spaces. 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 (indicated by the / character) SYSERR message 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.

Implementing 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.

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 modify 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, you 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, see **Statement Models**, page 483, *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 as the first line of your generated code.

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 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 **Natural Construct Generation Utility Subprograms (CSU*)**, page 307.

NEW MODEL EXAMPLE

This chapter provides a step-by-step example of creating a new model using the procedure described in **Building a New Model**, page 99. The model, Menu, generates a program that displays several choices to a user and allows the user to select one.

For an example of a generated menu program, refer to NCMAIN in the demo library.

Procedure to Build the Example Model

The example model, Menu, generates a program that displays a menu from which the user can select options.

➢ To build the model example:

- ❑ **Step 1: Define the Scope of the Model**, page 154
- ❑ **Step 2: Create the Prototype**, page 154
- ❑ **Step 3: Scrutinize the Prototype**, page 155
- ❑ **Step 4: Isolate the Parameters in the Prototype**, page 155
- ❑ **Step 5: Create the Code Frame and Define the Model**, page 156
- ❑ **Step 6: Create the Model PDA**, page 159
- ❑ **Step 7: Create Translation LDAs and Maintenance Maps**, page 160
- ❑ **Step 8: Create the Model Subprograms**, page 163
- ❑ **Step 9: Implement the Model**, page 173

The following sections describe these steps in detail.

Step 1: Define the Scope of the Model

A program generated by the Menu model must 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
Apr 02                         - 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
```

Output from the NCMAIN Menu Program

Step 3: Scrutinize the Prototype

After creating the prototype, follow the steps outlined in **Step 3: Scrutinize the Prototype**, page 101, 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. This data is described in the following sections.

Parameters for the Program Header

The parameters supplied for the program header are:

- Name of the program that is 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 the Code Frame and Define the Model

Creating the Code Frame

Once you have identified all data that must be supplied by parameters, create the code frame (CMNA?) for the model.

For an example of the code frame for the Menu model, read the CMNA? code frame (stored in the SYSCST library) into the Code Frame editor.

➢ To create the code frame:

1 Read the prototype into the Code Frame editor and define the substitution parameters. To identify a substitution parameter, locate the character strings that begin with an ampersand (&) character.

The following table shows examples of substitution parameters in the code frame:

Line Number	Code
4	`GLOBAL USING &GDA &WITH-BLOCK`
29	`01 #CODE-IN-LIST(A2/1:12) INIT<`
30	`&CODE-LIST>`
106	`USING MAP '&MAP-NAME'`

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.

User Exit	Description (continued)
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, read the Menu model code frame, CMNA?, into the Code Frame editor and refer to the following condition names:

- GDA-SPECIFIED
- DIRECT-COMMAND-PROCESSING
- MAP-USED

Defining the Model

At this point, you can define the model to Natural Construct using the Maintain Models function on the Administration main menu.

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.

The Menu model uses the following subprograms:

Subprogram	Description
CUMNPDA	Model parameter data area (PDA).
CUMNMA0	Map associated with the first modify subprogram.
CUMNMA	First modify subprogram.
CUMNMB0	Map associated with the second modify subprogram.
CUMNMB	Second modify subprogram.
CUMNC	Clear subprogram.
CUMNR	Read subprogram.
CUMNPR	Pre-generation subprogram.
CUMNPS	Post-generation subprogram.
CUMNS	Save subprogram.
CUMNGAAA	Generation subprogram.
CUMNSAAA	Sample subprogram.
CUMND	Document subprogram.
CMNA?	Code frame containing the header and main body for the generated program.

➢ To add the Menu model to Natural Construct:

1 Invoke the Maintain Models function from the Administration main menu.

2 Specify the following parameters on the Maintain Models panel:

```
CSDFM                N A T U R A L   C O N S T R U C T                CSDFM0
Oct 08                          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
```

Maintain Models Panel

Most of the components listed on the previous page are listed on this panel. The components that are not listed are assigned through subprograms or code frames. The CUMNMA0 and CUMNMB0 maps are invoked through the CUMNMA and CUMNMB modify subprograms, respectively. The generation subprogram is assigned through the CMNA? code frame. For more information about defining a model, see **Step 1: Define the Scope of the Model**, page 154.

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 Specify the following parameters on the Generation main menu:

– Type “M” in the Function field.

– Type “CUMNPDA” in the Module field.

– Type “CST-PDA” in the Model field.

2 Press Enter.
The Standard Parameters panel is displayed.

3 Enter “Menu” in the Model field:

```
CUPDMA                    CST-PDA Parameter Data Area                    CUPDMA1
Apr 02                        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
```

Standard Parameters Panel for the CST-PDA Model

You are returned to the Generation main menu.

4 Enter “G” in the Function field.
Natural Construct generates the PDA.

5 Enter “E” in the Function field.
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

Creating 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                 I    4 /* Explicit len to translate
------------------------------------------------------------------- S 17   L 1
```

Example of a Translation LDA

➢ 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 this example. For more information, see **Step 7: Create the Translation LDAs and Maintenance Maps**, page 131.

Creating 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 CUMNMB0 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).

Field	Description (continued)
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. This title identifies the program for the List Generated Modules function on the Generation main menu.
Description	Brief description of what the program does. This description is written in the program banner.
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. This field is marked by default.
Message numbers	Indicates whether the menu uses message numbers (if field is marked) or message text (if field is blank).
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.

Field	Description (continued)
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 (R 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.

Creating the Modify Subprograms

Use the CST-Modify model in the Generation subsystem to create the modify subprograms (CUMNMA and CUMNMB). These subprograms invoke the CUMNMA0 and CUMNMB0 maps, respectively.

For an example of the modify subprograms for the Menu model, refer to the CUMNMA and CUMNMB subprograms in the SYSCST library.

➢ To create the CUMNMA modify subprogram:

1 Specify the following parameters on the Standard Parameters panel:

```
 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
```

Standard Parameters Panel for the
CUMNMA Modify Subprogram

➢ To create the CUMNMB modify subprogram:

1 Specify the following parameters on the Standard Parameters panel:

```
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
```

Standard Parameters Panel for the
CUMNMB Modify Subprogram

Creating 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 Specify the following parameters on the Standard Parameters panel:

```
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
```

Standard Parameters Panel for the
CUMNPR Pre-Generation Subprogram

Creating 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 Specify the following parameters on the Standard Parameters panel:

```
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
```

Standard Parameters Panel for the
CUMNPS Post-Generation Subprogram

Creating 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 needs 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 Specify the following parameters on the Standard Parameters panel:

```
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
```

Standard Parameters Panel for the
CUMNC Clear Subprogram

Creating 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 Specify the following parameters on the Standard Parameters panel:

```
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
```

Standard Parameters Panel for the
CUMNS Save Subprogram

Creating 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 Specify the following parameters on the Standard Parameters panel:

```
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
```

Standard Parameters Panel for the
CUMNR Read Subprogram

Creating 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 Specify the following parameters on the Standard Parameters panel:

```
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
```

Standard Parameters Panel for the
CUMNGGL Generation Subprogram

Creating the Document Subprogram

Use the CST-Document model in the Generation subsystem to create the document subprogram.

For an example of the document subprogram for the Menu model, refer to the CUMND subprogram in the SYSCST library.

➢ To create the CUMND document subprogram:

1 Specify the following parameters on the Standard Parameters panel:

```
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
```

Standard Parameters Panel for the
CUMND Document Subprogram

2 Press PF11 (right).
The Additional Parameters panel is displayed.

3 Specify the following parameters:

```
CUGDMB                    CST-Document Subprogram                    CUGDMB0
Oct 09                     Additional Parameters                      2 of 2

  Help Text .......... Type ..... O
                       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
```

Specific Parameters Panel for the
CUMND Document Subprogram

Testing the Model Subprograms

Natural Construct supplies a utility to help you 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 **Testing the Model Subprograms**, page 143.

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:

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.

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 chapter describes how to use the CST-PDA model to generate the model PDA.

The following topics are covered:

- **Introduction**, page 176
- **Parameters for the CST-PDA Model**, page 177

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**, page 235.
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 to Natural Construct. Natural Construct uses information in the model code frames to generate the model PDA, such as:

- substitution parameters
- condition codes

For information about isolating the parameters for your model PDA, see **Step 4: Isolate the Parameters in the Prototype**, page 101.

For information about creating code frames and defining models, see **Step 5: Create Code Frame(s) and Define the Model**, page 103.

For more information about creating the model PDA, see **Step 6: Create the Model PDA**, page 115.

For an example of a generated model PDA, refer to CUMNPDA in the SYSCST library.

Parameters for the CST-PDA Model

After you create the code frames and define your model, use the CST-PDA model to generate the model PDA. The CST-PDA model has one specification panel: Standard Parameters. This panel is described in the following section.

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
```

Standard Parameters Panel for the CST-PDA Model

The fields on this panel are:

Field	Description
Module name	Name of the model PDA (the name specified on the Generation main menu). The name must be alphanumeric and no more than 8 characters in length. Use the following naming convention: CX*xx*PDA where *xx* uniquely identifies your model.
Model name	Name of the model that uses the model PDA. (The specified model and its corresponding code frames must be 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.

Layout of the Generated Model PDA

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 redefinition 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 more information about substitution parameters, see the Natural documentation.)

For each condition code, the model generates a logical field (prefixed by #PDAC) within the redefinition of the #PDA-CONDITION-CODES field in the model PDA.

CST-CLEAR MODEL

This chapter 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.

The following topics are covered:

- **Introduction**, page 180
- **Parameters for the CST-Clear Model**, page 181
- **User Exits for the CST-Clear Model**, page 182

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.

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. The CST-Clear model has one specification panel, Standard Parameters, and one user exit panel. These panels are described in the following sections.

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
```

Standard Parameters Panel for the CST-Clear Model

The 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 8 characters in length. Use the following naming convention: CX*xx*C where *xx* uniquely identifies your model.

Field	Description (continued)
Parameter data area	Name of the parameter data area (PDA) for your model. Natural Construct determines the name of the PDA by the Module name was 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. Use the following naming convention: CX*xx*PDA where *xx* uniquely identifies your model.
Title	Title for the clear subprogram. The title identifies the generated clear subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the clear subprogram. The description is inserted in the banner at the beginning of the clear 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
```

User Exits Panel for the CST-Clear Model

For information about these user exits, see **Supplied User Exits**, page 249. For information about using the User Exit editor, see **User Exit Editor**, page 98, *Natural Construct Generation*.

CST-READ MODEL

This chapter describes the CST-Read model used to generate the read subprogram for your model. The read subprogram reads the specifications for the model.

The following topics are covered:

- **Introduction**, page 184
- **Parameters for the CST-Read Model**, page 185
- **User Exits for the CST-Read Model**, page 186

Introduction

After defining the model PDA and clear subprogram, you must create a read 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.

Note: Natural Construct invokes the clear subprogram before invoking the read subprogram. It is not necessary to save null parameter values.

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 in the Generation subsystem to generate the read subprogram. The CST-Read model has one specification panel, Standard Parameters, and one user exit panel. These panels are described in the following sections.

Standard Parameters Panel

```
CUGRMA                        CST-Read Subprogram                        CUG-MA1
Feb 04                        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
```

Standard Parameters Panel for the CST-Read Model

The fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. This name must be alphanumeric and no more than 8 characters in length. Use the following naming convention: CX*xx*R where *xx* uniquely identifies your model.

Field	Description
Parameter data area	Name of the parameter data area (PDA) for your model. Natural Construct determines the name of the PDA by the Module name was specified on the Generation main menu. For example, if you entered CXMNR as the read subprogram name, Natural Construct assumes the PDA name is CXMNPDA. Use the following naming convention: CX*xx*PDA where *xx* uniquely identifies your model.
Title	Title for the read subprogram. The title identifies the generated read subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the read subprogram. The description is inserted in the banner at the beginning of the read subprogram and is used internally for program documentation.

User Exits for the CST-Read Model

```
CSGSAMPL                    CST-Read Subprogram                        CSGSM0
Oct 09                           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
```

CST-Read User Exits Panel

For information about these user exits, see **Supplied User Exits**, page 249. For information about using the User Exit editor, see **User Exit Editor**, page 98, *Natural Construct Generation*.

CST-SAVE MODEL

This chapter describes the CST-Save model, which you use to generate the save subprogram for your model. The save subprogram writes the specification parameters to the source buffer.

The following topics are covered:

- **Introduction**, page 188
- **Parameters for the CST-Save Model**, page 189
- **User Exits for the CST-Save Model**, page 190

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 in the Generation subsystem to generate the save subprogram. The CST-Save model has one specification panel, Standard Parameters, and one user exit panel. These panels are described in the following sections.

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
```

Standard Parameters Panel for the CST-Save Model

The 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 8 characters in length. Use the following naming convention: CX*xx*S where *xx* uniquely identifies your model.

Field	Description
Parameter data area	Name of the parameter data area (PDA) for your model. Natural Construct determines the name of the PDA from the Module name specified on the Generation main menu. For example, if you entered CXMNS as the save subprogram name, Natural Construct assumes the PDA name is CXMNPDA. Use the following naming convention: CX*xx*PDA where *xx* uniquely identifies your model.
Title	Title for the save subprogram. The title identifies the generated save subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the save subprogram. The description is inserted in the banner at the beginning of the save 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
```

CST-Save User Exits Panel

For information about these user exits, see **Supplied User Exits**, page 249. For information about using the User Exit editor, see **User Exit Editor**, page 98, *Natural Construct Generation*.

CST-MODIFY AND CST-MODIFY-332 MODELS

This chapter describes the CST-Modify and CST-Modify-332 models used to generate the modify (maintenance) subprograms for your model. The CST-Modify model generates specification panels that support dynamic translation. The CST-Modify-332 model is provided for users who want to continue using modify subprograms that were generated using previous versions of Natural Construct.

The following topics are covered:

- **Introduction**, page 192
- **CST-Modify Model**, page 193
- **Parameters for the CST-Modify Model**, page 194
- **User Exits for the CST-Modify Model**, page 202
- **CST-Modify-332 Model**, page 203
- **Parameters for the CST-Modify-332 Model**, page 205
- **User Exits for the CST-Modify-332 Model**, page 207

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 modify subprograms to collect user-supplied specification parameters (#PDAX variables), perform validation checks, and set the condition codes and #PDA variables (optional).

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

Note: A modify 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).

CST-Modify Model

The CST-Modify model generates model modify 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 modify subprogram.

The CST-Modify model generates either a main modify subprogram panel (defined on the Maintain Models panel) or a modify subprogram subpanel (invoked from the main modify 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, a subroutine can 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 modify subprogram panel.

All modify subprograms require a VALIDATE-INPUT subroutine to process mandatory edits. At generation time, the edits for the modify subprogram subpanel are processed first, then the edits for the main modify subprogram panel are processed. Therefore, any subroutine edits should also be included in the VALIDATE-INPUT subroutine. To avoid confusion about the order of execution of the panel and subpanel subroutines, place edit checks in programs rather than in subroutines.

Note: The CST-Modify model also allows you to override the headers and PF-keys defined on the Subprogram record.

For an example of a modify subprogram panel generated by the CST-Modify model, refer to CUMNMB in the SYSCST library. For an example of a modify subprogram subpanel generated by the CST-Modify model, refer to CUMNMBA in SYSCST.

Parameters for the CST-Modify Model

Use the CST-Modify model to generate a modify subprogram that supports dynamic translation. This model has one specification panel, Standard Parameters, and one user exit panel. These panels are described in the following sections.

Standard Parameters Panel

```
CUGIMA                      CST-Modify Subprogram                      CUGIMA0
Jun 25                       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
```

Standard Parameters Panel for the CST-Modify Model

Use this panel to define standard parameters, such as the map and translation LDAs used with the modify 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 fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name of the modify subprogram must be alphanumeric and no more than eight characters in length. Use the following naming convention: Panel: CX*xx*M*y* Subpanel: CX*xx*M*yz* where *xx* uniquely identifies your model and *y* is a letter from A to J that identifies the maintenance panel (A for the first maintenance panel, B for the second, etc.), and *z* is a letter from A to 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 entered CXMNMA as the modify subprogram name, Natural Construct assumes the PDA name is CXMNPDA. Use the following naming convention: CX*xx*PDA where *xx* uniquely identifies your model.
Title	Title for the modify subprogram. The title identifies the generated modify subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the modify subprogram. The description is inserted in the banner at the beginning of the modify subprogram and is used internally for program documentation.

Field	Description (continued)
Map name	Name of the map for the modify subprogram. Natural Construct determines the name of the map based on the Module name specified on the Generation main menu. For example, if you entered “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 modify 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 modify 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 modify subprogram name, with the addition of “L”. For example: **Program** **Translation LDA** CXMNMA CXMNMAL CXMNMB CXMNMBL
Cursor translation	If this field is marked, the generated subprogram panel supports cursor translation (users can modify the text on this panel in translation mode).

Field	Description (continued)
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 on the model record. To override this default, specify the new header in this field. 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, 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 **Using SYSERR References**, page 406, 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 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.

Field	Description (continued)
	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 the ",+/" indicates the heading displayed. The number after the ",+/" indicates the number of bytes in which the heading is centered. For information about using SYSERR message numbers, see **Using SYSERR References**, page 406, or refer to the SYSERR utility in the Natural Utilities documentation.
Note:	Data substitution within SYSERR references is not supported in this context.
Subpanel	If this field is marked, the generated subprogram is created as a subpanel that is invoked from a main panel (such as a help selection window). 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 on the Standard Parameters panel. For more information, see **PF6 (pfkey)**, page 200.
Window support	If this field is marked, the generated subprogram is displayed in a window. 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 on the Standard Parameters panel. For more information, see **PF5 (windw)**, page 199.

PF5 (windw)

To change the default window settings, press PF5 (windw). The Window Parameters window is displayed:

```
CUGIDWM                     Natural Construct                    CUGIDWM0
Jun 25                      Window Parameters                      1 of 1

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

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

Frame OFF .... _

Control screen _

Title ........ ____________________________________________________________
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF
      help  retrn quit                                                  ma
```

Window Parameters Window

Use this window to change the size and position of the window, indicate whether it is displayed with a border, or define CONTROL processing (SCREEN or WINDOW).

The fields in this window are:

Field		Description
Size	Height	Number of lines the window spans. This value is included in the DEFINE WINDOW command generated by the subprogram.
	Width	Number of columns the window spans. This value is included in the DEFINE WINDOW command generated by the subprogram. If the defined width is too small, Natural will adjust the size of the window.
Position	Line	Number of lines between the top of the panel and the top of the window begins. This value is included in the DEFINE WINDOW command generated by the subprogram.
	Column	Number of columns between the left edge of the panel and the left edge of the window. This value is included in the generated DEFINE WINDOW command. The specified line and column form the top left corner of the window.

Field	Description (continued)
Frame OFF	If this field is marked, the window does not use a frame (border).
Control screen	If this field is marked, the PF-keys and messages are displayed within the generated window (CONTROL SCREEN). If this field is blank, the PF-keys and messages are displayed outside the window (CONTROL WINDOW).
Title	Title for the window (either text or the name of the variable that supplies the title). The title is automatically centered in the window frame. By default, the window does not have a title.

PF6 (pfkey)

To define processing for non-standard PF-keys, press PF6 (pfkey) on the Standard Parameters panel. The PF-Key Parameters window is displayed:

```
CUGIMAA                     Natural Construct                     CUGIMAA0
Jun 25                      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
```

PF-Key Parameters Window

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 by:

- Defining the processing and name for a non-standard PF-key
- Changing the processing and/or name for a non-standard PF-key

The fields in this window are:

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 (text or valid SYSERR message number). If this field is blank, the default key names are used. For information about SYSERR message numbers, see **Using SYSERR References**, page 406, or refer to the SYSERR utility in the Natural Utilities documentation.

Note: The left and right PF-keys are available only if the modify subprogram is a subpanel.

User Exits for the CST-Modify Model

```
CSGSAMPL                    CST-Modify Subprogram                       CSGSM0
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
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
frwrd help  retrn quit                    bkwrd frwrd
```

CST-Modify User Exits Panel 1

```
CSGSAMPL                    CST-Modify Subprogram                       CSGSM0
Oct 09                            User Exits                             1 of 1

               User Exits             Exists    Sample   Required Conditional
    -------------------------------- -------- ---------- -------- ------------
 _  END-OF-PROGRAM                             Example
```

CST-Modify User Exits Panel 2

For information about these user exits, see **Supplied User Exits**, page 249. For information about using the User Exit editor, see **User Exit Editor**, page 98, *Natural Construct Generation*.

CST-Modify-332 Model

Use the CST-Modify-332 model to generate a standard modify subprogram that does not support dynamic translation. This model is provided for users who want to continue using modify subprograms that were generated under previous versions of Natural Construct.

Note: We recommend that you use the CST-Modify model to create new models. For information, see **CST-Modify Model**, page 193.

Example of a Model Modify Subprogram

The following example shows the first 40 lines of the CUMNMA modify subprogram:

```
0010 **SAG GENERATOR: CST-MODIFY                        VERSION: 4.4.1
0020 **SAG TITLE: Menu Model Modify Subp
0030 **SAG SYSTEM: NATURAL-CONSTRUCT
0040 **SAG DATA-AREA: CUMNPDA
0050 **SAG MAP: CU--MA0
0060 **SAG DESCS(1): This subprogram is used as modify panel 1
0070 **SAG DESCS(2): 1 of 2
0080 **SAG HEADER2: *0311.1,+/54
0090 **SAG TRANSLATION-LDA(1): CU--MAL
0100 **SAG DYNAMIC-TRANSLATION: X
0110 ************************************************************
0120 * Program  : CUMNMA
0130 * System   : NATURAL-CONSTRUCT
0140 * Title    : Menu Model Modify Subp
0150 * Generated: May 03,02 at 05:33 PM by REGEN41
0160 * Function : This subprogram is used as modify panel 1
0170 *            1 of 2
0180 *
0190 *
0200 * History
0210 ************************************************************
0220 DEFINE DATA
0230   PARAMETER USING CUMNPDA       /* Model specific data
0240   PARAMETER USING CU--PDA       /* Standard model parameters
0250   PARAMETER USING CSASTD        /* Standard message passing area
0260   LOCAL USING CNAMSG            /* Message retrieval passing area
0270   LOCAL USING CSLRCODE          /* Message return codes
0280   LOCAL USING CSAMARK           /* Field mark information
0290   LOCAL USING CSLPHASE          /* Valid generation phases
0300   LOCAL USING CSLSTD            /* Local message passing area
0310   LOCAL USING CSACURS      /* Used by CSUCURS to translate prompts
0320   LOCAL USING CU--MAL      /* Translation LDA
0330   LOCAL
0340     01 #PROGRAM (A8)
0350     01 LOCAL-TRANSLATION
0360       02 TEXT
0370         03 #HEADER2 (A54)
0380             INIT<'*0311.1,+/54'>
0390       02 REDEFINE TEXT
0400         03 TRANSLATION-TEXT
....
```

Parameters for the CST-Modify-332 Model

The CST-Modify-332 model has one specification panel, Standard Parameters, and one user exit panel. These panels are described in the following sections.

Standard Parameters Panel

```
CUGMMA                    CST-Modify-332 Subprogram                    CUGMMA0
Mar 25                       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
```

Standard Parameters Panel for the CST-Modify-332 Model

Use this panel to specify standard parameters for the modify subprogram. The fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name of the modify subprogram must be alphanumeric and no more than eight characters in length. Use the following naming convention: CX*xx*M*y* where *xx* uniquely identifies your model and *y* is a letter from A to J that identifies the maintenance panel (A for the first maintenance panel, B for the second, etc.).

Field	Description (continued)
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 entered CXMNMA as the modify subprogram name, Natural Construct assumes the PDA name is CXMNPDA. Use the following naming convention: CX*xx*PDA where *xx* uniquely identifies your model.
Map name	Name of the map for the modify subprogram. Natural Construct determines the name of the map based on the Module name specified on the Generation main menu. For example, if you entered “CXMNMA” as the modify subprogram name, Natural Construct assumes the map name is CXMNMA1. The map must exist in the current library, and the map name should correspond to the modify subprogram name, with the addition of the language code. For example: **Program** **Map** CXMNMA CXMNMA1 (for English)
Title	Title for the modify subprogram. The title identifies the generated modify 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                    CSGSM0
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
```

User Exits Panel for the CST-Modify-332 Model

For information about these user exits, see **Supplied User Exits**, page 249. For information about using the User Exit editor, see **User Exit Editor**, page 98, *Natural Construct Generation*.

CST-PREGEN MODEL

This chapter describes the CST-Pregen model used to generate the pre-generation subprogram for your model. The pre-generation subprogram is invoked after all modify subprograms are executed during the generation phase or when the SAMPLE command is issued from the User Exit editor.

The following topics are covered:

- **Introduction**, page 210
- **Parameters for the CST-Pregen Model**, page 211
- **User Exits for the CST-Pregen Model**, page 212

Introduction

After generating the modify subprograms, you can 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 modify 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. (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 modify subprograms.)

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 generate the pre-generation subprogram. The CST-Pregen model has one specification panel, Standard Parameters, and one user exit panel. These panels are described in the following sections.

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
```

Standard Parameters Panel for the CST-Pregen Model

The fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name of the pre-generation subprogram must be alphanumeric and no more than eight characters in length. Use the following naming convention: CX*xx*PR where *xx* uniquely identifies your model.

Field	Description (continued)
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 entered “CXMNPR” as the pre-generation subprogram name, Natural Construct assumes the PDA name is CXMNPDA. Use the following naming convention: CX*xx*PDA where *xx* uniquely identifies your model.
Title	Title for the pre-generation subprogram. The title identifies the generated pre-generation subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the pre-generation subprogram. The description is inserted in the banner at the beginning of the pre-generation subprogram and is used internally for program documentation.

User Exits for the CST-Pregen Model

```
CSGSAMPL                    CST-Pregen Subprogram                      CSGSM0
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
```

User Exits Panel for the CST-Pregen Model

For information about these user exits, see **Supplied User Exits**, page 249. For information using about the User Exit editor, see **User Exit Editor**, page 98, *Natural Construct Generation*.

CST-POSTGEN MODEL

This chapter describes the CST-Postgen model used to generate the pre-generation subprogram for your model. The post-generation subprogram supplies values for the substitution parameters in the code frames. This is the final stage of the generation process.

The following topics are covered:

- **Introduction**, page 214
- **Parameters for the CST-Postgen Model**, page 215
- **User Exits for the CST-Postgen Model**, page 216

Introduction

After defining the pre-generation subprogram, you can generate the post-generation subprogram to supply values for substitution parameters in the code frames (identified by &). Generated using the CST-Postgen model, this subprogram is invoked as the final stage of the generation process when the application developer enters “G” 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.

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 generate the post-generation subprogram. The CST-Postgen model has one specification panel, Standard Parameters, and one user exit panel. These panels are described in the following sections.

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
```

Standard Parameters Panel for the CST-Postgen Model

The fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name of the post-generation subprogram must be alphanumeric and no more than 8 characters in length. Use the following naming convention: CX*xx*PS where *xx* uniquely identifies your model.
Model name	Name of the model that uses the post-generation subprogram. The model must be defined.

Field	Description (continued)
Title	Title for the subprogram. The title identifies the generated 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                      CSGSM0
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
```

User Exits Panel for the CST-Postgen Model

For information about these user exits, see **Supplied User Exits**, page 249. For information about using the User Exit editor, see **User Exit Editor**, page 98, *Natural Construct Generation*.

CST-FRAME MODEL

This chapter describes the CST-Frame model. This model creates sample subprograms for user exits and generation subprograms to supply parameters to the model.

The following topics are covered:

- **Sample Subprograms**, page 218
- **Generation Subprograms**, page 219
- **Parameters for the CST-Frame Model**, page 220
- **User Exits for the CST-Frame Model**, page 221

Sample Subprograms

Sample subprograms are invoked from a user exit. Generated using the CST-Frame model, these subprograms help the developer create user exit code by providing a starting sample. They 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 subprogram, use the CU—PDA.#PDAX-FRAME-PARM variable.

Before invoking the sample subprograms, Natural Construct invokes all modify subprograms and the pre-generation subprogram. This ensures that the current specification parameters are valid and the conditions are set.

You can define a sample subprogram by entering ".E" at the beginning of a user exit line in the Code Frame editor.

For more information about defining a sample subprogram, see **Parameters Supplied by User Exits**, page 106.

Generation Subprograms

Generation subprograms are invoked from a code frame. 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. The 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, 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, specify line type N at the > prompt in the Code Frame editor. You can also specify the constant parameter value at this prompt.

For more information about generation subprograms, see **Parameters Supplied by Generation Subprograms**, page 104.

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. The CST-Frame model has one specification panel, Standard Parameters, and one user exit panel. These panels are described in the following sections.

Standard Parameters Panel

```
CUGFMA                        CST-Frame Subprogram                      CUG-MA0
May 26                        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
```

Standard Parameters Panel for the CST-Frame Model

The 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 convention: CX*xx*G*yyy* where *xx* uniquely identifies your model and *yyy* identifies your generation subprogram. or CX*xx*S*yyy* where *xx* uniquely identifies your model and *yyy* identifies your sample subprogram.

Field	Description (continued)
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 entered CXMNGAAA, Natural Construct assumes the PDA name is CXMNPDA. Use the following naming convention: CX*xx*PDA where *xx* uniquely identifies your model.
Title	Title for the frame subprogram. The title identifies the generated frame subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the frame subprogram. The description is inserted in the banner at the beginning of the frame subprogram and is used internally for program documentation.

User Exits for the CST-Frame Model

```
CSGSAMPL                    CST-Frame Subprogram                      CSGSM0
May 26                           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
```

User Exits Panel for the CST-Frame Model

For information about these user exits, see **Supplied User Exits**, page 249. For information about using the User Exit editor, see **User Exit Editor**, page 98, *Natural Construct Generation*.

CST-DOCUMENT MODEL

This chapter describes the CST-Document model used to generate a document subprogram for your model. The document subprogram writes information about Natural Construct-generated modules in the Predict data dictionary.

The following topics are covered:

- **Introduction**, page 224
- **Parameters for the CST-Document Model**, page 225
- **User Exits for the CST-Document Model**, page 227

Introduction

After defining the generation and sample subprograms, you must generate the document 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 document subprogram writes the model description to Predict when the developer invokes the Save Specification and Source or Stow Specification and Source function on the Generation main menu and presses PF5 (optns). For a description of the Generation main menu, see **Generation Main Menu**, page 60, *Natural Construct Generation*.

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

Parameters for the CST-Document Model

Use the CST-Document model to generate the document subprogram. The CST-Document model has two specification panels, Standard Parameters and Additional Parameters, and one user exit panel. These panels are described in the following sections.

Standard Parameters Panel

```
CUGDMA                    CST-Document Subprogram                    CUGDMA0
Mar 27                      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
```

Standard Parameters Panel for the CST-Document Model

The fields on the Standard Parameters panel are:

Field	Description
Module name	Name specified on the Generation main menu. The name of the document subprogram must be alphanumeric and no more than eight characters in length. Use the following naming convention: CX*xx*D where *xx* uniquely identifies your model.
Model name	Name of the model that uses the document subprogram. The model must be defined.

Field	Description (continued)
Maps	Names of all maps (specification panels) used by the model. The document 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**, page 131.
Title	Title for the document subprogram. The title identifies the generated document subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the document subprogram. The description is inserted in the banner at the beginning of the document subprogram and is used internally for program documentation.

Additional Parameters Panel

```
CUGDMB                    CST-Document Subprogram                    CUGDMB0
Apr 09                     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
```

Additional Parameters Panel for the CST-Document Model

On this panel, you can either:

- 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.

  or

- 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
Mar 27                    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
```

User Exits Panel for the CST-Document Model

For information about these user exits, see **Supplied User Exits**, page 249. For information about using the User Exit editor, see **User Exit Editor**, page 98, *Natural Construct Generation*.

CST-VALIDATE MODEL

This chapter describes the CST-Validate model used to generate the validation subprogram for your model. During the generation process, the validation subprogram verifies inputs for the model.

The following topics are covered:

- **Introduction**, page 230
- **Parameters for the CST-Validate Model**, page 231
- **User Exits for the CST-Validate Model**, page 233

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 generate a validation subprogram. The CST-Validate model has one specification panel, Standard Parameters, and one user exit panel. These panels are described in the following sections.

Standard Parameters Panel

```
CUVAMA                    CST-Validate Subprogram                    CUVAMA0
Jul 28                      Standard Parameters                       1 of 1

  Module ............. ________
  System ............. C421_____________________________

  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
```

Standard Parameters Panel for the CST-Validate Model

The fields on the Standard Parameters panel are:

Field	Description
Module	Name specified on the Generation main menu. The name of the validate subprogram must be alphanumeric and no more than eight characters in length. Use the following naming convention: CX*xx*VAL where *xx* uniquely identifies your model.
System	Name of the system (by default, the name of the current library). This is a required field. The system name must be alphanumeric and need not 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.)

Field	Description (continued)
Title	Title for the validate subprogram. The title identifies the generated subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the validate subprogram. The description is inserted in the banner at the beginning of the validate 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                          CSGSM0
Jul 24                              User Exits                            1 of 1
               User Exit              Exists    Sample   Required Conditional
    -------------------------------- -------- ---------- -------- ------------
 _  CHANGE-HISTORY                             Subprogram
 _  LOCAL-DATA
 _  GENERATE-VALIDATIONS
 _  GENERATE-SUBROUTINES                       Subprogram
```

User Exits Panel for the CST-Validate Model

For information about these user exits, see **Supplied User Exits**, page 249. For information about using the User Exit editor, see **User Exit Editor**, page 98, *Natural Construct Generation*.

Coding 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

To retrieve SYSERR messages, use the CU--VERR copycode.

For more information about coding validations, see **GENERATE-SUBROUTINES**, page 259.

Validating 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.

Tip: If you do not want to exit the current subroutine, as with array processing, use the CU--VERZ copycode instead of CU--VERR.

Tip: If you want to return a warning message, rather than an error, use the CU--VWAR copycode.

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
```

CST-STREAM MODEL

This chapter describes the CST-Stream model. This model generates a stream subprogram that converts the contents of a model PDA between internal and streamed format.

The following topics are covered:

- **Introduction**, page 236
- **Parameters for the CST-Stream Model**, page 237
- **User Exits for the CST-Stream Model**, page 238

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**, page 175.

Parameters for the CST-Stream Model

Use the CST-Stream model to generate a stream subprogram for your model. The CST-Stream model has one specification panel, Standard Parameters, and one user exit panel. These panels are described in the following sections.

Standard Parameters Panel

```
CUGTMA                      CST-Stream Subprogram                      CUGTMA0
Jul 24                       Standard Parameters                        1 of 1
 Module ............ ________
 System ............ C421____________________________


 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
```

Standard Parameters Panel for the CST-Stream Model

The 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: CX*xx*T where *xx* uniquely identifies your model.
System	Name of the system (by default, the name of the current library). This is a required field. The system name must be alphanumeric and 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.)

Field	Description (continued)
Title	Title for the stream subprogram. The title identifies the generated stream subprogram for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the stream subprogram. The description is inserted in the banner at the beginning of the stream 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.
Generate trace code	If this field is marked, extra code is generated into the stream subprogram to help trace inconsistencies between data sent by the client and data expected by the server.

User Exits for the CST-Stream Model

```
CSGSAMPL                      Natural Construct                        CSGSM0
Jul 24                            User Exits                          1 of 1
               User Exit              Exists    Sample   Required Conditional
    ------------------------------- --------- ---------- -------- ------------
 _  CHANGE-HISTORY                             Subprogram
 _  LOCAL-DATA
 _  ADDITIONAL-INITIALIZATIONS                  Example
 _  END-OF-PROGRAM
```

User Exits Panel for the CST-Stream Model

Note: Normally, this model does not require user exits.

For information about these user exits, see **Supplied User Exits**, page 249. For information about using the User Exit editor, see **User Exit Editor**, page 98, *Natural Construct Generation*.

CST-SHELL MODEL

This chapter describes the CST-Shell model used to generate a template for a model subprogram.

The following topics are covered:

- **Introduction**, page 240
- **Parameters for the CST-Shell Model**, page 241
- **User Exits for the CST-Shell Model**, page 243

Introduction

The CST-Shell model generates a template for a model subprogram. It is similar to the supplied Shell model. For information about the Shell model, see **Parameters for the Shell Model**, page 410, *Natural Construct Generation*.

The main differences between the CST-Shell model and the Shell model are that the CST-Shell model:

- Supports regeneration
- Supports messaging

The CST-Shell model creates a 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. It also includes the required REPEAT loops and messaging subroutines. You can use this time-saving model to generate startup modules for your model subprograms.

For an example of a generated shell program, refer to CUMPSLFV in the SYSCST library.

Parameters for the CST-Shell Model

Use the CST-Shell model to generate a shell program. The CST-Shell model has one specification panel, Standard Parameters, and one user exit panel. These panels are described in the following sections.

Standard Parameters Panel

```
CUGSMA                        CST-Shell Program                        CUGSMA0
Jul 11                       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
```

Standard Parameters Panel for the CST-Shell Model

The fields on the Standard Parameters panel are:

Field	Description
Module name	Name of the shell program you are creating (by default, the name specified in the Module name field on the Generation main menu). This is a required field. The module name must follow standard Natural naming conventions, must be alphanumeric, and cannot be 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)

Field	Description (continued)
System name	Name of the system (by default, the name of the current library). This is a required field. 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 shell program. The title identifies the generated shell program for the List Generated Modules function on the Generation main menu and is used internally for program documentation.
Description	Brief description of the shell program. The description is inserted in the banner at the beginning of the shell program and is used internally for program documentation.
Messaging support	If this field is marked, the shell program supports the dynamic translation of messages.
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 (program) or S (subroutine), 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
Oct 10                            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
```

User Exits Panel for the CST-Shell Model

For information about these user exits, see **Supplied User Exits**, page 249. For information about using the User Exit editor, see **User Exit Editor**, page 98, *Natural Construct Generation*.

USER EXITS FOR THE ADMINISTRATION MODELS

This chapter describes the user exits supplied for the Natural Construct administration models. The administration models generate the model subprograms used by all models.

The following topics are covered:

- **What are User Exits?**, page 246
- **Supplied User Exits**, page 249

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 you are generating. Some exits contain sample code or subroutines, while others generate the DEFINE EXIT…END-EXIT lines only — 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

Reusing 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 that 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          _
```

CLEAR Source Area Window

To retain the code in the source buffer for use with the new module, press Enter in this window. The first specification panel for the new model is displayed. 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), enter any non-blank character in the Mark if you wish to clear the source area field. The source buffer is cleared and the first specification panel for the specified model is displayed.

Invoking the User Exit Editor

You can invoke the User Exit editor from the Generation main menu (using the User Exit Editor function) or from the last specification panel for a model that supports user exits (by pressing PF11).

To invoke the User Exit editor from the Generation main menu, see **User Exit Editor**, page 98, *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 no user exits are defined for the specified module, a list of the user exits available for that model is displayed on the User Exits panel.
- If user exits are defined for the specified module, the existing user exit code is displayed in the User Exit editor. To select additional exits, enter SAMPLE at the > prompt to display the User Exits panel.

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 following example shows the first User Exits panel for the CST-Clear model:

```
CSGSAMPL                     CST-Clear Subprogram                      CSGSM0
Oct 09                            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
```

User Exits for the CST-Clear Model

The User Exits panel is similar for all models. 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	If the corresponding user exit is defined (exists), “X” is displayed. If the user exit does not exist, this field is blank.
Sample	If the user exit is empty (contains DEFINE EXIT … END-EXIT lines), this field is blank. If the user exit contains a subprogram, “Subprogram” is displayed. If the user exit contains sample code, “Example” is displayed.
Required	If the user exit must be specified, X is displayed. If the user exit is optional, this field is blank.
Conditional	If the user exit is conditional (its existence is based on condition codes in the code frames), X is displayed.

➢ 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: You can also define user exits in the User Exit editor without using the SAMPLE command.

Note: Fully qualify all references to database fields with the file name.

Defining User Exits

The code you can specify in a user exit depends on the type of module and user exit you are using. 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

The following sections describe the user exits available for the Natural Construct administration models. The sections are listed in alphabetical order, based on the user exit name. For many exits, one or more examples are also included.

ADDITIONAL-INITIALIZATIONS

This user exit generates the framework for any additional initializations performed in the INITIALIZATIONS subroutine.

Example of code in the ADDITIONAL-INITIALIZATIONS user exit

```
** SAG DEFINE EXIT ADDITIONAL-INITIALIZATIONS
*
* Assign parameters for help routine CD-HELPR
MOVE  'CU'  TO #MAJOR-COMPONENT
MOVE *PROGRAM   TO  #MINOR-COMPONENT
*
**SAG END-EXIT
*
END-SUBROUTINE /* INITIALIZATIONS
```

ADDITIONAL-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 in the ADDITIONAL-SUBSTITUTION-VALUES user exit

```
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 in the ADDITIONAL-TRANSLATIONS user exit

```
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
3150     ASSIGN CNAMSG.MSG-DATA(1) = #PDA-FRAME-PARM
3160     ASSIGN CNAMSG.MSG = CUBASRPL.#RETURN-MESSAGE
3170     PERFORM GET-MESSAGE-TEXT
3180     ASSIGN CUBASRPL.#RETURN-MESSAGE = CNAMSG.MSG
3190     RESET CNAMSG.INPUT-OUTPUTS
3200     /*
3210     /* Assign available keys
3220     ASSIGN CU--PDA.#PDA-AVAILABLE1-NAME = #AVAILABLE1-NAME
3230     ASSIGN CU--PDA.#PDA-AVAILABLE2-NAME = #AVAILABLE2-NAME
3240     ASSIGN CU--PDA.#PDA-AVAILABLE3-NAME = #AVAILABLE3-NAME
3250     RESET #FIRST-TRANSLATION
3260     /*
3270     /* Override pfkey settings
3280     RESET #LOCAL-PFKEYS-REQUIRED
3290     /*
3300     /* Set all PF-keys named off
3310     INCLUDE CU--SOFF
3320     /*
3330     /* Set Help and Return keys
3340     SET KEY DYNAMIC CU--PDA.#PDA-PF-HELP = HELP
3350             NAMED CU--PDA.#PDA-HELP-NAME
3360     SET KEY DYNAMIC CU--PDA.#PDA-PF-RETURN
3370             NAMED CU--PDA.#PDA-RETURN-NAME
3380     END-IF
3390     **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 or to 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.

Note: If you do not include the PERFORM NEW-SCREEN code and continue with execution after processing this exit, an Invalid PF-key error message is displayed.

Example of code in the AFTER-INPUT user exit

```
2730 **SAG DEFINE EXIT AFTER-INPUT
2740   IF #FORMAT-HELP
2750     RESET #FORMAT-HELP
2760     ASSIGN  CU--FHL.#TEXT-REQUIRED = TRUE
2770     PERFORM GET-PROMPT-TEXT
2780     INPUT WINDOW = 'FRMT' USING MAP 'CU--FH0'
2790     ASSIGN CSAMARK.ERROR-POS = POS(#PDAX-VARIABLE-FORMAT)
2800     ESCAPE BOTTOM (NEW-SCREEN.)
2810   END-IF
2820 *
2830 **SAG END-EXIT
```

AFTER-INVOKE-SUBPANELS

This user exit generates the framework for any processing performed after subpanels are invoked.

Example of code in the AFTER-INVOKE-SUBPANELS user exit

```
0100 DEFINE EXIT AFTER-INVOKE-SUBPANELS
0110   PERFORM SET-MORE-INDICATORS
0120 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.

Note: 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 in the ASSIGN-DERIVED-VALUES user exit

```
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 in the BEFORE-CHECK-ERROR user exit

```
1320 **SAG DEFINE EXIT BEFORE-CHECK-ERROR
1330 *
1340 * Use this user exit for specific error checking
1350   IF CSASTD.RETURN-CODE = CSLRCODE.#INTERRUPT(*)
1360     ASSIGN C--PDA.#PDA-PHASE = #SAVE-PHASE
1370   END-IF
1380  **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 in the BEFORE-INPUT user exit

```
0160 DEFINE EXIT BEFORE-INPUT
0170 *
0180 * Assign external value
0190   FOR #I = 1 TO 7
0200     IF #PDAX-BACKGROUND-COLOUR = #CD(#I) THEN
0210       ASSIGN #REVERSED-CD(#I) = TRUE
0220       ESCAPE BOTTOM
0230     END-IF
0240   END-FOR
0250 END-EXIT
```

BEFORE-INVOKE-SUBPANELS

This user exit generates the framework for any processing performed before subpanels are invoked.

Example of code in the BEFORE-INVOKE-SUBPANELS user exit

```
0680 DEFINE EXIT BEFORE-INVOKE-SUBPANELS
0690   IF CU--PDA.#PDA-PHASE NE CSLPHASE.#TRANSLATE THEN
0700     PERFORM VALIDATE-FILE-INFO
0710   END-IF
0720 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 in the BEFORE-REINPUT-MESSAGE user exit

```
0010 END-SUBROUTINE /* INPUT-SCREEN
0020 *
0030 * DEFINE SUBROUTINE REINPUT-MESSAGE
0040 *
0050 **SAG DEFINE EXIT BEFORE-REINPUT-MESSAGE
0060    IF CSASTD.RETURN-CODE = CSLRCODE.#COMMUNICATION THEN
0070       ESCAPE BOTTOM(PROG.) IMMEDIATE
0080    END-IF
0090 **SAG END-EXIT
0100    DECIDE FOR FIRST CONDITION
0110       WHEN CSASTD.RETURN-CODE = CSLRCODE.#CONTINUE(*)
0120              IGNORE
0130       WHEN CSASTD.RETURN-CODE = CSLRCODE.#INTERRUPT(*)
0140                   ESCAPE BOTTOM(NEW-SCREEN)
0150        WHEN NONE
0160              IGNORE
0170 END-DECIDE
```

BEFORE-STANDARD-KEY-CHECK

The code in this user exit checks any additional PF-keys defined for a modify subprogram or prepares for standard PF-key validations.

Example of code in the BEFORE-STANDARD-KEY-CHECK user exit

```
DEFINE EXIT BEFORE-STANDARD-KEY-CHECK
*
* Use this user exit to check additional PF-keys or prepare for the
* standard PF-key check.
END-EXIT BEFORE-STANDARD-KEY-CHECK
```

CHANGE-HISTORY

This user exit keeps a record of changes to the generated module. It generates comment lines indicating the date, the user ID of the user who created or modified the module, and a description of any change.

Example of code in the CHANGE-HISTORY user exit

```
DEFINE EXIT CHANGE-HISTORY
* Changed on Aug 27,02 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 in the DESCRIBE-INPUTS user exit

```
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 in the END-OF-PROGRAM user exit

```
3310 **SAG DEFINE EXIT END-OF-PROGRAM
3320 *
3330 * Actions to be performed before program exit.
3340   IF #PDAX-GDA NE ' ' AND #PDA-PHASE = 'M' THEN
3350     ASSIGN CNAEXIST.#OBJECT-SOURCE = 'O'
3360     ASSIGN CNAEXIST.#LIBRARY-NAME = *LIBRARY-ID
3370     ASSIGN CNAEXIST.#INCLUDE-STEPLIB-SEARCH = TRUE
3380     ASSIGN CNAEXIST.#OBJECT-NAME   = #PDAX-GDA
3390     CALLNAT 'CNUEXIST' CNAEXIST
3400                        CSASTD
3410     PERFORM CHECK-ERROR
3420     IF NOT CNAEXIST.#OBJECT-EXISTS THEN
3430       ASSIGN CNAMSG.RETURN-CODE = CSLRCODE.#WARNING
3440       ASSIGN CNAMSG.MSG-DATA(1) = CU--MAL.#GDA
3450       ASSIGN CNAMSG.MSG-DATA(2) = #PDAX-GDA
3460       INCLUDE CU--GMSG '2128'
3470         ''':1::2::3:not in current library or STEPLIBs'''
3480     END-IF
3490   END-IF
3500 **SAG END-EXIT
```

GENERATE-CODE

This user exit generates the framework for any code generated by a model subprogram.

Example of code in the GENERATE-CODE user exit

```
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

To retrieve SYSERR messages, use the CU--VERR copycode.

Tip: If you want to return a warning message, rather than an error, use the CU--VWAR copycode.

For more information about coding validations, see **CST-Validate Model**, page 229. For information about validating array fields, see **Validating Array Fields**, page 234.

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. For more information, see **CST-Validate Model**, page 229.

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 in the INPUT-ADDITIONAL-PARAMETERS user exit

```
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 modify subprogram.

Example of code in the INPUT-SCREEN user exit

```
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 in the LOCAL-DATA user exit

```
0480 **SAG DEFINE EXIT LOCAL-DATA
0490     01 #HELPR(A8) INIT<'CD-HELPR'>
0500   LOCAL USING CNAEXIST
0510 **SAG END-EXIT
```

MISCELLANEOUS-SUBROUTINES

This user exit generates the framework for any additional subroutines used by a modify subprogram.

Example of code in the MISCELLANEOUS-SUBROUTINES user exit

```
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 prompt and field values to Predict. To generate the correct code, translation LDAs must adhere to the following naming standards:

Field Name	Prompt
#PDA-GEN-PROGRAM	CUMNMAL.#GEN-PROGRAM
#PDAX-TITLE	CUMNMAL.#TITLE

Example of code in the MISCELLANEOUS-VARIABLES user exit

```
0010 DEFINE EXIT MISCELLANEOUS-VARIABLES
0020 ****************************************************************
0030 DEFINE SUBROUTINE MISCELLANEOUS
0040 ****************************************************************
0050 *
0060   WRITE(SRC) NOTITLE 20T CU--DOCL.#MISC-SPECIFICATIONS
0070   WRITE(SRC) NOTITLE     CU--PDA.#PDA-UNDERSCORE-LINE (AL=70)
0080   WRITE(SRC) NOTITLE ' '
0090 END-SUBROUTINE /* MISCELLANEOUS
0100 END-EXIT
```

PARAMETER-DATA

This user exit generates the framework to process any additional parameters used in conjunction with other programs.

Example of code in the PARAMETER-DATA user exit

```
DEFINE EXIT PARAMETER-DATA
** PARAMETER USING PDAname
** PARAMETER
**  01 #Additional-parameter1
**  01 #Additional-parameter2
END-EXIT PARAMETER-DATA
```

PF-KEYS

This user exit documents information about PF-keys supported by a generated subprogram. To document information about PF-keys, mark this exit and press Enter. A window is displayed, in which you can specify the supported PF-keys. Descriptions of the specified keys are added to Predict.

Example of code in the PF-KEYS user exit

```
0090 * Translate pfkey functions
0100   PERFORM GET-CDKEYFL-TEXT
0110 *
0120 * Write pfkey names and functions
0130   PRINT(SRC) NOTITLE / 20T CU--DOCL.#PFKEY-SUPPORT
0140     / ' '
0150     / 3T CU--DOCL.#PFKEY 14T CU--DOCL.#FUNCTION
0160     / 3T CU--PDA.#PDA-UNDERSCORE-LINE (AL=10)
0170          CU--PDA.#PDA-UNDERSCORE-LINE (AL=60)
0180     / 3T CDKEYLDA.#KEY-NAME(2)
0190      14T CDKEYFL.#KEY-FUNCTION(2)
0200 END-SUBROUTINE /* PF-KEYS
0210 END-EXIT
0220 DEFINE EXIT PF-KEYS
0230 ****************************************************************
0240 DEFINE SUBROUTINE PF-KEYS
0250 ****************************************************************
0260 *
0270 * Translate pfkey names
0280   INCLUDE CU--DOC
0290 *
0300 * Translate pfkey functions
0310   PERFORM GET-CDKEYFL-TEXT
0320 *
0330 * Write pfkey names and functions
0340   PRINT(SRC) NOTITLE / 20T CU--DOCL.#PFKEY-SUPPORT
0350     / ' '
0360     / 3T CU--DOCL.#PFKEY 14T CU--DOCL.#FUNCTION
0370     / 3T CU--PDA.#PDA-UNDERSCORE-LINE (AL=10)
0380          CU--PDA.#PDA-UNDERSCORE-LINE (AL=60)
0390     / 3T CDKEYLDA.#KEY-NAME(3)
0400      14T CDKEYFL.#KEY-FUNCTION(3)
0410 END-SUBROUTINE /* PF-KEYS
0420 END-EXIT
```

PROCESS-SPECIAL-KEYS

This user exit is required for the CST-Modify-332 model if the subprogram supports special PF-keys (all keys other than Enter and help, return, quit, right, and left PF-keys).

Define the special PF-keys on the Maintain Subprograms panel. For information, see **Maintain Subprograms Function**, page 52. 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 in the PROCESS-SPECIAL-KEYS user exit

```
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. To specify default values for additional parameters in your model PDA, regenerate this user exit. This adds the new variables but does not modify the code from the previous generation.

Example of code in the PROVIDE-DEFAULT-VALUES user exit

```
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.

Note: If you add specification parameters to the 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 in the SAVE-PARAMETERS user exit

```
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. For example, 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 in the SET-CONDITION-CODES user exit

```
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 (such as initializing input values from globals, setting window or page sizes, or capturing 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 this user exit, STACK statements for all substitution variables are generated in this user 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 ADDITIONAL-SUBSTITUTION-VALUES user exit (or no user exit), the model will automatically stack any model PDA variables that match the &SUBSTITUTION values in the code frame. For example:

```
STACK TOP DATA FORMATTED '&PRIME-FILE' #PDAX-PRIME-FILE
```

- If you use this user exit, code all substitution values in the user exit since default code will not be generated.

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 the maintenance map (specified in the Map name field on the Standard Parameters panel).

The following sections contain 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 user exit code for the CST-Modify model

```
0010 DEFINE EXIT VALIDATE-DATA
0020   DECIDE FOR EVERY CONDITION
0030     WHEN #HEADER1 = ' '
0040       ASSIGN CNAMSG.MSG-DATA(1) = #HEADER1
0050       INCLUDE CU--RMSG '2001'
0060       ''':1::2::3:is required'''
0070       '#HEADER1'
0080     WHEN #HEADER2 = ' '
0090       ASSIGN CNAMSG.MSG-DATA(1) = #HEADER2
0100       INCLUDE CU--RMSG '2001'
0110       ''':1::2::3:is required'''
0120       '#HEADER2'
0130     WHEN #PDA-GEN-PROGRAM = ' '
0140       ASSIGN CNAMSG.MSG-DATA(1) = #GEN-PROGRAM
0150       INCLUDE CU--RMSG '2001'
0160       ''':1::2::3:is required'''
0170       '#PDA-GEN-PROGRAM'
0180     WHEN #PDA-SYSTEM = ' '
0190       ASSIGN CNAMSG.MSG-DATA(1) = #SYSTEM
0200       INCLUDE CU--RMSG '2001'
0210       ''':1::2::3:is required'''
0220       '#PDA-SYSTEM'
0230     WHEN #PDA-TITLE = ' '
0240       ASSIGN CNAMSG.MSG-DATA(1) = #TITLE
0250       INCLUDE CU--RMSG '2001'
0260       ''':1::2::3:is required'''
0270       '#PDA-TITLE'
0280     WHEN CUBAPDA.#PDAX-DESCS = ' '
0290       ASSIGN CNAMSG.MSG-DATA(1) = #DESCS
0300       INCLUDE CU--RMSG '2001'
0310       ''':1::2::3:is required'''
0320       'CUBAPDA.#PDAX-DESCS'
0330     WHEN CUBAPDA.#PDAX-GDA = ' '
0340       ASSIGN CNAMSG.MSG-DATA(1) = #GDA
0350       INCLUDE CU--RMSG '2001'
0360       ''':1::2::3:is required'''
0370       'CUBAPDA.#PDAX-GDA'
0380     WHEN CUBAPDA.#PDAX-GDA-BLOCK = ' '
0390       ASSIGN CNAMSG.MSG-DATA(1) = #GDA-BLOCK
0400       INCLUDE CU--RMSG '2001'
0410       ''':1::2::3:is required'''
0420       'CUBAPDA.#PDAX-GDA-BLOCK'
0430     WHEN CUBAMAL.#DESCRIPTION = ' '
0440       ASSIGN CNAMSG.MSG-DATA(1) = #DESCRIPTION
0450       INCLUDE CU--RMSG '2001'
0460       ''':1::2::3:is required'''
0470       'CUBAMAL.#DESCRIPTION'
0480     WHEN CUBAMAL.#GDA = ' '
0490       ASSIGN CNAMSG.MSG-DATA(1) = #GDA
0500       INCLUDE CU--RMSG '2001'
0510       ''':1::2::3:is required'''
0520       'CUBAMAL.#GDA'
```

```
0530     WHEN CUBAMAL.#GDA-BLOCK = ' '
0540       ASSIGN CNAMSG.MSG-DATA(1) = #GDA-BLOCK
0550       INCLUDE CU--RMSG '2001'
0560       ''':1::2::3:is required'''
0570       'CUBAMAL.#GDA-BLOCK'
0580     WHEN CUBAMAL.#GEN-PROGRAM = ' '
0590       ASSIGN CNAMSG.MSG-DATA(1) = #GEN-PROGRAM
0600       INCLUDE CU--RMSG '2001'
0610       ''':1::2::3:is required'''
0620       'CUBAMAL.#GEN-PROGRAM'
0630     WHEN CUBAMAL.#SYSTEM = ' '
0640       ASSIGN CNAMSG.MSG-DATA(1) = #SYSTEM
0650       INCLUDE CU--RMSG '2001'
0660       ''':1::2::3:is required'''
0670       'CUBAMAL.#SYSTEM'
0680     WHEN CUBAMAL.#TITLE = ' '
0690       ASSIGN CNAMSG.MSG-DATA(1) = #TITLE
0700       INCLUDE CU--RMSG '2001'
0710       ''':1::2::3:is required'''
0720       'CUBAMAL.#TITLE'
0730 END-EXIT
```

Example of user exit code for the 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.

MODIFYING THE SUPPLIED MODELS

This chapter 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).

The following topics are covered:

- **Introduction**, page 270
- **Modify the Supplied Models**, page 271
- **Example of Modifying a Model**, page 274
- **Using Steplibs to Modify Models**, page 277

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 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). 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**, page 42.

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 **Using Steplibs to Modify Models**, page 277.

Modify the Supplied Models

Typically, the Natural Construct administrator modifies 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.

Modifying Code Frames

Do not modify the supplied code frames. Instead, copy the code frames you want to customize and modify these. 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.

Note: 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.

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, invoke the Menu2 model. You can 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 **Parameters Supplied by Nested Code Frames**, page 105.

Modifying 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 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 **Testing the Model Subprograms**, page 143.

3 After testing the model subprograms in the SYSCST library, copy the modified modules to the SYSLIBS library in the FNAT system file.
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.

Modifying 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. If the changes only apply 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-generate 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 implement this feature:

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.
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 a panel; the map name is displayed in the top right corner of a panel.

   - Copy the current versions and change the “CU” prefix in the names 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 marking the Options field when generating a program using the Maint model. When the Status window is displayed, mark 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-generate 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 test copy of the model and have the test model refer to the X copies.

Note: Add the new PF-keys to CDKEYLDA. For information, see **Adding a New PF-Key**, page 130, *Natural Construct Generation.*

5 Modify the model subprograms as follows:

– Make 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:

```
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 **Testing the Model Subprograms**, page 143).

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.

8 Document all modifications to the model.

Using 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. 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 (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.

Invoking Natural Construct From a Steplib

To invoke Natural Construct from a steplib, define the SYSCST and SYSLIBS libraries as steplibs of all development libraries requiring Natural Construct. You should also define a higher level steplib where modules can be stored that override the supplied objects. This steplib should also contain a module called NCSTG, which is coded as follows:

```
FETCH 'CSTG'
END
```

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**, page 56.

EXTERNAL OBJECTS

This chapter describes the 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.

The following topics are covered:

- **Introduction**, page 280
- **Natural-Related Subprograms (CNU*)**, page 286
- **Natural-Related Helproutines (CNH*)**, page 304
- **Natural Construct Generation Utility Subprograms (CSU*)**, page 307
- **Predict-Related Subprograms (CPU*)**, page 361
- **Predict-Related Helproutines (CPH*)**, page 384
- **General Purpose Generation Subprograms (CU--*)**, page 388

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 PDA

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.

Example of a model subprogram requiring 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 chapter 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).

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**, page 66.

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 Category
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.

Processing Errors

Many of the supplied subprograms return information through the CSASTD parameter data area (PDA). You should check the value in the RETURN-CODE field 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.

Example of 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 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 only have to 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 (CP**R**ELNX, 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.

Note: 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 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 that CPURL-RD 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 Construct 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.

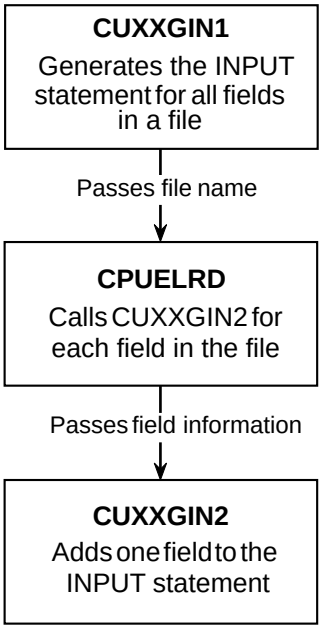
To avoid these problems, some subprograms do not return the information to the calling object. Instead, the calling object passes the name of a subprogram that is invoked for each record encountered. To generate an INPUT statement containing all fields in a file, for example, you can use the CPUELNX and CPUELRD subprograms. These subprograms are described in the following sections.

No 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.

Using Subprogram Chaining (CPUELRD)

The CPUELRD subprogram can be invoked once by your 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:



Example of Subprogram Chaining

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.

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 the following sections 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.

Note: Driver programs for many of the supplied programs and 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. For information about invoking the driver programs, see **Drivers Menu Function**, page 66.

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 SYSTEM-FNAT

Drivers Menu Option

```
 CTEELN            ***** Natural Related Subprograms *****         CTEELN1
 Oct 09                - Driver for subprogram CNUEL -             12:52 PM

*Data Area Name : ________
 Field Name.....: ________________________________
 Structure Name : ________________________________

 View Of Name...:

 Field Found....:     Field Format:        Lvl Number....:
 Constant Field :     Field Length:        Lvl Type Trail:
 Field Redefined:     Rank........:

 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
```

Driver for Subprogram CNUEL Window
Natural-Related Drivers Menu, Data Areas Menu,
Field Information Option

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. For information about INPUT/OUTPUT parameters, refer to the CNAELNX data area in the SYSCST library.
PDAs used	CNAELNX CNRELNX CSASTD
File 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

```
 CTENLNX           ***** Natural Related subprograms *****         CTENLNX1
 Dec 06               - Driver for subprogram CNUELNX -            09:50 PM

*Data Area Name...: ________      Field Count:       Constant Field :
 First Time.......: X             End Of File:       Field Redefined:
 Basic Occurrences: _

 Structure Name...:                                  Field Format...:
 Field Name.......:                                  Field Length...:
 View Of Name.....:                                  Units..........:
 Level Number.....:                                  Decimals.......:
 Level Type Trail.:                                  Rank...........:
 Occurrences Found:

                                       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
```

Driver for Subprogram CNUELNX Window
Natural-Related Drivers Menu, Data Areas Menu,
Get Next Field Information Option

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”.
PDA used	CSASTD
File accessed	SYSTEM-FNAT

Note: This subprogram returns system error messages, rather than application error messages. For information about application error messages, see **CNUMSG Subprogram**, page 296.

Drivers Menu 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
```

Driver for Subprogram CNUERMSG Window
Natural-Related Drivers Menu, Retrieve System Error Msg Option

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) where 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

```
 CTEEXIST          ***** Natural Related Subprograms *****          CTEXIST
 Oct 09              - Driver for subprogram CNUEXIST -             01:14 P

*Object/Source Name......: _______    Source             Object
 Library Name............: CST341__   -----------------  -----------------
                                      Exists.:           Exists.:
 Multi Steplib Search....: _          Type...:           Type...:
    Default DBID/FNR Only: _          Library:           Library:
                                      DBID...:           DBID...:
 Object/Source or Both...: _          FNR....:           FNR....:

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

Driver for Subprogram CNUEXIST Window
Natural-Related Drivers Menu, Verify Source/Object Existence Option

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

```
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
```

Driver for Subprogram CNUGDABL Window
Natural-Related Drivers Menu, Build Path Name for GDA Block Option

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

```
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
```

Driver for Subprogram CNUGDAEL Window
Natural-Related Drivers Menu, Verify GDA Field Existence Option

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

Drivers Menu Option

```
 CTEGENDA            N a t u r a l  C o n s t r u c t            CTEGEND1
 Aug 14               Driver for subprogram CNUGENDA               1 of 1

Field Name: ______________________________

Field Type: _ Format: _     Occurrences: ____
Level.....: _ Length: _____ Comment....: ________________________________


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

Driver for Subprogram CNUGENDA Window
Natural-Related Drivers Menu, Data Areas Menu,
Add a Field to a Data Area Option

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
File accessed	SYSTEM-FNAT

Note: This routine is not available on all platforms.

Drivers Menu 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
```

Driver for Subprogram CNUMPPRF Window
Natural-Related Drivers Menu, Map Settings Information Option

CNUMSG Subprogram

CNUMSG	Description
What it does	Returns application message text from the SYSERR message file. 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. A message number can be entered in either the Message Number field or the 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”.

CNUMSG	Description (continued)
S	Performs text substitutions in the Message Text (Input) field. A substitution will occur if typing placeholders are found in the message text. Placeholders are replaced at runtime with a value entered in one of the Message Substitution Data fields (1, 2, and 3). Placeholders are entered in the following format: ":N:", where "N" identifies one of the three Message Substitution Data fields. For example, if you entered 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. For more information about message numbers and placeholders, see **Using SYSERR References**, page 406.
B	Performs text retrieval using methods R and S which are explained earlier in this section. 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 entered 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. 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, see **Using SYSERR References**, page 406.

CNUMSG	Description (continued)
PDA used	CNAMSG CSASTD
File accessed	SYSTEM-FUSER

Drivers Menu 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
```

Driver for Subprogram CNUMSG Window
Natural-Related Drivers Menu, Retrieve Application Error Message
Retrieve Single Message Option

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.
PDA used	CNAPEXST
File accessed	SYSTEM-FNAT

Note: This module is not available on all platforms.

Drivers Menu 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
```

Driver for Subprogram CNUPEXST Window
Natural-Related Drivers Menu, Verify Map Profile Existence Option

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 09,96            - 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
```

Driver for Subprogram CNUSEL Window

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

```
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
```

Driver for Subprogram CNUSRCNX Window
Natural-Related Drivers Menu, Get Next Source Line Option

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

```
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
```

Driver for Subprogram CNUSRCRD Window
Natural-Related Drivers Menu, CALLNAT For Each Source Line Option

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 following helproutines 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.
Parameter used	#PDA-RESTRICTION(A1) #PDA-KEY(A32) (model name)
File 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)
File 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) • 3 (subprogram/helproutine/subroutine) • 4 (program/subprogram/helproutine/subroutine) • 5 (command processor) • D (data area)
Attached to	Input of a Natural object name field.

CNHOBJ	Description
Parameters used	#PDA-TYPE(A1) #PDA-KEY(A8) /* Start/Return key
File accessed	SYSTEM-FUSER

Natural Construct Generation Utility Subprograms (CSU*)

The following subprograms 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's description. For more information about the supplied driver programs, see **Drivers Menu Function**, page 66.

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

```
 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
```

Driver for Subprogram CSU-VAR Window
Additional Drivers Menu, Validation Subroutines Menu
Validate a Variable Name Option

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.
Parameters 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.
	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. This subprogram is used in conjunction with the CSUSUB command. For more information, see **JCL Submit Utility (Mainframe)**, page 751, *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

```
 CTEBYTES            N a t u r a l  C o n s t r u c t            CTEBYTE1
 Aug 14               Driver for subprogram CSUBYTES               1 of 1

Field Format: _        Bytes.......:
Field Length: _____
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                                 mai
```

Driver for Subprogram CSUBYTES Window
Additional Drivers Menu, General Utility Subprograms Menu
Storage Required for a Variable Option

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

```
 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
```

Driver for Subprogram CSUCASE Window
Additional Drivers Menu, General Utility Subprograms Menu
Convert Text to Upper/Lower/Mix Option

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:	See the CSLCCMD local data area for valid command values.
Note:	You cannot use nested command blocks.
Parameters used	CSACCMD CSASTD
Files accessed	None

CSUCENTR Subprogram

CSUCENTR	Description
What it does	Centers a text string. This subprogram centers text, such as headings, in 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

```
 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
```

Driver for Subprogram CSUCENTR Window
Additional Drivers Menu, Text Related Subprograms Menu
Center a Text String Option

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 the following section for a description of this subprogram).
PDAs used	CSACOMPR CSASTD
Files accessed	None

Drivers Menu 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
```

Driver for Subprogram CSUCOMPR Window
Additional Drivers Menu, DB2 Related Subprograms Menu
Generate an IF for a Superdescriptor Option

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.
Parameters 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.
Parameters 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**, page 355.
Parameters 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 inequations.

CSUDB2SP	Description (continued)
PDAs used	CSADB2SP CU--PDA CSASTD
Files accessed	None

Drivers Menu 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
```

Driver for Subprogram CSUDB2SP Window
Additional Drivers Menu, DB2 Related Subprograms Menu
Generate a FIND for a Superdescriptor Option

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

```
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: DEVPR___

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

Driver for Subprogram CSUDELFF Window
Additional Drivers Menu, Edit Buffer Related Subprograms Menu
Delete Lines Containing */ Option

CSUDEFLT Subprogram

CSUDEFLT	Description
What it does	Provides defaults used by Natural Construct models. This subprogram provides an interface between generated applications and the user-maintained CSXDEFLT subprogram. To override default settings, modify CSXDEFLT. The CCDEFLTA, CCDEFLTL, and CCDEFLTN copycode members return defaults for alphanumeric, logical, and numeric values, respectively.
Parameters used	CSADEFLT CSASTD
Files accessed	None

CSUDYNAT Subprogram

CSUDYNAT	Description

What it does	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:
	`#ATTRIBUTE-CHARS(1) = '}'`
	`#ATTRIBUTE-CHARS(2) = '{'`
	This subprogram returns:
	`#DY-PARAMETER = DY={I`
	If the caller's attributes are printable special characters, they are represented literally. Otherwise, they are represented using the HH syntax. Note that programs containing those represented in hex may not be portable.

Note:	The dynamic attribute character specified in position 1, which corresponds to normal intensity, is always coded at the end of the DY= parameter.
PDAs used	CSADYNAT CSASTD
Files accessed	None

Drivers Menu 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
```

Driver for Subprogram CSUDYNAT Window
Additional Drivers Menu, Text Related Subprograms Menu
Build Dynamic Attribute Option

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

```
 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
```

Driver for Subprogram CSUEMLEN Window
Additional Drivers Menu, Text Related Subprograms Menu
Calculate Bytes to Display Emask Option

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. (You do not need to call this subprogram after the last user exit.)
PDAs used	None
Files accessed	None

Drivers Menu 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: DEVPR___

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

Driver for Subprogram CSUENDX Window
Additional Drivers Menu, User Exit Related Subprograms Menu
Generate End to User Exit Option

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

```
 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
```

Driver for Subprogram CSUFDEF Window
Additional Drivers Menu, Validation Subroutines Menu
Validate Field Definition Option

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

```
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
```

Driver for Subprogram CSUFRVAR Window
Additional Drivers Menu, NCST Model/Frame Related Subprograms Menu
Model Code Frame Information Option

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. The specified model PDA must contain the model parameters required for generation.
Note:	This subprogram requires a NATPARM SSIZE of 55 or greater.
Parameters used	CSAGEN CSAMODEL 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

```
 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
```

Driver for Subprogram CSUHEADS Window
Additional Drivers Menu, Text Related Subprograms Menu
Separate a Line of Headings Option

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

```
 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: DEVPR___

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

Driver for Program CSUINCL Window
Additional Drivers Menu, Edit Buffer Related Subprograms Menu
Expand All Copy Codes Option

CSUIS Subprogram

CSUIS	Description
What it does	Checks 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

```
 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
```

Driver for Subprogram CSUIS Window
Additional Drivers Menu, Validation Subroutines Menu
Validate Format for Input Value Option

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 valid, CSASTD.MSG is blank; if the label is not valid, CSASTD.MSG contains an error message.
Parameters used	#PDA-LABEL(A32) CSASTD
Files accessed	None

Drivers Menu 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
```

Driver for Subprogram CSULABEL Window
Additional Drivers Menu, Validation Subroutines Menu,
Validate a Label Option

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

```
 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
```

Driver for Subprogram CSULENGT Window
Additional Drivers Menu, Text Related Subprograms Menu
Calculate Length of a Heading Option

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.
Parameters used	#PDA-TRANSLATION-REQUIRED (L) CSASTD
Files accessed	SYSDIC-FI

CSUMAX Subprogram

CSUMAX	Description
What it does	Generates the assignment of a maximum field value. 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

```
 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
```

Driver for Subprogram CSUMAX Window
Additional Drivers Menu, Edit Buffer Related Subprograms Menu
Generate Assign of Max Field Val Option

CSUMIMAX Subprogram

CSUMIMAX	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

```
 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
```

Driver for Subprogram CSUMIMAX Window
Additional Drivers Menu, Edit Buffer Related Subprograms Menu
Generate Assign of Min Field Val Option

CSUMODEL Subprogram

CSUMODEL	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

```
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
```

Driver for Subprogram CSUMODEL Window
Additional Drivers Menu, NCST Model/Frame Related Subprograms Menu
NCST Models Information Option

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 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:
	
	`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:	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.
Note:	If the value of *Language is not 1 during generation, the word “more” is not included in the initial values.
PDAs used	CSAMORE CSASTD
Files accessed	None

Drivers Menu 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
```

Driver for Subprogram CSUMORE Window
Additional Drivers Menu, Edit Buffer Related Subprograms Menu
Build Left/Right/More Prompt Option

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 checks 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

```
 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
```

Driver for Subprogram CSUMPLAY Window
Additional Drivers Menu, Map Related Subprograms Menu
Load Map Layout and Settings Option

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
```

Driver for Subprogram CSUMPREG Window
Additional Drivers Menu, Map Related Subprograms Menu
Find Available Area in Layout Option

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). 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 map specifications for the map currently in the edit buffer.
PDAs used	CSAMPTST CSASTD
Files accessed	None

Drivers Menu 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....: DEVPR___

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

Driver for Subprogram CSUMPTST Window
Additional Drivers Menu, Map Related Subprograms Menu
Test Map Specifications Option

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 specify:
	`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

```
 CTENATFM            N a t u r a l  C o n s t r u c t            CTENTFM1
 Aug 14               Driver for subprogram CSUNATFM               1 of 1

  Field Format: _         Natural Format:
  Field Length: _____


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

Driver for Subprogram CSUNATFM Window
Additional Drivers Menu, General Utility Subprograms Menu
Build Natural Format Option

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.
PDA used	CSANEWX
Files accessed	None

Drivers Menu 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
```

Driver for Subprogram CSUNEWX Window
Additional Drivers Menu, User Exit Related Subprograms Menu
Generate New User Exit Prompt Option

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, and UL 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

```
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
```

Driver for Subprogram CSUPARMS Window
Additional Drivers Menu, General Utility Subprograms Menu
Get a NATPARM Parameter Value Option

CSUPARTY Subprogram

CSUPARTY	Description
What it does	Identifies 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.
Parameters 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

```
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: DEVPR___

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

Driver for Program CSUPPER Window
Additional Drivers Menu, Edit Buffer Related Subprograms Menu
Convert Source to Upper Case Option

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 used	#READ-THIS-MODULE(A8) CSAMODEL CU--PDA CSASTD
Files accessed	NCST-ADA SYSTEM-FUSER

Note: 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.
Parameters 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.
PDA used	CSASCAN
Files accessed	None

Drivers Menu 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
```

Driver for Subprogram CSUSCAN Window
Additional Drivers Menu, Edit Buffer Related Subprograms Menu
Scan for Existence of a String Option

CSUSELFV Subprogram

CSUSELFV	Description

What it does	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(*)`

PDAs used	CSASELFV CSASTD
Files accessed	None

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

```
CTESETKY            N a t u r a l  C o n s t r u c t            CTESETK1
Sep 01               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
```

Driver for Subprogram CSUSETKY Window
Additional Drivers Menu, General Utility Subprograms Menu
Find PF Key Related Information Option

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:
	`CSASETW.SET-CONTROL.PARM='WBFL70C5'`

PDAs used	CSASETW CSASTD
Files accessed	None

Drivers Menu 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
```

Driver for Subprogram CSUSETW Window
Additional Drivers Menu, Text Related Subprograms Menu
Build Window Settings Option

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. For an example of how to call this subprogram, refer to the CSASORT data area.
Parameters 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

```
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: DEVPR___

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

Driver for Program CSUSPLIT Window
Additional Drivers Menu, Edit Buffer Related Subprograms Menu
Split Long Lines in Edit Buffer Option

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, see **JCL Submit Utility (Mainframe)**, page 751, *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 Subprogram function.
PDAs used	CSASUBP CSASTD
Files accessed	NCST-SUBPROGRAM

Drivers Menu 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
```

Driver for Subprogram CSUSUBP Window
Additional Drivers Menu, NCST Model/Frame Related Subprograms Menu,
NCST Subprograms Information Option

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 **Testing the Model Subprograms**, page 143.
PDAs used	None
File accessed	NCST-SUBPROGRAM NCST-CONTROL

Drivers Menu 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
```

Single Module Test Program Panel
Additional Drivers Menu, NCST Model/Frame Related Subprograms Menu
Test NCST Generated Models Option

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.
Parameters 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. For more information about SYSERR message numbers, see **Using SYSERR References**, page 406. CSUTRANS reads the supplied data structure, searching for one of two message number patterns: *NNNN or *NNNN.A, where *NNNN identifies the message number and .A identifies a position within the message number. If a message number of the type *NNNN is located, the entire SYSERR message is retrieved. If a message number of type *NNNN.A is located, the portion of the message corresponding to the .A notation is retrieved. A message number can have up to 15 positions: the values 1 to 9 represent the first nine positions, and the values A to F represent the 10th to 15th 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. If 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. For more information, see **Formatting SYSERR Message Text**, page 412. CSUTRANS requires a specific data structure. The example on the following page shows the translation LDA for the Standard Parameters panel for the Batch model:

CSUTRANS Description (continued)

```
* * **SAG TRANSLATION LDA
* * * used by CTETRANS.
1 CTE-MAL

  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,+/16'>
    3 #DESCS                        A   20 INIT<'*2001.2,.'>
    3 #GDA-BLOCK                    A   20 INIT<'*2001.3,>'>
    3 #MAP-HEADER1                  A   20 INIT<'*2049.1,./18'>
    3 #MAP-HEADER2                  A   20 INIT<'*2049.2,>/18'>
    3 #USE-MSG-NR                   A   20 INIT<'*2050.1,.'>
    3 #PASSWORD-CHECK               A   20 INIT<'*2050.2,./20'>
  2 TEXT
    3 TRANSLATION-TEXT
      4 TEXT-ARRAY                  A    1 (1:200)
  2 ADDITIONAL-PARMS
    3 #MESSAGE-LIBRARY              A    8 INIT<'CSTLDA'>
    3 #LDA-NAME                     A    8 INIT<'CTE-MAL'>
    3 #TEXT-REQUIRED                L      INIT<TRUE>
    3 #LENGTH-OVERRIDE              I    4  /* Length to translate
```

Other details about the structural elements include:

- 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 CTE-MAL level 1 structure name is typically the LDA name; use this qualifier whenever the variables are accessed.
- The level 3 variables (#GEN-PROGRAM, #SYSTEM, #GDA, etc.) are screen prompts that are initialized with a valid SYSERR number. All SYSERR numbers use the *NNNN.A notation and are listed in sequential order.

Note: This sequence does not apply to positions after the period within the *NNNN.A notation. For example, you can list *2000.2 before *2001.1.

- The TEXT-ARRAY value must match the total number of bytes in all prompt variables to be translated.
- The #MESSAGE-LIBRARY value indicates the SYSERR library where the text is stored.
- The #TEXT-REQUIRED logical indicates whether translation is required, If it is, this field ensures that translation is performed only once.

CSUTRANS	Description (continued)
PDA used	CSATRANS CSASTD
File accessed	SYSTEM-FUSER

Drivers Menu 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
```

Sample Driver for Subprogram CSUTRANS Window
Natural-Related Drivers Menu, Retrieve Application Error Messages
Retrieve Block Messages Option

Note: This driver program is provided as a sample only. Because the screen prompts translated by CSUTRANS vary depending on the application you are developing, 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.
PDA used	CSAXCHK
Files accessed	None

Drivers Menu 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: DEVPR___


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

Driver for Subprogram CSUXCHK Window
Additional Drivers Menu, User Exit Related Subprograms Menu
Validate Format for Input Value Option

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”.
Note:	Use this subprogram when you add characters to a file or field name that is already 32 characters long.
PDA used	CSA2LONG
Files accessed	None

Drivers Menu 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
```

Driver for Subprogram CSU2LONG Window
Additional Drivers Menu, General Utility Subprograms Menu,
Shorten a Long Variable Name Option

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.

Note: 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.

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's description. For more information about the supplied driver programs, see **Drivers Menu Function**, page 66.

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.

Example of using the FILE-CODE field

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.

The following sections describe subprograms that retrieve information from Predict.

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. 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

```
 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..: ___

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

Driver for Subprogram CPU-OBJ Window
Predict-Related Drivers Menu, Generate Data Areas Menu, External Option

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.
Parameters 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. For more information, see CPA-OREL.ENTITY(*).
Parameters 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 logicals 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.
Note:	This subprogram differs from CPU-OBJ in that it generates internal rather than external data structures. 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. For information about INPUT/OUTPUT parameters, refer to the CPA-VIEW data area in the SYSCST library.
PDAs used	CPA-VIEW CSASTD

CPU-VIEW	Description (continued)
Files accessed	SYSDIC-EL SYSDIC-FI

Drivers Menu 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
```

Driver for Subprogram CPU-VIEW Window
Predict-Related Drivers Menu, Generate Data Areas Menu, Internal Option

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.
PDAs used	CPAEL CSASTD
File accessed	SYSDIC-EL

Drivers Menu Option

```
 CTEEL                N a t u r a l  C o n s t r u c t          CTEEL11
 Aug 14                 Driver for subprogram CPUEL               1 of 2

*File Name..: ________________________________ DDM Prefix: _______________
*Field Name : ________________________________
 Simple Outputs: _

Fld Found...:       Adabas Fld Name:       Fld Format....:     Field Uq :
Ver Found...:       Fld Length.....:       Predict Format:     De Type..:
Lvl Number..:       Sign...........:       Suppression...:     Gr Struct:
Occurrences.:       Fld Type.......:       A/Descend.....:     Pe Ind...:
                    Fld Redefined  :       Rank..........:

Edit Mask...:                                          Field Headings:
DDM Fld Name:
Index Name..:
Fld Sequence:

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

Driver for Subprogram CPUEL Window 1
Predict-Related Drivers Menu, Field Information Menu, Single Field Option

```
CTEEL                N a t u r a l  C o n s t r u c t            CTEEL21
Aug 14                  Driver for subprogram CPUEL               2 of 2

File Name..:
Field Name :

LEVEL
-----
DDM Field Name                    Field Type    Is Redefined









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

Driver for Subprogram CPUEL Window 2
Predict-Related Drivers Menu, Field Information Menu, Single Field Option

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.
Parameters 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.
Parameters 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.
Parameters 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. For information about INPUT/OUTPUT parameters, refer to the CPAELNX data area in the SYSCST library. **Note:** CPRELNX contains reserved variables that keep track of the current position; it must not be modified by the calling program.
PDAs used	CPAELNX CPRELNX CSASTD
Files accessed	SYSDIC-EL SYSDIC-FI

Drivers Menu Option

```
 CTEELNX             N a t u r a l  C o n s t r u c t            CTEENX11
 Aug 14                Driver for subprogram CPUELNX                1 of 2

*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
```

Driver for Subprogram CPUELNX Window 1
Predict-Related Drivers Menu, Field Information Menu, Get Next Field Option

```
 CTEELNX              N a t u r a l  C o n s t r u c t            CTEENX21
 Aug 14                 Driver for subprogram CPUELNX               2 of 2

        Field Headings
-------------------------------
                                 IMS Offset....:          Access Lvl:
                                 IMS Fld Name..:          Update Lvl:
                                 IMS Fld Length:

Index Name..:
DDM Fld Name:

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
```

Driver for Subprogram CPUELNX Window 2
Predict-Related Drivers Menu, Field Information Menu, Get Next Field Option

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 (redefined as required) CSASTD
Note:	The CSAPASS parameter data area can be redefined as required and used to store additional information that must be preserved between CALLNATs.
File accessed	SYSDIC-EL

Drivers Menu 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
```

Driver for Subprogram CPUELRD Window
Predict-Related Drivers Menu, Field Information Menu
CALLNAT For Each Field in a File Option

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
File accessed	SYSDIC-EL

Drivers Menu 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
```

Driver for Subprogram CPUELVE Window
Predict-Related Drivers Menu, Field Information Menu
Get Verification Rule Names Option

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

```
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
```

Driver for Subprogram CPUEXIST Window
Predict-Related Drivers Menu, Verify Object Existence Option

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
File accessed	SYSDIC-FI

Drivers Menu 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
```

Driver for Subprogram CPUFI Window
Predict-Related Drivers Menu, File Information Option

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.
Parameters 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.
Parameters 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
File accessed	SYSDIC-EL

Drivers Menu 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
```

Driver for Subprogram CPUREDEF Window
Predict-Related Drivers Menu, Field Information Menu,
Generate Field Redefinition Option

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
File 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
```

Driver for Subprogram CPURL Window
Predict-Related Drivers Menu, Relationship Information Option

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 (redefined as required) CSASTD
Note:	The CSAPASS parameter data area can be redefined as required and used to store additional information that must be preserved between CALLNATs.
Files accessed	SYSDIC-FI SYSDIC-KL

Drivers Menu 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
```

Driver for Subprogram CPURLRD Window
Predict-Related Drivers Menu, CALLNAT for Each Relationship Option

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.

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
File accessed	SYSDIC-EL

Drivers Menu 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
```

Driver for Subprogram CPUSUPER Window
Predict-Related Drivers Menu, Field Information Menu
Expand Superde or Redefine Option

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

```
 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
```

Driver for Subprogram CPUVE Window
Predict-Related Drivers Menu, Field Information Menu
Generate Verification Rules Option

CPUVERUL Subprogram

CPUVERUL	Description
What it does	Returns information about Predict verification rules.
Parameters 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
File accessed	SYSDIC-EL

Drivers Menu 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
```

Driver for Subprogram CPUXPAND Window 1
Predict-Related Drivers Menu, Field Information Menu
CALLNAT For Sup/Subde Components Option

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.

```
 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
```

Driver for Subprogram CPUXPAND Window 2
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            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
```

Driver for Subprogram CPUXPAND Window 3
Predict-Related Drivers Menu, Field Information Menu
CALLNAT For Sup/Subde Components Option

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.

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.
Parameters used	CPAHEL CSASTD
File 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. For information about INPUT/OUTPUT parameters, refer to the CPHELBA data area in the SYSCST library.
PDAs used	CPAHEL CSASTD
File 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.
Parameter used	#PDA-FILE(A32)
Files accessed	SYSDIC-FI

CPHFIB Subprogram

CPHFIB	Description
What it does	Browses Predict views and files for selection.
Parameters used	#PDA-KEY(A32) CSASTD
File 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) #PDA-KEY(A32)

CPHPRED	Description (continued)
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.
Parameter 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.
Parameters 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 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

CU--MSG	Description

What it does	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:
	
	`USE-MSG-NR 1`
	`ASSIGN MSG-INFO.##MSG-NR = 8123 "`
	`ELSE 1`
	`ASSIGN MSG-INFO.##MSG = "`
	`SUBPROGRAM:CU--MSG PARAM: 8123 N "`

Parameters used	CU--PDA CSASTD
File accessed	Application error message file.

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 used	#PDA-UNDERSCORE(A4) #PDA-UNDERSCORE-LINE(A80) CSASTD

SUPPLIED ADMINISTRATION UTILITIES

This chapter describes the utilities supplied with Natural Construct for use in the Administration subsystem.

The following topics are covered:

- **Introduction**, page 392
- **Import and Export Utilities**, page 393
- **Frame Hardcopy Utility**, page 396
- **Comparison Utilities**, page 397
- **Upper Case Translation Utility**, page 401
- **Additional Utilities**, page 402

Introduction

The following sections describe 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).

Mainframe Note:
When a description refers to “your print file,” it refers to Print File 1.

Unix Note:
When a description refers to “your print file,” 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's 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	*Natural Construct Administration and Modeling*
CSHLOAD CSHUNLD	*Natural Construct Help Text*
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 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

Value entered	Results
*	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.

Multiple Code Frame Import Utility

The CSFLOAD frame import utility allows you to import 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.

Enter each code frame name one name at a time. As you enter the names, they are automatically displayed on the panel.

Use the Replace Option field to replace the existing code frames with code frames with the same names in work file 1. To replace the existing code frames, mark the field. If you do not want to replace the existing code frames, leave the field blank.

Examples of input values

Value entered	Results
*	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.
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.
MENU X	Imports the “MENU” code frame from work file 1. If the “MENU” code frame exists in the code frame file, it is replaced.
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.

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

Value entered	Results
*	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

The following sections describe 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 ===> CST411M_ CSGCMPS_   017    029           _
New version ===> CST411M_ 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
```

Compare Criteria Window

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

Using CSGCMPL 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)
```

Source Range Compare Facility Window

The Dominant field 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

Object Type	Description
L	Local data area
M	Map
N	Subprogram
P	Parameter data area
S	Subroutine
T	Text

Using CSGCMPL 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 any 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). This 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. Since this conversion requires a significant amount of processing, only run this utility in a batch environment.

Note: Before running this utility, ensure that the batch job defines the correct Natural Construct logical file, FUSER system file, and FNAT system file.

You can 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 following table describes additional utilities you can use. For more information, see **Utilities**, *Natural Construct Generation.*

Utility	Command	Description
INCLUDE Code Insertion	CSUINCL	Inserts all INCLUDE members into a program. Use this utility during the debugging process to display hidden INCLUDE members.
JCL Submit (mainframe)	CSUSUB	Submits the contents of the source buffer to the internal reader. This utility requires the availability of the NATRJE module.
Multiple Generation	NCSTBGEN	Regenerates multiple Natural modules using their corresponding Natural Construct models.
Upper Case Translation	CSUPPER	Translates the contents of the source buffer into upper case. You can optionally translate any combination of the following elements: comments, statements, or quoted strings.

USING SYSERR FOR MULTILINGUAL SUPPORT

This chapter 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.

The following topics are covered:

- **Introduction**, page 404
- **Defining SYSERR References**, page 405
- **Using SYSERR References**, page 406
- **Supported Areas in Natural Construct**, page 413
- **CSUTRANS Utility**, page 415
- **CNUMSG Utility**, page 419
- **One Language (Static) Mode**, page 421

Introduction

Natural Construct supports the dynamic translation of text and messages on many model 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 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 **One Language (Static) Mode**, page 421.

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.

Defining 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.

Using 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 **Message Numbers**, page 25, *Natural Construct Help Text.*

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, see the Natural Construct 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 Natural Construct’s dedicated library and contains all language-independent prompt text.

For more information, see **CSUTRANS Utility**, page 415.

For Panel Headings and PF-Key Names

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 model modify subprogram to override these defaults. Modify subprograms reference the #HEADER1 and #HEADER2 internal variables to display panel headings. If these headings are not overridden by the model modify 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**, page 189.

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” and the message is “:1::2::3:is required”, the message is displayed as: “Module 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, invokes 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:

```
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
```

Translate Short Message Window

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.

Note: You can also use the context translation mechanism to perform the original translation (instead of mass translation).

Note: 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, invokes 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
```

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 (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**, page 413.

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 the REINPUT statement in the 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
```

Formatting 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 with periods.
+	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”):

Format Specified	Result
*0210.1,+/24	Centers text in 24 bytes. At runtime, text is displayed as: `|  Field Help  |`
*0210.1,>/24	Right-justifies text. At runtime, text is displayed as: `|  Field Help |`
*0210.1,/24 or *0210.1,</24	Left-justifies text (the default). At runtime, text is displayed as: `|Field Help  |`
*0210.1,./24	Left-justifies text and fills the remaining blank spaces with periods. At runtime, text is displayed as: `|Field Help.....|`

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. For information on substitution values, see **With Substitution Values**, page 411. For information on formatting, see **Formatting SYSERR Message Text**, page 412.

Location	Substitutions?	Formatting?
Maintain Control Record panel:		
• PF-key names	No	No
• Indicators	No	No
Maintain Models panel:		
• Description	Yes	No
Maintain Subprogram panel:		
• Description	Yes	No
• PF-key names	No	No
CST-Modify Standard Parameters panel:		
• 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

The following table lists chapters 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	See
Maintain Control Record panel	**Maintain Control Record Function**, page 53
Maintain Models panel	**Maintain Models Function**, page 42
Maintain Subprogram panel	**Maintain Subprograms Function**, page 52
CST-Modify Standard Parameters panel	**Parameters for the CST-Modify Model**, page 194
Translation LDA utilities (CNUMSG and CSUTRANS)	**CNUMSG Subprogram**, page 296, **CNUMSG Utility**, page 419, **CSUTRANS Subprogram**, page 356, **CSUTRANS Utility**, page 415
Help Text editor	**Editing Help Text**, page 47, *Natural Construct 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 retrieve a valid message, you must also specify the SYSERR library name (CSTLDA, by default). To change the library name, use the #MESSAGE-LIBRARY variable.

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 example on the following page 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            I    4 /* Explicit length to translate
```

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.
Byte 6	Can be a period (.), which indicates that the next byte is a position value.

Position	Format (continued)
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. 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.
Note:	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 >)
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), 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.

The CNUMSG utility retrieves message text in one of two ways. If a reference number is specified (CNAMSG.MSG-NR), CNUMSG uses that number to retrieve the SYSERR message text. If a reference number is not specified, CNUMSG checks the message text (CNAMSG.MSG) for the *NNNN or *NNNN.A notation and uses the specified notation to retrieve the SYSERR message text.

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 if 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.

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 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).

To perform a desired function, CNUMSG can also be called with a method. Natural Construct supports the following methods:

Method	Description
R	Retrieves the SYSERR message as is without any text substitution. This method is good 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).
S	Substitutes the data into the :1::2::3: place holders without retrieving the main message text. For example, you may have a text string created programmatically to which you want to apply substitutions. This method only substitutes the available (passed) data into the place holders. Unused place holders are removed.
B	Retrieves the message text and performs the substitutions. This is the most commonly used method and is the default setting when the method is blank.
blank	If the method is blank, the default is B.

All other method settings will return a fatal error without performing any actions.

One Language (Static) Mode

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.

Before using the Static option, check with your local Software AG office to ensure that your language is supported. English is always supported.)

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 subprograms and static text LDAs. For information, see the Natural Construct installation documentation.
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).

Natural Construct can also use the English text supplied with each INCLUDE code member and bypass the SYSERR retrieval process (see **In Messages**, page 407).

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 for you to copy and use in INCLUDE statements.
Cursor-sensitive or Cursor sensitivity	Created by moving the cursor at an item and pressing Enter. If you are using 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 stores parameters that are passed between subprograms, and a global data area stores data that is used by all programs in 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 add 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**.

Term	Definition (continued)
Mark a field	Type any non-blank character in the field.
Note:	You may also be required to press the Enter key.
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.
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, for example, 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 you enter 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 always have the same format and different values at generation time.

Term	Definition (continued)
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 generated functions for their own requirements. User exit code is preserved when the program is regenerated.
Utility	Program supplied to perform 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).

INDEX

Symbols

#HEADER1
internal variable, 407
#HEADER2
internal variable, 407
#PDA variables
assigning, 210
#PDA-AVAILABLEn fields
CU--PDA parameter data area, 125
#PDA-AVAILABLEn-NAME fields
CU--PDA parameter data area, 124
#PDA-BACKWARD field
CU--PDA parameter data area, 125
#PDA-BACKWARD-NAME field
CU--PDA parameter data area, 124
#PDA-FORWARD field
CU--PDA parameter data area, 125
#PDA-FORWARD-NAME field
CU--PDA parameter data area, 124
#PDA-FRAME-PARM field
invoking generation subprograms, 136
#PDA-HELP field
CU--PDA parameter data area, 125
#PDA-HELP-NAME field
CU--PDA parameter data area, 124
#PDA-LEFT field
CU--PDA parameter data area, 125
#PDA-LEFT-MORE-PROMPT field
CU--PDA parameter data area, 123
#PDA-LEFT-NAME field
CU--PDA parameter data area, 124
#PDA-MAIN field
CU--PDA parameter data area, 125
#PDA-MAIN-NAME field
CU--PDA parameter data area, 124
#PDA-PF-AVAILABLE fields (1/2/3)
CU--PDA parameter data area, 126
#PDA-PF-BACKWARD field
CU--PDA parameter data area, 126
#PDA-PF-FORWARD field
CU--PDA parameter data area, 126
#PDA-PF-HELP field
CU--PDA parameter data area, 126
#PDA-PF-LEFT field
CU--PDA parameter data area, 126
#PDA-PF-MAIN field
CU--PDA parameter data area, 126
#PDA-PF-QUIT field
CU--PDA parameter data area, 126
#PDA-PF-RETURN field
CU--PDA parameter data area, 126
#PDA-PF-RIGHT field
CU--PDA parameter data area, 126
#PDA-PF-TEST field
CU--PDA parameter data area, 126
#PDA-QUIT field
CU--PDA parameter data area, 125
#PDA-QUIT-NAME field
CU--PDA parameter data area, 124
#PDA-RETURN field
CU--PDA parameter data area, 125
#PDA-RETURN-NAME field
CU--PDA parameter data area, 124
#PDA-RIGHT field
CU--PDA parameter data area, 125
#PDA-RIGHT-MORE-PROMPT field
CU--PDA parameter data area, 123
#PDA-RIGHT-NAME field
CU--PDA parameter data area, 124
#PDA-TEST field
CU--PDA parameter data area, 125
#PDA-TEST-NAME field
CU--PDA parameter data area, 124
& (ampersand) character
identifying substitution parameters, 104

* (asterisk)
displaying help, 35
* edit command
Code Frame editor, 91
*Language hierarchy
defining for CSUTRANS utility, 132, 416
*Language variable
implementing multilingual support, 67, 404
++ edit command
Code Frame editor, 92
+H edit command
Code Frame editor, 92
+nnnn edit command
Code Frame editor, 92
+P edit command
Code Frame editor, 92
. (period)
terminating Natural Construct, 33
. edit command
Code Frame editor, 90
.C(nn) line command
Code Frame editor, 88
.CX(nn) line command
Code Frame editor, 88
.CX-Y(nn) line command
Code Frame editor, 88
.CY(nn) line command
Code Frame editor, 88
.D(nn) line command
Code Frame editor, 88
.E line command
Code Frame editor, 88
.G line command
Code Frame editor, 88
.I line command
Code Frame editor, 88–89
.IF line command
Code Frame editor, 89
.L line command
Code Frame editor, 89
.MX line command
Code Frame editor, 89
.MX-Y line command
Code Frame editor, 89
.MY line command
Code Frame editor, 89
.N line command
Code Frame editor, 89, 92
.P line command
Code Frame editor, 89
.W(nn) line command
Code Frame editor, 89
.X line command
Code Frame editor, 89
.Y line command
Code Frame editor, 89
> prompt (command line)
Code Frame editor, 84
? (question mark)
displaying help, 33
indicating nested code frames, 105
used in code frame names, 47

A

ABS field
Code Frame editor, 85
Act field
Frame Compare Facility panel, 64
Add
message text, 37
ADD edit command
Code Frame editor, 90
Additional Parameters panel
CST-Document model, 226
ADDITIONAL-INITIALIZATIONS user exit
description, 249
example, 249
ADDITIONAL-SUBSTITUTION-VALUES user exit
description, 250
example, 250
ADDITIONAL-TRANSLATIONS user exits
description, 251
example, 251

Administration main menu
  displaying
    PF12 (main), 33
  Drivers Menu function, 281
  invoking
    translation mode, 28
  translation mode, 67
Administration subsystem
  invoking
    translation mode, 28
AFTER-INPUT user exit
  description, 252
AFTER-INVOKE-SUBPANELS user exit
  description, 252
  example, 252
Alphanumeric fields
  changing format and length, 328
  converting to another format/length, 328
  CST-PDA model, 178
Ampersand (&) character
  identifying substitution parameters, 104
Application library
  CSTAPPL, 273
Array fields
  validating, 234
Arrays
  sorting 2-dimensional, 351
ASSIGN-DERIVED-VALUES user exit
  description, 253
  example, 253
Asterisk (*)
  displaying help, 35
  exporting a range of code frames, 394
  loading a range of frames, 395
  printing a range of code frames, 396
Attributes
  returning description, 367
Auto save numbers field
  Maintain Current Editor Profile window, 93–94

B

B type line
  Code Frame editor, 86
BEFORE-CHECK-ERROR user exit
  description, 254
  example, 254
BEFORE-INPUT user exit
  description, 254
BEFORE-INVOKE-SUBPANELS user exit
  description, 255
  example, 255
BEFORE-REINPUT-MESSAGE user exit
  description, 255
  example, 255
BEFORE-STANDARD-KEY-CHECK user exit
  description, 256
  example, 256
Blank line type
  Code Frame editor, 87
BOTTOM edit command
  Code Frame editor, 92
Browse a file
  glossary definition, 423
Browse program
  glossary definition, 423

C

C field
  Code Frame editor, 87
Callback functions, 283
CALLNAT
  issuing for each field in Predict file, 371
  issuing to #CALLNAT subprogram, 363
Case
  converting source buffer into upper, 344
  converting text string to another, 311
CHANGE edit command
  Code Frame editor, 90
  used with ABS field, 85
CHANGE-HISTORY user exit
  description, 256
  example, 256

Checks
generating referential integrity, 346
CLEAR edit command
Code Frame editor, 90
Clear specification field
Maintain Models panel, 45
Clear subprogram
creating for a new model, 168
defining model defaults, 75
description, 139
example, 140, 180
generating, 179–180
PROVIDE-DEFAULT-VALUES user exit, 76
when invoked, 139
CNAEL parameter data area
CNUEL subprogram, 287
CNAEL.INPUTS field
CNUEL subprogram, 287
CNAEL.OUTPUTS field
CNUEL subprogram, 287
CNAELNX parameter data area
CNUELNX subprogram, 288
CNAEXIST parameter data area
CNUEXIST subprogram, 291
CNAGDABL parameter data area
CNUGDABL subprogram, 292
CNAGDAEL parameter data area
CNUGDAEL subprogram, 293
CNAGENDA parameter data area
CNUGENDA subprogram, 294
CNAMSG local data area
defining *Language hierarchy, 132, 416
CNAMSG parameter data area
CNUMSG subprogram, 298
CNAPEXST parameter data area
CNUPEXST subprogram, 299
CNASEL parameter data area
CNUSEL subprogram, 300
CNASRCNX parameter data area
CNUSRCNX subprogram, 301
CNASRCRD parameter data area
CNUSRCRD subprogram, 302
CNHMDL helproutine
browsing models for selection, 304
CNHMSG helproutine
displaying application message text, 305
CNHOBJ helproutine
browsing objects in current library, 305
CNRELNX parameter data area
CNUELNX subprogram, 288
CNRGENDA parameter data area
CNUGENDA subprogram, 294
CNRSRCNX parameter data area
CNUSRCNX subprogram, 301
CNUGDAEL subprogram
verifying field in GDA, 293
CNUGENDA subprogram
adding field to data area, 294
CNUMPPRF subprogram
reading map profile from system file, 295
CNUMSG subprogram
returning application message text, 296, 419
CNUMSG utility
retrieving text for one message, 419
CNUPEXST subprogram
checking for existence of map profile, 299
CNUSEL subprogram
selecting fields from data areas, 300
CNUSRCNX subprogram
returning next line of source, 301
CNUSRCRD subprogram
reading source text, 302
Code field
Single Module Test Program panel, 143
Code frame
glossary definition, 423
Code Frame Compare Utility panel
PF5 (top), 63
PF6 (hcopy), 63
Code frame conditions
example, 109, 111
Code Frame editor
adding a user exit point, 107
description, 38, 84
edit commands, 90
edit commands not supported, 50
line commands, 88

order of command execution, 87
passing DEFINE and INIT parameters
example, 137
scrolling, 92
valid condition levels, 87
Code Frame field
Code Frame editor, 84
Code Frame menu
Clear Edit Buffer function, 52
description, 49
Print Saved Code Frame function, 52
Purge Code Frame function, 52
Code frame name field
Maintain User Exit window, 108
Code frame(s) field
Maintain Models panel, 44
Code frames
adding a user exit point, 106–107
comparing
Compare Frames panel, 61
comparing different, 61
comparing in different files, 63
comparing in two models, 65
creating, 50
conditions for a new model, 157
for a new model, 156
editing, 50
exporting, 394
exporting nested, 394
importing, 395
modifying, 270
nested, 47
example, 106
line types, 86
printing hardcopy, 396
purging from file, 52
retrieving model parameters and conditions, 325
supplying parameters
generation subprograms, 104
nested code frames, 105
supplying values for substitution parameters, 214
Column field
Window Parameters window, 199
Command blocks
generating delimiters for, 312
Command line (> prompt)
Code Frame editor, 84, 90
Comment end indicator field
Maintain Models panel, 44
Comment start indicator field
Maintain Models panel, 44
Comments
generating into source buffer, 309
Compare
models, 57
Compare Code Frames function
Compare Frames panel, 61
Compare Criteria window
description, 397
Compare Frames panel
description, 61
Com-Plete
printing code frames, 52
Com-Pose
printing code frames, 52
Compound keys
generating redefinitions, 375
Condition codes field
Optional Parameters window, 147
Condition levels
Code Frame editor
valid, 87
Conditional field
Maintain User Exit window, 108
User Exits panel, 248
Conditions
assigning true values
CST-Pregen model, 210
code frame
example, 111
generating initialization statements, 265
setting in model PDA, 265
Constant
glossary definition, 423
Context translation mode
description, 410
Control record
retrieving information from, 315
returning PF-key definitions, 349
setting PF-keys, 315
Control screen field
Window Parameters window, 200
Conventions
naming model subprograms, 113
used in this documentation, 21

Convert Text to Upper/Lower/Mix option
General Utility Subprograms submenu, 311
COPY LINE command
Code Frame editor, 88
Copycode
glossary definition, 423
modifying CC*, 273
Corporate defaults
assigning, 78
Courses
related Natural Construct, 23
CPAEL parameter data area
CPUEL subprogram, 366
example, 280
CPAELNX parameter data area
CPUELNX subprogram, 369
CPAELRD parameter data area
CPUELRD subprogram, 370
CPAELVE parameter data area
CPUELVE subprogram, 372
CPAEXIST parameter data area
CPUEXIST subprogram, 373
CPAFI parameter data area
CPUFI subprogram, 374
CPAHEL parameter data area
CPHEL subprogram, 384
CPHELB subprogram, 384
CPA-OBJ parameter data area
CPU-OBJ subprogram, 362
CPAREDEF parameter data area
CPUREDEF subprogram, 375
CPARL parameter data area
CPURL subprogram, 376
CPARLRD parameter data area
CPURLRD subprogram, 377
CPASUPER parameter data area
CPUSUPER subprogram, 378
CPAVE parameter data area
CPUVE subprogram, 379–380
CPA-VIEW parameter data area
CPU-VIEW subprogram, 364
CPAXPAND parameter data area
CPUXPAND subprogram, 381
CPHEL subprogram
browsing fields in file for selection, 384
CPHELB subprogram
browsing fields in file for selection, 384
CPHFI helproutine
browsing views/files for selection, 385
CPHFIB subprogram
browsing views/files for selection, 385
CPHPRED helproutine
browsing Predict objects for selection, 385
CPHRL helproutine
browsing relationships for selection, 386
CPHSET helproutine
setting flag for help field, 387
CPRELNX parameter data area
CPUELNX subprogram, 369
CPUEL subprogram
example, 282
FILE-CODE field, 361
returning Predict field information, 366
CPUELDE subprogram
returning description attributes, 367
CPUELKY subprogram
returning keywords, 368
CPUELNX subprogram
no subprogram chaining, 284
returning field information, 369
CPUELRD subprogram
calling subprograms for fields, 370
subprogram chaining, 285
CPUELVE subprogram
returning type N verification rules, 372
CPUEXIST subprogram
verifying existence of Predict object, 373
CPUFI subprogram
returning Predict file information, 374
CPU-FREL subprogram
retrieving information about foreign relationship, 368
CPUHOLD subprogram
finding hold field, 375
CPUKY subprogram
retrieving Predict keyword information, 375
CPU-OBJ subprogram
generating external data area, 362

CPU-OBJ2 subprogram
issuing CALLNAT to #CALLNAT subprogram, 363
CPU-OREL subprogram
adding entity information to table, 364
CPUREDEF subprogram
generating field redefinitions, 375
CPURL subprogram
returning relationship information, 376
CPURLRD subprogram
retrieving relationships for specified files, 377
CPUSUPER subprogram
returning DB2 compound key definition, 378
returning super/subdescriptor definitions, 378
CPUVE subprogram
printing verification rules to source buffer, 380
CPUVERUL subprogram
returning verification rule information, 381
CPU-VIEW subprogram
generating field definitions, 364
CPUXPAND subprogram
expanding super/subdescriptors/ redefined fields, 381
Creating model subprograms
description, 134
Creating new models
Natural Construct, 99
CSA2LONG parameter data area
CSU2LONG subprogram, 360
CSABLDRP parameter
CSUBLDRP subprogram, 309
CSABYTES parameter data area
CSUBYTES subprogram, 310
CSACASE parameter data area
CSUCASE subprogram, 311
CSACENTR parameter data area
CSUCENTR subprogram, 313
CSACOMPR parameter data area
CSUCOMPR subprogram, 314
CSADB2SP parameter data area
CSUDB2SP subprogram, 317
CSADYNA.#ATTRIBUTE-CHARS field
CSUDYNAT subprogram, 320
CSADYNAT parameter data area
CSUDYNAT subprogram, 320
CSAEMLEN parameter data area
CSUEMLEN subprogram, 322
CSAFDEF parameter data area
CSUFDEF subprogram, 324
CSAFDEF. INVALID FORMATS field
CSUFDEF subprogram, 324
CSAFRVAR parameter data area
CSUFRVAR subprogram, 325
CSAHEADS parameter data area
CSUHEADS subprogram, 326
CSAIS parameter data area
CSUIS subprogram, 328
CSALENGT parameter data area
CSULENGT subprogram, 330
CSAMAX parameter data area
CSUMAX subprogram, 331
CSAMIMAX parameter data area
CSUMIMAX subprogram, 332
CSAMODEL parameter data area
CSUMODEL subprogram, 333
CSAMORE parameter data area
CSUMORE subprogram, 334
CSAMPBOX parameter data area
CSUMPBOX subprogram, 335
CSAMPFLD parameter data area
CSUMPDUP subprogram, 336
CSUMPOVL subprogram, 338
CSUMPTAB subprogram, 340
CSAMPFLD.FIELD-INFO(*) structure
checking for duplicate fields on map, 336
CSAMPOUT parameter data area
CSUMPMMS subprogram, 338
CSAMPREG parameter data area
CSUMPREG subprogram, 339
CSAMPSET parameter data area
CNUMPPRF subprogram, 295
CSUMPCPR subprogram, 336
CSUMPLAY subprogram, 337
CSUMPMMS subprogram, 338
CSAMPTST parameter data area
CSUMPTST subprogram, 340

CSAMPX-Y parameter data area
CSUMPTAB subprogram, 340
CSANATFM parameter data area
CSUNATFM subprogram, 341
CSANEWX parameter data area
CSUNEWX subprogram, 342
CSAPARMS parameter data area
CSUPARMS subprogram, 343
CSAPASS parameter data area
CNUSRCRD subprogram, 302
CPUELRD subprogram, 370
CPURLRD subprogram, 377
CPUXPAND subprogram, 381
CSASCAN parameter data area
CSUSCAN subprogram, 347
CSASELFV parameter data area
CSUBLDRP subprogram, 309
CSUSELFV subprogram, 348
CSASETKY parameter data area
CSUSETKY subprogram, 349
CSASETKY. PF-NAME(*) array
returning PF-key names, 349
CSASETW parameter data area
CSUSETW subprogram, 350
CSASTD parameter data area
CNUEL subprogram, 287
CPHEL subprogram, 384
CPHELB subprogram, 384
CPHFIB subprogram, 385
CSULABEL subprogram, 329
CSU-VAR subprogram, 308
processing errors, 282
required fields, 130
CSASTD.ERROR-FIELD field
CSUFDEF subprogram, 324
CSASTD.MSG field
CSUFDEF subprogram, 324
CSUIS subprogram, 328
CSULABLE subprogram, 329
CSASUBP parameter data area
CSUSUBP subprogram, 353
CSATRANS parameter data area
CSUTRANS subprogram, 358
CSAXCHK parameter data area
CSUXCHK subprogram, 359
CSFHCOPY utility
printing hardcopy of code frames, 396
CSFLOAD utility
loading code frames, 393, 395
CSFUNLD utility
unloading code frames, 393–394
CSGCMPL utility
Source Range Compare Facility
window, 398
using in batch, 399
using online, 398
CSGCMPS utility
Compare Criteria window, 397
comparing two Natural source modules, 397
CSHLOAD utility
loading help modules, 393
CSHUNLD utility
unloading help modules, 393
CSLRCODE LDA
valid codes for RETURN-CODE field, 130
CST-Clear model
description, 179–180
example, 180
parameters, 181
Standard Parameters panel, 181
CST-Document model
description, 224
example, 224
parameters, 225
Standard Parameters panel, 225
user exits, 227
CST-Frame model
generation subprogram
example, 219
parameters, 220
Standard Parameters panel, 220
user exits, 221
CSTG command
invoking Natural Construct from a steplib, 277
CSTLDA library
SYSERR
description, 406

CST-Modify model
description, 193
example, 193
parameters, 194
Standard Parameters panel, 195
supporting dynamic translation, 194
user exits, 202
CST-Modify-332 model
description, 203
Standard Parameters panel, 205
example, 205
user exits, 207
CSTMSG library
SYSERR
description, 407
storing messages, 407
CST-Postgen model
description, 214
example, 214
parameters, 215
Standard Parameters panel, 215
SUBSTITUTION-VALUES user exit, 266
user exits, 216
CST-Pregen model
description, 210
example, 210
parameters, 211
Standard Parameters panel, 211
user exits, 212
CST-Read model
example, 184
parameters, 185
reading input parameters, 259
Standard Parameters panel, 185
CST-Save model
example, 188
parameters, 189
Standard Parameters panel, 189
CST-Shell model
description, 240
example, 240
parameters, 241
Standard Parameters panel, 241
CST-Stream model
description, 236
parameters, 237
Standard Parameters panel, 237
user exits, 238
CST-Validate model
CHECK-AFTER-EDIT subroutine, 234
coding validations, 233
description, 230
GENERATE-SUBROUTINES user exit, 233
parameters, 231
Standard Parameters panel, 231
user exits, 233
validating array fields, 234
CSU2LONG subprogram
converting variable name to abbreviation, 360
CSUBANN subprogram
generating standard banner, 309
CSUBLDRP subprogram
building report layout, 309
CSUBMIT subprogram
submitting JCL job (mainframe), 310
CSUBYTES subprogram
calculating required bytes for field, 310
CSUCCMD subprogram
multiple-module generation utility, 312
CSUCTRL subprogram
retrieving information from Control record, 315
CSUCURS subprogram
determining cursor position, 315
CSUCURS1 subprogram
determining cursor position, 316
CSUDB2SP subprogram
generating FIND statement for superdescriptor, 316
CSUDEFLT subprogram
providing model defaults, 319
supplying default model parameters, 76
CSUDELFF subprogram
deleting lines containing ‘*/’, 318
CSUDYNAT subprogram
building parameters containing dynamic attributes, 320
CSUEMLEN subprogram
number of characters to display edit mask, 322
CSUENDX subprogram
generating end of user exit prompt, 323
CSUFDEF subprogram
validating field definition, 324

CSUFRVAR subprogram
return parameters and conditions from model, 325
CSUGEN subprogram
issuing CALLNAT to Generate function, 326
CSUHEADS subprogram
separating line of headings, 326
CSUINCL program
expanding copycodes in edit buffer, 327
CSUINCL utility
inserting INCLUDE members, 402
CSUIS subprogram
checking contents of alphanumeric field, 328
CSULABEL subprogram
verifying a Natural looping label, 329
CSULENGT subprogram
building input prompts, 330
calculating length of heading, 330
CSULPS subprogram
changing languages, 331
CSUMAX subprogram
assigning maximum field value, 331
CSUMIMAX subprogram
assigning minimum field value, 332
CSUMODEL subprogram
returning information about Natural Construct mode, 333
CSUMORE subprogram
initial value for LEFT-MORE/RIGHT-MORE array, 334
CSUMPBOX subprogram
handling map edit buffer, 335
CSUMPCPR subprogram
replacing map settings, 336
CSUMPDUP subprogram
checking for duplicate fields on map, 336
CSUMPLAY subprogram
loading map layout, 337
returning map settings, 337
CSUMPMMS subprogram
merging two map settings, 338
CSUMPOVL subprogram
checking for overlapping fields on map, 338
CSUMPREG subprogram
finding available area on map, 339
CSUMPTAB subprogram
calculating abs field coordinates on map, 340
creating field prompts on map, 340
CSUMPTST subprogram
testing map specifications, 340
CSUNATFM subprogram
building valid Natural format definition, 341
CSUNEWX subprogram
generating new user exit prompt, 342
CSUPARMS subprogram
returning NATPARM value, 343
CSUPARTY subprogram
returning byte length of Natural data types, 344
CSUPPER program
converting source buffer to upper case, 344
CSUPPER utility
upper case translation, 402
CSUREADS subprogram
reading specification parameters, 345
CSUREF subprogram
generating referential integrity checks against foreign files, 346
CSUSCAN subprogram
scanning edit buffer for string, 347
CSUSELFV subprogram
selecting fields from views, LDAs, or PDAs, 348
CSUSETKY subprogram
returning PF-key definitions, 349
CSUSETW subprogram
separating line of headings, 350
CSUSORT subprogram
sorting 2-dimensional array, 351
CSUSPLIT program
splitting source buffer lines, 351
CSUSUB command
CSUSUB program, 352

CSUSUB program
submitting JCL job (mainframe), 352
submitting source buffer to internal reader (mainframe), 402
CSUSUBP subprogram
returning information about subprograms, 353
CSUTEST command
Single Module Test Program panel, 143
CSUTEST program
testing model subprograms, 354
CSUTLATE subprogram
translating runtime messages, 355
CSUTRANS subprogram
returning application message text, 356
CSUTRANS translation utility
creating translation LDAs, 132, 415
defining *Language hierarchy, 132, 416
description, 415
CSU-VAR subprogram
validating variable name, 308
CSUXCHK subprogram
scanning for user exits, 359
CSXCNAME subprogram
implementing security, 74
supplying default model parameters, 78
CSXDEFLT subprogram
defining model defaults, 76
modifying default model parameters, 77
supplying default model parameters, 76
CSXDUEXT subprogram
implementing security, 74
CSXFUEXT subprogram
implementing security, 74
CSXHUEXT subprogram
implementing security, 74
CSXMUEXT subprogram
implementing security, 74
CSXPSCHG subprogram
implementing security, 74
CSXSUEXT subprogram
implementing security, 74
CUBAMAL translation LDA
example, 131
CU--EM subprogram
returning edit masks for date/time fields, 388
CU--LRP subprogram
returning prompts for panels, 388
CU--MSG subprogram
returning text for application error messages, 389
CU--PDA parameter data area
assigning PDA-OBJECT-TYPE parameter, 44
CNUSRCRD subprogram, 302
CPUELRD subprogram, 370
CPURLRD subprogram, 377
CPUXPAND subprogram, 381
CSUDB2SP subprogram, 317
CU--QUOT
changing single quotation marks to double, 148
Cursor
determining field position, 315–316
Cursor translation field
Standard Parameters panel
CST-Modify model, 196
Cursor-sensitive
glossary definition, 423
Customize
models, 269
CU--UL subprogram
returning underscore lines for panels, 389
CU--VERR copycode
validating array fields, 234
CU--VERZ copycode
validation subprograms, 234
CU--VWAR copycode
return warning message, 234
CVUPPERC utility
converting Natural Construct components to upper case, 401
CX prefix
modifying subprograms and code frames, 270

D

Data area
glossary definition, 423

Data areas
generating external, 362
generating internal, 364
retrieving information about next field, 289
returning information about next field, 288
Data scrolling
backward with PF7 (bkwrd), 32
forward with PF8 (frwrd), 32
Database field
Model Compare Utility window, 58
Date field
Select Frames window, 51
DB2 compound key
expanding, 381
Debugging
inserting INCLUDE code, 402
DEFAULT keyword
defining model defaults, 75
modifying model defaults, 76
Default language
for Natural Construct, 404
Default return field
Maintain Control Record panel, 56
Default user exit code field
Maintain User Exit window, 109
Defaults
assigning corporate, 78
defining model, 75–76
modifying model, 75
supplying logical hold field, 80
supplying object description, 80
supplying primary key, 79
DEFINE parameter
Code Frame editor, 137
DEFINE PRINTER(SRC=1) OUTPUT 'SOURCE' statement
writing to the edit buffer, 219
DEFINE PRINTER(SRC=1) OUTPUT ’SOURCE’ statement
writing to edit buffer, 136
Definitions
validating field, 324
Delete Lines Containing ‘*/’ option
Edit Buffer Related Subprograms submenu, 318
DEPTH-KEYS condition
modifying code frames, 275
DESCRIBE-INPUTS user exit
description, 256
example, 256
Description field
Additional Parameters panel
CST-Document model, 227
Code Frame editor, 84
Maintain Models panel, 43
Select Frames window, 51
Standard Parameters panel
CST-Clear model, 182
CST-Document model, 226
CST-Frame model, 221
CST-Modify-332 model, 206
CST-Postgen model, 216
CST-Pregen model, 212
CST-Shell model, 242
CST-Stream model, 238
CST-Validate model, 232
Direct command
direct command line, 37
sample, 38
Direct command box
specifying direct commands (Unix), 38
Direction indicator
Code Frame editor, 85, 88
Disable indicator field
Maintain Control Record panel, 55
Display help
PF1 (help), 32
DISPLAY statement
NOHDR option, 136, 219
NOTITLE option, 136, 219
Document in Predict field
Optional Parameters window, 147
Document specification field
Maintain Models panel, 46
Document subprogram
creating for a new model, 172
description, 142, 224
example, 142
Documentation
related Natural Construct, 22
Drivers Menu function
description, 66

Drivers Menu options
CNUELNX, 289
CNUERMSG, 290
CNUEXIST, 291
CNUGDABL, 292
CNUGDAEL, 293
CNUGENDA, 294
CNUMPPRF, 295
CNUMSG, 298
CNUPEXST, 299
CNUSEL, 300
CNUSRCNXA, 301
CNUSRCRD, 302
CPUEL, 366
CPUELNX, 369
CPUELRD, 371
CPUELVE, 372
CPUEXIST, 373
CPUFI, 374
CPU-OBJ, 362
CPUREDEF, 376
CPURL, 377
CPURLRD, 378
CPUSUPER, 379
CPUVE, 380
CPU-VIEW, 365
CPUXPAND, 382
CSU2LONG, 360
CSUCASE, 311
CSUCENTR, 313
CSUCOMPR, 314
CSUDB2SP, 317
CSUDELFF, 318
CSUDYNAT, 321
CSUEMLEN, 322
CSUENDX, 323
CSUFDEF, 324
CSUFRVAR, 325
CSUHEADS, 327
CSUINCL, 327
CSUIS, 328
CSULABEL, 329
CSULENGT, 330
CSUMAX, 331
CSUMIMAX, 332
CSUMODEL, 333
CSUMORE, 335
CSUMPLAY, 337
CSUMPREG, 339
CSUMPTST, 340
CSUNATFM, 341
CSUNEWX, 342
CSUPARMS, 343
CSUPPER, 344
CSUSCAN, 347
CSUSETKY, 349
CSUSETW, 350
CSUSPLIT, 352
CSUSUBP, 353
CSUTEST, 354
CSUTRANS, 358
CSU-VAR, 308
CSUXCHK, 359
CSYBYTES, 310
DX edit command
Code Frame editor, 90
DX-Y edit command
Code Frame editor, 90
DY edit command
Code Frame editor, 90
Dynamic attributes
building DY parameter, 320
Dynamic Attributes fields
Maintain Control Record panel, 55
Dynamic translation
modify subprograms, 194
using CSUTRANS, 415

E

Edit buffer
expanding copycode, 327
saving current code frame, 50
scanning for existence of text strings, 347
scanning for user exits, 359
splitting long lines, 352
Edit Code Frame function
Administration main menu, 50
Edit commands
Code Frame editor, 90
positional, 92
Edit masks
writing information without embedded spaces, 148
Editors
Code Frame, 84
Embedded statements field
Optional Parameters window, 147

END edit command
Code Frame editor, 90
END-OF-PROGRAM user exit
description, 257
Entering a value
glossary definition, 423
Entity information
adding to table, 364
Error messages
multilingual support for, 404
ERROR-FIELD-INDEX fields (1/2/3)
CSASTD parameter data area, 131
Errors
processing, 282
return warning message instead, 234
EX edit command
Code Frame editor, 90
Execute
glossary definition, 423
Exists field
User Exits panel, 248
Export utility
code frame, 394
description, 393
External data areas
generating, 362
modifying, 273
External parameter data areas
description, 280
EX-Y edit command
Code Frame editor, 90
EY edit command
Code Frame editor, 90

F

F line type
Code Frame editor, 86
Field
glossary definition, 423
Field names
changing dynamically, 71, 404
maintaining consistency, 404
Field-level help
active, 35
description, 35
passive, 35
Fields
adding to a data area, 294
assigning maximum values, 331
assigning minimum values, 332
CALLNAT subprogram for each, 371
defining a logical hold, 375
determining cursor position, 315–316
expanding redefined, 381–382
generating redefinitions, 375–376
retrieving information about, 287
retrieving information about next, 369
retrieving information from Predict about, 366
retrieving verification rule names, 372
returning Predict information about, 366, 369
supplying default alphanumeric values, 77
supplying default logical values, 77
supplying default numeric values, 77
validating definitions for, 324
File field
Model Compare Utility window, 58
FILE-CODE field
Natural Security, 361
Files
restricting access to Predict, 361
Find Available Area in Layout option
Map Related Subprograms submenu, 339
Find PF Key Related Information option
General Utility Subprograms submenu, 349
FIND statement
generating for superdescriptor, 317
First header field
Standard Parameters panel
CST-Modify model, 197
FNAT system file
copying modified modules, 272
Foreign files
generating referential integrity checks against, 346

Foreign relationships
retrieving information about, 368
Formatting
supported for SYSERR message text, 413
SYSERR message text, 412
Frame Compare Facility panel
description, 64
Frame field
Select Frames window, 51
Frame OFF field
Window Parameters window, 200
Frames
exporting, 394
importing, 395
FREE field
Code Frame editor, 84
Function
glossary definition, 423
Function field
Code Frame menu, 82
Single Module Test Program panel, 143

G

Generate as subroutine field
Maintain User Exit window, 108
Generate function
issuing CALLNAT to, 326
Generate trace code field
Standard Parameters panel
CST-Stream model, 238
GENERATE-CODE user exit
description, 258
example, 258
GENERATE-SUBROUTINES user exit
CST-Validate model, 233
description, 259
GENERATE-VALIDATIONS user exit
description, 259
Generating multiple modules
utility, 402
Generation main menu
PF5 (optns), 146
Generation options
Optional Parameters window, 146
Generation subprograms
creating for a new model, 171
description, 136
example, 137
NOTITLE and WRITE TITLE ' '
statements, 148
NOTITLE or WRITE TITLE ' '
statements, 148
PDA-FRAME-PARM field, 136, 219
supplying code frame parameters, 104
Global data area field
Standard Parameters panel
CST-Shell model, 242
Global data areas (GDAs)
defining
CST-Shell model, 240
Glossary of terms
used in this documentation, 423
GUI
sample subprogram
user exit examples, 94
GUI sample subprogram
Maintain User Exit window, 108

H

-H edit command
Code Frame editor, 92
Height field
Window Parameters window, 199
Help
displaying
PF1 (help), 32
Help and return codes
on menus, 33
HELP edit command
Code Frame editor, 90
Help indicator field
Maintain Control Record panel, 54
Help text
multilingual support for, 407
Helproutine
glossary definition, 423
Helproutines
CNHMDL, 304
CNHMSG, 305
CNHOBJ, 305
CPHESET, 387

Natural data
prefix used, 151
Predict data
prefix used, 151
Hold field
determining for a file, 375
HOLD-FIELD keyword
supplying default hold field, 80

I

IF comparison clause
generating for two structures, 314
Import utility
using, 393, 395
In member field
Maintain Current Editor Profile window, 93
INCLUDE code members
description, 409
retrieving message text with, 409
Include recursive code frames field
CSFLOAD Multiple Code Frame Export utility, 394
INIT parameter
Code Frame editor, 137
INIT values
SYSERR, 415
format, 132
SYSERR format, 416
Initialization statements
generating for PDA variables in model PDA, 253
generating for PDAC variables, 265
Input prompts
building, 330
INPUT statements
for PDAX variables
supplying, 184
INPUT-ADDITIONAL-PARAMETERS user exit
description, 259
example, 259
INPUT-OUTPUTS structure
parameter data areas (PDAs), 280
INPUTS structure
parameter data areas (PDAs), 280
INPUT-SCREEN user exit
description, 260
example, 260
Installation
overriding SYSERR message text, 421
Internal data areas
generating, 365
Internal reader
submitting JCL source buffer to, 402
Invoke
glossary definition, 423
User Exit editor, 247

J

JCL job
submitting (mainframe), 352

K

Keys
defining default primary, 79
Keywords
retrieving information about, 375
returning, 368

L

L field
Code Frame editor, 85
Language
static installation to override SYSERR, 421
Language Preference window
description, 68
Layout
generated model PDA, 178
LET edit command
Code Frame editor, 90
Libraries
Natural Construct, 29
Line commands
Code Frame editor, 88
Line field
Window Parameters window, 199

Line types
  nested code frames, 86
List Code Frames function
  Administration main menu, 51
LIST edit command
  Code Frame editor, 90
Load Map Layout and Settings option
  Map Related Subprograms submenu, 337
Load utilities
  transferring data between platforms, 393
Local data areas (LDAs)
  creating translation, 131
  CST-Shell model
    defining, 240
  retrieving field information, 287
  Standard Parameters panel
    CST-Shell model, 242
  static or dynamic installation, 406
LOCAL-DATA user exit
  description, 260
Looping label
  verifying a Natural looping label, 329
Lower case
  converting text string, 311

M

Maintain Control Record panel
  description, 53–54
Maintain Current Editor Profile window
  description, 93
  edit recovery, 94
  example, 93
Maintain Models function
  Administration main menu, 42
Maintain Models panel
  defining new model
    example, 112
Maintain Subprograms panel
  description, 52
Maintain User Exit window
  description, 107
Maintenance maps
  creating, 134
  creating translation LDAs, 131
Major field
  Additional Parameters panel
    CST-Document model, 227
Map name field
  Standard Parameters panel
    CST-Modify model, 196
    CST-Modify-332 model, 206
Map settings
  loading into edit buffer, 337
Maps
  calculating absolute field coordinates, 340
  loading layout into edit buffer, 337
Maps field
  Standard Parameters panel
    CST-Document model, 226
Mark a field
  glossary definition, 424
Mass translation
  translating screen prompts, 409
Mass translation mode
  description, 409
Maximum value
  generating field assignments, 331
MENUT command
  invoking translation mode, 409
Message numbers
  using with substitution values, 411
Message text
  adding, 37
  formatting, 412
  multilingual support, 37
  multilingual support for, 404, 407–408
  overriding multilingual support, 408
  retrieving, 290
  supported areas for formatting, 413
  translating at runtime, 355
Messaging support field
  Standard Parameters panel
    CST-Shell model, 242
Methods
  calling CNUMSG within, 420
Minimum value
  generating field assignments, 332
Minor field
  Additional Parameters panel
    CST-Document model, 227

MISCELLANEOUS-SUBROUTINES
user exit
  description, 260
  example, 260
MISCELLANEOUS-VARIABLES user exit
  description, 261
  example, 261
Mixed case
  converting text string, 311
Model
  glossary definition, 424
Model field
  Maintain Models panel, 43
  Model Compare Utility window, 58
  Single Module Test Program panel, 145
Model name field
  CST-PDA model, 177
  Standard Parameters panel
    CST-Document model, 225
    CST-Postgen model, 215
Model PDA
  alphanumeric fields, 178
  assigning variable values, 210
  creating, 177
  creating for a new model, 159
  example, 176
  example of layout, 118
  FILE-CODE field, 361
  PDAX variables
    supplying INPUT statements, 184
  RESET-STRUCTURE field, 180
  setting conditions, 265
  substitution parameters, 178
  supplying default values for PDAX variables, 263
Model PDA field
  Standard Parameters panel
    CST-Stream model, 238
    CST-Validate model, 232
Model subprograms
  description, 53
  example, 281–282
  modifying parameter data area (PDA), 148
  parameter data areas (PDAs), 115
  precautions when using, 148
  testing, 143
  tips for using, 148
  using DISPLAY statements
    NOHDR option, 136
Models
  clear subprogram example, 180
  creating
    clear subprogram, 168
    code frame example, 156
    document subprogram, 172
    generation subprogram, 171
    maintenance map, 161
    model PDA example, 159
    modify subprograms, 163
    parameters for program header, 155
    post-generation subprogram, 167
    pre-generation subprogram, 166
    prototype example, 154
    read subprogram, 170
    save subprogram, 169
    subprograms, 163
  creating code frames, 102
  creating new
    scope and functionality, 99
    utility subprograms and helproutines, 151
    what to generate?, 100
  creating parameter data areas (PDAs), 115, 176
  creating prototype
    isolate parameters, 101
  CST-Clear, 179–180
  CST-Document, 224
    example, 224
    parameters, 225
  CST-Frame model
    example of generation subprogram, 219
    parameters, 220
  CST-Modify, 193
    example, 193
  CST-PDA, 176
    example, 176
    parameters, 177
  CST-Postgen, 214
    example, 214
    parameters, 215
  CST-Pregen, 210
    example, 210
    parameters, 211

CST-Read
parameters, 185
CST-Save model
parameters, 189
CST-Shell
description, 240
example, 240
parameters, 241
CST-Stream, 236
customizing, 269
defining defaults, 75
defining new
description, 112
description of components, 98
maximum number of code frames, 44
modifying model subprograms, 270
new
implementing, 149
parameter data areas (PDAs), 176
production copy of model subprograms, 272
providing defaults for, 319
read subprogram
example, 184
retrieving information about Natural Construct, 333
retrieving parameters and conditions from code frames, 325
save subprogram
example, 188
statement
creating, 149
description, 149
supplied
modifying, 114
supplied components, 96
testing Natural Construct-generated, 354
testing subprograms, 143

Modify
code frames, 271, 275
external data areas, 273
maintenance maps, 274
model subprograms, 274
subprograms (CD*), 273

Modify client specificatn field
Maintain Models panel, 45

Modify models
using steplibs, 277

Modify server specificatn field
Maintain Models panel, 45

Modify subprograms
creating, 134
creating for a new model, 163
description, 192
order of execution, 134, 192
supporting dynamic translation, 194

Module
glossary definition, 424

Module name field
CST-PDA model, 177
Standard Parameters panel
CST-Clear model, 181
CST-Document model, 225
CST-Frame model, 220
CST-Modify model, 195
CST-Modify-332 model, 205
CST-Postgen model, 215
CST-Pregen model, 211
CST-Read model, 185
CST-Save model, 189
CST-Shell model, 241
CST-Stream model, 237
CST-Validate model, 231

Module type field
Maintain Models panel, 149
Standard Parameters panel
CST-Shell model, 241

Module types
defining a new model, 99
Maintain Models panel, 44

Modules
comparing in different libraries, 398
comparing Natural source, 397
generating startup
CST-Shell model, 240
regenerating multiple, 402
storing saved, 37

Multilingual support
creating translation LDAs, 131
for Natural Construct, 67

Multiple Code Frame Export utility, 394

Multiple Code Frame Import utility
description, 395

N

N line type
  Code Frame editor, 86
Naming conventions
  defining a model, 113
NATPARM parameters
  supported, 343
Natural
  returning byte length of data types, 344
  verifying looping label, 329
Natural Construct
  accessing Predict from model subprograms, 270
  creating new models, 99
  documentation and course information, 22
  general purpose generation subprograms, 388
  generated modules
    advantages of using, 26
  generation utility, 307
  glossary of terms, 423
  invoking from a steplib, 31, 277–278
  libraries, 29
  multilingual support, 404
  nucleus, 53
    description, 98
    PF-key control, 200
  PF-key descriptions, 32
  retrieving information about models, 333
  retrieving information about subprograms, 353
  saving objects, 37
  supplied models
    modifying, 114
  terminating
    PF3 (quit), 32
  terminating and saving, 28
  testing generated modules, 354
Natural Construct Administration and Modeling documentation
  conventions used throughout, 21
  purpose and structure, 19
Natural Construct client
  translating specification data passed to the server, 236
Natural Construct components
  converting to upper case, 401
Natural editor profile
  Code Frame editor, 94
Natural Next prompt
  specifying direct commands, 38
Natural objects
  checking for existence of, 291
  types, 305
Natural Security
  retrieving information from Predict, 361
Natural source modules
  comparing, 397
Natural stack
  placing substitution values, 273
Natural-related helproutines (CNH), 304
Natural-related subprograms
  description, 286
NCSTBGEN command
  batch generation, 278
NCSTBGEN utility
  regenerating multiple modules, 402
NCST-CONTROL file
  CSUSETKY subprogram, 349
NCSTDEM2 library
  invoking the demo system
    DB2, 31
NCSTDEMO library
  invoking the demo system
    Adabas, 31
NCSTDEMS library
  invoking the demo system
    Unix SQL, 31
NCSTDEMV library
  invoking the demo system
    VSAM, 31
NCST-FRAME-LINES file
  CSUFRVAR subprogram, 325
NCSTG module
  invoking Natural Construct from a steplib, 278
NCSTG program
  invoking Natural Construct from a steplib, 31

NCST-MODEL file
  CNHMDL utility subprogram, 304
  CSUFRVAR subprogram, 325
NCST-SUBPROGRAM file
  CSUSUBP subprogram, 353
Nested code frames
  description, 47
  example, 106
  exporting, 394
  line types, 86
  supplying parameters, 105
-nnnn edit command
  Code Frame editor, 92
NOHDR option
  DISPLAY statement, 136, 219
NOTITLE option
  WRITE, DISPLAY, or PRINT statements, 136, 219
Nucleus
  description, 98

O

Object
  glossary definition, 424
Object category
  CN* prefix, 282
  CP* prefix, 282
  CS* prefix, 282
Object description
  defining default, 80
Objects
  passing information about elements of, 363
  saving Natural Construct-generated, 37
Of indicator field
  Maintain Control Record panel, 55
Optional field
  glossary definition, 424
Optional Parameters window
  description, 146
  PF5 (optns) key, 146
OUTPUTS structure
  parameter data areas (PDAs), 280
Overriding SYSERR numbers
  in message text, 408

P

-P edit command
  Code Frame editor, 92
PA-key settings
  Code Frame editor
    modifying, 92
Panel
  glossary definition, 424
Panels
  creating multilingual, 69
  displaying next
    PF11 (right), 33
  displaying previous
    PF10 (left), 32
    PF2 (retrn), 32
  editing text, 71
  translating text, 73
Parameter
  glossary definition, 424
Parameter data area field
  Standard Parameters panel
    CST-Clear model, 182
    CST-Frame model, 221
    CST-Modify model, 195
    CST-Modify-332 model, 206
    CST-Pregen model, 212
    CST-Read model, 186
    CST-Save model, 190
    CST-Shell model, 242
Parameter data areas (PDAs)
  CPAEL, 280
  creating new models, 115
  defining
    CST-Shell model, 240
  external, 280
  model PDA, 176
  model subprograms, 176
    modifying redefinitions, 148
  retrieving field information, 287
  selecting fields from, 348
Parameter field
  Code Frame editor
    N line types, 86
PARAMETER-DATA user exit
  description, 261
  example, 261

Parameters
calculating values at runtime, 102
containing dynamic attributes, 320
model components, 101
prototype
isolating, 101
reading specification, 345
removing redundant, 102
substitution, 104
model PDA, 178
supplying code frame
generation subprograms, 104
nested code frames, 105
supplying default corporate, 78
supplying default library-specific, 77
supplying default model, 76–78
supplying default Predict, 79
user exit
description, 106
using substitution values, 408
PDA name field
Maintain Models panel, 43
PDA variables
generating initialization statements, 253
PDAC variables
generating initialization statements, 265
PDAC-DEPTH-KEYS field
model PDA, 274
PDA-CONDITION-CODES field
CST-PDA model, 178
model PDA, 274
PDA-FILE parameter
CPHFI helproutine, 385
CPHRL helproutine, 386
PDA-FRAME-PARM field
CU--MSG subprogram, 389
invoking generation subprograms, 219
PDA-KEY parameter
CNHMDL helproutine, 304
CNHMSG helproutine, 305
CPHFIB subprogram, 385
PDA-LABEL parameter
CSULABEL subprogram, 329
PDA-LEFT-INDICATOR parameter
CU--LRP subprogram, 388
PDA-LEFT-PROMPT parameter
CU--LRP subprogram, 388
PDA-MSG parameter
CNHMSG helproutine, 305
PDA-OBJECT-TYPE parameter
CNHOBJ helproutine, 306
PDA-RELATIONSHIP-NAME parameter
CPHRL helproutine, 386
PDA-RESTRICTION parameter
CNHMDL helproutine, 304
PDA-RIGHT-INDICATOR parameter
CU--LRP subprogram, 388
PDA-RIGHT-PROMPT parameter
CU--LRP subprogram, 388
PDA-RIGHT-PROMPT-OF parameter
CU--LRP subprogram, 388
PDA-SET-HELP parameter
CPHSETHL helproutine, 387
PDA-STRING parameter
CSU-VAR subprogram, 308
PDA-TYPE parameter
CPHPRED helproutine, 385
PDA-UNDERSCORE parameter
CU--UL subprogram, 389
PDA-USER-AREA field
CST-PDA model, 178
model PDA, 274
resetting fields, 180
PDAX variables
supplying default values, 263
supplying INPUT statements, 184
writing to source buffer, 264
PDAX-DEPTH-KEYS field
maintenance maps, 274
model PDA, 274
PDAX-DIRECT-COMMAND-PROCESS field
maintenance maps, 277
Period (.)
return code
description, 33
PF1 (help)
description, 32
PF1= fields
Maintain Current Editor Profile window, 93

PF10 (left)
  description, 32
PF11 (right)
  description, 33
PF12 (lang)
  description, 68
PF12 (main)
  description, 33
PF2 (retrn)
  description, 32
PF3 (quit)
  description, 32
PF5 (optns)
  Generation main menu, 146
PF5 (top)
  Code Frame Compare Utility panel, 63
PF5 (windw)
  Standard Parameters panel
    CST-Modify model, 199
PF6 (hcopy)
  Code Frame Compare Utility panel, 63
PF6 (pfkey)
  Standard Parameters panel
    CST-Modify model, 200
PF7 (bkwrd)
  description, 32
PF8 (frwrd)
  description, 32
PF-key
  glossary definition, 424
PF-key Parameters window
  description, 200
PF-key settings
  Code Frame editor
    modifying, 92
  returning information about, 353
PF-Key Settings fields
  Maintain Control Record panel, 54
PF-keys
  13 to 24, 54
  PF12 (lang), 68
  PF5 (optns)
    Generation main menu, 146
  PF5 (windw)
    Standard Parameters panel, 199
  PF6 (pfkey)
    Standard Parameters panel, 200
  retrieving definitions, 349
  setting Natural Construct, 53
  used in Natural Construct
    descriptions, 32
PF-KEYS user exit
  description, 262
  example, 262
Place holders
  setting up SYSERR text with, 411
  supported areas in Natural Construct, 413
Platforms
  differences between mainframe and Unix, 18
  transferring data between, 393
POINT edit command
  Code Frame editor, 89
Position fields
  Window Parameters window, 199
Position indicator(s) field
  Maintain Control Record panel, 55
Positional edit commands
  Code Frame editor, 92
Post-generation field
  Maintain Models panel, 45
Post-generation modifications field
  Optional Parameters window, 147
Post-generation subprogram
  creating for a new model, 167
  description, 137
  example, 138
  STACK TOP DATA FORMATTED statement, 148
Predict
  accessing from model subprograms, 270
  related
    helproutines (CPH), 384
    subprograms (CPU), 361
      with Natural Security installed, 361
  retrieving information from, 361
  retrieving information when using Natural Security, 361
  valid object types, 385
Predict file
  CALLNAT subprogram for each field, 371
  retrieving information about fields, 366
  retrieving information about next field, 369

Predict keywords
retrieving information about, 375
supplying default model parameters, 79
Predict relationships
browsing for selection, 386
retrieving for specified file, 377
Predict verification rules
printing to source buffer, 380
returning information about, 381
Pre-generation field
Maintain Models panel, 46
Pre-generation subprogram
creating for a new model, 166
CST-Pregen model, 210
description, 136
example, 136
for a model, 210
Primary key
defining default, 79
Print
verification rules to source buffer, 380
PRINT (SRC) NOTITLE statement
model subprograms, 148
Print files
DEVICE LPT 1 (Unix), 392
Print File 1 (mainframe), 392
PRINT statements
model subprograms
tips when using, 148
NOTITLE option, 136, 219
Processing
read logical, 284
Processing errors, 282
PROCESS-SPECIAL-KEYS user exit
description, 263
example, 263
Production copy
model subprogram, 272
PROFILE edit command
Code Frame editor, 90
Program
glossary definition, 424
Programming mode field
Maintain Models panel, 44
Prompts
multilingual support for, 404
translating, 356
Prototype
removing parameters, 102
Prototype of a model
scrutinizing, 101
PROVIDE-DEFAULT-VALUES user exit
defining model defaults, 76
description, 263
example, 263

Q

Query program
glossary definition, 424
Question mark (?)
displaying help, 33
indicating nested code frames, 105
used in code frame names, 47
QUIT edit command
Code Frame editor, 90
Quotation marks
changing single to double
CU--QUOT, 148

R

Read logical processing
model subprograms, 284
READ PROGRAM edit command
Code Frame editor, 90
Read specification field
Maintain Models panel, 45
Read subprogram
creating for a new model, 170
description, 141, 184
example, 142, 184
Recovery member
Code Frame editor, 92
Recursive code frames
exporting, 394
Redefined fields
expanding, 381

Redefinitions
generating for fields, 375
Reference numbers
using with substitution values, 413
Referential integrity checks
generating against foreign files, 346
Regenerating multiple modules
utility, 402
Relationships
retrieving for specified file, 377
retrieving information about foreign, 368
returning information about, 376
Replace Option field
CSFLOAD Multiple Code Frame Import utility, 395
Required field
glossary definition, 424
User Exits panel, 248
RESET edit command
Code Frame editor, 90
RESET-STRUCTURE field
model PDA, 180
Restricted data areas
description, 283
Return code
glossary definition, 424
RETURN-CODE field
CSASTD parameter data area, 130, 282
Rules
returning information about Predict verification, 381

S

S field
Code Frame editor, 85
Sample field
User Exits panel, 248
Sample subprogram
description, 142
example, 142
Sample subprogram field
Maintain User Exit window, 108
Sample subprograms
GUI, 94
Save Code Frame function
Administration main menu, 50
Save specification field
Maintain Models panel, 45
Save subprogram
creating for a new model, 169
description, 140
example, 141
for a model
example, 188
SAVE-PARAMETERS user exit
description, 264
example, 264
SCAN edit command
Code Frame editor, 91
used with ABS field, 85
Screen prompts
translating, 356, 358
Scroll
glossary definition, 424
Scroll indicator field
Maintain Control Record panel, 55
Scrolling
backward through data
PF7 (bkwrd), 32
Code Frame editor, 92
forward through data
PF8 (frwrd), 32
generating depth, 274
three-dimensional arrays, 274
Second header field
Standard Parameters panel
CST-Modify model, 197
Server
translating specification data from the client, 236
SET CONTROL parameters
separating line of headings, 350
SET-CONDITION-CODES user exit
description, 265
example, 265
Show Model Differences window
example, 59, 64–65
Single Module Test Program panel
description, 143
testing Natural Construct-generated models, 354
Size fields
Code Frame editor, 84
Window Parameters window, 199

Source buffer
  submitting JCL, 402
Source Range Compare Facility window
  description, 398
Special Hardware field
  Maintain Control Record panel, 56
Specification parameters
  reading, 345
  writing to source buffer, 264
Specifications only field
  Optional Parameters window, 147
Specify a value
  glossary definition, 424
SPUELNX
  no chaining, 284
STACK TOP option
  example, 139
Standard Parameters panel
  CST-Clear model, 181
  CST-Document model, 225
  CST-Frame model, 220
  CST-Modify model, 195
    PF5 (windw), 199
    PF6 (pfkey), 200
  CST-Modify-332 model, 205
    example, 205
  CST-PDA model, 177
  CST-Postgen model, 215
  CST-Pregen model, 211
  CST-Read model, 185
  CST-Save model, 189
  CST-Shell model, 241
  CST-Stream model, 237
  CST-Validate model, 231
START-OF-PROGRAM user exit
  description, 266
Startup modules
  generating
    CST-Shell model, 240
Statement models
  alignment of generated code, 150
  description, 149
Static mode installation
  overriding SYSERR references, 421
Static option
  retrieving and replacing references, 421
Status window field
  Maintain Models panel, 43
  Optional Parameters window, 146
Step field
  Optional Parameters window, 146
Steplibs
  invoking Natural Construct, 277–278
    with Natural Security installed, 31
Stream subprogram
  generating, 236
Structure names
  passing, 283
Structures
  generating IF comparison clause, 314
Subdescriptors
  expanding, 381
  returning definition for, 378
Subpanel field
  Standard Parameters panel
    CST-Modify model, 198, 201
Subprogram
  CSUCOMPR, 314
  glossary definition, 424
Subprogram chaining
  CPUELRD, 285
Subprogram field
  Code Frame editor
    N line types, 86
Subprograms
  chaining, 285
  CNUERMSG, 290
  CNUEXIST, 291
  CNUGDABL, 292
  CNUGDAEL, 293
  CNUGENDA, 294
  CNUMPPRF, 295
  CNUMSG, 296
  CNUPEXST, 299
  CNUSEL, 300
  CNUSRCNX, 301
  CNUSRCRD, 302
  CPHEL, 384
  CPHELB, 384
  CPHFI, 385
  CPHFIB, 385
  CPHPRED, 385
  CPHRL, 386

CPUEL, 366
CPUELDE, 367
CPUELKY, 368
CPUELNX, 369
CPUELRD, 370
CPUELVE, 372
CPUEXIST, 373
CPUFI, 374
CPU-FREL, 368
CPUHOLD, 375
CPUKY, 375
CPU-OBJ, 362
CPU-OBJ2, 363
CPU-OREL, 364
CPUREDEF, 375
CPURL, 376
CPURLRD, 377
CPUSUPER, 378
CPUUNIQ, 379
CPUVERUL, 381
CPU-VIEW, 364
CPUXPAND, 381
CSU2LONG, 360
CSUBANN, 309
CSUBLDRP, 309
CSUBMIT, 310
CSUBYTES, 310
CSUCASE, 311
CSUCCMD, 312
CSUCENTR, 313
CSUCTRL, 315
CSUCURS, 315
CSUCURS1, 316
CSUDB2SP, 316
CSUDELFF, 318
CSUDELFLT, 319
CSUDYNAT, 320
CSUEMLEN, 322
CSUENDX, 323
CSUFDEF, 324
CSUFRVAR, 325
CSUGEN, 326
CSUHEADS, 326
CSUINCL, 327
CSUIS, 328
CSULABEL, 329
CSULENGT, 330
CSULPS, 331
CSUMAX, 331
CSUMIMAX, 332
CSUMODEL, 333
CSUMORE, 334
CSUMPBOX, 335
CSUMPCPR, 336
CSUMPDUP, 336
CSUMPLAY, 337
CSUMPMMS, 338
CSUMPOVL, 338
CSUMPREG, 339
CSUMPTAB, 340
CSUMPTST, 340
CSUNATFM, 341
CSUNEWX, 342
CSUPARMS, 343
CSUPARTY, 344
CSUPPER, 344
CSUREADS, 345
CSUREF, 346
CSUSCAN, 347
CSUSELFV, 348
CSUSELTKY, 349
CSUSETW, 350
CSUSORT, 351
CSUSPLIT, 351
CSUSUBP, 352–353
CSUTEST, 354
CSUTLATE, 355
CSUTRANS, 356
CSU-VAR, 308
CSUXCHK, 359
CU--EM, 388
CU--L, 388
CU--MSG, 389
CU--UL, 389
document, 142, 224
implementing user exit security, 74
issuing CALLNAT for fields in Predict file, 371
model
  creating, 134
  modify, 192
  modifying parameter data areas (PDAs), 148
  precautions when using, 148
  tips for using, 148
modify
  creating, 134
modifying CD*, 273
modifying model, 270
Natural Construct utility
  prefix used, 151

Natural data retrieval
prefix used, 151
passing relationship information to, 377
Predict data retrieval
prefix used, 151
pre-generation
description, 136
retrieving information about Natural Construct, 353
returning information about, 353
sample user exit, 142
save, 141
SPUVE, 380
testing model, 354
using DISPLAY statements
NOHDR option, 219
validating a field definition, 324
Subroutine
glossary definition, 424
Substitution data members
supported areas in Natural Construct, 413
using with SYSERR, 408, 411
Substitution parameters
default format, 178
description, 104
glossary definition, 424
model PDA, 178
stack order, 139
supplying values for, 214
Substitution values
replacing with a null string, 139
Summary Report window
description, 62
Superdescriptors
expanding, 381
generating a FIND statement, 316
returning definition for, 378
SYSCST library
description, 30
SYSCSTX library
description, 30
SYSDIC-DB file
CPUEXIST subprogram, 373
SYSDIC-EL file
CPUELNX subprogram, 369
CPU-OBJ subprogram, 362
CPU-VIEW subprogram, 365
SYSDIC-FI file
CPHFI helproutine, 385
CPHRL helproutine, 386
CPHSETHL helproutine, 387
CPUELNX subprogram, 369
CPUEXIST subprogram, 373
CPU-OBJ subprogram, 362
CPURLRD subprogram, 377
CPU-VIEW subprogram, 365
SYSDIC-KL file
CPURLRD subprogram, 377
SYSDIC-KY file
CPHPRED helproutine, 386
CPUEXIST subprogram, 373
SYSDIC-MO file
CPHPRED helproutine, 386
SYSDIC-PR file
CPHPRED helproutine, 386
CPUEXIST subprogram, 373
SYSDIC-RE file
CPHPRED helproutine, 386
SYSDIC-RL file
CPUEXIST subprogram, 373
SYSDIC-SY file
CPHPRED helproutine, 386
CPUEXIST subprogram, 373
SYSDIC-UE file
CPUEXIST subprogram, 373
SYSDIC-VE-ACT file
CPUVE subprogram, 379–380
SYSERR INIT values
format, 132, 416
SYSERR message library
specifying a valid name for message text, 415
SYSERR numbers
formatting, 412
multilingual support
translation LDAs, 132
overriding in a one language install, 421
overriding multilingual support, 408
supported areas for formatting, 413
supported areas in Natural Construct, 413
translation LDAs, 415

SYSERR references
format, 405
formatting, 412
multilingual support for text and messages, 404
referencing text, 405
translating prompts and message text, 404
SYSERR utility
adding multilingual messages, 37
multilingual model panels, 67
multilingual support, 404
SYSLIBS library
description, 30
SYSMAIN utility
using to copy new model components, 149
System field
Standard Parameters panel
CST-Stream model, 237
CST-Validate model, 231
SYSTEM library
description, 30
System name field
Standard Parameters panel
CST-Shell model, 242
SYSTEM-FNAT file
CNUEL subprogram, 287
CNUEXIST subprogram, 291
CNUGDAEL subprogram, 293
CNUSRCNX subprogram, 301
CNUSRCRD subprogram, 302
SYSTEM-FUSER file
CNHOBJ helproutine, 306
CNUEL subprogram, 287
CNUEXIST subprogram, 291
CNUGDABL subprogram, 292
CNUGDAEL subprogram, 293
CNUSRCNX subprogram, 301
CNUSRCRD subprogram, 302

T

T field
Code Frame editor, 86
Terminate
glossary definition, 425
Natural Construct
PF3 (quit), 32
Text
multilingual support for message, 404
Text field
Optional Parameters window, 146
Text strings
centering, 313
scanning edit buffer for, 347
Text translation
context translation, 409
mass translation, 409
Three-dimensional arrays
scrolling, 274
Time field
Select Frames window, 51
Title field
Standard Parameters panel
CST-Clear model, 182
CST-Document model, 226
CST-Frame model, 221
CST-Modify model, 195
CST-Modify-332 model, 206
CST-Postgen model, 216
CST-Pregen model, 212
CST-Read model, 186
CST-Save model, 190
CST-Shell model, 242
CST-Stream model, 238
CST-Validate model, 232
Window Parameters window, 200
TOP edit command
Code Frame editor, 92
Trace facilities
Natural Construct
debugging a model, 146
Translate Short Message window
example
editing text, 71
translating text, 72
description, 410
Translation
lower case to upper case, 402
Translation LDAs
creating, 132, 415
creating for a new model, 160
defining a larger field, 418
example, 134
format, 132
implementing multilingual support, 131

multilingual support, 131, 406
naming conventions, 132
Translation LDAs field
Standard Parameters panel
CST-Document model, 226
CST-Modify model, 196
Translation mode
context translation, 410
editing panel text, 71
invoking Administration main menu, 28
mass translation, 409
translating panel text, 72
using, 70
Type field
Additional Parameters panel
CST-Document model, 227
Maintain Models panel, 44
Types
valid object code, 44

U

U line type
Code Frame editor, 86
Underscore character field
Maintain Control Record panel, 55
Unload utilities
transferring data between platforms, 393
Upper case
converting contents of source buffer to, 344
converting text string, 311
Upper case translation utility
description, 402
UPPER edit command
Code Frame editor, 91
User exit
glossary definition, 425
User exit code
reusing, 246
User Exit editor
description, 248
User exit name field
Maintain User Exit window, 108
User exit point
adding to code frames, 106
User exit required field
Maintain User Exit window, 108
User exit security
implementing, 74
User exits
ADDITIONAL-INITIALIZATIONS, 249
ADDITIONAL-SUBSTITUTION-VALUES, 250
ADDITIONAL-TRANSLATIONS, 251
AFTER-INPUT, 252
AFTER-INVOKE-SUBPANELS, 252
ASSIGN-DERIVED-VALUES, 253
BEFORE-CHECK-ERROR, 254
BEFORE-INPUT, 254
BEFORE-INVOKE-SUBPANELS, 255
BEFORE-REINPUT-MESSAGE, 255
BEFORE-STANDARD-KEY-CHECK, 256
CHANGE-HISTORY, 256
Code Frame editor
line commands, 88
CST-Clear model, 182
CST-Document model, 227
CST-Frame model, 221
CST-Modify model, 202
CST-Modify-332 model, 207
CST-Postgen model, 216
CST-Pregen model, 212
CST-Read model, 186
CST-Save model, 190
CST-Stream model, 238
CST-Validate model, 233
DESCRIBE-INPUTS, 256
displaying a list of available, 247
END-OF-PROGRAM, 257
format, 249
GENERATE-CODE, 258
GENERATE-SUBROUTINES, 259
GENERATE-VALIDATIONS, 259
generating end of prompt, 323
generating new prompts, 342
INPUT-ADDITIONAL-PARAMETERS, 259
INPUT-SCREEN, 260
LOCAL-DATA, 260
MISCELLANEOUS-SUBROUTINES, 260

MISCELLANEOUS-VARIABLES, 261
PARAMETER-DATA, 261
parameters, 106
PF-KEYS, 262
PROCESS-SPECIAL-KEYS, 263
PROVIDE-DEFAULT-VALUES, 263
referencing database fields, 248
SAVE-PARAMETERS, 264
scanning edit buffer for, 359
selecting from User Exits panel, 248
SET-CONDITION-CODES, 265
START-OF-PROGRAM, 266
SUBSTITUTION-VALUES, 266
supplied for the administration models, 246
translating screen prompts, 358
VALIDATE-DATA, 266
User Exits field
User Exits panel, 248
User Exits panel
CST-Clear model, 182
CST-Save model, 190
description, 248
User field
Select Frames window, 51
Utilities
comparing modules in different libraries, 398
comparison, 397
converting Natural Construct components to upper case, 401
creating new models, 151
CSFHCOPY Frame Hardcopy, 396
CSFLOAD Multiple Frame Import, 395
CSFUNLD Multiple Code Frame Export, 394
INCLUDE code insertion, 402
invoking supplied, 392
loading code frames, 271
multiple module regeneration, 402
Natural Construct generation, 307
Predict-related, 361
submitting JCL source buffer, 402
supplied with Natural Construct
accessing from Drivers menu, 66
transferring data between platforms, 393
unloading code frames, 271
upper case translation, 402
Utility
glossary definition, 425

V

VALIDATE-DATA user exit
description, 266
example
CST-Modify model, 267
CST-Modify-332 model, 268
Validation edits
placing, 275
Validation subprogram
coding validations, 233
description, 230
example, 230
Validations
coding, 233
Variable
glossary definition, 425
Variables
converting to abbreviation, 360
Verification rules
printing to source buffer, 380
retrieving names, 372
returning information about Predict, 381
returning type N, 372
Version field
Model Compare Utility window, 58
Views
selecting fields from, 348
Views field
Standard Parameters panel
CST-Shell model, 242

W

Warning message
instead of error, 234
Width field
Window Parameters window, 199
Window
glossary definition, 425
Window Parameters window
description, 199
Window support field
Standard Parameters panel
CST-Modify model, 198

Work files
reading from other platforms, 393
saving in ASCII format (mainframe), 393
saving in ASCII format (Unix), 393
WRITE statements
format, 264
generating for new specification parameters, 264
NOTITLE option, 136, 219
writing PDAX variables to source buffer, 264

X

X and Y delimiters
Code Frame editor
SCAN edit command, 91
X delimiter
Code Frame editor, 89
X line type
Code Frame editor, 87
X-Y field
Code Frame editor, 85, 91

Y

Y delimiter
Code Frame editor, 89
Y edit command
Code Frame editor, 92