

## **Natural Business Services**

### **Natural Construct Object Models**

Version 8.2.1

November 2013

This document applies to Natural Business Services Version 8.2.1.

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

Copyright © 2006-2013 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, United States of America, and/or their licensors.

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

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://documentation.softwareag.com/legal/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

**Document ID: NBS-OBJECTMODELS-821-20131119**

## Table of Contents

Preface .....	v
1 Overview of Object-Oriented Development .....	1
Define Natural Construct Objects .....	2
Define Object Relationships in Predict .....	5
2 Using the Object-Browse Models .....	13
Introduction .....	14
Object-Browse-Subp Model .....	15
Object-Browse-Static Model .....	31
Object-Browse-Dialog Model .....	36
Object-Browse-Dialog-Driver Model .....	94
Object-LDA Model .....	96
Object-Browse-Select-Subp Model .....	101
3 Using the Object-Generic-Subp Model .....	111
Introduction .....	112
Parameters for the Object-Generic-Subp Model .....	112
User Exits for the Object-Generic-Subp Model .....	115
4 Using the Object-Maint Models .....	125
Introduction .....	126
Object-Maint-Subp and Object-Maint-Enhanced-Subp Models .....	128
Object-Maint-Dialog Model .....	152
Object-Maint-Dialog-Subp Model .....	167



---

# Preface

---

*Natural Construct Object Models* describes the Object series of models. These models generate the Natural subprograms used by Natural Business Services.

This documentation is intended for developers who are familiar with Natural Construct and want to use the Object series of models to create the business service subprograms.

*Natural Construct Object Models* covers the following topics:

- |  |  |
|--|--|
| <i>Overview of Object-Oriented Development</i> | Provides an overview of object-oriented development. Includes information on defining Natural Construct objects and defining object relationships in Predict.  |
| <i>Using the Object-Browse Models</i>          | Describes the Object-Browse series of models and how to use the models to generate the browse components of an object-browse process. It also contains information on changing the default PF-key style. |
| <i>Using the Object-Generic-Subp Model</i>     | Describes the Object-Generic-Subp model, which generates a business service (subprogram) associated with up to 10 subprograms and 20 methods.  |
| <i>Using the Object-Maint Models</i>           | Describes the Object-Maint series of models and how to use the models to generate the maintenance components of an object-maintenance process.   |

---

# 1 Overview of Object-Oriented Development

---

- Define Natural Construct Objects ..... 2
- Define Object Relationships in Predict ..... 5

The concept behind object-oriented development is that an application consists of object modules (Natural subprograms) and dialog modules (Natural programs or subprograms). The object module builds an object-level interface to a complex data structure, while the dialog module communicates with the end user and invokes methods (data actions) implemented by the object module.



**Note:** For more information on object-oriented development, see *Design Methodology, Natural Construct Generation*.

Because object implementation is hidden in object-oriented development, applications:

- Are simplified
- Can use more complex data models
- Are more reliable and easier to maintain because semantic integrity is enforced within the object

An important aspect of object orientation and component-based, client/server development is the granularity of application components. Components with coarse granularity tend to contain large-scale, complex operations that provide much functionality from a single request. However, they usually do not allow the calling program to fine-tune the object's behavior or request additional functionality.

On the other hand, finely grained components allow greater control over how their functions are performed, but they often place an additional burden on the calling program as it has to call the object multiple times in a procedural fashion. Well-conceived objects strike a balance between these approaches to optimize ease-of-use and performance.

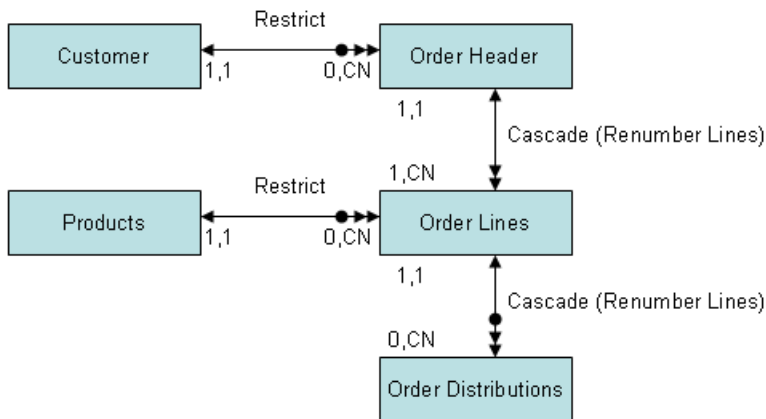
For maximum reusability, components should achieve a separation of dialog handling from business logic. This distinction allows for consistent handling of business rules, regardless of whether the rules are enforced from a GUI dialog, a browser, a mainframe screen, or through a batch process. Similarly, by distinctly separating business logic from database access, it is much easier to change the underlying database technology without affecting the business logic within applications.

## Define Natural Construct Objects

---

An object is a group of related entities that must be maintained within a single transaction. This section discusses some of the characteristics all objects possess and describes how to define intra-object relationships (relationships that define the bounds of an object), and inter-object relationships (relationships between two objects) in Predict. The following diagram illustrates the difference between inter-object and intra-object relationships:





1. Customer object
2. Products object
3. Order object

In this example, Customer (1) and Products (2) are related to Order (3) via inter-object relationships. The Order object is defined by two intra-object relationships between all objects.

When defining the entities within an object, you should follow certain rules. These rules are described in the following sections:

- Level 1 (Primary) File Rules
- Level 2 (Secondary) File Rules
- Level 3 (Tertiary) File Rules
- Level 4 (Quaternary) File Rules
- Primary Key Relationships

### Level 1 (Primary) File Rules

All objects must have a single primary header file that contains a unique primary key field. Although you do not need to define the primary key with the unique option in the database (since the subprogram ensures its uniqueness), we recommend that you do. The primary key can be a descriptor or superdescriptor.

The primary file of a maintenance object must contain a hold field. This field is used internally by the object-maintenance subprogram to logically hold an occurrence of an object between a Get action and a corresponding Update or Delete action. The subprogram tests and updates this field to determine whether the object was updated or deleted by another user. The following table lists the valid data types for the hold field and the value assigned for each type:

Hold Field Format	Value Assigned at Updating
T	*TIMX
A10	*TIME
B8	*TIMESTAMP
N7	*TIMN
A26	*TIMX (DB2 timestamp format)
If format is none of the above, it must be numeric.	Increase current value by 1.

### Level 2 (Secondary) File Rules

The primary file can be related to multiple secondary files with cardinality 1:1, 1:C (1 to optionally 1), 1:N (1 to many), or 1:CN (1 to, optionally, many). Specify the maximum value for N.

### Level 3 (Tertiary) File Rules

The secondary files can be related to multiple tertiary files with cardinality 1:1, 1:C, 1:N, or 1:CN. Tertiary files cannot contain multiple-valued fields (MUs) within periodic groups (PEs).

### Level 4 (Quaternary) File Rules

The tertiary files can be related to multiple quaternary files with cardinality 1:1, 1:C, 1:N or 1:CN. Quaternary files cannot contain MUs or PEs.

### Primary Key Relationships

Each sub-entity (any level under the primary file) must contain a key (descriptor or superdescriptor) that relates the entity to its parent entity. Because each primary key for a sub-entity (child entity) must be prefixed by the primary key of the parent entity and usually contains a suffix (used to return the sub-entities in sorted order), the primary key grows in length as you move down an object from the primary file to the secondary, tertiary, and quaternary files.

The following example shows the primary and secondary file keys for a Customer object:

File	Key
Primary	Customer-Number
Secondary	Customer-Number + Contact-Name

The following example shows the primary, secondary, and tertiary file keys for an Order object:

File	Key
Primary	Order-Number
Secondary	Order-Number + Order-Line-Number
Tertiary	Order-Number + Order-Line-Number + Distribution-Line-Number

## Define Object Relationships in Predict

Use the Predict Modify File Relation panel to define intra-object and inter-object relationships in Predict. The following topics are covered:

- [Intra-Object Relationships](#)
- [Inter-Object Relationships](#)

### Intra-Object Relationships

The following example shows the Predict Modify File Relation panel with information entered for an intra-object relationship:

```

16:44:22          ***** P r e d I c t *****
                  - Modify File Relation -
File relation ... NCST-ORDER-HAS-LINES          Modified: 13-10-21 at 15:40
Type .....* N Natural Construct                by: DEVPM
Keys ..                                         Zoom: N

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

Abstract      Zoom: N

EDIT:  Owner: N   Desc: N
    
```

The following table describes the fields on this panel and the values specified or displayed in each field for an intra-object relationship:

Field	Description
File relation	Relationship ID name. This name should not match the name of any entity within the object or any of their attributes.
Type	Relationship type. Natural Construct models process type N (Natural Construct) relationships.  <b>Note:</b> For relational databases, type R relationships are also processed.
Keys	Keywords. This is an optional field.
Cardinality	
File 1	Cardinality of File 1. For an intra-object relationship, specify "1" in this field.
File-ID	Name of the file on the left side of the 1 to many (1:N) relationship.
Field-ID	Primary key for File 1. The key can be a descriptor or superdescriptor.  <b>Note:</b> For relational databases, the key can be a compound key.
Minimum	Minimum number of occurrences of a record in File 1. For an intra-object relationship, specify "1" in this field.
Average	Average number of occurrences of a record in File 1. For an intra-object relationship, specify "1" in this field.
Maximum	Maximum number of occurrences of a record in File 1. For an intra-object relationship, specify "1" in this field.
File 2	Cardinality of File 2. For an intra-object relationship, specify one of the following codes:  <ul style="list-style-type: none"> <li>■ 1 If an occurrence of a record in File 1 must be related to exactly one occurrence of a record in File 2 and vice versa.</li> <li>■ C If a record in File 1 can be related to zero or one record in File 2, but a record in File 2 must be related to one record in File 1.</li> <li>■ N If an occurrence of a record in File 1 must be related to one or more occurrences of a record in File 2, but an occurrence of a record in File 2 must be related to exactly one occurrence of a record in File 1.</li> <li>■ CN If an occurrence of a record in File 1 can be related to zero or more occurrences of a record in File 2, but an occurrence of a record in File 2 must be related to exactly one occurrence of a record in File 1.</li> </ul>
File-ID	Name of the file on the right side of the 1 to many relationship. This file is always subordinate to File 1.

Field	Description
Field-ID	<p>Primary key for File 2. The key can be a descriptor or superdescriptor (or compound key).</p> <p>The length of this key must be greater than or equal to the length of File 1. If the length is greater, the key must be either a superdescriptor or a redefined field. The sum of the lengths of the underlying fields that prefix the superdescriptor or redefinition must also match the length of File 1.</p>
Minimum	<p>Minimum number of occurrences of a record in File 2 for each occurrence of a record in File 1. For an intra-object relationship, specify one of the following:</p> <ul style="list-style-type: none"> <li>■ "1" for 1:1 cardinality</li> <li>■ "0" for C or CN cardinality</li> <li>■ "N" for minimum value of N for N cardinality</li> </ul> <p>The generated subprogram ensures that this minimum cardinality is satisfied.</p>
Average	<p>Average number of occurrences of a record in File 2 for each occurrence of a record in File 1. This value is not used by Natural Construct.</p>
Maximum	<p>Maximum number of occurrences of a record in File 2 for each occurrence of a record in File 1. This value defines the upper bounds of the object.</p> <p><b>Note:</b> When setting these maximum values, remember that generated objects must fit within the available memory.</p>
Update type	<p>Rules for updating the primary keys for sub-entities or testing for the existence or deletion of a sub-entity record.</p> <p>In the following example, you do not have to specify the line number for each order line because line numbers are determined by the position of the line within the Order object:</p> <pre>Object:          Order Primary File Key: Order-Number Secondary File Key: Order-Number + Order-Line-                   Number Tertiary File Key: Order-Number + Order-Line-                   Number + Distribution-Line-Number</pre> <p>In the following example, the secondary file contains contact names for the Customer object and has a key of Customer-Number + Contact-Name. Specify the suffix portion of the secondary file key:</p> <pre>Object:          Customer Primary File Key: Customer-Number Secondary File Key: Customer-Number + Contact-Name</pre> <p>The distinction between the two types of relationships is identified by the Update constraint type. For an intra-object relationship, specify one of the following codes:</p> <ul style="list-style-type: none"> <li>■ C (Cascade)</li> </ul>

Field	Description
	<p>Suffix portion of the sub-entity key represents data you must specify. The object-maintenance subprogram determines whether to save the sub-entity to the database (irrespective of the values of other non-primary key attributes) by the existence of a key suffix value.</p> <ul style="list-style-type: none"> <li>■ L (Suffix is a line number)</li> </ul> <p>Suffix portion of the sub-entity key, which is a line assigned by the object-maintenance subprogram, is determined by the occurrence of the sub-entity within the object array. The existence of any non-primary key attribute within an occurrence of the sub-entity determines whether to save the sub-entity.</p> <ul style="list-style-type: none"> <li>■ N (Renumber suffix)</li> </ul> <p>This option is similar to L, except the object-maintenance subprogram collapses empty lines (lines containing only null-valued attributes) and subsequent lines are renumbered.</p>
Delete type	Delete constraint type. For intra-object relationships, specify "C" (Cascade), since one of the properties of an object is that all sub-entities must be deleted if the primary header file is deleted.
Constraint name	This field is not used by Natural Construct.

### Inter-Object Relationships

In addition to defining relationships within objects, you can also define relationships and specify referential integrity constraints between objects. For example, you can prohibit the addition of an order for a customer who does not have a record in the Customer file (Customer object). Similarly, you can prevent the deletion of a customer who has outstanding orders in the Order file (Order object).

All referential integrity relationships must relate a foreign key of one object to a primary key of another object. The following example shows the Predict Modify File relation panel with information entered for an inter-object relationship:

```

16:54:26          ***** P r e d I c t *****
                  - Modify File relation -
File relation ... NCST-CUSTOMER-ORDER-HEADER      Modified: 13-10-21 at 13:29
Type .....* N Natural Construct                  by: DEVNG
Keys ..                                           Zoom: N

Cardinality ..* 1 : CN
File 1                                           Minimum ...
  File ID ....* NCST-CUSTOMER                    Average ... 1.00
  Field ID ...* CUSTOMER-NUMBER                  Maximum ... 1
File 2                                           Minimum ...
  File ID ....* NCST-ORDER-HEADER                Average ... 50.00
  Field ID ...* ORDER-CUSTOMER-NUMBER            Maximum ...

Constraint attributes
Update type .....* R Restrict
Delete type .....* R Restrict
Constraint name ..
Usage .....* (none)

Abstract      Zoom: N

EDIT:  Owner: N  Desc: N
    
```

The following table describes the fields on this panel and the values specified or displayed in each field for an inter-object relationship:

Field	Description
File relation	Relationship ID name. This name should not match the name of any entity within the object or any of their attributes.
Type	Relationship type. Natural Construct models process type N (Natural Construct) relationships.  <b>Note:</b> For relational databases, type R relationships are also processed.
Keys	Keywords. This is an optional field.
Cardinality	
File 1	Cardinality for File 1. For inter-object relationships, specify one of the following codes: <ul style="list-style-type: none"> <li>■ 1  If the value of the foreign key (File 2) must match the value of the primary key (File 1).</li> <li>■ C  If the value of the foreign key (File 2) can be blank, but if a value is specified, it must match the value of the primary key (File 1). In other words, only a non-null value in the foreign key needs to match the value of the primary key.</li> </ul>

Field	Description
	<p>For example, for C:N cardinality for the NCST-CUSTOMER file, it is not necessary to specify a value in the ORDER-CUSTOMER-NUMBER field in the NCST-ORDER-HEADER file. However, if a value is specified, it must have a corresponding value in the CUSTOMER-NUMBER field in the NCST-CUSTOMER file. An order does not require a customer (perhaps to signify an internal order), but if a customer is specified, it must be a valid customer.</p> <p><b>Note:</b> If the foreign key is a member of a periodic group and/or a multiple-valued field, 1:CN cardinality for File 1 implies that all occurrences are required and C:CN cardinality for File 1 implies that all occurrences are optional.</p>
File-ID	Name of the primary file for the object to which the foreign key in File 2 is related.
Field-ID	<p>Primary key for File 1. The key can be a descriptor or superdescriptor.</p> <p><b>Note:</b> For relational databases, the key can be a compound key.</p>
Minimum	Minimum number of occurrences of a record in File 1. For an inter-object relationship, specify "1" or "0" (based on the value specified for the cardinality of File 1).
Average	Average number of occurrences of a record in File 1. For an inter-object relationship, specify "1". This value is not used by Natural Construct during code generation.
Maximum	Maximum number of occurrences of a record in File 1. For an inter-object relationship, specify "1".
File 2	<p>Cardinality of File 2. For an intra-object relationship, specify one of the following codes:</p> <ul style="list-style-type: none"> <li>■ C                     <p>If a record in File 1 can be related to a maximum of one record in File 2, but a record in File 2 must be related to one record in File 1. The value of the primary key (File 1) can be blank, but if a value is specified, it must have a corresponding value in the foreign key (File 2).</p> <p>For example, for 1:C cardinality for the NCST-ORDER-HEADER file, each customer specified in the CUSTOMER-NUMBER field of the NCST-CUSTOMER file can have either zero or one order specified in the ORDER-CUSTOMER-NUMBER field in the NCST-ORDER-HEADER file.</p> <p>If an occurrence of a record in File 1 must be related to exactly one occurrence of a record in File 2 and vice versa.</p> </li> <li>■ CN                     <p>If a record in File 1 does not have to be specified, but if a record is specified, it can match zero, one, or many records in File 2 (1:CN or C:CN cardinality).</p> <p>For example, for 1:CN cardinality for the NCST-ORDER-HEADER file, a customer specified in the CUSTOMER-NUMBER field in the NCST-CUSTOMER file can have 0, 1, or many orders specified in the ORDER-CUSTOMER-NUMBER field in the NCST-ORDER-HEADER file.</p> </li> </ul>
File-ID	Name of the secondary file that has a foreign key related to File 1.



Field	Description
Field-ID	Foreign key for File 2. The length of this key must be equal to the length of the key for File 1.
Minimum	Minimum number of occurrences of a record in File 2 for each occurrence of a record in File 1. For an inter-object relationship, specify "0" in this field.
Average	This value is not used by Natural Construct.
Maximum	For an inter-object relationship, specify the maximum number of occurrences of a record in File 2 for each occurrence of a record in File 1.  <b>Note:</b> Natural Construct does not check this value to ensure it is not exceeded.
Constraint attributes	
Update type	For an inter-object relationship, specify "R" (Restrict).
Delete type	For an inter-object relationship, specify "R" (Restrict) to enforce the Restricted Delete rule (for information, see <a href="#">Support for Foreign Referential Constraints</a> ).
Constraint name	This field is not used by Natural Construct.

### Support for Foreign Referential Constraints

Natural Construct supports two types of foreign referential constraints (inter-object relationships):

- Restricted Update for Insertion (RUI) rule

When adding or updating an object entity with a foreign key related to the primary key for a related object, the action is allowed only when the value of the foreign key is either null or equal to the value of the primary key for the related object.

- Restricted Delete (RD) rule

When deleting an object entity with a primary key that is used as a foreign key in a related object, the action is allowed only when the value of the primary key does not match any values of the foreign key for the related object.

### Support for Predict Automatic Rules

Object-oriented methodology makes a distinction between data access and dialog. All data is validated and manipulated within the object-maintenance subprogram, whereas the object-maintenance dialog program communicates with the object by sending messages and parameters through the interface provided by the object parameter data area (PDA).

Since maps used by the object-maintenance dialog program do not refer to data fields directly, Natural cannot incorporate Predict automatic rules into the fields on these maps. To support object-oriented methodology, the object-maintenance subprogram incorporates the Predict automatic rules in its generated code.

### Conventions for Automatic Rules

- When an error occurs, the rule should assign either the message text or number to MSG-INFO.##MSG or MSG-INFO.##MSG-NR, respectively. Optionally, the rule should also assign dynamic substitution data to MSG-INFO.##MSG-DATA(\*) and then perform an ESCAPE ROUTINE. Natural Construct generates the code required to handle the error processing.
- Within the rule, specify the field name attached to the rule as "&1&".



**Note:** Only Predict verification rules with VE-STATUS = "N" (Natural Construct) are included.

### Features of Automatic Rules

- You can attach a rule to any field, including a multiple-valued field (MU), a periodic group (PE), or an MU within a PE. No reference to occurrences of the array is required since the rule should not have any knowledge of what type of field it is attached to. When the rule is attached to an MU or PE, Natural Construct generates additional code to handle the array.
- A rule can declare and use external local data areas (LDAs). These LDAs can be used by another rule or by the host subprogram, since Natural Construct recognizes such usage and only generates the declaration for the LDA once.
- If a rule needs inline local data, that data must belong to a level 1 structure called #PRD-*rulename*, where *rulename* is the name of the Predict rule. This is necessary to avoid naming conflicts.
- If you want to maintain all data semantic processing within Predict and reuse any generic data processing that can be shared by multiple fields within an application, automatic rules can eliminate the need for user exit routines.

## 2 Using the Object-Browse Models

---

■ Introduction .....	14
■ Object-Browse-Subp Model .....	15
■ Object-Browse-Static Model .....	31
■ Object-Browse-Dialog Model .....	36
■ Object-Browse-Dialog-Driver Model .....	94
■ Object-LDA Model .....	96
■ Object-Browse-Select-Subp Model .....	101

The Object-Browse series of models generate the modules for an object-browse process. The generated modules provide easy access to database tables in a transparent and flexible way, regardless of where data resides and whether data is used to create a mainframe screen, GUI dialog, hardcopy report, Web page, spreadsheet, or business service. The modules are also ideal when access to the data is required for validation purposes.



**Note:** For more information on object-oriented development, see [Overview of Object-Oriented Development](#).

## Introduction

---

The Object-Browse models encapsulate database calls within a Natural subprogram that provides a high-level, flexible interface for retrieving rows. Some features of the generated browse modules include:

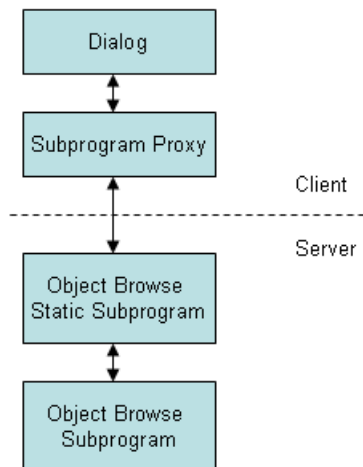
- Support for logical keys that combine multiple key components.
- Support for multiple logical keys within a single browse object.
- Support for complex wildcard handling to retrieve a range of records.
- Ability to efficiently read a large number of records and return them to the caller in manageable (configurable) blocks — without relying on the database to perform the necessary cursor management and without requiring that the connection to the browse object remain active between calls.
- Ability to browse by a repeating key (MU/PE) and by superdescriptors containing repeating components.
- Ability to begin browsing from a particular instance of a non-unique key.
- Ability to combine ascending and descending components within a single logical key.
- Ability to toggle between histogram mode and normal mode, depending on whether or not the caller wants distinct key values to be returned.
- Ability to confine a search by locking portions of the key.
- Ability to determine the most efficient record selection logic based on the specified search criteria.

The Object-Browse models generate several modules required for client/server applications. These include:

Model	Modules Generated
Object-Browse-Subp	<ul style="list-style-type: none"> <li>■ Object-browse subprogram</li> <li>■ Object parameter data area (defines returned row data)</li> <li>■ Key parameter data area (defines search key values)</li> <li>■ Restricted parameter data area (private data used internally by the browse object to maintain context)</li> </ul>
Object-Browse-Static	<ul style="list-style-type: none"> <li>■ Object-browse static subprogram</li> <li>■ Static object parameter data area</li> </ul>

The Object-Browse models also generate several data areas. For example, all object-browse subprograms access the CDPDA-M data area for message information and the CDBRPDA data area for standard parameters. For information, see [Parameters for the Object-Browse-Subp Model](#).

The following diagram illustrates how to implement the components of a browse object in a client/server configuration:



## Object-Browse-Subp Model

The Object-Browse-Subp model generates the browse subprogram for an object, as well as three parameter data areas: the object PDA (defines returned row data), key PDA (defines search key values), and the restricted PDA (private data used internally by the browse object to maintain context).

To view examples of an object-browse subprogram and its PDAs generated using the Object-Browse-Subp model, refer to CUSBRSUB, CUSBRROW, CUSBRKEY, and CUSBRPRI in the Nat-

ural Construct demo system. The demo system also contains CUSBRPGM, an example of a dialog module that calls other modules. For more information, see [Parameters for the Object-Browse-Subp Model](#).

This section covers the following topics:

- [Use Multiple Browse Keys](#)
- [Use Compound Browse Keys with Multiple Components](#)
- [Specify Minimum and Maximum Key Values](#)
- [Allow Lower Case Input Values](#)
- [Use Wildcard Characters](#)
- [Read Consecutive Sets of Records](#)
- [Position to a Specific Record](#)
- [Example of Using an Object-Browse Subprogram](#)
- [Parameters for the Object-Browse-Subp Model](#)
- [User Exits for the Object-Browse-Subp Model](#)

## Use Multiple Browse Keys

You can define a single browse object that can return records using multiple sort orders. The calling object must indicate the sort order by assigning the CDBRPDA.SORT-KEY field. If this field is not assigned, the default sort order is used (the first logical key defined in the specification).

The sort keys cannot be arbitrarily chosen. Each browse object supports a predefined number of sort key values, which are specified at generation.

The sort key values represent logical key names that may map to more than one key component. For example, a browse object may support the following key configurations for the NCST-CUSTOMER file:

Logical Sort Key	Physical Keys	Sort Order
NAME	BUSINESS-NAME	Ascending
NAME-BACKWARDS	BUSINESS-NAME	Descending
NAME-WAREHOUSE	BUSINESS-NAME	Ascending
	CUSTOMER-WAREHOUSE-ID	Ascending
CUSTOMER-NUMBER	CUSTOMER-NUMBER	Ascending
CUSTOMER-NUMBER-BACKWARDS	CUSTOMER-NUMBER	Descending

The union of all physical keys is always passed to the browse object as starting values. In this example, BUSINESS-NAME, CUSTOMER-WAREHOUSE-ID, and CUSTOMER-NUMBER are always passed to the browse object. Depending on the value of SORT-KEY, one or more of these starting values are processed by the browse object.

For an example of multiple sort keys, see [Example of Using an Object-Browse Subprogram](#).

## Use Compound Browse Keys with Multiple Components

To browse by compound keys that involve many components, define the browse object so that it requires a prefix. If you specify a logical sort key that contains many key components and do not specify a prefix or starting value, browsing begins at the first record on file and continues to the last.

You can also specify two prefixes for the sort key. Using the COMPANY and DIVISION fields as an example, users must specify exact values for these fields, and all rows returned must pertain to this company and division. While this may reduce the flexibility of the browse object, the resulting database accesses are much more efficient.



**Note:** When browsing Adabas or VSAM files by a single superdescriptor, efficiency is not affected by specifying prefix key components. When more than three components are required (and for maximum efficiency), define the logical key as a superdescriptor.

For an example of prefix keys, see [Example of Using an Object-Browse Subprogram](#).

When generating browse objects that access relational tables, there is a limit of four unbounded (non prefix) keys. If you want to browse by compound keys with more than four components, define the browse object so that it requires specific (absolute) values for the leading components. To do this, specify a number of prefix components so that the total number of key components minus the prefix components is less than or equal to four. This optimizes the generated SELECT statements.



**Note:** When browsing Adabas or VSAM files by a logical key with multiple components, the maximum number of unbounded keys permitted is three.

For example, assume a logical sort key called ACCOUNT-NUMBER that is made up of the key components: COMPANY, DIVISION, COST-CENTER, ACCOUNT-CODE, and PROJECT. If you specify this sort key for a browse object, one of the following conditions must be met:

- The key is defined as an Adabas or VSAM superdescriptor. In this case, a prefix component is not required.
- The key is defined as a compound key or a series of individual keys within a relational table. In this case, you must specify a minimum of one prefix component.
- The key is defined as a series of individual fields on an Adabas or VSAM table. In this case, you must specify at least two prefix components.

## Specify Minimum and Maximum Key Values

Programs generated using the Browse models allow you to provide a starting and ending value for the browse. The combination of the minimum and maximum keys creates a logical window within the file. The program will not browse before or after these values.

Client code for an object-browse subprogram allows the same functionality as minimum/maximum key values do for browse programs. For example, when browsing from an order header to the order lines, you can restrict the selection to that order number only. To include this functionality in an object-browse subprogram, specify the Limit components and Prefix components options in the Optional Parameters panel (see [Define Optional Parameters](#)).

You can also change the range option for the object-browse subprogram on the client (for example, you can specify a *starts with* value).

Additional minimum/maximum functionality can be coded within user exits (for example, READ-INPUT-CRITERIA).



**Tip:** Generated browse programs use only one key, which handles both ascending and descending order. This allows you to specify minimum and maximum key values. However, an object-browse subprogram handles up to six keys. If your object-browse subprogram only requires one key (or at the most two keys, one representing ascending and the other representing descending order for the same database field), you can take advantage of the Transform-Browse model to implement the minimum/maximum key functionality. For example, you can generate a browse program containing these options and then use the Transform-Browse model to transform the browse program into an object-browse subprogram. This feature, however, is not exposed on the specification panels so the values can only be changed by manually changing the \*\*SAG lines that contain these values. For more information about transforming browse modules, see *Transform-Browse Model, Natural Construct Generation*.

## Allow Lower Case Input Values

By default, Natural Construct-generated object-browse subprograms convert the starting values for all supplied alphanumeric key components (contained in the KEY parameter data area) to upper case. If the input values may include lower case characters, you can use the Predict Modify Field panel to add the ALLOW-LOWER-CASE keyword to the key components you do not want converted.

For example, if the database contains both upper case and lower case values, such as iXpress and IBM for the BUSINESS-NAME field, you can add the ALLOW-LOWER-CASE keyword to the field as follows:



```

21:08:57          ***** P R E D I C T 8.2.1 *****          2013-08-12
                    - Modify Field -
Field ID ..... BUSINESS-NAME          Modified 2013-08-12 at 20:47
File ID ..... NCST-CUSTOMER          by CND SHE1
Keys .. DESCRIPTION,ALLOW-LOWER-CASE          Zoom: N

Ty L Field ID          F Cs Length   Occ   D U DB S NAT-1 Cnv
*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  1 BUSINESS-NAME          A    30.0          D   XZ N  ←

```

Users can then search for lower case or upper case values. For example, "I\*" for iXpress and "I\*" for IBM.

You should only use the ALLOW-LOWER-CASE keyword if the data stored in the database contains lower case characters. Otherwise, Natural Construct will not convert input values that should be converted. For example, assume the database contains keys for the Canadian provinces: BC, NB, NS, ON, etc. By default, if a user enters "n" as a search value, NB and NS are returned (even though these begin with upper case "N"). If you add the ALLOW-LOWER-CASE keyword to the field, no records are returned because there are no values matching a lower case "n".

### Use Wildcard Characters

Normally, users enter a starting value in the key input field for a browse object. For all alphanumeric fields, users can specify wildcard characters to limit the range of records returned. Wildcard characters may only be used for single keys or in the last specified field of a compound key. These characters must always be the last character specified in the input field. Valid wildcard characters are:

Wildcard Character	Description
*	Returns keys that begin with the value preceding the *.
<	Returns keys that are less than the value preceding the <.
>	Returns keys that are greater than the value preceding the >.
=	Returns keys that are equal to the value preceding the =.

In addition to specifying wildcard support for the browse object, you can indicate the key value to be used as a starting value, ending value, prefix, etc., by setting a wildcard option.

## Read Consecutive Sets of Records

A browse object can be called many times to read consecutive sets of records. This ensures that the caller is returned the next set of records. This feature is achieved by means of a restricted data area, where the browse object stores data as the last sort key, the last starting value, the last row returned, etc. If the browse object is called repeatedly using the same sort key value and start values, it assumes the user wants to resume where the last call left off.

The search is automatically restarted if the user specifies a new sort key, start value, range option, etc. The user can request an explicit restart using the same search criteria.



**Note:** For more information about the restricted data area, see [Parameters Passed to the Object-Browse Subprogram](#).

## Position to a Specific Record

When reading records by non-unique key values, the user can specify a particular record from which browsing begins. To achieve this, each row must contain a unique key value (or unique combination of keys). For Adabas files, the ISN of the record is used. When starting a browse, the user can specify a starting value (SMITH, for example) and a value to uniquely identify the exact SMITH (ISN 9238). For more information, see [CDBRPDA](#).

## Example of Using an Object-Browse Subprogram

The following example uses a telephone directory to illustrate how the various features of the browse object can be combined to meet typical data access requirements. For example, a telephone company has a database consisting of the following columns:

```
1 AREA-CODE (N3)
1 CITY (A30)
1 NAME (A30)
1 STREET (A30)
1 PHONE-NUMBER (N7)
```

The browse object can be defined with the following logical keys and components:

Logical Key	Key Components
Key 1	AREA-CODE + CITY + NAME + STREET
Key 2	AREA-CODE + NAME + CITY

Since a telephone operator providing directory assistance typically deals with a single area code, the AREA-CODE can be defined as a prefix component at generation, so that all searches are restricted to one area code.

If the caller supplies a city, the operator can use Key 1 and specify the city, increasing the number of prefix components to two at runtime. Since exact values must be supplied for all prefix compon-

ents, these are normally assigned programmatically, for example, by allowing the operator to choose from a list of valid cities.

If the exact spelling of the name is known, the operator can also supply this information and designate a third prefix component. If the exact name is not known, the operator can supply a partial name and a wildcard character, for example, "THOM\*".

If the city is not known, the operator can use Key 2 to help find possible cities. If it is an Adabas or VSAM file, the Histogram option can be used to avoid returning duplicate rows.

## Parameters for the Object-Browse-Subp Model

The Object-Browse-Subp model has two specification panels: Standard Parameters and Additional Parameters. This section describes these panels. The following topics are covered:

- [Standard Parameters Panel](#)
- [Additional Parameters Panel](#)
- [Parameters Passed to the Object-Browse Subprogram](#)

### Standard Parameters Panel

The following example shows the first specification panel, the Standard Parameters panel:

```

CUBOMA          Object-Browse-Subp Multi-Module          CUBOMA0
Oct 02          Standard Parameters                      1 of 2

Module ..... OBJBRSUB
System ..... CST821S_____

Title ..... Object Browse ..._____
Description ..... This subprogram is used to encapsulate data access_____
                    for ..._____
                    _____
                    _____

Message numbers .... _

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

```

This panel is similar for all models. For a description of this panel, see *Common Fields on the Standard Parameters Panel*.

### Additional Parameters Panel

The following example shows the second specification panel, the Additional Parameters panel:

```

CUBOMB          Object-Browse-Subp Multi-Module          CUBOMB0
Oct 02          Additional Parameters                    2 of 2

Predict view ..... _____ *
Natural (DDM) ..... _____
Program view ..... _____

Logical keys          Key components          Option
1 _____          _____          *      -
2 _____          _____          *      -
3 _____          _____          *      -
4 _____          _____          *      -
5 _____          _____          *      -
6 _____          _____          *      -

Object PDA ..... OBJBRROW *          X          Source          Object
Key PDA ..... OBJBRKEY *          X
Restricted PDA ..... OBJBRPRI *          X

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

Use this panel to specify additional parameters for your object-browse subprogram. The fields on this panel are:

Field	Description
Predict view	<p>Name of the Predict view used by the browse subprogram. The specified view must be defined in Predict. This view determines which fields are generated into the row PDA returned to the caller.</p> <p>After selecting the view, the first six key components in the view are displayed.</p> <p><b>Tip:</b> If you only want to use a subset of fields in the DDM, create a Predict view that only contains these fields. The generated object-browse subprogram and PDAs will contain this subset of fields.</p>
Natural (DDM)	<p>Name of the data definition module (DDM) that corresponds to the primary file. If you do not specify a DDM, the DDM name defaults to the primary file name.</p> <p>The Predict definition of the primary file determines which fields are included in the DEFINE DATA section of the generated code. The format of the generated code in the DEFINE DATA section has the following structure:</p> <pre> 1 Primary-file-name VIEW OF Data-definition-module 2 fields pulled from Predict of Primary-file-name                     </pre>

Field	Description
Program view	<p>Name of the view for the primary file in the object-browse subprogram. If you specify this parameter, you must define the view in the LOCAL-DATA user exit or a local data area (LDA). User-defined views must contain all fields defined as key components, as well as a field that uniquely identifies a record.</p> <p>If you do not specify a program name, a view is automatically generated containing all fields in the Predict view. The MAX. OCCURS value in Predict determines how many occurrences of MU/PE fields are included.</p>
Logical keys	<p>Key(s) that determines the sort order of the returned records. You can define a browse object that returns records using up to six sort orders. The calling program indicates the sort order by assigning CDBRPDA.SORT-KEY. If a sort key value is not assigned, the first logical key is used as the default.</p> <p>The logical key names can map to as many as five components. If a logical key contains only one component, the logical key name is optional. If you do not specify a logical key name, this field defaults to the name of the key component.</p> <p>If the key field contains MU or PE fields, the rows returned also contain an index value that identifies which occurrence of the MU/PE field satisfies the read condition.</p>
Key components	<p>Key components for a logical key that maps to more than one component. You can add key components and set options in the Optional Parameters window. To display this window, press PF5 (optns) or mark the Option field and select Enter.</p> <p>Using this window, you can add as many as five components and specify options for them. For information, see <a href="#">Define Optional Parameters</a>.</p> <p>To define key components, specify either one superdescriptor or multiple individual descriptors.</p> <p><b>Note:</b> When browsing by non-unique keys, performance will deteriorate with each call to the browse object that returns keys identical to the previous call. Avoid defining keys for which a large number of records correspond to each key value. For example, a key consisting of CITY + NAME results in more efficient access than just CITY alone.</p>
Option	<p>A plus sign (+) indicates that options have been set for the logical key. To display the Optional Parameters window, press PF5 (optns) or mark the Option field and select Enter. For more information, see <a href="#">Define Optional Parameters</a>.</p>
Object PDA	<p>Object PDA that defines the rows returned to the browse object and the columns within each row. The generated object PDA contains one column for each field defined in the specified Predict view (as well as additional columns). You can edit the generated object PDA to alter the list of fields returned by the subprogram.</p> <p>When creating a new specification, this field is filled in by default with the first five bytes of the module name, plus the suffix "ROW".</p> <p>For more information, see <a href="#">Object PDA</a>.</p>

Field	Description
Key PDA	<p>Key PDA that contains all of the components contained in the logical keys, as well as a unique ID field.</p> <p>When creating a new specification, this field is filled in by default with the first five bytes of the module name, plus the suffix "KEY".</p> <p>For more information, see <a href="#">Key PDA (Search Key)</a>.</p>
Restricted PDA	<p>Restricted PDA that stores data, such as the last sort key, the last starting value, the last row returned, etc. so that the next set of consecutive records is returned to the caller. The contents of this data area should not be altered by the calling module.</p> <p>When creating a new specification, this field is filled in by default with the first five bytes of the module name, plus the suffix "PRI".</p> <p>For more information, see <a href="#">Restricted PDA</a>.</p>
Generate	<p>If the PDA source and/or object code is found in the Natural steplib chain, this field displays the name of the library where the source and/or object code is located. If a generated PDA is not found, it is generated by default (and this field is marked and protected).</p>
Source	<p>Name of the first library in which the source code for the module is found. The source code may exist in multiple libraries in the Natural steplib chain.</p> <p>If the source code resides in the current library, regenerating it will execute a STOW command and overwrite the previous version.</p>
Object	<p>Name of the first library in which the object code for the module is found. The object code for the module may exist in multiple libraries in the Natural steplib chain.</p> <p>If the object code resides in the current library, regenerating it will execute a STOW command and overwrite the previous version.</p> <p><b>Note:</b> If the Generate field is marked, the PDAs specified in this window are generated and stowed when the object-browse subprogram is generated — regardless of whether the subprogram is stowed.</p>

### Define Optional Parameters

You can define optional parameters for your object-browse subprogram.

#### ► To define optional parameters, either:

- 1 Press PF5 (optns) on the Additional Parameters panel.

Or:

Mark the Option field for a logical key and select Enter.

The Optional Parameters window is displayed. For example:

```

CUBOMBA                               Natural Construct                               CUBOMBA0 ↵
Sep 12                                 Optional Parameters                               1 of ↵
1
Logical keys                            ↵
>> 1 _____ ↵
Key components                            Descending ↵
PERSONNEL-ID_____ *                    _ ↵
_____ *                                _ ↵
_____ *                                _ ↵
_____ *                                _ ↵
_____ *                                _ ↵
Histogram support ..... _ ↵
Limit components ..... _ ↵
Prefix components ..... _ ↵
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF ↵
help retrn quit                          bkwrđ frwrđ ↵
ma ↵

```



**Note:** If you marked the Option field for a logical key, that key is displayed in this window. To view other logical keys, press PF8 (frwrđ) and PF7 (bkwrđ).

- 2 Use the following fields to define the additional parameters:

Field	Description
Logical keys	Name of the logical key for which you are defining options.
>> 1	Scroll indicator. You can enter the number of a logical key in this field to scroll to that key.
Key components	Components of the key. Each logical key can have up to five components.

Field	Description
Descending	<p>To have the key component values listed in descending sequence in the generated browse, mark this option. Otherwise, values are sorted in ascending sequence.</p> <p><b>Note:</b> For Adabas and VSAM files, all components of a logical key must use the same sort order.</p>
Histogram support	<p>If this parameter is marked, the browse has an additional histogram version of one or more logical key values. This allows the calling program to request a histogram be returned. Rather than returning all of the predefined columns of the browse object, only the specific key column is returned, along with a count of the number of records containing the key value.</p> <p><b>Note:</b> This option is only allowed when the associated key has one key component.</p>
Limit components	<p>Number of components of a superdescriptor (compound key) used in the logical key (if the relational database table contains a superdescriptor with many components).</p> <p>To restrict the number of components, specify the limit in this field. For example, to use the first two components of the superdescriptor, enter "2".</p> <p><b>Note:</b> Using fewer components in the key may make accessing the key more efficient.</p>
Prefix components	<p>Prefix used for components of a superdescriptor (compound key).</p> <p>When browsing by compound keys that have many components, you can define the browse object so that it requires specific values for the leading components. This optimizes the generated SELECT statements.</p> <p><b>Note:</b></p> <ol style="list-style-type: none"> <li>1. For more information about prefix components, see <a href="#">Use Compound Browse Keys with Multiple Components</a>.</li> <li>2. When browsing Adabas or VSAM files by a single superdescriptor, efficiency is not affected by specifying prefix key components.</li> </ol>

### Parameters Passed to the Object-Browse Subprogram

Generated object-browse subprograms accept parameters from the following parameter data areas (PDAs):

```

DEFINE DATA
  PARAMETER USING CUSBRKEY /* Search key values
  PARAMETER USING CUSBRROW /* Returned row data
  PARAMETER USING CUSBRPRI /* Private (restricted) data
  PARAMETER USING CDBRPDA /* Generic browse object parameters
  PARAMETER USING CDPDA-M /* Msg info
END-DEFINE

```



These data areas are described in the following sections:

- Key PDA (Search Key)
- Object PDA
- Restricted PDA
- CDBRPDA
- CDPDA-M

### Key PDA (Search Key)

Each browse object can support multiple logical keys, where a logical key is comprised of one or more key components. The key PDA defines the union of all fields that are components of a logical key. Additionally, the generated key PDA contains a field that can be used to begin the browse at a specific record.

The logical key definitions and sort order for the sample key PDA are:

Logical Sort Key	Physical Key(s)	Sort Order
NAME	BUSINESS-NAME	Ascending
NAME-BACKWARDS	BUSINESS-NAME	Descending
NAME-WAREHOUSE	BUSINESS-NAME	Ascending
	CUSTOMER-WAREHOUSE-ID	Ascending
CUSTOMER-NUMBER	CUSTOMER-NUMBER	Ascending
CUSTOMER-NUMBER-BACKWARDS	CUSTOMER-NUMBER	Descending

The following example shows the sample key PDA:

```

1 CUSBRKEY                               /* All key components
 2 BUSINESS-NAME                           A   20
 2 CUSTOMER-WAREHOUSE-ID                   A   3
 2 CUSTOMER-NUMBER                         N   5
 2 UNIQUE-ID                               P  10 /* Unique row id      ↵

```

When calling the browse object, the caller can assign starting values for the fields that make up the sort key. The range option (described in [CDBRPDA](#)) determines whether the specified key value represents a starting value, ending value, exact value, or prefix. By default (when range option = 0), the specified value is assumed to be a starting value for columns that are sorted in ascending sequence and an ending value for columns that are sorted in descending sequence.

The default range option also allows users to specify wildcard characters (for example, <, >, \*, and =).

## Object PDA

The object PDA contains one field for each field defined in the specified Predict view. These fields are defined within a 1:V structure so the browse object can support an arbitrary number of return rows. The following example shows a generated object PDA:

```

1 CUSBRROW
2 ROW                                     (1:V)
3 CUSTOMER-NUMBER                       N    5
3 BUSINESS-NAME                         A   30
3 PHONE-NUMBER                          N   10
3 MAILING-ADDRESS
4 M-STREET                              A   25
4 M-CITY                                 A   20
4 M-PROVINCE                            A   20
4 M-POSTAL-CODE                         A    6
3 SHIPPING-ADDRESS
4 S-STREET                              A   25
4 S-CITY                                 A   20
4 S-PROVINCE                            A   20
4 S-POSTAL-CODE                         A    6
3 CONTACT                               A   30
3 CREDIT-RATING                         A    3
3 CREDIT-LIMIT                          P 11.2
3 DISCOUNT-PERCENTAGE                 P   3.2
3 CUSTOMER-WAREHOUSE-ID                 A    3
3 CUSTOMER-TIMESTAMP                   T
3 COUNT                                 I    4
3 UNIQUE-ID                             P   10

```



**Note:** The Predict STRUCT parameter indicates whether to generate periodic groups with the occurrences at the structure or individual field level.

In addition to fields defined in the Predict view, the object PDA contains an additional field called UNIQUE-ID. This field uniquely identifies a row in the object PDA. If the row is from an Adabas file, the record ISN uniquely identifies the file. For other file types, the primary key is used. Some additional fields may be generated into the object PDA, depending on the keys and options selected. These fields are:

### ■ COUNT

If one or more keys support the Histogram option, the COUNT field is added to the object PDA. This field is assigned the number of rows that match the returned key value during Histogram processing.

### ■ MU-PE-MATCH-INDEX

If one or more keys is a multiple-valued (MU) field and/or an element of a periodic group (PE) or a superdescriptor containing such a field, the MU-PE-MATCH-INDEX field is added to the

object PDA. The contents of the field indicate which occurrence of the repeating field resulted in the current row being returned.

### Restricted PDA

To determine state information, all browse objects are passed information that the object PDA maintains internally. This data should not be disturbed by the caller.

### CDBRPDA

The fields in this data area control the behavior of the browse object. The following table shows the structure of the CDBRPDA parameter data area:

Field	Description
1 CDBRPDA	
2 INPUTS	
3 METHOD (N1)	Currently not used, always pass 0.
3 SORT-KEY (A32)	Name of the logical key for the browse sort order. Starting values may be supplied for the physical keys that make up this key. If no sort key is provided, the first defined logical key is used.
3 HISTOGRAM (L)	If this field is true, only distinct key values are returned, along with a count of the number of records having the key. This option is ignored if the browse does not support a histogram for the specified sort key.
3 LEADING-FIXED-COMPONENTS (N2)	Override for the default number of leading fixed key values for the logical key.  To increase the default number of leading fixed key values for the logical key, specify the number in this field. All key values supplied up to the specified number of components must match the value in the return row.
3 ROWS-REQUESTED (N4)	Maximum number of rows requested to be returned for the current call. This number must be less than or equal to the number of rows allocated.
3 RANGE-OPTION (N2)	<ul style="list-style-type: none"> <li>■ 0 = DEFAULT (embedded wildcard in key)</li> <li>■ 1 = LESS-THAN</li> <li>■ 2 = LESS-THAN-OR-EQUAL</li> <li>■ 3 = EQUAL</li> <li>■ 4 = GREATER-THAN-OR-EQUAL</li> <li>■ 5 = GREATER-THAN</li> <li>■ 6 = BEGINS-WITH</li> </ul>

Field	Description
	<b>Note:</b> For relational database searches, increasing this number (and thereby specifying more absolute values) will improve the performance of the search.
3 USE-UNIQUE-ID (L)	<p>If this flag is set, browsing by a non-unique key begins at a specified record.</p> <p>You can specify from which record to begin browsing. For example, if browsing the NCST-CUSTOMER file by BUSINESS-NAME, you can specify to begin at the SMITH with ISN 1234. All prior SMITH records are not returned.</p> <p>This feature is primarily used to simulate backward scrolling. For non-Adabas files, the primary key determines uniqueness. If there is no primary key, the record sequence is used.</p>
2 INPUT-OUTPUTS	
3 RESTART (L)	If this flag is set, the browse object begins a new browse even though the starting key may not have changed. This field is reset by the called browse object.
2 OUTPUTS	
3 ACTUAL-ROWS-RETURNED (N4)	Number of rows returned. This number will be less than or equal to the ROWS-REQUESTED value.
3 END-OF-DATA (L)	This flag is set when all rows in the database matching the selection criteria have been displayed.

**CDPDA-M**

This parameter data area contains standard message information for the browse object.

**User Exits for the Object-Browse-Subp Model**

The following example shows the User Exits panel for the Object-Browse-Subp model:

CSGSAMPL Aug 18	OBJECT-BROWSE-SUBP User Exits	Multi-Module User Exits	Sample	Required	Conditional	CSGSMO 1 of 1
User Exits		Exists	Sample	Required	Conditional	
—	NAT-DOCS				X	
—	CHANGE-HISTORY		Subprogram			
—	PARAMETER-DATA		Example			
—	LOCAL-DATA					
—	START-OF-PROGRAM					
—	READ-INPUT-CRITERIA		Subprogram		X	
—	END-OF-PROGRAM					
—	ADDITIONAL-INITIALIZATIONS		Example			
—	BEFORE-ROW-ASSIGNMENT		Example			
—	AFTER-ROW-ASSIGNMENT		Example			
—	SELECT-STATEMENTS		Subprogram		X	
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---						
help retrn quit			bkwrđ frwrđ			↩



#### Notes:

1. For information about these user exits, see *User Exits for the Generation Models, Natural Construct Generation*.
2. For information about the User Exit editor, see *User Exit Editor, Natural Construct Generation*.

## Object-Browse-Static Model

You can create a browse object as part of a client/server application. This object communicates with the server via a remote procedure call (RPC).

### ▶ To create a browse object that communicates via a remote procedure call (RPC):

- 1 Create the static object-browse subprogram.
- 2 Create the RPC subprogram.

Because the RPC subprogram cannot process parameter values having occurrences defined using 1:V, create at least one static subprogram and object PDA to replace 1:V with a fixed-length value for the ROW parameter. To generate the static subprogram and its object PDA, use the Object-Browse-Static model.

The following example shows a subprogram generated by the Object-Browse-Static model:

```

>
> + Subprogram CUSBR020 Lib SYSCSTDE
Top  ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
0010 **SAG GENERATOR: OBJECT-BROWSE-STATIC          VERSION: 8.2.1
0020 **SAG TITLE: Object Browse Static ...
0030 **SAG SYSTEM: SYSCSTDE
0040 **SAG DESC(1): Object Browse Static for ....
0050 **SAG SUBPROGRAM-NAME: CUSBRSUB
0060 **SAG STATIC-OBJECT-PDA: CUSBP020
0070 **SAG STATIC-OCCURRENCES: 00020
0080 *****
0090 * Program   : CUSBR020
0100 * System    : SYSCSTDE
0110 * Title     : Object Browse Static ...
0120 * Generated: Oct 13,13 at 01:03 PM by SAG
0130 * Function  : Object Browse Static for ....
0140 *
0150 *
0160 *
0170 * History
0180 *****
0190 DEFINE DATA
0200     PARAMETER USING CUSBRKEY
0210     PARAMETER USING CUSBP020
0220     PARAMETER USING CUSBRPRI
0230     PARAMETER USING CDBRPDA
0240     PARAMETER USING CDPDA-M
0250 LOCAL
0260     01 #PROGRAM (A8)
0270 END-DEFINE
0280 PROG.      /* to allow escape from routine.
0290 REPEAT
0300 *
0310 PERFORM INITIALIZATIONS
0320     CALLNAT 'CUSBRSUB'
0330             CUSBRKEY
0340             CUSBRROW
0350             CUSBRPRI
0360             CDBRPDA
0370             MSG-INFO
0380 *
0390 *****
0400 DEFINE SUBROUTINE INITIALIZATIONS
0410 *****
0420 *
0430     ASSIGN #PROGRAM = *PROGRAM
0440 *
0450 END-SUBROUTINE /* INITIALIZATIONS
0460 *
0470 ESCAPE BOTTOM(PROG.) IMMEDIATE
0480 END-REPEAT /* PROG.
0490 END

```

The following example shows an object PDA generated by the Object-Browse-Static model:

1	CUSBRROW		
2	ROW	(1:20)	
3	CUSTOMER-NUMBER	N	5
3	BUSINESS-NAME	A	30
3	PHONE-NUMBER	N	10
3	MAILING-ADDRESS		
4	M-STREET	A	25
4	M-CITY	A	20
4	M-PROVINCE	A	20
4	M-POSTAL-CODE	A	6
3	SHIPPING-ADDRESS		
4	S-STREET	A	25
4	S-CITY	A	20
4	M-STREET	A	25
4	M-CITY	A	20
4	M-PROVINCE	A	20
4	M-POSTAL-CODE	A	6
3	SHIPPING-ADDRESS		
4	S-STREET	A	25
4	S-CITY	A	20
4	S-PROVINCE	A	20
4	S-POSTAL-CODE	A	6
3	CONTACT	A	30
3	CREDIT-RATING	A	3
3	CREDIT-LIMIT	P	11.2
3	DISCOUNT-PERCENTAGE	P	3.2
3	CUSTOMER-WAREHOUSE-ID	A	3
3	CUSTOMER-TIMESTAMP	T	
3	COUNT	I	4
3	UNIQUE-ID	P	10



**Note:** To view the specifications for these examples, refer to the CUSBRSUB (Object-Browse-Subp) and CUSBR020 (Object-Browse-Static) subprograms in the Natural Construct demo system.

This section covers the following topics:

- [Parameters for the Object-Browse-Static Model](#)

- [User Exits for the Object-Browse-Static Model](#)

### Parameters for the Object-Browse-Static Model

The Object-Browse-Static model has one specification panel, the Standard Parameters panel.



**Note:** An object-browse subprogram must exist before you can create its corresponding static object-browse subprogram.

### Standard Parameters Panel

The following example shows the only specification panel for the Object-Browse-Static model:

```

CUBRMA          Object-Browse-Static Subprogram          CUBRMA0
Oct 02          Standard Parameters                      1 of 1

Module .....
System ..... CST821S_____

Title ..... Object Browse Static ..._
Description ..... Object Browse Static for ...._____
_____
_____

Object Browse Subp . _____ *      Source      Object
Object PDA .....

Static Occurrences . 10_ _____   Generate   Source      Object
                                PDA           _

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

The fields in the upper portion of this panel are similar for all models. For a description of these fields, see *Common Fields on the Standard Parameters Panel*.

The fields in the lower portion of this panel are:



Field	Description
Object Browse Subp	Name of the object-browse subprogram called by the static object-browse subprogram. The object-browse subprogram must currently exist.  After you enter the object-browse subprogram name, Natural Construct fills in the default values.
Object PDA	Name of the object PDA for the object-browse subprogram. Natural Construct determines this name based on the specified object-browse subprogram name.
Source	Name of the first library in which source code for the module is found. The source code may exist in multiple libraries in the Natural steplib chain.
Object	Name of the first library in which object code for the module is found. The object code may exist in multiple libraries in the Natural steplib chain.
Static Occurrences	Number of occurrences the static object-browse subprogram returns. By default, it returns 10 occurrences.
PDA	Name of the static object PDA for the object-browse subprogram. By default, Natural Construct creates a name composed of the first four bytes of the module name, plus "Pnnn", where <i>nnn</i> is the number of occurrences.  If the specified static object PDA already exists in the current library, an additional field is displayed, which you can mark to have the module regenerated. If the static object PDA does not exist, it is automatically generated.
Generate	Mark this field to generate the PDA. If the PDA does not already exist, this field is marked by default.
Source	Name of the first library in which source code for the module is found. The source code may exist in multiple libraries in the Natural steplib chain.  If the source code resides in the current library and you regenerate the module, the module is stowed and the previous version is overwritten.
Object	Name of the first library in which object code for the module is found. The object code may exist in multiple libraries in the Natural steplib chain.  If the object code resides in the current library and you regenerate the module, the module is stowed and the previous version is overwritten.

### User Exits for the Object-Browse-Static Model

The following example shows the User Exits panel for the Object-Browse-Static model:

```

CSGSAMPL          Object-Browse-Static Subprogram          CSGSM0
Oct 02              User Exits                               1 of 1

          User Exits          Exists          Sample          Required Conditional
-----
- NAT-DOCS                                     X
- CHANGE-HISTORY                               Subprogram
- PARAMETER-DATA                               Example
- LOCAL-DATA                                   Example
- START-OF-PROGRAM
- ADDITIONAL-INITIALIZATIONS                   Example
- END-OF-PROGRAM

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



**Notes:**

1. For information about these user exits, see *User Exits for the Generation Models, Natural Construct Generation*.
2. For information about the User Exit editor, see *User Exit Editor, Natural Construct Generation*.

## Object-Browse-Dialog Model

Using the Object-Browse-Dialog model, you can create character-based dialogs to work with your object-browse subprograms. These dialogs provide an interim step in the progression from centralized to client/server applications. You can create object subprograms that users access from a character-based dialog. These same object subprograms can be accessed from a GUI client using Construct Spectrum or another client/server technology.

The Object-Browse-Dialog model preserves most of the functionality of the Browse and Browse-Select series of models. When used with other object-browse models (Object-Browse-Dialog-Driver and Object-Browse-Subp, for example), this model creates browse programs, subprograms, or help routines — with or without selection capabilities.

In addition, the Object-Browse-Dialog model:

- Provides user exit modules to support the regeneration of user exits. User exit models generate text member modules you can use with the User-Exit statement model to generate user exits. For information, see [User Exit Models](#).
- Supports international applications. The model can generate text in any language for which translations are available in the SYSERR library. SYSERR references are replaced by text dynamically at runtime. The model can also generate modules that support cursor translation, which allows users to translate prompts and headings while running the generated application. For

information, see *Define International Parameters* and *Use SYSERR References for Headings and Prompts*.

- Contains mechanisms for handling PF-key and action parameters, which allow more flexibility when customizing applications.

This section covers the following topics:

- [Uses for Object-Browse Dialogs](#)
- [Change the Default PF-key Style](#)
- [Example of a Generated Object-Browse Dialog](#)
- [Parameters for the Object-Browse-Dialog Model](#)
- [User Exits for the Object-Browse-Dialog Model](#)

### Uses for Object-Browse Dialogs

Object-browse dialogs display record values on the screen. The record values are retrieved by an object-browse subprogram. Users can scroll backward, forward, left, and right when using the generated dialogs. These dialogs are useful to an application as:

- Stand-alone programs invoked from a menu by an object-browse dialog driver.
- Active help routines invoked by an object-browse dialog driver to display a list of valid values for a field and allow the user to select one.
- Subprograms called from another program, such as a maintenance program.

When a selection column is added to the dialog, users can:

- Select multiple records on the same panel.
- Apply actions to the selected records.

### Export and Report Options

Object-browse dialogs support export and report functions. The export data function writes records with delimiters to an ASCII file and downloads the file to a user's PC, where it can be inserted into a spreadsheet or word processor program. (This feature requires Entire Connection.) The report data function sends records to a local printer.

The layout of the export and report functions are independent of the screen layout. You can design and generate the layouts using the EXPORT-DATA-FIELDS, INPUT-KEY, REPORT-DATA-FIELDS, and WRITE-DATA-FIELDS user exits (available by entering SAMPLE in the User Exit editor). Or you can use the user exit models to create regeneratable text members you can generate into the User Exit editor using the User-Exit statement model. For more information, see *User Exit Models* and *User-Exit Statement Model*.

## Change the Default PF-key Style

A generated object-browse dialog module and browse module use different styles of PF-keys. To use the browse style of PF-keys in the generation of all new object-browse dialogs, remove the comment indicators from the following line in the CUBDC subprogram:

```
CUBDPDA.#PDAX-USE-BROWSE-PFKEYS := TRUE
```

To use the browse style of PF-keys for existing object-browse dialogs:

1. Remove the comment indicators from the following line in the CUBDR subprogram:

```
CUBDPDA.#PDAX-USE-BROWSE-PFKEYS := TRUE
```

2. Use the SYSMAIN utility to copy the object code for CUBDR to the SYSLIBS library.
3. Regenerate all object-browse dialogs using the NCSTBGEN batch generator.



**Note:** Although you can use the same procedure to define object-browse dialog style PF-keys for all transformed modules (by specifying CUBDPDA.#PDAX-USE-BROWSE-PFKEYS := FALSE), this may interfere with the specifications in the user exit code.

## Example of a Generated Object-Browse Dialog

The following example shows the first panel of a typical browse panel generated using the Object-Browse-Dialog model:

Product	Description	Reorder point	Unit cost
111111	DOG FOOD	5000	110.00
111116	CHEESE DOODLE	70	0.15
145688	HOT CHOCOLATE DRINK	300	10.00
187361	CAT NUGGETS	70	150.00
199210	COOPER GLOVES	50	100.00
222222	BIRD SEED	88	50.00
256733	OATS AND BARLEY CEREAL	22	20.00
324597	COOPER GLOVES	100	100.00
333333	DOG BONES	22	50.00
335977	DOMESTIC KITTY LITTER	40	7.00
342723	ORANGE DRINK CRYSTALS	4000	15.00
444444	CORN FLAKES	1000	1.22

Product .... \_\_\_\_\_  
 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF  
 help retrn quit exptr bkwrdr frwrdr reprt left right ma ←

To reposition the data, users enter a value in the input field near the bottom of the panel (Product). If the key is alphanumeric or numeric, users can include a wildcard character (\*, >, or <) to limit the range of records displayed.

Modules generated using the Object-Browse-Dialog model support an export PF-key, which routes a delimited report to an ASCII file on the user's PC. These modules also support a report PF-key, which routes a report to a printer. Users can use these PF-keys, along with wildcard characters, to generate a report based on a range of records. Export data and report data support are optional.

To view the specifications for the Object-Browse-Dialog model example, refer to the NCPRDOBD program in the Natural Construct demo system.

### Parameters for the Object-Browse-Dialog Model

The Object-Browse-Dialog model accepts parameters from the following parameter data areas:

Data Area	Description
Key PDA	Parameter data area (PDA) containing all fields that are components of the logical keys supported by the object-browse subprogram. Additionally, the generated key PDA contains a field you can use to request that the browse begin from a specific record.  You can also use the key PDA to return the selected value in the PROCESS-SELECTED-RECORD user exit.
CDBRPDA	PDA containing fields that control the behavior of the object-browse subprogram, such as the sort key, the numbers of records requested, the range option, and the actual number of records returned. For information, see <a href="#">CDBRPDA</a> .
CDPDA-D	External PDA containing standard parameters for dialogs.
CDPDA-M	External PDA containing standard parameters for exchanging message information.
CDPDA-P	External PDA containing global information shared with the object-browse subprogram.



**Note:** You can specify additional parameters in the PARAMETER-DATA user exit.

The Object-Browse-Dialog model has two specification panels: Standard Parameters and Additional Parameters. This section describes these panels. The following topics are covered:

- [Standard Parameters Panel](#)
- [Additional Parameters Panel](#)
- [Define PF-Keys](#)

▪ Define Actions

**Standard Parameters Panel**

The following example shows the first specification panel:

```

CUBDMA          OBJECT-BROWSE-DIALOG Subprogram          CUBDMA0
Mar 30          Standard Parameters                      1 of 2

Module ..... TEST_____
System ..... DEMO_____

Title ..... Object Browse Dialog for_
Description ..... This dialog is used for the object browse ..._____
_____
_____
_____

First heading ..... _____ *
Second heading ..... _____ *


Do not populate first input screen ..... _
Generate page title (when not on input map) ..... _

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

The fields in the upper portion of this panel are similar for all models. For a description of these fields, see *Common Fields on the Standard Parameters Panel*.

The fields in the lower portion of this panel are:

Field	Description
Do not populate first input screen	Indicates whether default data is automatically entered in the fields on the first input screen. This option can be used to provide consistency between Natural Construct-generated browse and object-browse modules. By default, the first input screen is not populated for a generated browse module and is populated for a generated object-browse-dialog module.
Generate page title (when not on input map)	Indicates whether to automatically code the page title when a map is not being used. By default, the page title is coded for a generated browse module and is not coded for a generated object-browse-dialog module.

 **Note:** These fields are used by the Transform-Browse model to determine how to transform a Natural Construct-generated browse module into object-browse modules. For information

about transforming browse modules, see *Transform-Browse Model, Natural Construct Generation*.

## Define or Customize PF-Keys

► To define or customize the PF-keys used by your object-browse dialog:

- 1 Press PF6 (pfkey) on the Standard Parameters panel.

The PF-Key Parameters window is displayed:

```

CUBDMAB                               Natural Construct                               CUBDMAB0
Dec 19                                PF-Key Parameters                               1 of 1

Key ID template .... MC 1___ *   Object browse dialog template

Key ID          NAMED          Subprogram          Subroutine
__ __          PF1 help          _____          _____
__ __          PF2 retrn          _____          _____
__ __ *        PF3 quit          _____ *          _____
__ __ *        PF4 trans          _____ *          _____
__ __ *        PF5 actns          _____ *          _____
__ __ *        PF6 exprt          _____ *          _____
__ __          PF7 bkwr          _____          _____
__ __          PF8 frwr          _____          _____
__ __ *        PF9 reprt          _____ *          _____
__ __ *        PF10 left          _____ *          _____
__ __ *        PF11 right          _____ *          _____
__ __ *        PF12 main          _____ *          _____

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

```

- 2 Use this window to do one of the following:
  - Specify the name of a key template containing a pre-defined group of single PF-keys
  - Override the default PF-keys by specifying individual PF-keys, their positions, and the names of the subprograms or subroutines that perform the keys functions
- 3 Select Enter to confirm the changes.

## Specify a Key Template

► To specify a key template, do one of the following:

- Type the name of a Natural Construct or application-specific key template in the Key ID template field and select Enter.

Or:

Press the help PF-key when the cursor is in the Key ID template field to select from a list of available key templates (both Natural Construct and application-specific).

### Override the Default PF-Keys

#### ▶ To override the default single PF-keys:

- 1 Type the ID for the single PF-key in the PF-key ID field associated with the appropriate position on the PF-key line.

Or:

Press the help PF-key when the cursor is in the PF-key ID field associated with the appropriate position on the PF-key line to select from a list of single PF-keys (both Natural Construct and application-specific).

- 2 In the corresponding Subprogram or Subroutine field, type the name of the subprogram or subroutine that is invoked when the key is pressed.
- 3 Select Enter.

### Change the Reserved PF-Keys

You cannot change the reserved PF-keys: PF1 (help), PF2 (retrn), PF7 (bkwrđ), and PF8 (frwrđ).

### Use the Null PF-Key

The null PF-key (PF0) disables the default PF-key in the position to which you assign it. Use this key when you want to disable the functionality of a PF-key and not replace it with another function. For example, to disable PF12 (main), enter "PF0 null" in the corresponding Key ID field in the PF-Key Parameters window.

### Define International Parameters

To define international parameters for your object-browse dialog, press PF9 (intl) on the Standard Parameters panel. The International Parameters window is displayed. For a description of this window, see [Define International Parameters](#).



## Additional Parameters Panel

The following example shows the second specification panel:

```

CUBDMB          Object-Browse-Dialog Subprogram          CUBDMB0
Jan 29          Additional Parameters                    2 of 2

                                Source      Object

Object browse subp ..... _____ *
  Object PDA ..... _____ *
  Key PDA ..... _____ *
  Object LDA ..... _____ *

Input key ..... _____ *
  Prompt ..... _____ *

Records displayed ..... 10__
Selection column format .. _ __ *
Input using map ..... _____ *
Horizontal panels ..... 1_
Backward scroll pages .... 10

Export data support ..... _
Report data support ..... _
Use BROWSE-SELECT actions. _

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

```

Use this panel to specify additional parameters and options for the browse dialog. The fields on this panel are:

Field	Description
Object browse subp	Name of the object-browse subprogram used by the generated module to retrieve records for display. Field-level help is available to select an object-browse subprogram.  <b>Note:</b> Use the Object-Browse-Subp model to generate the object-browse subprogram.
Object PDA	Name of the object parameter data area (PDA) used by the generated module. Natural Construct supplies this value by default, based on the name of the object-browse subprogram. Field-level help is available to select an object PDA.
Key PDA	Name of the key parameter data area (PDA) used by the generated module. The key PDA is comprised of all fields that are components of the logical keys supported by the specified object-browse subprogram. Natural Construct supplies this value by default, based on the name of the object-browse subprogram. Field-level help is available to select a key PDA.
Object LDA	Name of the object local data area (LDA) used by the generated module. The object LDA contains the default field headings used when generating user exits. Field-level help is available to select an object LDA.

Field	Description
Input key	Name of the logical key by which scrolling takes place in the generated module. The specified key must be defined in the key PDA. Field-level help is available to select an input key.
Prompt	Field prompt (name) displayed for the input key when the module is invoked. This name can either be text or a SYSERR reference (with or without formatting). Field-level help is available to select an existing SYSERR number or create a new SYSERR entry. For more information, see <a href="#">Use SYSERR References for Headings and Prompts</a> .
Records displayed	Number of records displayed on the screen at one time. By default, the generated module displays 10 records at one time.
Selection column format	Indicates the format for the selection column. The following formats are available: <ul style="list-style-type: none"> <li>■ A (alphanumeric) 1–253 units</li> <li>■ B (binary) 1–126 units</li> <li>■ C (attribute control)</li> <li>■ D (date)</li> <li>■ F (floating point) 4, 8 units</li> <li>■ I (integer) 1, 2, 4 units</li> <li>■ L (logical)</li> <li>■ N (numeric, unpacked) 1–29 units, 1–7 decimals</li> <li>■ P (numeric, packed) 1–29 units, 1–7 decimals</li> <li>■ T (time)</li> </ul>
Input using map	Name of the layout map used by the generated module. Natural Construct supplies many layout maps you can use with the supplied models. Field-level help is available to select a map.
Horizontal panels	Number of panels used by the generated module. Users can display the next panel by pressing the right PF-key; they can return to the previous panel by pressing the left PF-key. By default, the module uses one horizontal panel.
Backward scroll pages	Maximum number of scroll pages supported by the module. By default, the module supports 10 scroll pages; users can scroll forward and backward within a 10-page range. If they scroll forward 11 pages, page 1 is forced out of the range and they cannot scroll back to it. <p><b>Note:</b> Natural Construct-generated modules do not allow backward scrolling through data that has not been previously scrolled through in a forward direction.</p>
Export data support	When this field is selected, records are exported to a work file (instead of the screen). Users can use these work files in other environments or on other platforms (for example, in a PC spreadsheet application). To export the generated report to a work file, users specify a starting value and press the export PF-key. <p>When generating the module, you must indicate which records to export by defining the EXPORT-DATA-FIELDS user exit using one of the following methods:</p>

Field	Description
	<ul style="list-style-type: none"> <li>■ Use the Export-Data-Fields user exit model to generate the exit (this model is available on the mainframe only)</li> <li>■ Select and define the EXPORT-DATA-FIELDS user exit by right-clicking the generated the module in the Natural for Windows editor</li> </ul> <p>You can customize the work file number and delimiter character (character used to delimit fields on the report) for your site.</p> <p><b>Note:</b> If you mark this field and do not define the EXPORT-DATA-FIELDS user exit, Natural Construct generates a default WRITE WORK FILE statement that includes all fields in the specified input key.</p>
Report data support	<p>When this field is selected, records are exported to a local printer (instead of the screen). To print the generated report, users specify a starting value and press the report PF-key.</p> <p>When generating the module, you must indicate which records to export by defining the REPORT-DATA-FIELDS user exit using one of the following methods:</p> <ul style="list-style-type: none"> <li>■ Use the Report-Data-Fields user exit model to generate the exit (this model is available on the mainframe only)</li> <li>■ Select and define the REPORT-DATA-FIELDS user exit by right-clicking the generated the module in the Natural for Windows editor</li> </ul> <p><b>Note:</b> If you mark this field and do not define the REPORT-DATA-FIELDS user exit, Natural Construct generates a default WRITE statement that includes all fields in the specified input key.</p>
Use BROWSE-SELECT actions	<p>When this field is selected, the browse dialog uses the same actions as those used by a Browse-Select model. For information, see <a href="#">Define or Customize Browse-Select Actions</a>.</p>

This section covers the following topics:

- [Change the Default Window Settings](#)
- [Define or Customize Standard Actions](#)
- [Define or Customize Browse-Select Actions](#)

- Define Screen Layout Parameters

### Change the Default Window Settings

▶ To change the default window settings for your object-browse dialog:

- Press PF5 (windw) on the Additional Parameters panel.

The Window Parameters window is displayed. For a description of this window, see *Change the Default Window Settings*.

### Define or Customize Standard Actions

▶ To define or customize the standard actions used by your object-browse dialog:

- 1 Press PF6 (actns) on the Additional Parameters panel.

The Action Parameters window is displayed:

```

CUBDMBA                               Natural Construct                               CUBDMBA0
Dec 30                                 Action Parameters                               1 of 1

Template ..... MC 1___ *   Object browse dialog template

Action ID   Action           Subprogram           Subroutine
___ ___ *   add             _____ *         _____
___ ___ *   detail          _____ *         _____
___ ___ *   display         _____ *         _____
___ ___ *   modify          _____ *         _____
___ ___ *   purge           _____ *         _____
___ ___ *   select          _____ *         _____
___ ___ *
___ ___ *
___ ___ *
___ ___ *
___ ___ *
___ ___ *
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12
      help  retrn quit                                     main
    
```

- 2 Use this window to do one of the following:

- Specify the name of an action template (pre-defined group of single actions).
- Override the default actions by specifying single actions and the names of the subprograms or subroutines that perform the actions.

## Specify an Action Template

### ▶ To specify an action template:

- Type the name of a Natural Construct or application-specific action template in the Template field and select Enter.

Or:

Press the help PF-key when the cursor is in the Template field to select from a list of available action templates (both Natural Construct and application-specific).

## Override the Default Actions

### ▶ To override the default single actions:

- 1 Type the ID for the single action in the appropriate Action ID field.

Or:

Press the help PF-key when the cursor is in the Action ID field to select from a list of single actions (both Natural Construct and application-specific).

- 2 In the corresponding Subprogram or Subroutine field, type the name of the subprogram or subroutine that is invoked when the action is requested.
- 3 Select Enter.

## Define or Customize Browse-Select Actions

You can define Browse-Select-style actions for your object-browse dialog, as opposed to the standard actions for the object-browse dialog (for example, add, detail, display, modify, purge, and select).

### ▶ To define or customize Browse-Select-style actions for your object-browse dialog:

- 1 Mark the Use BROWSE-SELECT actions field on the Additional Parameters panel.
- 2 Press PF6 (actns).

The Action Parameters window is displayed:

```
CUBDMD          Object-Browse-Dialog Subprogram          CUBDMD0
Jun 08          Action Parameters                          1 of 1
                                                         ↵
  _ Add         _ Browse         _ Clear         _ Display         _ Modify         ↵
  _ Next        _ Purge          _ Copy          _ Recall          _ Replace        ↵
  _ Select      _ Detail          _ Former       ↵
                                                         ↵
                                                         ↵
                                                         ↵
                                                         ↵
                                                         ↵
                                                         ↵
                                                         ↵
                                                         ↵
                                                         ↵
                                                         ↵
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF1
      help  retrn quit                                     mai
```

This window displays the actions used by a Browse-Select model.

 **Note:** For DB2 users, the Former action is disabled in this window.

- 3 Mark the actions you want to expose on your object-browse dialog.
- 4 Select Enter to confirm the changes.

## Define Screen Layout Parameters

### ▶ To define the screen layout parameters for your object-browse dialog:

- 1 Press PF7 (scrn) on the Additional Parameters panel.

The Screen Layout Parameters window is displayed:

```

CUBDMBB          Natural Construct          CUBDMBB0
Feb 07           Screen Layout Parameters    1 of 1

Screen header lines ..... 2_
Field heading lines ..... 1_
Underline headings ..... X
Blank lines after headings ... _
Record display lines ..... 1_
Input key lines ..... 1_
      Position . Bottom ... X
                  Top ..... _
Starting column ..... _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---
      help  retrn quit  test
  
```

- 2 Use the fields in this window to define how information is displayed on your generated object-browse dialog.

The fields in this window are:

Field	Description
Screen header lines	Number of screen heading lines displayed when your generated module is invoked. By default, two lines are reserved for screen headings.
Field heading lines	Number of field heading lines displayed when your generated module is invoked. By default, one line is reserved for each field heading line.
Underline headings	When this field is selected, field headings are underlined when the generated module is invoked. By default, this field is selected and field headings are underlined on the generated dialog.
Blank lines after headings	Number of blank lines inserted after the field heading lines. For example, if you specify 1 in this field, one blank line is inserted below each field heading line.
Record display lines	Number of screen lines required to display each record and its attributes. By default, one line is reserved for each record.
Input key lines	Number of screen lines required to display input keys. By default, one line is reserved for each input key.
Position	

Field	Description
Bottom	When this field is selected, the input key lines are displayed at the bottom of the screen. By default, the input key lines are displayed at the bottom of the generated dialog.
Top	When this field is selected, the input key lines are displayed at the top of the screen.
Starting column	Number of the column in which the selection column begins.

- 3 Select Enter to confirm the changes.

### Test the Modified Screen Layout

You can view a test version of the window with the characteristics specified in the Screen Layout Parameters window.

#### ▶ To test the modified screen layout:

- Press PF4 (test).

### Define PF-Keys

You can specify a PF-key's position on the PF-key line, the Natural variable name used in the generated code, and international support for the text displayed on the PF-key line. You can also use a pre-defined set of PF-keys (called a key template), which you can select from a list of existing key templates or create on your own.



**Note:** For information on defining the Natural variable name used in the generated code, see *Defining PF-Keys for Generated Applications, Natural Construct Generation*.

To access PF-keys for your applications, Natural Construct provides two APIs (Application Programming Interfaces) and six methods in the CD--PFK module. Within this module, you can:

- Create new methods and APIs to access PF-keys
- Define single PF-keys and their attributes
- Define sets of PF-keys and attributes (key templates)

Any model can use the CD--PFK module during the modify and generation phases.

#### ▶ To define PF-keys:

- Press PF6 (pfkey) on the Standard Parameters panel for the Object-Browse-Dialog model.

The PF-Key Parameters window is displayed, from which you can select a key template or single PF-keys for your dialog. For more information, see [Define or Customize PF-Keys](#).





**Note:** If the object-browse dialog module was generated by the Transform-Browse model, the PF6 (pfkey) option is not available. For more information, see [Change the Default PF-key Style](#) and [PF-Key Styles](#).

## Define Single PF-Keys

Single PF-keys perform specific functions at runtime. Typically, they are used to add one or two functions to the standard PF-key set supplied by a model.

Single PF-keys are divided into two types and maintained in separate lists: PF-keys used by the Natural Construct models (identified by an SC prefix) and application-specific PF-keys that you create (identified by an SA prefix). You can use any single PF-key with any model.

The attributes of a single PF-key are:

Attribute	Description
SOURCE-ID	Type of PF-key and whether the key is defined by Natural Construct or by the user. Valid single source IDs are defined in the CD--PFKM local data area. They are: <ul style="list-style-type: none"> <li>■ SC (Single Construct)</li> <li>■ SA (Single Application)</li> </ul>
PFKEY-ID	Internal identifier for a single PF-key. Valid single PF-key IDs are defined in the CD--PFKM local data area.
PFKEY-POSITION	Position of a single PF-key on the PF-key line (for example, PF2 for the return PF-key). The value in this field is assigned to PFKEY-POSITION-VARIABLE in the generated code.
PFKEY-POSITION-VARIABLE	Natural variable used in the generated code to refer to the position of a single PF-key (for example, #PF-RETURN for the return PF-key).
PFKEY-NAME-VARIABLE	Natural variable containing the name displayed for a single PF-key at runtime (for example, #RETURN-NAME for the return PF-key).
PFKEY-NAME	Text or SYSERR reference used to identify a single PF-key. The value in this field is assigned to PFKEY-NAME-VARIABLE in the generated code. <p><b>Note:</b> If you use SYSERR references, text is retrieved from the CSTPFK library in SYSERR.</p> <p>The generated contents of PFKEY-NAME depend on the international parameters defined for the model. You can generate text in a specific language or generate SYSERR references. By default, English text is generated.</p>
PFKEY-NAME-LONG	Text or SYSERR reference used as a long description of a single PF-key.

Attribute	Description
PFKEY-STATUS	Status code for a single PF-key. This code determines if a single PF-key may be overridden by the user. Valid status codes are: <ul style="list-style-type: none"> <li>■ C (Conditional; PF-key is reserved by the model and is conditional on the selection of options)</li> <li>■ O (Optional; PF-key can be overridden by the user)</li> <li>■ R (Required; PF-key is required by the model)</li> </ul>



**Note:** The values specified in these fields are used for defaulting purposes. In most cases, these values can be overridden by the user.

### Natural Construct Single PF-Keys

Natural Construct single PF-keys are defined in the CD--PFKM local data area and the CD--PFK module. There are 34 Natural Construct single PF-keys provided.

The default constant that identifies the current number of PF-keys is defined in CD--PFKM as follows:

```

*
* * Default constants
  1 CD--PFKD
  2 MAX-SPC                                I      2 CONST<34> /* Max CST singl
    
```

Natural Construct single PF-key IDs are defined in CD--PFKM as follows:

```

* *
* * Construct Single Pfkey Id List
I  2 SPC-PFKEY-ID          I  2 (1:MAX-SPC)
R  2 SPC-PFKEY-ID          /* REDEF. BEGIN :
* *                          /* Construct list
  3 HELP-PFKEY             I  2 /* Pfkey Id 1
  3 RETURN-PFKEY           I  2 /* Pfkey Id 2
  3 BACKWARD-PFKEY         I  2 /* Pfkey Id 3
  3 FORWARD-PFKEY          I  2 /* Pfkey Id 4
  3 LEFT-PFKEY             I  2 /* Pfkey Id 5
  3 RIGHT-PFKEY            I  2 /* Pfkey Id 6
  3 QUIT-PFKEY             I  2 /* Pfkey Id 7
  3 MAIN-PFKEY             I  2 /* Pfkey Id 8
  3 FLIP-PFKEY            I  2 /* Pfkey Id 9
  3 PLACE-PFKEY           I  2 /* Pfkey Id 10
  3 HARDCOPY-PFKEY         I  2 /* Pfkey Id 11
  3 EXPORT-PFKEY           I  2 /* Pfkey Id 12
  3 PREFERENCES-PFKEY     I  2 /* Pfkey Id 13
  3 CONFIRM-PFKEY         I  2 /* Pfkey Id 14
  3 DIRECT-COMMAND-PFKEY  I  2 /* Pfkey Id 15
  3 ADD-PFKEY             I  2 /* Pfkey Id 16
  3 BROWSE-PFKEY          I  2 /* Pfkey Id 17
  3 CLEAR-PFKEY           I  2 /* Pfkey Id 18
  3 COPY-PFKEY            I  2 /* Pfkey Id 19
  3 DETAIL-PFKEY          I  2 /* Pfkey Id 20
  3 DISPLAY-PFKEY         I  2 /* Pfkey Id 21
  3 MODIFY-PFKEY          I  2 /* Pfkey Id 22
  3 NEXT-PFKEY            I  2 /* Pfkey Id 23
  3 PURGE-PFKEY           I  2 /* Pfkey Id 24
  3 RECALL-PFKEY          I  2 /* Pfkey Id 25
  3 REPLACE-PFKEY         I  2 /* Pfkey Id 26
  3 SELECT-PFKEY          I  2 /* Pfkey Id 27
  3 ACTIONS-PFKEY         I  2 /* Pfkey Id 28
  3 REPORT-PFKEY          I  2 /* Pfkey Id 29
  3 TRANSLATE-PFKEY       I  2 /* Pfkey Id 30
  3 NULL-PFKEY            I  2 /* Pfkey Id 31
  3 STORE-PFKEY           I  2 /* Pfkey Id 32
  3 UPDATE-PFKEY          I  2 /* Pfkey Id 33
  3 MAINTAIN-PFKEY        I  2 /* Pfkey Id 34

```



**Note:** The internal ID is derived from the occurrence of each PF-key within the SPC-PFKEY-ID array.

Natural Construct single PF-key IDs (referenced in CD--PFKM) are defined in the CD--PFK module as follows:

```

*
* Single key initial values Construct
01 SINGLE-CST
  02 INITIAL-VALUES (A150/1:MAX-SPC)
*   Position  \Pos var.\ Name variable/Name/ Name long/ Status override
*           \      /      \      /      \      /
*           \      /      \      /      \      /
*           \      /      \      /      \      /
*           \      /      \      /      \      /
INIT (1)<'PF1/#PF-HELP/#HELP-NAME/*8001.1/*8001.2/0'>
      (2)<'PF2/#PF-RETURN/#RETURN-NAME/*8002.1/*8002.2/0'>
      (3)<'PF7/#PF-BACKWARD/#BACKWARD-NAME/*8003.1/*8003.2/0'>
      (4)<'PF8/#PF-FORWARD/#FORWARD-NAME/*8004.1/*8004.2/0'>
      (5)<'PF10/#PF-LEFT/#LEFT-NAME/*8005.1/*8005.2/0'>
      (6)<'PF11/#PF-RIGHT/#RIGHT-NAME/*8006.1/*8006.2/0'>
      (7)<'PF3/#PF-QUIT/#QUIT-NAME/*8007.1/*8007.2/0'>
      (8)<'PF12/#PF-MAIN/#MAIN-NAME/*8008.1/*8008.2/0'>
      (9)<'PF5/#PF-FLIP/#FLIP-NAME/*8009.1/*8009.2/0'>
      (10)<'PF6/#PF-PLACE/#PLACE-NAME/*8010.1/*8010.2/0'>
      (11)<'PF9/#PF-HARDCOPY/#HARDCOPY-NAME/*8011.1/*8011.2/0'>
      (12)<'PF6/#PF-EXPORT/#EXPORT-NAME/*8012.1/*8012.2/0'>
      (13)<'PF6/#PF-PREFERENCE/#PREFERENCE-NAME/*8013.1/*8013.2/0'>
      (14)<'ENTR/#PF-CONFIRM/#CONFIRM-NAME/*8014.1/*8014.2/0'>
      (15)<'PF41/#PF-DIRECT-CMD/#DIRECT-CMD-NAME/*8015.1/*8015.2/0'>
      (16)<'PF25/#PF-ADD/#ADD-NAME/*8016.1/*8016.2/0'>
      (17)<'PF26/#PF-BROWSE/#BROWSE-NAME/*8017.1/*8017.2/0'>
      (18)<'PF27/#PF-CLEAR/#CLEAR-NAME/*8018.1/*8018.2/0'>
      (19)<'PF28/#PF-COPY/#COPY-NAME/*8019.1/*8019.2/0'>
      (20)<'PF29/#PF-DETAIL/#DETAIL-NAME/*8020.1/*8020.2/0'>
      (21)<'PF30/#PF-DISPLAY/#DISPLAY-NAME/*8021.1/*8021.2/0'>
      (22)<'PF31/#PF-MODIFY/#MODIFY-NAME/*8022.1/*8022.2/0'>
      (23)<'PF32/#PF-NEXT/#NEXT-NAME/*8023.1/*8023.2/0'>
      (24)<'PF33/#PF-PURGE/#PURGE-NAME/*8024.1/*8024.2/0'>
      (25)<'PF34/#PF-RECALL/#RECALL-NAME/*8025.1/*8025.2/0'>
      (26)<'PF35/#PF-REPLACE/#REPLACE-NAME/*8026.1/*8026.2/0'>
      (27)<'PF36/#PF-SELECT/#SELECT-NAME/*8027.1/*8027.2/0'>
      (28)<'PF5/#PF-ACTIONS/#ACTIONS-NAME/*8028.1/*8028.2/0'>
      (29)<'PF9/#PF-REPORT/#REPORT-NAME/*8029.1/*8029.2/0'>
      (30)<'PF4/#PF-TRANSLATE/#TRANSLATE-NAME/*8030.1/*8030.2/0'>
      (31)<'PF0/#PF-NULL/#NULL-NAME/*8031.1/*8031.2/0'>
      (32)<'PF4/#PF-STORE/#STORE-NAME/*8032.1/*8032.2/0'>
      (33)<'PF4/#PF-UPDATE/#UPDATE-NAME/*8033.1/*8033.2/0'>
      (34)<'PF4/#PF-MAINTAIN/#MAINTAIN-NAME/*8034.1/*8034.2/0'>

```



#### Notes:

1. SYSERR references are defined in the CSTPFK library in SYSERR; this library is reserved for PF-key attribute definitions.
2. All models using this mechanism generate the PF-key attributes inline. To reflect any changes to the PF-key definitions, you must regenerate the module.

## Application-Specific Single PF-Keys

Application-specific single PF-keys are defined in the CD--PFKM local data area and the CD--PFK module. There are two sample single PF-keys provided.

The default constant that identifies the current number of single PF-keys is defined in CD--PFKM as follows:

```
*
* * Default constants
  1 CD--PFKD
  2 MAX-SPA                                I    2 CONST<2> /* Max App single pfkey
```

Application-specific single PF-keys are defined in CD--PFKM as follows:

```
* *
* * Application Single Pfkey Id List
I  2 SPA-PFKEY-ID                        I    2 (1:MAX-SPA)
R  2 SPA-PFKEY-ID                        /* REDEF. BEGIN : SPA-PFKEY-ID
* *                                     /* Application list
  3 SAMPLE-PFKEY1                        I    2 /* Pfkey Id 1
  3 SAMPLE-PFKEY2                        I    2 /* Pfkey Id 2 ←
```



**Note:** The internal ID is derived from the occurrence of each PF-key within the SPA-PFKEY-ID array.

Application-specific single PF-key IDs (referenced in CD--PFKM) are defined in the CD--PFK module as follows:

```
*
* Single key initial values Application
01 SINGLE-APP
  02 INITIAL-VALUES (A113/1:MAX-SPA)
*   Position \Pos var.\ Name variable/Name/ Name long/ Status override
*
*
*
*
  INIT (1)<'PF5/#PF-JUMP/#JUMP-NAME/*9001.1/*9001.2/0'>
        (2)<'PF9/#PF-STOP/#STOP-NAME/*9002.1/*9002.2/0'>
```



### Notes:

1. SYSERR references are defined in the CSTPFK library in SYSERR; this library is reserved for PF-key attribute definitions. User-defined SYSERR references start at 9000. For more information, see the online help.
2. All models using the PF-key mechanism generate the PF-key attributes inline. To reflect changes to the PF-key definitions, you must regenerate the module.

► To add an application-specific single PF-key:

- 1 Increase the MAX-SPA default constant by one in the CD--PFKM local data area.

For example, change the MAX-SPA line in the following:

```
*
* * Default constants
  1 CD--PFKD
  2 MAX-SPA                                I    2 CONST<2> /* Max App single pfkey
```

to:

```
2 MAX-SPA                                I    2 CONST<3> /* Max App single pfkey
```

- 2 Add the internal PF-key name to the redefinition of the SPA-PFKEY-ID array in CD--PFKM.

For example:

```
* *
* * Application Single Pfkey Id List
I   2 SPA-PFKEY-ID                        I    2 (1:MAX-SPA)
R   2 SPA-PFKEY-ID                        /* REDEF. BEGIN : SPA-PFKEY-ID
* *                                       /* Application list
   3 SAMPLE-PFKEY1                        I    2 /* Pfkey Id 1
   3 SAMPLE-PFKEY2                        I    2 /* Pfkey Id 2
*
* add new single pfkey redefine
*
   3 NEW-KEY                              I    2 /* Pfkey Id 3 ↵
```

- 3 Edit the SPA-PFKEY-ID array to initialize the internal ID value.

For example:

```
11:59:15          ***** EDIT FIELD *****                               13-10-16
                   - Initial Values - Single Mode -

Local   CD--PFKM  Library CSTDEM
Command +

      Index          SPA-PFKEY-ID(I2/1:3)
-----
(1)                1
(2)                2
*
* add new single pfkey internal id
*
(3)                3
```

- 4 Stow CD--PFKM.
- 5 Define the attributes of the application-specific single PF-key IDs in the CD--PFK module (referenced in CD--PFKM).

For example, add the INIT clause for the new occurrence of INIT-VALUES:

```
* Single key initial values Application
01 SINGLE-APP
  02 INITIAL-VALUES (A113/1:MAX-SPA)
*   Position \Pos var.\ Name variable/Name/ Name long/ Status override
*
*
*
*
INIT (1)<'PF5/#PF-JUMP/#JUMP-NAME/*9001.1/*9001.2/0'>
      (2)<'PF9/#PF-STOP/#STOP-NAME/*9002.1/*9002.2/0'>
*
* add new single pfkey attributes
*
      (3)<'PFnn/#PF-name/#key-NAME/*9003.1/*9003.2/0'>
```



**Note:** You can use either SYSERR references or text for the NAME and NAME-LONG values. User-defined SYSERR references start at 9000. For more information, see the online help.

- 6 Stow CD--PFK.

### Define Key Templates

A key template is a group of single PF-keys relevant to a given model. If a model supports PF-key parameters, the logical grouping (template) can be displayed to the user as default values. The user can then customize the template as desired.

Key templates are divided into two types and maintained on separate lists: key templates used by the Natural Construct models (identified by an MC prefix) and application-specific key templates that you create (identified by an MA prefix). You can use any single PF-key with any key template.

The attributes for a key template are:

Attribute	Description
SOURCE-ID	Type of key template and whether the template is defined by Natural Construct or user-defined. Valid source IDs are defined in the CD--PFKM local data area. They are: <ul style="list-style-type: none"> <li>■ MC (Model Construct)</li> <li>■ MA (Model Application)</li> </ul>

Attribute	Description
PFKEY-ID	Internal identifier for a key template. Valid key templates are defined in the CD--PFKM local data area.
NAME	Text or SYSERR reference used as a long description of the key template. <b>Note:</b> If you use SYSERR references, the text is retrieved from the CSTPFK library in SYSERR.
MODEL-PFKEY-VALUES	Key template containing a group of single PF-key definitions. <b>Note:</b> The maximum number of PF-keys assigned to one key template is 12.
SOURCE-ID	Source IDs for the single PF-keys in the key template.
PFKEY-ID	Internal identifiers for single PF-keys in the key template.
PFKEY-POSITION-OVERRIDE	Override positions for single PF-keys in the key template. If you do not specify an override position, the position value for that single PF-key is assigned to this field.
PFKEY-STATUS-OVERRIDE	Override statuses for single PF-keys in the key template. If you do not specify an override status, the status value for the single PF-key is assigned to this field.



**Note:** The values in these fields are used for defaulting purposes. In most cases, you can override these attributes.

### Natural Construct Key Templates

Natural Construct key templates are defined in the CD--PFKM local data area and the CD--PFK module. There is one Natural Construct key template provided.

The default constant that identifies the current number of key templates is defined in CD--PFKM as follows:

```
*
* * Default constants
  1 CD--PFKD
  2 MAX-MPC                                I    2 CONST<1> /* Max CST model pfkeys
```

Natural Construct key templates are defined in CD--PFKM as follows:

```
* *
* * Construct Model Pfkey Id List
I  2 MPC-PFKEY-ID                        I    2 (1:MAX-MPC)
R  2 MPC-PFKEY-ID                        /* REDEF. BEGIN : MPC-PFKEY-ID
* *                                     /* Construct list
  3 OBJECT-BROWSE-DIALOG                  I    2 /* Model pfkey Id 1
```





**Note:** The internal ID is derived from the occurrence of each key template within the MPC-PFKEY-ID array.

Natural Construct key templates (referenced in CD--PFKM) are defined in the CD--PFK module as follows:

```

*
* Model pfkey initial values Construct
01 MODEL-CST
  02 INITIAL-MODEL-VALUES (A40/1:MAX-MPC)
*   model source id/model pfkey id/ model name /
*       /           /           /
*       /           /           /
*       /           /           /
*       /           /           /
INIT (1) <'MC/0001/*8501.1'>
*
*
02 INITIAL-MODEL-PFKEY-VALUES (A15/1:MAX-MPC,1:12)
* Model\   source id/single pfkey id/position override/status override
*   \     /           /           /           /
*   \     /           /           /           /
*   \     /           /           /           /
*   \     /           /           /           /
INIT (1,1) <'SC/0001/ /R'>
      (1,2) <'SC/0002/ /R'>
      (1,3) <'SC/0007/ /O'>
      (1,4) <'SC/0030/ /C'>
      (1,5) <'SC/0028/ /C'>
      (1,6) <'SC/0012/ /C'>
      (1,7) <'SC/0003/ /R'>
      (1,8) <'SC/0004/ /R'>
      (1,9) <'SC/0029/ /C'>
      (1,10) <'SC/0005/ /C'>
      (1,11) <'SC/0006/ /C'>
      (1,12) <'SC/0008/ /O'>

```

### Application-Specific Key Templates

Application-specific key templates are defined in the CD--PFKM local data area and the CD--PFK module. One sample key template is provided.

The default constant that identifies the current number of key templates is defined in CD--PFKM as follows:

```
*
* * Default constants
1 CD--PFKD
2 MAX-MPA I 2 CONST<1> /* Max App model pfkeys
```

Application-specific key templates are defined in CD--PFKM as follows:

```
* *
* * Application Model Pfkey Id List
I 2 MPA-PFKEY-ID I 2 (1:MAX-MPA)
R 2 MPA-PFKEY-ID /* REDEF. BEGIN : MPA-PFKEY-ID
* * /* Application list
3 SAMPLE-MODEL I 2 /* Model pfKey Id 1
```



**Note:** The internal ID is derived from the occurrence of each key template within the MPA-PFKEY-ID array.

Application-specific key templates (referenced in CD--PFKM) are defined in the CD--PFK module as follows:

```
*
* Model key initial values Application
01 MODEL-APP
02 INITIAL-MODEL-VALUES (A40/1:MAX-MPA)
* source id/key id/ key name /
* / / / /
* / / / /
* / / / /
* INIT (1) <'MA/0001/*9501.1'>
*
02 INITIAL-MODEL-PFKEY-VALUES (A15/1:MAX-MPA,1:12)
* Model\ source id/single key id/position override/status override
* \ / / / / /
* \ / / / / /
* \ / / / / /
* INIT (1,1) <'SA/0001/ /R'>
(1,2) <'SA/0002/ /R'>
(1,3) <'SC/0003/ /0'>
(1,4) <'SC/0004/ /R'>
(1,5) <'SC/0005/ /C'>
(1,6) <'SC/0007/ /0'>
(1,7) <'SC/0008/ /0'>
(1,8) <'SC/0009/ /0'>
```

► **To add an application-specific key template:**

- 1 Increase the MAX-MPA default constant by one in the CD--PFKM local data area.

For example, change the MAX-MPA line in the following:

```
*
* * Default constants
  1 CD--PFKD
  2 MAX-MPA                                I    2 CONST<1> /* Max App model pfkeys
```

to:

```
  2 MAX-MPA                                I    2 CONST<2> /* Max App model pfkeys
```

- 2 Add the internal key template name to the redefinition of the MPA-PFKEY-ID array in CD--PFKM.

For example:

```
* *
* * Application Model Pfkey Id List
I   2 MPA-PFKEY-ID                        I   2 (1:MAX-MPA)
R   2 MPA-PFKEY-ID                        /* REDEF. BEGIN : MPA-PFKEY-ID
* *                                       /* Application list
   3 SAMPLE-MODEL                        I   2 /* Model pfkey Id 1
*
* add new model redefine
*
   3 NEW-MODEL                            I   2 /* New Model pfkey Id 2
```

- 3 Edit the MPA-PFKEY-ID array to initialize the internal ID value.

For example:

```
11:59:15          ***** EDIT FIELD *****                               13-10-16
                  - Initial Values - Single Mode -

Local   CD--PFKM   Library CSTDEM
Command +

          Index          MPA-PFKEY-ID(I2/1:2)
-----
(1)                1
*
* add new model internal id
*
(2)                2
```

- 4 Stow CD--PFKM.
- 5 Define the attributes of the application-specific key template in the CD--PFK module (referenced in CD--PFKM).

For example, add the INIT clause for the new occurrence of INIT-MODEL-VALUES:

```
*
* Model key initial values Application
01 MODEL-APP
  02 INITIAL-MODEL-VALUES (A40/1:MAX-MPA)
*      source id/key id/ key name /
*      / / / /
*      / / / /
*      / / / /
*      / / / /
INIT (1) <'MA/0001/*9501.1'>
*
* add new model attributes
*
INIT (2) <'MA/0002/*9502.1'>
```

6 Define the key template IDs in the CD--PFK module (referenced in CD--PFKM).

For example, add the INIT clause for the new occurrence of INITIAL-MODEL-PFKEY-VALUES:

```
02 INITIAL-MODEL-PFKEY-VALUES (A15/1:MAX-MPA,1:12)
* Model\      source id/single key id/position override/status override
* \ / / / / /
* \ / / / / /
* \ / / / / /
* \ / / / / /
INIT (1,1) <'SA/0001/ /R'>
      (1,2) <'SA/0002/ /R'>
      (1,3) <'SC/0003/ /O'>
      (1,4) <'SC/0004/ /R'>
      (1,5) <'SC/0005/ /C'>
      (1,6) <'SC/0007/ /O'>
      (1,7) <'SC/0008/ /O'>
      (1,8) <'SC/0009/ /O'>
*
* add new model group of single pfkey ids
*
      (2,1) <'SC/0006/ /R'>
      (2,2) <'SA/0001/ /C'>
      (2,3) <'SC/0012/ /O'>
      (2,4) <'SA/0002/ /R'>
      (2,5) <'SC/0031/ /C'>
```

7 Stow CD--PFK.

## Select PF-Keys and Key Templates

You can display helproutine windows listing valid IDs for the single PF-keys and key templates. The CU-PSOBD helproutine (listing the single PF-key IDs) is invoked by the CU-PSH driver. The CU-PMOBD helproutine (listing the key templates) is invoked by the CU-PMH driver. You can also create your own helproutine drivers and pass a different set of parameters.

The helproutines are available by invoking online help for the single PF-keys or the key templates in the PF-Key Parameters window. The following examples show the Select Key ID and the Select Template windows:

```
CU-PSOBD          Natural Construct
Dec 19           Select Key ID                1 of 1

  Key ID  Key Name  Position  Description
  -----
_ SC0001  help    PF1      Display available help
_ SC0002  retrn   PF2      Return to previous screen
_ SC0003  bkwrtd  PF7      Scroll backward
_ SC0004  frwrtd  PF8      Scroll forward
_ SC0005  left    PF10     Scroll left
_ SC0006  right   PF11     Scroll right
_ SC0007  quit    PF3      Terminate the application
_ SC0008  main    PF12     Return to main menu
_ SC0009  flip    PF5      Toggle between action and pfkeys
_ SC0010  place   PF6      Position to place
_ SC0011  hcopy   PF9      Hardcopy records
_ SC0012  exprt   PF6      Export records
Key ID ..... _____
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12
      help  retrn quit                bkwrtd frwrtd                main
```

```
CU-PMOBD          Natural Construct
Dec 19           Select Template            1 of 1

  Template Description
  -----
_ MC0001  Object browse dialog template
_ MA0001  User model template
                        End of Data

Template ... _____
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7
      help  retrn quit                bkw
```

All PF-keys are defined and controlled by the CD--PFK module. To access the PF-keys, the module supports six different methods and two APIs (Application Programming Interfaces): CDSPPFK and CDAPPFK. You can also add your own APIs and methods to the CD--PFK module.

### CD--PFK Module

The CD--PFK module controls the PF-keys. The PF-keys and the methods that access the PF-keys are defined in this module. CD--PFK accepts the following parameters:

Data Area	Description
CD1VPFKA	Parameter data area (PDA) containing the following fields: <ul style="list-style-type: none"> <li>■ API-NAME (A8)</li> <li>■ INPUTS (A1/1:V)</li> <li>■ INPUT-OUTPUTS (A1/1:V)</li> <li>■ OUTPUTS (A1/1:V)</li> </ul> The value specified in the API-NAME field determines how the INPUTS, INPUT-OUTPUTS, and OUTPUTS fields are redefined internally. For more information, refer to CD--PFK.
CDPDA-M	External PDA containing standard parameters for exchanging message information.

### PF-Key Methods

PF-key methods are defined in the CD--PFKM local data area. For example:

```
* *
* * Defined methods for CD--PFK
I  2 METHODS          A  2 (1:MAX-PM)
R  2 METHODS          /* REDEF. BEGIN : METHODS
  3 SINGLE-PFKEY      A  2
  3 MODEL-PFKEYS      A  2
  3 ALL-SINGLE-PFKEYS  A  2
  3 ALL-MODEL-PFKEYS  A  2
  3 ALL-PFKEYS        A  2
  3 MODEL-PFKEY-DETAIL A  2
```

The supported PF-key methods are:

Method	Description
SINGLE-PFKEY	Retrieves attributes for the specified single PF-key ID. For an example of using this method, refer to the CUBDMAB module.
MODEL-PFKEYS	Retrieves attributes for the specified key template (group of single PF-key IDs and their attributes). For an example of using this method, refer to the CUBDMAB module.
ALL-SINGLE-PFKEYS	Retrieves all single PF-key IDs. For an example of using this method, refer to the CU-PSOBS module.

Method	Description
ALL-MODEL-PFKEYS	Retrieves all key templates. For an example of using this method, refer to the CU-PMOBS module.
ALL-PFKEYS	Retrieves all single PF-key IDs and key templates.
MODEL-PFKEY-DETAIL	Retrieves the attributes for the specified key template, but not the attributes for the group of single PF-keys. For an example of using this method, refer to the CUBDMAB module.

### PF-Key APIs

The following PF-key API (Application Programming Interface) modules interface with the CD--PFK module:

- CDSPPFK

The CDSPPFK API module passes one occurrence of a PF-key ID and its attributes to the CD--PFK module. CDSPPFK accepts the following parameters:

- CDSPPFKA

Parameter data area (PDA) containing the following fields:

Field	Redefined As
INPUTS (A1/9)	<ul style="list-style-type: none"> <li>■ METHOD (A2)</li> <li>■ VERSION (N1)</li> <li>■ SOURCE-ID (A2)</li> <li>■ PFKEY-ID (N4)</li> </ul>
INPUT-OUTPUTS (A1/113)	<ul style="list-style-type: none"> <li>■ PFKEY-POSITION (A4)</li> <li>■ PFKEY-POSITION-VARIABLE (A32)</li> <li>■ PFKEY-NAME-VARIABLE (A32)</li> <li>■ PFKEY-NAME (A12)</li> <li>■ PFKEY-NAME-LONG (A32)</li> <li>■ PFKEY-STATUS (A1)</li> </ul>
OUTPUTS (A1/68)	<ul style="list-style-type: none"> <li>■ METHOD-LIBRARY (A8)</li> <li>■ METHOD-DESCRIPTION (A60)</li> </ul>

- CDPDA-M

External PDA containing standard parameters for exchanging message information.



**Note:** For an example of using this API, refer to the CUBDMAB module.

■ CDAPPFK

The CDAPPFK API module passes 12 occurrences of PF-key IDs and their attributes to the CD-PFK module. CDAPPFK accepts the following parameters:

■ CDAPPFKA

Parameter data area (PDA) containing the following fields:

Field	Redefined As
INPUTS (A1/15)	<ul style="list-style-type: none"> <li>■ METHOD (A2)</li> <li>■ VERSION (N1)</li> <li>■ START-SOURCE-ID (A2)</li> <li>■ START-PFKEY-ID (N4)</li> <li>■ MAX-PFKEY-REQUESTED (N4)</li> <li>■ RESTRICT-SOURCE-ID (A2)</li> </ul>
INPUT-OUTPUTS (A1/1428)	<ul style="list-style-type: none"> <li>■ PFKEY-ARRAY (1:12)</li> </ul> <p style="margin-left: 20px;">Redefined as:</p> <ul style="list-style-type: none"> <li>■ PFKEY-IDENTIFIER (A6)</li> </ul> <p style="margin-left: 20px;">Redefined as:</p> <ul style="list-style-type: none"> <li>■ PFKEY-SOURCE-ID (A2)</li> <li>■ PFKEY-PFKEY-ID (N4)</li> <li>■ PFKEY-POSITION (A4)</li> <li>■ PFKEY-POSITION-VARIABLE (A32)</li> <li>■ PFKEY-NAME-VARIABLE (A32)</li> <li>■ PFKEY-NAME (A12)</li> <li>■ PFKEY-NAME-LONG (A32)</li> <li>■ PFKEY-STATUS (A1)</li> </ul>
OUTPUTS (A1/72)	<ul style="list-style-type: none"> <li>■ METHOD-LIBRARY (A8)</li> <li>■ METHOD-DESCRIPTION (A60)</li> <li>■ MAX-PFKEY-RETURNED (N4)</li> </ul>

■ CDPDA-M

External PDA containing standard parameters for exchanging message information.



**Note:** For an example of using this API, refer to the CUBDMAB module.



## Define Actions

You can specify which Natural variable names are used in the generated code, the valid action codes, and whether international support is available for displaying the action text. You can also use a pre-defined set of actions (called an action template), which you can select from a list of existing action templates or create on your own.

To access actions for your applications, Natural Construct provides two APIs (Application Programming Interfaces) and six methods in the CD--ACT module. Within this module, you can:

- Create new methods and APIs to access actions
- Define single actions and their attributes
- Define sets of actions and their attributes (action templates)

Any model can use the CD--ACT module during the modify and generation phases.

To define actions, press PF6 (actns) on the Additional Parameters panel for the Object-Browse-Dialog model. The Action Parameters window is displayed, from which you can select an action template or single actions for your dialog. For more information, see [Define or Customize Standard Actions](#).

## Define Single Actions

Single actions execute specific functions at runtime. They are divided into two types and maintained on separate lists: actions used by the Natural Construct models (identified by an SC prefix) and application-specific actions that you create (identified by an SA prefix). You can use any single action with any model.

The attributes for a single action are:

Attribute	Description
SOURCE-ID	Type of action and whether the action is defined by Natural Construct or by the user. Valid single source IDs are defined in the CD--ACTM local data area. They are: <ul style="list-style-type: none"> <li>■ SC (Single Construct)</li> <li>■ SA (Single Application)</li> </ul>
ACTION-ID	Internal identifier for a single action. Valid single action IDs are defined in the CD--ACTM local data area.
ACTION-NAME-VARIABLE	Natural variable containing the name displayed for a single action at runtime (for example, #ADD-NAME for the Add action).
ACTION-CODE	Text or SYSERR reference used to identify a valid code for a single action. The specified code is used for action validation.

Attribute	Description
ACTION-NAME	Text or SYSERR reference used to identify a single action. The value in this field is assigned to ACTION-NAME-VARIABLE in the generated code.  <b>Note:</b> If you use SYSERR references, text is retrieved from the CSTACT library in SYSERR.  <b>Note:</b> The generated contents of ACTION-NAME depend on the international parameters defined for the model. You can generate text in a specific language or generate SYSERR references. By default, English text is generated.
ACTION-NAME-LONG	Text or SYSERR reference used as a long description of a single action.
ACTION-STATUS	Status code for a single action. This code determines if a single action may be overridden by the user. Valid status codes are:  <ul style="list-style-type: none"> <li>■ C (Conditional; action is reserved by the model and is conditional on the selection of options)</li> <li>■ O (Optional; action can be overridden by the user)</li> <li>■ R (Required; action is required by the model)</li> </ul>



**Note:** The values specified in these fields are used for defaulting purposes. In most cases, these values can be overridden by the user.

### Natural Construct Single Actions

Natural Construct single actions are defined in the CD--ACTM local data area and CD--ACT module. There are 12 Natural Construct single actions provided.

The default constant that identifies the current number of actions is defined in CD--ACTM as follows:

```
*
* * Default constants
  1 CD--ACTD
  2 MAX-SAC I 2 CONST<12> /* Max CST single act.
```

Natural Construct single action IDs are defined in CD--ACTM as follows:

```

* *
* * Construct Single Action Id List
I 2 SAC-ACTION-ID          I 2 (1:MAX-SAC)
R 2 SAC-ACTION-ID          /* REDEF. BEGIN : SAC-ACTION-ID
* *                        /* Construct list
 3 ADD-ACTION              I 2 /* Action Id 1
 3 BROWSE-ACTION           I 2 /* Action Id 2
 3 CLEAR-ACTION            I 2 /* Action Id 3
 3 COPY-ACTION             I 2 /* Action Id 4
 3 DETAIL-ACTION           I 2 /* Action Id 5
 3 DISPLAY-ACTION          I 2 /* Action Id 6
 3 MODIFY-ACTION           I 2 /* Action Id 7
 3 NEXT-ACTION             I 2 /* Action Id 8
 3 PURGE-ACTION            I 2 /* Action Id 9
 3 RECALL-ACTION           I 2 /* Action Id 10
 3 REPLACE-ACTION          I 2 /* Action Id 11
 3 SELECT-ACTION           I 2 /* Action Id 12

```



**Note:** The internal ID is derived from the occurrence of each action within the SAC-ACTION-ID array.

Natural Construct single action IDs (referenced in CD--ACTM) are defined in the CD--ACT module as follows:

```

*
* Single key initial values Construct
01 SINGLE-CST
 02 INITIAL-VALUES (A95/1:MAX-SAC)
*   action name variable/action code/action name/act. name long/status
*
*
*
*
*
*
INIT (1)< '#ADD/*8001.1/*8001.2/*8001.3/0'>
      (2)< '#BROWSE/*8002.1/*8002.2/*8002.3/0'>
      (3)< '#CLEAR/*8003.1/*8003.2/*8003.3/0'>
      (4)< '#COPY/*8004.1/*8004.2/*8004.3/0'>
      (5)< '#DETAIL/*8005.1/*8005.2/*8005.3/0'>
      (6)< '#DISPLAY/*8006.1/*8006.2/*8006.3/0'>
      (7)< '#MODIFY/*8007.1/*8007.2/*8007.3/0'>
      (8)< '#NEXT/*8008.1/*8008.2/*8008.3/0'>
      (9)< '#PURGE/*8009.1/*8009.2/*8009.3/0'>
      (10)< '#RECALL/*8010.1/*8010.2/*8010.3/0'>
      (11)< '#REPLACE/*8011.1/*8011.2/*8011.3/0'>
      (12)< '#SELECT/*8012.1/*8012.2/*8012.3/0'>

```



**Notes:**

1. SYSERR references are defined in the CSTACT library in SYSERR; this library is reserved for action attribute definitions.
2. All models using this functionality generate the action attributes inline. To reflect changes to the action definitions, you must regenerate the module.

### Application-Specific Single Actions

Application-specific single actions are defined in the CD--ACTM local data area and the CD--ACT module. There are two sample single actions provided. You can expand this list as required.

The default constant that identifies the current number of single actions is defined in CD--ACTM as follows:

```
*
* * Default constants
1 CD--ACTD
2 MAX-SAA I 2 CONST<2> /* Max App single act.
```

Application-specific single actions are defined in CD--ACTM as follows:

```
* *
* * Application Single Action Id
I 2 SAA-ACTION-ID I 2 (1:MAX-SAA)
R 2 SAA-ACTION-ID /* REDEF. BEGIN : SAA-ACTION-ID
* * /* Application list
3 SAMPLE-ACTION1 I 2 /* Action Id 1
3 SAMPLE-ACTION2 I 2 /* Action Id 2
```



**Note:** The internal ID is derived from the occurrence of each action within the SAA-ACTION-ID array.

Application-specific single actions (referenced in CD--ACTM) are defined in the CD--ACT module as follows:

```
*
* Single action initial values Application
01 SINGLE-APP
02 INITIAL-VALUES (A90/1:MAX-SAA)
* action name variable/action code/action name/act. name long/status
*
*
*
*
*
INIT (1)<'#GET/*9001.1/*9001.2/*9001.3/0'>
(2)<'#SKIP/*9002.1/*9002.2/*9002.3/0'>
```



**Note:** SYSERR references are defined in the CSTACK library in SYSERR; this library is reserved for action attribute definitions. User-defined SYSERR references start at 9000. For more information, see the online help.

▶ **To add an application-specific single action:**

- 1 Increase the MAX-SAA default constant by one in the CD--ACTM local data area.

For example, change the MAX-SAA line in the following:

```
*
* * Default constants
  1 CD-ACTD
  2 MAX-SAA                                I    2 CONST<2> /* Max App single act.
```

to:

```
  2 MAX-SAA                                I    2 CONST<3> /* Max App single act. ←
```

- 2 Add the internal action name to the redefinition of the SAA-ACTION-ID array in CD--ACTM.

For example:

```
* *
* * Application Single Action Id
I  2 SAA-ACTION-ID                        I    2 (1:MAX-SAA)
R  2 SAA-ACTION-ID                        /* REDEF. BEGIN : SAA-ACTION-ID
* *                                       /* Application list
  3 SAMPLE-ACTION1                        I    2 /* Action Id 1
  3 SAMPLE-ACTION2                        I    2 /* Action Id 2
*
* add new single ACTION redefine
*
  3 NEW-ACTION                            I    2 /* Action Id 3 ←
```

- 3 Edit the SAA-ACTION-ID array to initialize the internal ID value.

For example:

```

11:59:15          ***** EDIT FIELD *****                               13-10-16
          - Initial Values - Single Mode -
Local      CD-ACTM  Library CSTDEM
Command +

          Index          SAA-ACTION -ID(I2/1:3)
-----
(1)                1
(2)                2
*
* add new single action internal ID
*
(3)                3

```

- 4 Stow CD-ACTM.
- 5 Define the attributes of the application-specific single action IDs in the CD--ACT module (referenced in CD--ACTM).

For example, add the INIT clause for the new occurrence of INIT-VALUES:

```

* Single action initial values Application
01 SINGLE-APP
  02 INITIAL-VALUES (A90/1:MAX-SAA)
*   action name variable/action code/action name/act. name long/status
*
*
*
*
*
*
INIT (1)<'#GET/*9001.1/*9001.2/*9001.3/0'>
      (2)<'#SKIP/*9002.1/*9002.2/*9002.3/0'>
*
* add new single action attributes
*
      (3)<'#NEW-ACTION/*9003.1/*9003.2/*9003.3/0'>

```



**Note:** You can use either SYSERR references or text for the NAME and NAME-LONG values. User-defined SYSERR references start at 9000. For more information, see the online help.

- 6 Stow CD--ACT.

## Define Action Templates

An action template is a group of single actions relevant to a given model. If a model supports action parameters, the logical grouping (template) can be displayed to the user as default values. The user can then customize the template as desired.

Action templates are divided into two types and maintained on separate lists: action templates used by the Natural Construct models (identified by an MC prefix) and application-specific action templates that you create (identified by an MA prefix). All single actions are available for all action templates.

The attributes of an action template are:

Attribute	Description
SOURCE-ID	Type of action template and whether the template is defined by Natural Construct or user-defined. Valid source IDs are defined in the CD--ACTM local data area. They are: <ul style="list-style-type: none"> <li>■ MC (Model Construct)</li> <li>■ MA (Model Application)</li> </ul>
ACTION-ID	Internal identifier for an action template. Valid action templates are defined in the CD--ACTM local data area.
NAME	Text or SYSERR reference used as a long description of the action template. <p><b>Note:</b> If you use SYSERR references, the text is retrieved from the CSTACT library in SYSERR.</p>
MODEL-ACTION-VALUES	Action template containing a group of single action definitions. <p><b>Note:</b> The maximum number of actions assigned to one action template is 12.</p>
SOURCE-ID	Source IDs for single actions in the action template.
ACTION-ID	Internal identifiers for single actions in the action template.
ACTION-STATUS-OVERRIDE	Override statuses for single actions in the action template. If you do not specify an override status, the status value for the single action is assigned to this field.



**Note:** The values in these fields are used for defaulting purposes. In most cases, you can override these attributes.

### Natural Construct Action Templates

Natural Construct action templates are defined in the CD--ACTM local data area and the CD--ACT module. There is one Natural Construct action template provided.

The default constant that identifies the current number of action templates is defined in CD--ACTM as follows:

```

*
* * Default constants
  1 CD--ACTD
  2 MAX-MAC                                I    2 CONST<1> /* Max CST model act.

```

Natural Construct action templates are defined in CD-ACTM as follows:

```

* *
* * Construct action template Id List
I  2 MAC-ACTION-ID                        I    2 (1:MAX-MAC)
R  2 MAC-ACTION-ID                        /* REDEF. BEGIN : MAC-ACTION-ID
* *                                       /* Construct list
  3 OBJECT-BROWSE-DIALOG                  I    2 /* action template Id 1

```



**Note:** The internal ID is derived from the occurrence of each action within the MAC-ACTION-ID array.

Natural Construct action templates (referenced in CD--ACTM) are defined in the CD--ACT module as follows:

```

*
* action template initial values Construct
01 MODEL-CST
  02 INITIAL-MODEL-VALUES (A40/1:MAX-MAC)
    model source ID/action template ID/model name/
    /
    /
    /
    /
INIT (1) <'MC/0001/*8501.1'>
*
*
  02 INITIAL-MODEL-ACTION-VALUES (A10/1:MAX-MAC,1:24)
    Model\   source ID/action ID/status override
    \
    \
    \
    \
INIT (1,1) <'SC/0001/ ' >
      (1,2) <'SC/0005/ ' >
      (1,3) <'SC/0006/ ' >
      (1,4) <'SC/0007/ ' >
      (1,5) <'SC/0009/ ' >
      (1,6) <'SC/0012/ ' >

```

### Application-Specific Action Templates

Application-specific action templates are defined in the CD--ACTM local data area and the CD--ACT module. There is one sample action template provided.

The default constant that identifies the current number of action templates is defined in CD--ACTM as follows:



```

*
* * Default constants
  1 CD-ACTD
  2 MAX-MAA                               I    2 CONST<1> /* Max App model act.

```

Application-specific action templates are defined in CD-ACTM as follows:

```

* *
* * Application action template Id List
I  2 MAA-ACTION-ID                       I    2 (1:MAX-MAA)
R  2 MAA-ACTION-ID                       /* REDEF. BEGIN : MAA-ACTION-ID
* *                                       /* Application list
  3 SAMPLE-MODEL                         I    2 /* action template Id 1

```



**Note:** The internal ID is derived from the occurrence of each action within the MAA-ACTION-ID array.

Application-specific action templates (referenced in CD--ACTM) are defined in the CD--ACT module as follows:

```

*
* action template initial values Application
01 MODEL-APP
  02 INITIAL-MODEL-VALUES (A40/1:MAX-MAA)
    source ID/action ID/ model name /
    / / / / /
    / / / / /
    / / / / /
    / / / / /
INIT (1) <'MA/0001/*9501.1'>
*
*
  02 INITIAL-MODEL-ACTION-VALUES (A15/1:MAX-MAA,1:24)
    Model\ source ID/action ID/status override/
    \ / / / /
    \ / / / /
    \ / / / /
INIT (1,1) <'SC/0012/C'>
    (1,2) <'SC/0010/ '>
    (1,3) <'SC/0008/ '>
    (1,4) <'SC/0006/ '>
    (1,5) <'SC/0004/ '>
    (1,6) <'SC/0002/ '>
    (1,7) <'SA/0001/ '>
    (1,8) <'SC/0003/ '>
    (1,9) <'SA/0002/ '>
    (1,10) <'SC/0009/ '>
    (1,11) <'SC/0008/ '>
    (1,12) <'SC/0011/ '>

```

► **To add an application-specific action template:**

- 1 Increase the MAX-MAA default constant by one in the CD--ACTM local data area.

For example, change the MAX-MAA line in the following:

```
*
* * Default constants
  1 CD--ACTD
  2 MAX-MAA          I    2 CONST<1> /* Max App model act.
```

to:

```
  2 MAX-MAA          I    2 CONST<2> /* Max App model act.
```

- 2 Add the internal action template name to the redefinition of the MAA-ACTION-ID array in CD--ACTM.

For example:

```
* *
* * Application action template Id List
I  2 MAA-ACTION-ID          I    2 (1:MAX-MAA)
R  2 MAA-ACTION-ID          /* REDEF. BEGIN : MAA-ACTION-ID
* *                          /* Application list
  3 SAMPLE-MODEL           I    2 /* action template Id 1
*
* add new model redefine
*
  3 NEW-MODEL              I    2 /* New action template Id 2
```

- 3 Edit the MAA-ACTION-ID array to initialize the internal ID value.

For example:

```
11:59:15          ***** EDIT FIELD *****                               13-10-16
                   - Initial Values - Single Mode -
Local    CD-ACTM   Library CSTDEM
Command  +
          Index          MAA-ACTION -ID(I2/1:2)
-----
(1)          1
*
* add new model internal ID
*
(2)          2
```

- 4 Stow CD--ACTM.
- 5 Define the attributes of the application-specific action template in the CD--PFK module (referenced in CD--ACTM).

For example, add the INIT clause for the new occurrence of INIT-MODEL-VALUES:

```
*
* action template initial values Application
01 MODEL-APP
  02 INITIAL-MODEL-VALUES (A40/1:MAX-MAA)
*      source ID/action ID/ model name /
*      /          /          /
*      /          /          /
*      /          /          /
*      /          /          /
INIT (1) <'MA/0001/*9501.1'>
*
* add new model attributes
*
INIT (2) <'MA/0002/*9502.1'>
```

- 6 Define the action template IDs in the CD--ACT module (referenced in CD--ACTM).

For example, add the INIT clause for the new occurrence of INITIAL-MODEL-ACTION-VALUES:

```

02 INITIAL-MODEL-ACTION-VALUES (A15/1:MAX-MAA,1:24)
*   Model\      source ID/action ID/status override/
*           \      /      /      /
*           \      /      /      /
*           \      /      /      /
INIT (1,1) <'SC/0012/C'>
      (1,2) <'SC/0010/ ' >
      (1,3) <'SC/0008/ ' >
      (1,4) <'SC/0006/ ' >
      (1,5) <'SC/0004/ ' >
      (1,6) <'SC/0002/ ' >
      (1,7) <'SA/0001/ ' >
      (1,8) <'SC/0003/ ' >
      (1,9) <'SA/0002/ ' >
      (1,10) <'SC/0009/ ' >
      (1,11) <'SC/0008/ ' >
      (1,12) <'SC/0011/ ' >
*
* add new action template of single action IDs
*
      (2,1) <'SC/0009/R'>
      (2,2) <'SA/0002/ ' >
      (2,3) <'SC/0011/0'>
      (2,4) <'SA/0001/ ' >
      (2,5) <'SC/0010/C'>

```

## 7 Stow CD-ACT.

### Select Actions and Action Templates

You can display helproutine windows listing valid IDs for single actions and action templates. The CU-ASOBD helproutine (listing the single actions) is invoked by the CU-ASH driver. The CU-AMOBD helproutine (listing the action templates) is invoked by the CU-AMH driver. You can also create your own helproutine drivers and pass a different set of parameters.

The helproutines are available by invoking online help for the single actions or the action templates in the Action Parameters window. The following examples show the Select Action ID and Select Template windows:

```

CU-ASOBD          Natural Construct
Dec 29           Select Action ID          1 of 1

  Action ID  Action name  Action code  Description
  -----
- SC0001    add           ADD         Add a record
- SC0002    browse        BROWSE     Browse a list of records
- SC0003    clear         CLEAR      Clear contents from screen
- SC0004    copy          COPY       Copy existing record
- SC0005    detail        DETAIL     Show record details
- SC0006    display       DISPLAY    Display a record
- SC0007    modify        MODIFY     Modify an existing record
- SC0008    next          NEXT       Display next record
- SC0009    purge         PURGE     Purge a record
- SC0010    recall        RECALL     Recall the purged record
- SC0011    replace       REPLACE    Replace the record
- SC0012    select        SELECT     Select a record
Action ID .. _____
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12
      help  retrn quit                    bkwrđ frwrđ                    main

```

```

CU-AMOBD          Natural Construct
Dec 29           Select Template          1 of 1

  Template  Description
  -----
- MC0001    Object browse dialog template
- MA0001    User model template
           End of Data

Template ... _____
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7
      help  retrn quit                    bkwrđ

```

### Access Actions and Methods

All actions are defined and controlled by the CD--ACT module. To access the actions, the module supports six different methods and two APIs (Application Programming Interfaces): CDSAACT and CDAAACT. You can also add your own APIs and methods to the CD--ACT module.

The CD--ACT module controls the actions. The actions and the methods that access the actions are defined in this module. CD--ACT accepts the following parameters:

Data Area	Description
CD1VACTA	<p>Parameter data area (PDA) containing the following fields:</p> <ul style="list-style-type: none"> <li>■ API-NAME (A8)</li> <li>■ INPUTS (A1/1:V)</li> <li>■ INPUT-OUTPUTS (A1/1:V)</li> <li>■ OUTPUTS (A1/1:V)</li> </ul> <p>The value specified in the API-NAME field determines how the INPUTS, INPUT-OUTPUTS, and OUTPUTS fields are redefined internally. For more information, see the CD--ACT module.</p>
CDPDA-M	External PDA containing standard parameters for exchanging message information.

### Action Methods

Action methods are defined in the CD--ACTM local data area. For example:

```

* *
* * Defined methods for CD--ACT
I  2 METHODS                A  2 (1:MAX-AM)
R  2 METHODS                /* REDEF. BEGIN : METHODS
  3 SINGLE-ACTION           A  2
  3 MODEL-ACTIONS           A  2
  3 ALL-SINGLE-ACTIONS       A  2
  3 ALL-MODEL-ACTIONS       A  2
  3 ALL-ACTIONS             A  2
  3 MODEL-ACTION-DETAIL     A  2

```

The supported action methods are:

Method	Description
SINGLE-ACTION	Retrieves attributes for the specified single action. For an example of using this method, refer to the CUBDMBA module.
MODEL-ACTIONS	Retrieves attributes for the specified action template (group of single action IDs and their attributes). For an example of using this method, refer to the CUBDMBA module.
ALL-SINGLE-ACTIONS	Retrieves all single action IDs. For an example of using this method, refer to the CU-ASOBS module.
ALL-MODEL-ACTION	Retrieves all action templates. For an example of using this method, refer to the CU-AMOBS module.
ALL-ACTIONS	Retrieves all single actions and action templates.
MODEL-ACTION-DETAIL	Retrieves the attributes for the specified action template, but not the attributes for the group of single actions. For an example of using this method, refer to the CUBDMBA module.

## Action APIs

Two action API (Application Programming Interface) modules interface with the CD--ACT module:

- CDSAACT

The CDSAACT API module passes one occurrence of an action ID and its attributes to the CD--ACT module. CDSAACT accepts the following parameters:

- CDSAACTA

Parameter data area (PDA) containing the following fields:

Field	Redefined As
INPUTS (A1/9)	<ul style="list-style-type: none"> <li>■ METHOD (A2)</li> <li>■ VERSION (N1)</li> <li>■ SOURCE-ID (A2)</li> <li>■ ACTION-ID (N4)</li> </ul>
INPUT-OUTPUTS (A1/89)	<ul style="list-style-type: none"> <li>■ ACTION-NAME-VARIABLE (A32)</li> <li>■ ACTION-CODE(A12)</li> <li>■ ACTION-NAME (A12)</li> <li>■ ACTION-NAME-LONG (A32)</li> <li>■ ACTION-STATUS (A1)</li> </ul>
OUTPUTS (A1/68)	<ul style="list-style-type: none"> <li>■ METHOD-LIBRARY (A8)</li> <li>■ METHOD-DESCRIPTION (A60)</li> </ul>

- CDPDA-M

External PDA containing standard parameters for exchanging message information.



**Note:** For an example of using this API, refer to the CUBDMBA module.

- CDAAACT

The CDAAACT API module passes 12 occurrences of action IDs and their attributes to the CD--ACT module. CDAAACT accepts the following parameters:

Parameter data area (PDA) containing the following fields:

- CDAAACTA

Field	Redefined As
INPUTS (A1/15)	<ul style="list-style-type: none"> <li>■ METHOD (A2)</li> <li>■ VERSION (N1)</li> <li>■ START-SOURCE-ID (A2)</li> <li>■ START-ACTION-ID (N4)</li> <li>■ MAX-ACTION-REQUESTED (N4)</li> <li>■ RESTRICT-SOURCE-ID (A2)</li> </ul>
INPUT-OUTPUTS (A1/1140)	<ul style="list-style-type: none"> <li>■ ACTION-ARRAY (1:12)</li> </ul> <p>Redefined as:</p> <ul style="list-style-type: none"> <li>■ ACTION-IDENTIFIER (A6)</li> </ul> <p>Redefined as:</p> <ul style="list-style-type: none"> <li>■ ACTION-SOURCE-ID (A2)</li> <li>■ ACTION-ACTION-ID (N4)</li> <li>■ ACTION-NAME-VARIABLE (A32)</li> <li>■ ACTION-CODE (A12)</li> <li>■ ACTION-NAME (A12)</li> <li>■ ACTION-NAME-LONG (A32)</li> <li>■ ACTION-STATUS (A1)</li> </ul>
OUTPUTS (A1/72)	<ul style="list-style-type: none"> <li>■ METHOD-LIBRARY (A8)</li> <li>■ METHOD-DESCRIPTION (A60)</li> <li>■ MAX-ACTION-RETURNED (N4)</li> </ul>

- CDPDA-M

External PDA containing standard parameters for exchanging message information.



**Note:** For an example of using this API, refer to the CUBDMBA module.



## User Exits for the Object-Browse-Dialog Model

The following examples show the User Exits panels for the Object-Browse-Dialog model:

CSGSAMPL Nov 21	Natural Construct User Exits	CSGSM0 1 of 1
User Exit	Exists	Sample Required Conditional
— NAT-DOCS		X
— CHANGE-HISTORY	Subprogram	
— PARAMETER-DATA	Example	
— LOCAL-DATA	Subprogram	
— DEFINE-REPORT-PRINTER		X
— START-OF-PROGRAM		
— WRITE-COLUMN-HEADERS	Example	
— REPORT-HEADERS		X
— REPORT-COLUMN-HEADERS		X
— USER-DEFINED-METHODS	Example	
— BEFORE-CHECK-PFKEYS		
— BEFORE-CALLNAT-SUBPROGRAMS		
— AFTER-CALLNAT-SUBPROGRAMS		
— WRITE-DATA-FIELDS	Subprogram	
— EXPORT-COLUMN-HEADERS		
— EXPORT-DATA-FIELDS	Subprogram	
— BEFORE-CHECK-ERROR	Example	
— ADDITIONAL-TRANSLATIONS		
— BEFORE-OBJECT-CALL		
— AFTER-OBJECT-CALL		
— REPORT-DATA-FIELDS	Subprogram	
— ADDITIONAL-INITIALIZATIONS	Example	
— BEFORE-INPUT		
— SCREEN-HEADERS	Example	X
— INPUT-KEY	Subprogram	X
— AFTER-INPUT		
— BEFORE-PROCESS-ACTIONS		
— AFTER-PROCESS-ACTIONS		
— PROCESS-SELECTED-RECORD	Example	X
— DEFINE-TRANSLATION-HEADERS		X
— TRANSLATE-SCREEN-HEADERS	Example	X
— TRANSLATE-INPUT-KEY		X
— ADDITIONAL-TRANSLATE-MAP		
— TRANSLATE-COLUMN-HEADERS		
— ADDITIONAL-TRANSLATE-TEXT		
— MISCELLANEOUS-SUBROUTINES		
— END-OF-PROGRAM		

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---  
frwr help retrn quit bkwr frwr ←

Using the user exit models, you can create regeneratable Export-Data-Fields, Input-Key, Report-Data-Fields, and Write-Data-Fields user exits.



**Notes:**

1. For information about the user exit models, see the following section.
2. For information about these user exits, see *User Exits for the Generation Models, Natural Construct Generation*.
3. For information about the User Exit editor, see *User Exit Editor, Natural Construct Generation*.

### User Exit Models

This section describes the user exit models supplied for use with the Object-Browse-Dialog model. User exit models generate (and regenerate) text member modules that contain field layout specifications for object-browse dialogs. To define user exits for your object-browse dialogs, use the User-Exit statement model to invoke the text members from the User Exit editor. For more information, see [User-Exit Statement Model](#).

The user exit models support international applications; these models generate text in any language for which translations are available within SYSERR. All SYSERR references are dynamically replaced by text at runtime. For more information, see [Define International Parameters](#). The user exit models also support cursor-sensitive translation, which allows users to change or translate panel headings and prompts while running the generated application. For more information, see [Use SYSERR References for Headings and Prompts](#).



**Note:** The user exit models share the same parameter data area. After modifying the specifications for a user exit model, clear the edit buffer before reading the specifications for another user exit model.

The following table lists the user exit models and describes when to use each:

Model Name	Use To:
Export-Data-Fields	Generate the layout of fields and column headers to be exported to an ASCII file on a PC.
Input-Key	Generate the input fields and prompts for inputting data.
Report-Data-Fields	Generate the layout of fields and column headers to be routed to a printer.
Write-Data-Fields	Generate the layout of fields and column headers to be displayed on a terminal screen.

This section covers the following topics:

- [Export-Data-Fields Model](#)
- [Input-Key Model](#)
- [Report-Data-Fields Model](#)

- Write-Data-Fields Model

### Export-Data-Fields Model

The following example shows a text member generated using the Export-Data-Fields model:

```
Product ; Description ; Unit cost ; Street; City ; Province ; Postal code;
111111 ; DOG FOOD ; 5000 ; 110 ; 21 JUMP STREET ; VICTORIA ; British Columbia; ←
X1E1X1 ;
111116 ; CHEESE DOODLE ; 70 ; 0 ; 2020 UNIVERSITY ; MONTREAL ; Quebec ; H3A2A5 ;
145688 ; HOT CHOCOLATE ; 300 ; 10 ; 5523 ROGERS ROAD ; TORONTO ; Ontario ; M4U1V1 ;
187361 ; CAT NUGGETS ; 70 ; 150 ; 13 SAUTE STREET ; SARNIA ; Ontario ; H1Q1X1 ;
```



#### Notes:

- To view the specifications for this example, refer to the NCPRDEX text member in the Natural Construct demo system.
- To view the generated code for this example, refer to NCPRDOBD in the demo system and scan for "NCPRDEX".

### Input-Key Model

The following example shows a text member generated using the Input-Key model:

```
NCPRDOBD                               Table Subsystem
Oct 14                                  Select Product                               1 of 2

Product          Description          Reorder point    Unit cost
-----
Product ....
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF
help retrn quit trans          exprt bkwrđ frwrđ reprt left right ma ←
```




#### Notes:

1. To view the specifications for this example, refer to the NCPRDIN text member in the Natural Construct demo system.
2. To view the generated code for this example, refer to NCPRDOB in the demo system and scan for "NCPRDIN".

**Report-Data-Fields Model**

The following example shows a text member generated using the Report-Data-Fields model:

NCPRDOB		Table Subsystem		Page 1
Oct 14		Select Product		
Product	Unit cost	Reorder point	Description	
111111	110.00	5000	DOG FOOD	
111116	0.15	70	CHEESE DOODLE	
145688	10.00	300	HOT CHOCOLATE DRINK	
187361	150.00	70	CAT NUGGETS	
199210	100.00	50	COOPER GLOVES	
222222	50.00	88	BIRD SEED	
256733	20.00	22	OATS AND BARLEY CEREAL	
324597	100.00	100	COOPER GLOVES	
333333	50.00	22	DOG BONES	
335977	7.00	40	DOMESTIC KITTY LITTER	
342723	15.00	4000	ORANGE DRINK CRYSTALS	
444444	1.22	1000	CORN FLAKES	↩

 **Note:** To view the specifications for this example, refer to the NCPRDRP text member in the Natural Construct demo system.

**Write-Data-Fields Model**

The following examples show text members generated using the Write-Data-Fields model:

NCPRDOBD		Table Subsystem	
Oct 14		Select Product	
		1 of 2	
Product	Description	Reorder point	Unit cost
111111	DOG FOOD	5000	110.00
111116	CHEESE DOODLE	70	0.15
145688	HOT CHOCOLATE DRINK	300	10.00
187361	CAT NUGGETS	70	150.00
199210	COOPER GLOVES	50	100.00
222222	BIRD SEED	88	50.00
256733	OATS AND BARLEY CEREAL	22	20.00
324597	COOPER GLOVES	100	100.00
333333	DOG BONES	22	50.00
335977	DOMESTIC KITTY LITTER	40	7.00
342723	ORANGE DRINK CRYSTALS	4000	15.00
444444	CORN FLAKES	1000	1.22

Product . . . . \_\_\_\_\_  
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF  
help retrn quit trans exprt bkwrđ frwrđ reprt left right ma

NCPRDOBD		Table Subsystem	
Oct 14		Select Product	
		2 of 2	
Product	Street	City	Province
111111	21 JUMP STREET	VICTORIA	British Columbia
111116	2020 UNIVERSITY	MONTREAL	Quebec
145688	5523 ROGERS ROAD	TORONTO	Ontario
187361	83 SAUTE STREET	SARNIA	Ontario
199210	32 HALL ST	COOPERS TOWN	Ontario
222222	47 FRONTENAC BLVD	QUEBEC CITY	Quebec
256733	45 CRESENT STREET	EDMONTON	Alberta
324597	398 DOWNE ST.	KAMLOOP	Ontario
333333	77 ALBONY CRES	REGINA	Saskatchewan
335977	85 MAIN ST.	STRATFORD	Ontario
342723	3476 BRANTFORD ST.	POINT PEELEY	Ontario
444444	FLAKEYS	FLAKE VILLE	Ontario

Product . . . . \_\_\_\_\_  
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF  
help retrn quit trans exprt bkwrđ frwrđ reprt left right ma



**Note:** To view the specifications for this example, refer to the NCPRDWF1 and NCPRDWF2 text members in the Natural Construct demo system.

## User-Exit Statement Model

The User-Exit statement model generates user exits. This model is invoked from the User Exit editor and uses text members generated by the user exit models. For information about user exit models, see [User Exit Models](#).

This section covers the following topics:

- [Generate Code into the User Exit Editor](#)
- [User-Exit Statement Window](#)

### Generate Code into the User Exit Editor

#### ▶ To generate user exit code using the User-Exit statement model:

- 1 Invoke the User Exit editor from the Generation main menu.

For information about using the User Exit editor, see *User Exit Editor, Natural Construct Generation*.

- 2 Enter the following line command:

```
.g(User-Exit,user exit model name,text member,text member,...)
```

where:

- `user exit model name` is the name of the model used to generate the user exit text members
- `text member` is the name of the user exit text member containing layout specifications (previously generated by one of the user exit models)

The number of text members you can specify varies depending on the target model. Although the user exit models were designed for the Object-Browse-Dialog model, you can also use the models with other compatible models (you may need to override the user exit names).

If you do not specify the name of a text member, the User-Exit Statement window is displayed. For a description of this window, see the following section.

## User-Exit Statement Window

```

USER-EXIT Statement

User exit name _____

User exit text member
  1 _____ *    2 _____ *    3 _____ *
  4 _____ *    5 _____ *    6 _____ *
  7 _____ *    8 _____ *    9 _____ *
 10 _____ *   11 _____ *   12 _____ *
 13 _____ *   14 _____ *   15 _____ *
 16 _____ *   17 _____ *   18 _____ *
 19 _____ *   20 _____ *   21 _____ *
 22 _____ *   23 _____ *   24 _____ *
 25 _____ *   26 _____ *   27 _____ *
 28 _____ *   29 _____ *   30 _____ *

```

The Natural Construct demo system contains examples of using the User-Exit statement model and several user exit text members. To view the sample code, refer to the NCPRDOBD module. The examples were generated using the User-Exit statement model and the following text members:

- NCPRDEX (generated using the Export-Data-Fields model)
- NCPRDIN (generated using the Input-Key model)
- NCPRDRP (generated using the Report-Data-Fields model)
- NCPRDWF1 and NCPRDWF2 (generated using the Write-Data-Fields model)

### Define International Parameters

You can define international parameters for modules generated using the Object-Browse-Dialog and user exit models. International parameters indicate the language used to display text on panels.

#### ▶ To define international parameters:

- 1 Press the intl PF-key (PF9) on the Standard Parameters panel for the model.

The International Parameters window is displayed:

```

CUBDMAA      Natural Construct      CUBDMAA0
Feb 07       International Parameters 1 of 1

Message numbers .... _
Construct prompts .. _

Generate language .. 1_
  Model library ... CSTAPPL
  App library ..... CSTAPPL_

Cursor translation . _
Translation LDAs ... _____ *
                  _____ *
                  _____ *
                  _____ *
                  _____ *

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---P
      help retrn quit
    
```

2 Use the following fields to define the international parameters:

Field	Description
Message numbers	Type of messages used. When this field is marked, the generated code uses message numbers rather than message text.
Construct prompts	Type of prompts used. When this field is marked, the model generates Natural Construct-style prompts (for example, 1 of 2).
Generate language	Code for the language used when generating message text. The default is 1 (English).
Model library	Name of the SYSERR message library used to retrieve common message text. The default is CSTAPPL.
App library	Name of the SYSERR library used to retrieve message text for user-defined SYSERR references. This parameter is only applicable to modules generated using the Object-Browse-Dialog model. If you do not specify an application library, the Model library value is used.
Cursor translation	When this field is selected, the generated code supports cursor-sensitive translation (users can modify or translate panel text dynamically in translation mode). For more information, refer to the following section.
Translation LDAs	Names of the translation local data areas (LDAs) used by modules generated with the Object-Browse-Dialog model. You can specify up to five translation LDAs.  <b>Note:</b> Use the Object-LDA model to create translation LDAs. For information, see <a href="#">Object-LDA Model</a> .

3 Select Enter.



## Cursor-Sensitive Translation

Cursor-sensitive translation allows users to display a specification panel or window in translation mode, select a prompt or heading with their cursor, select Enter, and be presented with a window in which they can change the text:

```

CSUTLATE          Natural Construct
Dec 31           Translate Short Message          1 of 1

Language Short Message ( CSTLDA2000 )
-----  ....+....1....+....2....+....3....+....4....+....5....+....6....+
English  Module/System/Global data area          /+20

```

This window displays the text defined for SYSERR number 2000 in the CSTLDA library for English. Users can also display this window in another language to define heading and prompt text in that language.

For performance and space considerations, multiple screen prompts may share the same SYSERR number and library. When using SYSERR references, each position within the number is identified by a decimal and number. For example, 2000.1 identifies the text, "Module", 2000.2 identifies the text, "System", and 2000.3 identifies the text, "Global data area". The "/+20" notation indicates the maximum number of bytes the corresponding text may occupy on a panel.



**Note:** For information on using SYSERR, see the following section.

## Use SYSERR References for Headings and Prompts

Natural Construct makes it easy to use SYSERR references to define text for some headings and prompts. In any applicable field, you can press the help PF-key to display the Select SYSERR Messages window. From this window, you can:

- Select an existing SYSERR reference to use as a panel heading or field prompt
- Access the Maintain SYSERR Messages window to:
  - change an existing SYSERR reference
  - create a new SYSERR reference

The following table lists the models, panels, and fields for which you can use SYSERR references:

Model	Panel	Field
Object-Browse-Dialog	Standard Parameters	First heading and Second heading
	Additional Parameters	Prompt
Object-LDA	Field Layout Parameters	Field Heading
All user exit models	Field Layout Parameters	Field Heading

This section covers the following topics:

- [Select a SYSERR Reference](#)
- [Add or Maintain a SYSERR Reference](#)

### Select a SYSERR Reference

► To select a SYSERR reference for an applicable field:

- 1 Press the help PF-key when the cursor is in the field.

The Select SYSERR Messages window is displayed:

```

CNHMOBD          Natural Construct
Dec 21           Select SYSERR Messages                1 of 1

Number          Short Message ( English )
-----
CSTAPPL0002    User:1:does not exist
CSTAPPL0003    No matching conversation found for:1:
CSTAPPL0004    API: No function possible after EOC
CSTAPPL0005    Partner finished the conversation
CSTAPPL0006    API: Last message not found
CSTAPPL0007    Service:1:/:2:/:3:not registered
CSTAPPL0008    No related text for error number:1:/:2:
CSTAPPL0009    Conversation found for:1:- no message
CSTAPPL0013    ATTR: Value for keyword too long
CSTAPPL0015    ATTR: Maximum possible number of clients reached
CSTAPPL0016    MQ/OMB entry is already free
CSTAPPL0018    ATTR: Maximum possible number of servers reached
Number ..... __1 Library .... CSTAPPL_ Language ... _1
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10-
      help retrn quit maint          exprt bkwrđ frwr          ←

```

You can use PF-keys in this window to perform other functions. For example:

- Using the maint PF-key, you can modify existing SYSERR references and/or create new references. For more information, see the following section.
- Using the exprt PF-key, you can download the SYSERR messages as an ASCII file to your PC.

- 2 Move the cursor to the reference you want to use.

If you do not see the reference, use the frwr or bkwr PF-keys to scroll to it.

- 3 Select Enter.

The selected SYSERR number is displayed in the field.



**Note:** To specify the library from which SYSERR messages are retrieved, access the International Parameters window. For information, see [Define International Parameters](#).

### Add or Maintain a SYSERR Reference

Use the Maintain SYSERR Messages window to add a new SYSERR reference or maintain existing references.

#### ▶ To create a new SYSERR reference:

- 1 Press the maint PF-key in the Select SYSERR Messages window.

The Maintain SYSERR Messages window is displayed:

```

CNMMOBD          Natural Construct
Dec 21           Maintain SYSERR Messages          1 of 1
                Short Message ( CSTAPPL0001 )
                .....1.....2.....3.....4.....5.....6.....+
English
English _____
Number ..... _1 Library .... CSTAPPL_ Language ... _1
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF
                help  retrn          cnfrm          bkwr frwr          ↵

```

- 2 Type the reference text on the line provided.
- 3 Press the cnfrm PF-key (PF4) to add the new reference.

#### ▶ To modify an existing SYSERR reference:

- 1 Move the cursor to the reference you want to modify in the Select SYSERR Messages window.

If you do not see the reference, use the frwr or bkwr PF-keys to scroll to it.

- 2 Press the maint PF-key.

The selected reference is displayed in the Maintain SYSERR Messages window.

- 3 Modify the text as desired.
- 4 Press the cnfrm PF-key to confirm changes to the text.

## Object-Browse-Dialog-Driver Model

---

Use this model to generate a help routine or driver program that invokes a specified object-browse dialog. This functionality allows you to use the same object-browse dialog for both a maintenance browse function and a field-level help function.

This section covers the following topics:

- [Parameters for the Object-Browse-Dialog-Driver Model](#)
- [User Exits for the Object-Browse-Dialog-Driver Model](#)

### Parameters for the Object-Browse-Dialog-Driver Model

The Object-Browse-Dialog-Driver model has one specification panel: Standard Parameters.

#### Standard Parameters Panel

The following example shows the only specification panel for the Object-Browse-Dialog-Driver model, the Standard Parameters panel:

```

CUODMA          Object-Browse-Dialog-Driver Model          CUODMA0
Nov 28          Standard Parameters                        1 of 1

Module ..... _____
Module type ..... _
System ..... CST821S_____

Title ..... Object Browse Dialog Driv
Description ..... This Object Browse Dialog Driver is used to invoke ...
_____
_____
_____

                                Source      Object
Object-Browse-Dialog _____ *
Messaging support .. _

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

The fields in the upper portion of this panel are similar for all models. For a description of these fields, see *Common Fields on the Standard Parameters Panel*.



**Note:** The Module type for an object-browse-dialog-driver module can be "P", "H", or "N". When the module type is "N" (subprogram), the primary key (#PDA-KEY) can be overridden (similar to a browse subprogram generated by the Browse-Subp model).

The fields in the lower portion of this panel are:

Field	Description
Object-Browse-Dialog	Name of the browse dialog invoked by this driver program.
Source	Name of the first library in which source code for the browse dialog is found. The source code may exist in multiple libraries in the Natural steplib chain.  Natural Construct displays the library name after you enter the name of the browse dialog.
Object	Name of the first library in which object code for the browse dialog is found. The object code may exist in multiple libraries in the Natural steplib chain.  Natural Construct displays the library name after you enter the name of the browse dialog.

### User Exits for the Object-Browse-Dialog-Driver Model

The following example shows the User Exits panel for the Object-Browse-Dialog-Driver model:

```

CSGSAMPL          Object-Browse-Dialog-Driver Model          CSGSM0
Jan 29              User Exits                               1 of 1

          User Exits          Exists   Sample   Required Conditional
-----
_  NAT-DOCS                                X
_  CHANGE-HISTORY                    Subprogram
_  BEFORE-CHECK-ERROR                Example
_  PARAMETER-DATA
_  LOCAL-DATA
_  START-OF-PROGRAM
_  BEFORE-CALLNAT
_  AFTER-CALLNAT
_  ADDITIONAL-INITIALIZATIONS
_  END-OF-PROGRAM

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

```



#### Notes:

1. For information about these user exits, see *User Exits for the Generation Models, Natural Construct Generation*.
2. For information about the User Exit editor, see *User Exit Editor, Natural Construct Generation*.

## Object-LDA Model

---

Using the Object-LDA model, you can generate a local data area (LDA) containing field headings and prompts. You can then use this data area to default field headings for the Object-Browse-Dialog model or user exit models.

You can also use the Object-LDA model to generate a translation LDA for international applications — either by specifying SYSERR references or by generating the module using text in a specified language (which improves performance when you require only one language).

If you specify SYSERR references as your field prompts and headings, instead of hardcoding the actual text, you reduce maintenance requirements and ensure a consistent interface throughout your application.



**Note:** For information, see [Use SYSERR References for Headings and Prompts](#).

### Parameters for the Object-LDA Model

The Object-LDA model has two specification panels: Standard Parameters and Field Layout Parameters. This section describes these panels. The following topics are covered:

- [Standard Parameters Panel](#)
- [Field Layout Parameters Panel](#)

#### Standard Parameters Panel


The following example shows the first specification panel for the Object-LDA model, the Standard Parameters panel:

CUOFMA	Object-LDA Local Data Area	CUOFMA0
Jan 29	Standard Parameters	1 of 2
Module .....	_____	
System .....	CST821S_____	
Title .....	Object LDA for ..._____	
Description .....	This Object LDA is used for ..._	
	_____	
	_____	
Predict view .....	_____	*
Data area .....	_____	*
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12--		
help retrn quit intnl left right main ←		

The fields in the upper portion of this panel are similar for all models. For a description of these fields, see *Common Fields on the Standard Parameters Panel*.

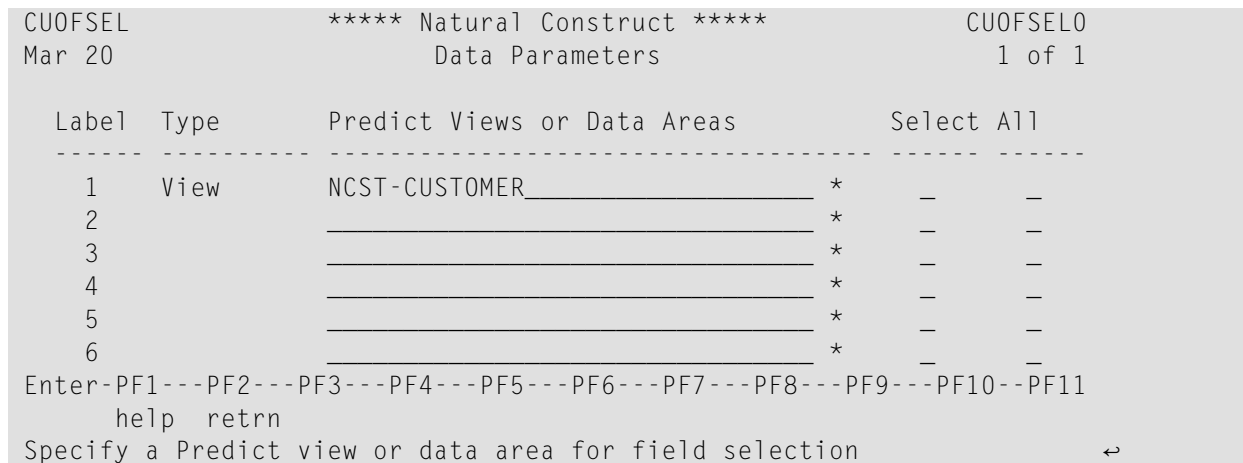
The fields in the lower portion of this panel are:

Field	Description
Predict view	Name of the Predict view from which fields are selected for the screen layout. The specified view must be defined in Predict; this view will be assigned to label 1 in the Data Parameters window (see the following section for more information).  <b>Note:</b> The primary file type can be Adabas, DB2, VSAM, or sequential. All type N (Natural Construct) relationships that specify a cascading delete option and are (directly or indirectly) related to the specified file are included in the generated module.  Field-level help is available to select a Predict view.
Data area	Name of the data area from which fields are selected for the screen layout. Field-level help is available to select a data area.

 **Note:** When using the field-level help on this panel, a window is displayed to select which type of help you require: Predict view, parameter data area, or local data area. After selecting one of these options, the corresponding field-level help window is displayed.

### Select Data Parameters

After selecting a Predict view or data area from the field-level help window, the Data Parameters window is displayed:



This window lists the views or data areas you specified on the Standard Parameters panel and allows you to select all or some of the fields in these files. You can select up to 96 fields from up to 6 different views, local data areas (LDAs), and/or parameter data areas (PDAs). Natural Construct appends the selected fields to CUOFFPDA; you cannot re-select existing fields in CUOFFPDA.

If you are selecting fields from a data area, you cannot select:

- constants
- more than one structure

The fields in this window are:

Field	Description
Label	Number that identifies the Natural label for fields selected for the screen layout. This number is assigned by the Object-LDA model after you select a view or data area (for example, if you select a view and then a data area, the view is assigned 1 and the data area is assigned 2). By using the label number instead of the Natural label (for example, NCST-CUSTOMER.CUSTOMER-NUMBER), field names are shorter and easier to display for selection.
Type	Type of corresponding view or data area (displays View or Structure).
Predict Views or Data Areas	Names of the Predict views, local data areas (LDAs), and/or parameter data areas (PDAs) from which fields are selected for the screen layout.
Select	To display a selection window from which you can select the fields for the screen layout, mark this field and select Enter.



Field	Description
All	<p>To select all fields from the corresponding view or data area, mark this field and select Enter.</p> <ul style="list-style-type: none"> <li>■ When selecting from a view, multiple-valued fields (MUs) within a periodic group (PE) are not included.</li> <li>■ When selecting from a data area, constants and arrays with a rank greater than 1 are not included.</li> </ul> <p><b>Note:</b> If there is more than one structure in a data area, only the first structure is included when you mark this field.</p>

### Field Layout Parameters Panel

The following example shows the second specification panel for the Object-LDA model, the Field Layout Parameters panel. For this example, NCST-CUSTOMER was selected from the Predict view field on the Standard Parameters panel:

CUOFMB		OBJECT-LDA Local Data Area		CUOFMB0		
Mar 20		Field Layout Parameters		2 of 2		
_1	Ord	Lbl	Field name	Field heading		
>>	----	----	-----	-----		
1	10_	1	CUSTOMER-NUMBER_____	Customer Number_____	*	+
2	20_	1	BUSINESS-NAME_____	Business Name_____	*	+
3	30_	1	PHONE-NUMBER_____	Phone Number_____	*	+
4	40_	1	MAILING-ADDRESS_____	Mailing Address_____	*	-
5	50_	1	SHIPPING-ADDRESS_____	Shipping Address_____	*	-
6	60_	1	CONTACT_____	Contact_____	*	+
7	70_	1	CREDIT-RATING_____	Credit Rating_____	*	+
8	80_	1	CREDIT-LIMIT_____	Credit Limit_____	*	+
9	90_	1	DISCOUNT-PERCENTAGE_____	Discount %_____	*	+
10	---	-	_____	_____	*	-
11	---	-	_____	_____	*	-
12	---	-	_____	_____	*	-
13	---	-	_____	_____	*	-
14	---	-	_____	_____	*	-
15	---	-	_____	_____	*	-
16	---	-	_____	_____	*	-
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---						
help retrn quit parms selfd bkwrld frwrld left main ←						

This panel defines the position of fields in the screen layout and the prompt displayed for each field. The fields on this panel are:

Field	Description
_1	<p>Number of the field currently at the top of the panel. Up to 16 fields can be displayed on this panel at one time. To display the specification lines for additional fields, enter the corresponding line number in this field. By default, 1 is displayed.</p> <p><b>Note:</b> You can also display the specification lines for other fields by pressing the frwrdr or bkwrdr PF-keys.</p>
Ord	<p>Current position of the corresponding field in increments of 10. This numbering convention allows you to easily add, remove, or reorder fields. For example, to add a field between the 1st and 2nd fields (positions 10 and 20):</p> <ol style="list-style-type: none"> <li>1. Scroll to the first empty line (you may have to press the frwrdr PF-key).</li> <li>2. Press the selfd PF-key and select a new field.</li> </ol> <p>By default, the new field is assigned the next available number.</p> <ol style="list-style-type: none"> <li>3. Type 15 over the default number (next to the new field).</li> <li>4. Press the reord PF-key.</li> </ol> <p>In this example, the fields are reordered so that the new field is now in the second position (position 20).</p>
Lbl	<p>Number that identifies the Natural label for fields selected for the screen layout. This number is assigned by the Object-LDA model after you select a view or data area (for example, if you select a view and then a data area, the view is assigned 1 and the data area is assigned 2). By using the label number instead of the Natural label (for example, NCST-CUSTOMER.CUSTOMER-NUMBER), field names are shorter and easier to display for selection.</p>
Field Name	<p>Name of a field selected for the screen layout (as defined in Natural). The Object-LDA model supplies this name.</p>
Field Heading	<p>Field prompts displayed in the screen layout. By default, the Predict heading (for views) or INIT clause (for data areas) is displayed. (If no default is available, the Object-LDA model converts the field name to mixed case and uses it as the prompt.) To enable cursor translation or internationalization, use SYSERR references as the field prompts.</p> <p>Field-level help is available to select an existing SYSERR number or create a new SYSERR entry. For more information, see <a href="#">Use SYSERR References for Headings and Prompts</a>.</p>
+ or -	<p>To add more key components to a logical key or set additional options for the logical keys, mark this field and select Enter. A window is displayed in which you can modify the logical key. A + in this field indicates that parameters have been defined for the corresponding field.</p> <p><b>Note:</b> You can also display the window by selecting the parms PF-key on this panel.</p>

## Object-Browse-Select-Subp Model

---

The Object-Browse-Select-Subp model generates a subprogram similar in functionality to a subprogram generated by the Browse-Select-Subp model. Both subprograms allow users to update multiple rows at one time. The primary difference between the two is that an object-browse-select subprogram can accommodate a client/server environment and you can use a subprogram proxy to access the generated code as a business service.

To create the business service, the object-browse-select subprogram accesses an object-browse subprogram and, optionally, an object-maintenance subprogram (generated by the Object-Browse-Subp and Object-Maint-Subp models). The subprogram proxy automatically copies the generated methods to the repository, using the domain and business service names indicated on the model specification panels.



**Note:** If an object-maintenance subprogram is not specified, the MultiMaint, Update, Store, and Delete methods are not generated.

The advantages of using an object-browse-select subprogram include:

- You can update a set of rows at the same time, which reduces the number of calls across the network. Set processing groups a specified number of rows together for faster, less congested, transportation across the network.
- By default, subprograms generated by the Object-Browse-Select model create 20 rows of data, but will reduce this number if the rows are extremely large. This functionality helps accommodate size limitations across the wire.
- You can create additional multiple methods to take advantage of the single methods in the object-maintenance subprogram. Multiple-method processing provides the flexibility to update a set of rows and associate a different method with each row (for example, Add one row, Delete another row, Approve another row, etc.).
- You can add business logic that is not available in the object-browse and/or object-maintenance subprogram.
- You can create one method to represent multiple methods, but also restrict a user at the single-method level. For example, the MultiMaint method can include the Update, Store, and Delete methods. Although two users may be permitted to access this method, one user may be able to perform the Delete action while the other user cannot.
- You can take advantage of the functionality available in an object-browse subprogram. For example, you can specify a non-unique key in the subprogram (such as Warehouse ID to browse the Order file) and use the Object-Browse-Select-Subp model to generate code for a histogram (count).

If the histogram code has been generated, the object-browse-select subprogram generates a method called *KeyNameCount* (for example, WarehouseIDCount). A user can query the Order

file to determine how many orders are stored in each warehouse; after viewing the results of this query, the user can query a specified warehouse in the Warehouse file to view details about orders stored in that warehouse. If required, the user can manipulate data based on the results of this query.

- During generation, the Object-Browse-Select-Subp model determines all the keys used by an object-browse subprogram and automatically generates business-friendly methods (for example, `FindByKeyName`). You can remove any methods that are not applicable and/or change the method names.



**Note:** For information on using the Adabas ISN as a unique primary key for maintenance, see *Use \*ISN as the Unique Primary Key for Maintenance*.

This section covers the following topics:

- [Object-Browse Model Differences](#)
- [Methods Generated](#)
- [Generated Code Differences](#)
- [Suffixes Used by Natural Construct Objects](#)
- [Compatibility with a Subprogram Proxy](#)
- [Specify Leading Fixed Components for the Logical Key](#)
- [Parameters for the Object-Browse-Select-Subp Model](#)
- [User Exits for the Object-Browse-Select-Subp Model](#)

### Object-Browse Model Differences

The following table lists the advantages and disadvantages of using the Object-Browse and Object-Browse-Select models:

Model	Advantage	Disadvantage
Object-Browse	<ul style="list-style-type: none"> <li>■ Speeds up the development process; code generated by this model may already exist.</li> <li>■ Combines data and business layers.</li> </ul>	<ul style="list-style-type: none"> <li>■ Has only one method (BROWSE).</li> <li>■ Places lookup and derived data in a separate level 1 data area; this data cannot be assigned at the ROW level.</li> </ul>
Object-Browse-Select	<ul style="list-style-type: none"> <li>■ Provides more methods; the method names are more descriptive.</li> <li>■ Separates data and business access layers.</li> <li>■ Assigns lookup and derived data at the ROW level with other data.</li> <li>■ Allows users to easily create tables for data; the data is handled like a dataset on the client.</li> </ul>	<ul style="list-style-type: none"> <li>■ Requires an additional subprogram layer.</li> </ul>

## Methods Generated

The Object-Browse-Select-Subp model generates three default methods for use on individual rows:

- Delete
- Store
- Update

This model also generates the MultiMaint method for use on a set of rows. By default, the Multi-Maint method processes the Update, Store, and Delete methods, but can contain any combination of methods.

An object-browse-select subprogram must contain at least one method. Additional methods are based on the key fields and the histogram option specified in the object-browse subprogram. If you select the histogram option for a key field, two methods are created for the field: `FindByKeyName` and `KeyNameCount`. If desired, you can change or delete these names on the specification panels and the method associated with the histogram will just return the key value and the count.

By default, logic is generated to treat each row as a separate transaction, but this can be overridden on the specification panels.

## Transaction States

Two LDAs in the SYSCST library represent the various transaction states and row actions and responses. These are:

- Transaction states are represented in the CDTRACT LDA
- Row actions and row responses are represented in a state format in the CDSTATE LDA

## Generated Code Differences

Code for the Object-Browse-Select-Subp model is generated based on whether the object-maintenance subprogram was generated with the hash-locking or with the timestamp record locking option.



**Note:** If the file is not normally maintained through a Natural Construct-generated object-maintenance module, the object-maintenance subprogram should use the hash-locking record locking option. If the file is only maintained through an object-maintenance module, the subprogram should use the timestamp record locking option (as it is more efficient).

If the object-maintenance subprogram uses the hash-locking option and the target file is Adabas, the GET-BY-ISN option is used to retrieve the data in the object-maintenance subprogram.

You can change the record-locking method on the client by specifying a transaction style. The transaction styles are:

- Aggressive Row Object

Each row is treated as a transaction, so each row is committed as it is processed. If an error occurs in a row, it is noted and processing continues to the next row. This transaction style is the default, unless the default is changed on the server in user exit code.

■ **Passive Row Object**

An End of Transaction statement is issued when all the rows are processed; if an error occurs in a row, processing is stopped and everything is backed out.

■ **Business Service Object**

No End of Transaction statement is issued by the business service; the client is expected to issue the ET. This allows the client to confirm that the data is committed to another proprietary solution before the data in the business service is committed.

■ **Unique Object**

The generated code makes no assumptions; the programmer must manually commit all transactions.



**Note:** If you do not specify a transaction style, the default is used (Aggressive Row Object).

**Suffixes Used by Natural Construct Objects**

The following table lists Natural Construct object types and the suffixes assigned to each:

Natural Construct Object Type	Suffix Assigned
Browse data array	D
Browse extended row PDA	E
Browse key information	K
Browse private information	P
Browse static object row PDA	S
Business service	B
Interface map data PDA (fields to be shown to the user)	M
Interface-retained PDA data (fields that maintain State data; they are required over a network call, but they are not shown to the user)	X
Object-maintenance LDA (used with the hash-locking option)	H
Object-maintenance PDA	A
Object-maintenance restricted PDA	R
Object subprogram	N
Subprogram proxy	Y

## Compatibility with a Subprogram Proxy

A subprogram generated by the Subprogram-Proxy model works with the object-browse-select subprogram to generate the appropriate methods: the methods specified in the Object-Browse-Select-Subp specifications and the MultiMaint, Update, Store, and Delete methods. The Subprogram-Proxy model is also used to name the business service and associate it with a domain and version. For more information on subprogram proxies, see *Natural Business Services Subprogram-Proxy-Client Model*.

## Specify Leading Fixed Components for the Logical Key

You can override the default number of leading fixed key values for the logical key by defining the LEADING-FIXED-COMPONENTS field in the CDBUPDA2 data area for the object-browse-select subprogram.

As a generated object-browse-select subprogram uses the CDBUPDA data area by default, you must activate the CDBUPDA2 subprogram using the CSXDEFLT subprogram.



### Notes:

1. For more information on LEADING-FIXED-COMPONENTS, see [CDBRPDA](#).
2. For information on CSXDEFLT, see *Use CSXDEFLT Overrides*.

## Parameters for the Object-Browse-Select-Subp Model

The Object-Browse-Select-Subp model has two specification panels: Standard Parameters and Additional Parameters. This section describes these panels. The following topics are covered:

- [Standard Parameters Panel](#)
- [Additional Parameters Panel](#)
- [Parameters Passed to the Object-Browse-Select Subprogram](#)

### Standard Parameters Panel

The following example shows the first specification panel, the Standard Parameters panel:

```

CUBUMA          Object-Browse-Select-Subp Multi-Module          CUBUMA1
Nov 16          Standard Parameters                              1 of 2

Module ..... BCUSTN__
System ..... S51PTYPE_____

Title ..... Object Browse Select ....
Description ..... This subprogram is used to encapsulate data access_____
                    for ..._____
                    _____
                    _____

                                Primary File .....

Object browse subp ..... * _____
Object maint subprogram . * _____
Time ..... _

Static occurrences ..... 20_ PDA      Generate      Source      Object
                                BCUSTS20 *      _

Message numbers .... _

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

The fields in the upper portion of this panel are similar for all models. For a description of these fields, see *Common Fields on the Standard Parameters Panel*.

The fields in the lower portion of this panel are:

Field	Description
Object browse subp	Name of the subprogram used to browse the object.
Object maint subprogram	Name of the subprogram used to maintain the object (optional). The object-maintenance subprogram cannot process intra-object relationships. This allows the data presented to the client to be manageable and all data to be modifiable.  <b>Note:</b> If you use an object-maintenance subprogram and object-browse subprogram, both subprograms must use the same primary file.
Time	When this field is selected, code is generated to time how long a business service takes to execute. The result is returned in the business service message. The utility used to time the response is available from the Natural Construct Administration main menu as follows:  Drivers > Additional Drivers > General Utilities > Calculate Seconds  <b>Note:</b> This field is only available for mainframe platforms.
Static occurrences	Number of rows processed and sent across the network at one time (by default, 20, unless the rows are extremely large). The PDA (parameter data area) associated with



Field	Description
	<p>the static occurrences hardcodes this value (which is used to identify the <math>\vee</math> value in the object-browse subprogram) in the object-browse-select subprogram.</p> <p>To identify the number of occurrences in this PDA, the default PDA name contains the first five characters of the module name and the number of static occurrences ("BCUSTS20" in the example).</p> <p><b>Note:</b> If you change the number of static occurrences on this panel, you should also change this number in the default PDA name.</p>
Message numbers	<p>When this field is selected, message numbers are used and messages are retrieved from SYSERR at runtime.</p> <p><b>Note:</b> If the object-browse and/or object-maintenance subprograms specified on this panel use message numbers, the object-browse-select subprogram will also use message numbers.</p>

### Additional Parameters Panel

The following example shows the second specification panel, the Additional Parameters panel:

```

CUBUMB          Object-Browse-Select-Subp Multi-Module          CUBUMB1
Nov 16          Additional Parameters                            2 of 2

      Method Name                Count   Browse Key Name
      -----
01  FindByCustomerNumber_____  _     CUSTOMER-NUMBER
02  FindByBusinessName_____    _     BUSINESS-NAME
03  FindByCustomerWarehouseId___  _     CUSTOMER-WAREHOUSE-ID
04  FindByBusinessWarehouse_____  _     BUSINESS-WAREHOUSE
05  CustomerWarehouseIdCount_____ X     CUSTOMER-WAREHOUSE-ID

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

```

This panel shows the default methods for the business service, which were generated based on the keys specified for the object-browse subprogram. Notice that there are two methods associated with CUSTOMER-WAREHOUSE-ID. This indicates that the histogram option in the object-browse

subprogram has been marked for this key. If the CustomerWarehouseIdCount method is used, only the unique key values and the count will be returned.

You can override or delete methods listed on this panel. If you delete a method, it will not be available to the business service.



**Note:** To restore the default values based on the current object-browse subprogram, press PF5 (deflt).

### Parameters Passed to the Object-Browse-Select Subprogram

Generated object-browse-select subprograms accept parameters from the same parameter data areas (PDAs) as generated object-browse subprograms. For information, see [Parameters Passed to the Object-Browse Subprogram](#).

In addition, the generated subprograms accept the following parameters:

```
03 EXTRA-ROW-DATA
  04 ROW-STATE(A2)
  04 ROW-ID (N5) /* for internal use; do not change
  04 ROW-ERROR-DATA
    05 ###ERROR-FIELD (A32)
    05 ###MSG-NR (N4)
    05 ###MSG (A79)
PARAMETER USING CDBUPDA          /* Business service status data
PARAMETER USING CDBUINFO        /* Business service message data ↵
```

These parameters are:

Parameter	Description
Row PDA	<p>Similar to the row PDA for object-browse subprograms, except it is internal to the object-browse-select subprogram and contains extra data. For example:</p> <ul style="list-style-type: none"> <li>■ ROW-STATE Contains maintenance instructions or maintenance message codes.</li> <li>■ ROW-ID Used internally on the client (do not change).</li> <li>■ ROW-ERROR-DATA Contains message information returned by the object-maintenance subprogram for the affected row (i.e., transaction states).</li> </ul>
CDBUPDA	<p>Subset of the CDBRPDA PDA for object-browse subprograms. CDBUPDA contains all the parameters in CDBRPDA except the following:</p> <ul style="list-style-type: none"> <li>■ METHOD</li> </ul>

Parameter	Description
	<p>Not required.</p> <ul style="list-style-type: none"> <li>■ SORT-KEY</li> </ul> <p>The object-browse-select subprogram sets this value for CDBRPDA based on the FindBy* methods.</p> <ul style="list-style-type: none"> <li>■ HISTOGRAM</li> </ul> <p>The object-browse-select subprogram sets this value for CDBRPDA based on the Count* methods).</p> <ul style="list-style-type: none"> <li>■ ROWS-REQUESTED</li> </ul> <p>Not required.</p> <ul style="list-style-type: none"> <li>■ LEADING-FIXED-COMPONENTS</li> </ul> <p>Set in the CDBUPDA2 parameter data area (for information, see <a href="#">Specify Leading Fixed Components for the Logical Key</a>).</p> <ul style="list-style-type: none"> <li>■ USE-UNIQUE-ID</li> </ul> <p>The object-browse-select subprogram sets this value.</p>
CDBUINFO	<p>Similar to the CDPDA-M parameter data area for object-browse subprograms; this PDA contains messages and transaction information for all rows. For example:</p> <pre> 1 CDBUINFO 2 ###TRANSACTION-STYLE          A          1 /*See CDTRACT for valid va 2 #BACKOUT- ISSUED              L 1 BUSINESS- INFO 2 ###MSG                        A          79 2 ###MSG- NR                    N          4 2 ###RETURN- CODE              A          1 </pre>

**User Exits for the Object-Browse-Select-Subp Model**

CSGSAMPL		Object-Browse-Select-Subp Multi-Module		CSGSM0	
Nov 16		User Exits		1 of 1	
User Exits		Exists	Sample	Required	Conditional
-----					
—	NAT-DOCS				X
—	CHANGE-HISTORY		Subprogram		
—	PARAMETER-ROW				
—	PARAMETER-DATA		Example		
—	LOCAL-DATA				
—	START-OF-PROGRAM				
—	USER-DEFINED-METHODS				
—	USER-DEFINED-SUCCESSFUL-STATE				X
—	USER-DEFINED-PENDING-STATE				X
—	BEFORE-BROWSE-OBJECT				
—	AFTER-BROWSE-OBJECT				
—	BEFORE-CHECK-BUSINESS-ERROR				
—	BEFORE-CHECK-ERROR				
—	BEFORE-BT-PROCESSING				X
—	AFTER-BT-PROCESSING		X		
—	BEFORE-ET-PROCESSING				X
—	AFTER-ET-PROCESSING		X		
—	END-BUSINESS-SERVICE				
—	ADDITIONAL-INITIALIZATIONS				
—	START-ROW-PROCESSING				X
—	BEFORE-CALL-TO-MAINT-OBJECT				X
—	AFTER-CALL-TO-MAINT-OBJECT				X
—	STATE-FOR-ABORTED-TRANSACTIONS				X
—	ROW-STATE-INPUT-CONVERSION				X
—	DEFAULT-TRANSACTION-STYLE				X
—	UNIQUE-TRANSACTION-STYLE				X
—	MISCELLANEOUS-SUBROUTINES				
—	END-OF-PROGRAM				
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---					
help retrn quit		bkwrđ frwrđ		←	



**Notes:**

1. Conditional user exits are based on whether an object-maintenance subprogram was specified.
2. For information about these user exits, see *User Exits for the Generation Models, Natural Construct Generation*.
3. For information about the User Exit editor, see *User Exit Editor, Natural Construct Generation*.

# 3 Using the Object-Generic-Subp Model

---

- Introduction ..... 112
- Parameters for the Object-Generic-Subp Model ..... 112
- User Exits for the Object-Generic-Subp Model ..... 115

This section describes the Object-Generic-Subp model, which generates a business service (a *wrapper* subprogram) associated with up to 10 subprograms and 20 methods. The following topics are covered:



**Note:** For information about object-oriented development, see [Overview of Object-Oriented Development](#).

## Introduction

---

The Object-Generic-Subp model:

- Creates a subroutine for each subprogram used by the business service (maximum of 10 subprograms)



**Note:** You can create additional subroutines within user exits to perform specialized functions.

- Creates methods to call the subroutines and user exits (maximum of 20 methods).

## Parameters for the Object-Generic-Subp Model

---

The Object-Generic-Subp model has two specification panels: Standard Parameters and Additional Parameters. This section describes these panels. The following topics are covered:

- [Standard Parameters Panel](#)
- [Additional Parameters Panel](#)

### Standard Parameters Panel

The following example shows the first specification panel, the Standard Parameters panel:

```

CUOGMA          OBJECT-GENERIC-SUBP Subprogram          CUOGMA0
Aug 20          Standard Parameters                    1 of 2

Module ..... NEW_____
System ..... BIZDEMO_____

Title ..... Generic Business Service_
Description ..... This subprogram is used to maintain the generic_____
                    business service ....._____
                    _____
                    _____


Message numbers .... _ Categorize parameters _

Subprograms
_____ *          _____ *          _____ *          _____ *          _____ *
_____ *          _____ *          _____ *          _____ *          _____ *

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

```

The fields in the upper portion of this panel are similar for all models. For a description of these fields, see *Common Fields on the Standard Parameters Panel*.

 **Note:** If you select the Categorize Parameters option, the PARAMETER-DATA user exit is required. For information on this exit, see [PARAMETER-DATA User Exit](#). For information on categorizing parameters, see *Categorize Parameters*.

The subprograms listed in Subprograms in the lower portion of this panel:

- Should have no screen I/O or navigation functionality (i.e., they cannot contain INPUT, WRITE, PRINT, DISPLAY, REINPUT statements or manage PF key functions)
- Must have at least one parameter

You can specify the names of up to 10 subprograms; you must specify at least one subprogram name.

## Additional Parameters Panel

The following example shows the second specification panel, the Additional Parameters panel:

Before Code	Order	Subroutine	After Code
—	—	CALLNAT-ACUSTN_____	—
—	—	CALLNAT-BCUSTN_____	—

Method ... 1 \_\_\_\_\_ of 20

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

Use this panel to name the methods used by your business service and to indicate the functionality of each method (i.e., which subprogram to execute and the order of execution for the subprograms specified on the Standard Parameters panel and wrapped in the subroutines listed). In addition:

- You must define at least one method
- A subprogram is part of a method if an order number is assigned to it
- Each order (sequence) number must be unique; you cannot use the same number more than once
- You cannot mark the Before Code or After Code fields unless an order (sequence) number is specified
- If a level 1 parameter grouping is in more than one subprogram, the first one encountered is the one that is used



## User Exits for the Object-Generic-Subp Model

The following example shows the User Exits panel for the Object-Generic-Subp model:

CSGSAMPL		OBJECT-GENERIC-SUBP Subprogram			CSGSM0	
Aug 19		User Exits			1 of 1	
User Exits	Exists	Sample	Required	Conditional		
— NAT-DOCS				X		
— CHANGE-HISTORY		Subprogram				
— PARAMETER-DATA		Subprogram	X	X		
— PARAMETER-DATA-UNCATEGORIZED				X		
— LOCAL-DATA						
— MOVE-TO						
— MOVE-TO-UNCATEGORIZED				X		
— UNDEFINED-METHOD						
— BEFORE-CODE		Subprogram		X		
— AFTER-CODE		Subprogram		X		
— MATERIALIZE-XARRAY-PDA-TO-LDA				X		
— MATERIALIZE-XARRAY-LDA-TO-PDA				X		
— RESET-TEMP-MATERIALIZED				X		
— MOVE-BACK						
— MOVE-BACK-UNCATEGORIZED				X		
— MISC-SUBROUTINES						

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---  
 help retrn quit bkwrdr frwrdr ↩



### Notes:

1. For information about the standard user exits, see *User Exits for the Generation Models, Natural Construct Generation*.
2. For information about the User Exit editor, see *User Exit Editor, Natural Construct Generation*.

The following user exits are either required by or specific to the Object-Generic-Subp model:

- AFTER-CODE User Exit
- BEFORE-CODE User Exit
- MATERIALIZE-XARRAY-LDA-TO-PDA User Exit
- MATERIALIZE-XARRAY-PDA-TO-LDA User Exit
- MOVE-BACK User Exit
- MOVE-BACK-UNCATEGORIZED User Exit
- MOVE-TO User Exit
- MOVE-TO-UNCATEGORIZED User Exit
- PARAMETER-DATA User Exit

- PARAMETER-DATA-UNCATEGORIZED User Exit
- RESET-TEMP-MATERIALIZED User Exit
- UNDEFINED-METHOD User Exit

## AFTER-CODE User Exit

The code in this exit is executed after all subprograms that had the After Code option selected on the Additional Parameters panel have been executed. If method-specific code is required, you can add it based on the value of +METHOD (indicates which business service method is executed).

To only execute the portion of the code associated with a specific subprogram, ensure that the appropriate code is specified in VALUE in the DECIDE clause associated with the corresponding subroutine (i.e., only the code in the VALUE "CALLNAT-GCDN" clause will be executed when the CALLNAT-GCDN subroutine invokes it). For example, the CALLNAT-GCDN subroutine contains a CALLNAT to the GCDN subprogram and this code is executed after that CALLNAT.

The following example shows code in the AFTER-CODE user exit for the BNUM subprogram in the SYSCSTDE library:

```

DEFINE EXIT AFTER-CODE
** Note +METHOD can also be used to
**     determine lines of execution
**     e.g. IF +METHOD = ... THEN
    DECIDE ON FIRST VALUE OF #SUBROUTINE-NAME
        VALUE "CALLNAT-GCDN"
            IF +METHOD = 'SolutionWithLowerNumbers'
/*         Lower the first number by the GCD
           #FUNCTION := 'Divide'
           INPUT-DATA.#SECOND-NUM := GCD-DATA.#RESULT
           PERFORM CALLNAT-CALC
/*         Instead of using temporary variables; temporarily used
/*         exposed field variables
           #BIZ-INPUT-OUTPUTS.#FIRST-NUM := OUTPUT-DATA.#RESULT
/*         Lower the second number by the GCD
           INPUT-DATA.#FIRST-NUM :=
               #BIZ-INPUT-OUTPUTS.#SECOND-NUM
           INPUT-DATA.#SECOND-NUM := GCD-DATA.#RESULT
           PERFORM CALLNAT-CALC
           #BIZ-INPUT-OUTPUTS.#SECOND-NUM := OUTPUT-DATA.#RESULT
/*         Move results to Calc input again to do actual division
/*         of reduced numbers
           MOVE BY NAME #BIZ-INPUT-OUTPUTS TO INPUT-DATA
        END-IF
        IF +METHOD = 'GreatestCommonDenominator'
            IF GCD-DATA.#RESULT > 1 THEN
                OUTPUT-DATA.#SUCCESS := TRUE
            ELSE
                OUTPUT-DATA.#SUCCESS := FALSE
            END-IF
        END-IF
    END-IF

```

```

    NONE
    IGNORE
    END-DECIDE
END-EXIT

```

### BEFORE-CODE User Exit

The code in this exit is similar to the code in the AFTER-CODE user exit except it is executed before the corresponding subroutine is executed. If method-specific code is required, you can add it based on the value of +METHOD (indicates which business service method is executed).

The following example shows code in the BEFORE-CODE user exit for the BSTRINGN subprogram in the SYSCSTDE library:

```

DEFINE EXIT BEFORE-CODE
** Note +METHOD can also be used to
**   determine lines of execution
**   e.g. IF +METHOD = ... THEN
    DECIDE ON FIRST VALUE OF #SUBROUTINE-NAME
        VALUE "CALLNAT-CSUCASE" /* U=Upper, L=Lower, M=Mixed Case
            DECIDE ON FIRST VALUE OF +METHOD
                VALUE 'ConvertToUpperCase'
                    CSACASE.#FUNCTION := 'U'
                VALUE 'ConvertToLowerCase'
                    CSACASE.#FUNCTION := 'L'
                VALUE 'ConvertToMixedCase'
                    CSACASE.#FUNCTION := 'M'
            ANY
                EXAMINE FULL #BIZ-INPUT-OUTPUTS.#STRING FOR ' '
                    GIVING LENGTH IN #BIZ-INPUT-OUTPUTS.STRING-LENGTH
            NONE
                IGNORE
            END-DECIDE
        IGNORE
    NONE
    IGNORE
    END-DECIDE
END-EXIT

```

### **MATERIALIZED-XARRAY-LDA-TO-PDA User Exit**

This exit is used when you add X-array fields to the object generic PDA. It is used in conjunction with the `MATERIALIZED-XARRAY-PDA-TO-LDA` and `RESET-TEMP-MATERIALIZED` user exits. The code in this exit prepares for a `MOVE BY NAME` from a local data area (LDA) containing X-arrays to a parameter data area (PDA) containing similar X-arrays. This code temporarily resizes X-arrays before performing the `MOVE BY NAME` to the PDA.

These exits are only required when an X-array parameter is added to the object generic PDA (in one of the `PARAMETER-DATA` exits) and has not been included in the supplied subprograms. This code eliminates runtime errors when X-arrays have not been sized before a `MOVE BY NAME` is performed.



**Tip:** To use these exits, refer to the code preceding the exits that was generated for known X-arrays.

### **MATERIALIZED-XARRAY-PDA-TO-LDA User Exit**

This exit is used when you add X-array fields to the object generic PDA. It is used in conjunction with the `MATERIALIZED-XARRAY-LDA-TO-PDA` and `RESET-TEMP-MATERIALIZED` user exits. The code in this exit prepares for a `MOVE BY NAME` from a parameter data area (PDA) containing X-arrays to a local data area (LDA) containing similar X-arrays.

For more information, see [MATERIALIZED-XARRAY-LDA-TO-PDA User Exit](#).

### **MOVE-BACK User Exit**

This exit is used in conjunction with the `MOVE-TO` user exit and the `PARAMETER-DATA` user exit, which contains the data that is exposed to the client from the object generic subprogram. After the internal subprograms have been invoked, the data must be exposed via the parameter data area (PDA). The local variables are moved to the parameter variables in the `MOVE-BACK` user exit.

For more information, see [PARAMETER-DATA User Exit](#).

### **MOVE-BACK-UNCATEGORIZED User Exit**

This exit is used in conjunction with the `MOVE-TO-UNCATEGORIZED` user exit and the `PARAMETER-DATA-UNCATEGORIZED` user exit, which contains the data that is exposed to the client from the object generic subprogram. After the internal subprograms have been invoked, the data must be exposed via the parameter data area (PDA). The local variables are moved to the parameter variables in the `MOVE-BACK-UNCATEGORIZED` user exit.

For more information, see [PARAMETER-DATA-UNCATEGORIZED User Exit](#).

## MOVE-TO User Exit

This exit is used in conjunction with the MOVE-BACK user exit and the PARAMETER-DATA user exit, which contains the data that is exposed to the client from the object generic subprogram. To pass this data to subprograms, the data must be moved to local data areas in the MOVE-TO user exit.

For more information, see [PARAMETER-DATA User Exit](#).

## MOVE-TO-UNCATEGORIZED User Exit

This exit is used in conjunction with the MOVE-BACK-UNCATEGORIZED user exit and the PARAMETER-DATA-UNCATEGORIZED user exit, which contains the data that is exposed to the client from the object generic subprogram. To pass this data to subprograms, the data must be moved to local data areas in the MOVE-TO-UNCATEGORIZED user exit.

For more information, see [PARAMETER-DATA-UNCATEGORIZED User Exit](#).

## PARAMETER-DATA User Exit

The PARAMETER-DATA user exit is required if you specified the Categorize Parameters option on the Standard Parameters panel. This exit is used in conjunction with two other exits: MOVE-TO and MOVE-BACK. (For more information on categorizing parameters, see *Categorize Parameters*.)

The object generic subprogram wraps up to 10 subprograms into one subprogram. The PARAMETER-DATA user exit contains the data that is exposed to the client from the object generic subprogram. To pass this data to the subprograms, it must be moved to local data areas in the MOVE-TO user exit. Similarly, after the internal subprograms have been invoked, the data must be exposed via the parameter data area (PDA). The local variables are moved to the parameter variables in the MOVE-BACK user exit.

The PARAMETER-DATA user exit allows the user to choose which level 1 parameter groupings will be input, input-output, state, and output. The same parameter name cannot be listed under the same input, input-output, state, or output groupings. If this occurs, you must revise the generated code.

To help select the level 1 parameter groupings, the following panel is displayed when you select Enter on the User Exits panel for the Object-Generic-Subp model:

CUOGMC Nov 16		Natural Construct Object-Generic-Subp Subprogram Build Report				CUOGMCO 1 of 1	
1__	Level Ones	Input	Input-Output	State	Output		
1	ACUSTNK	—	—	—	—		
2	ACUSTND	—	—	—	—		
3	ACUSTNP	—	—	—	—		
4	CDBRPDA	—	—	—	—		
5	MSG-INFO	—	—	—	—		
6	BCUSTE1	—	—	—	—		
7	CDBUPDA	—	—	—	—		
8	CDBUINFO	—	—	—	—		
9	BUSINESS-INFO	—	—	—	—		

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---  
 help retrn gen bkwrđ frwrđ ↵

Based on the grouping, data is moved from the exposed PDAs to the internal LDAs used for the subprograms. You can define up to 100 level 1 parameter groupings. Up to four unique PDAs can be duplicated across the subprograms.



**Note:** While using the NCSTBGEN command to regenerate multiple modules in batch mode, object generic subprograms may not be regenerated. For example, if the parameters have been categorized (i.e., defined within user exits), you must regenerate the PARAMETER-DATA user exit from the client.

### Structure of the Generated Code

The following example shows the skeleton view of code generated by the PARAMETER-DATA user exit:

```
DEFINE DATA
PARAMETER
1 #INPUT
...
1 #INPUT-OUTPUT
...
1 #STATE
...
1 #OUTPUT
LDAs
END-DEFINE
```

```

MOVE BY NAME Pdas to Ldas
** SAG EXIT POINT AFTER-PDA-TO-LDA-MOVE
DECIDE ON FIRST VALUE OF +METHOD
VALUE 'ABC'
    EXECUTE-BEFORE := TRUE (optional)
    EXECUTE-AFTER := TRUE (optional)
    PERFORM nnnn2-CALLNAT
    EXECUTE-AFTER := TRUE (optional)
    PERFORM nnnn1-CALLNAT
VALUE 'DEF'
    EXECUTE-BEFORE := TRUE (optional)
    EXECUTE-AFTER := TRUE (optional)
    PERFORM nnnn3-CALLNAT
    EXECUTE-AFTER := TRUE (optional)
    PERFORM nnnn1-CALLNAT
END-DECIDE
MOVE BY NAME LDAS to PDAs
** SAG EXIT POINT AFTER-LDA-TO-PDA-MOVE
*
DEFINE SUBROUTINE nnnn1-CALLNAT
    SUBROUTINE-NAME := 'nnnn1-CALLNAT'
    IF EXECUTE-BEFORE THEN
        PERFORM BEFORE
    END-IF
    CALLNAT 'nnnn1' ...
    IF EXECUTE-AFTER THEN
        PERFORM AFTER
    END-IF
    RESET EXECUTE-BEFORE EXECUTE-AFTER
END-SUBROUTINE
*
DEFINE SUBROUTINE nnnn2-CALLNAT
    SUBROUTINE-NAME := 'nnnn1-CALLNAT'
    IF EXECUTE-BEFORE THEN
        PERFORM BEFORE
    END-IF
    CALLNAT 'nnnn1' ...
    IF EXECUTE-AFTER THEN
        PERFORM AFTER
    END-IF
    RESET EXECUTE-BEFORE EXECUTE-AFTER
END-SUBROUTINE
*
DEFINE SUBROUTINE nnnn3-CALLNAT
    SUBROUTINE-NAME := 'nnnn1-CALLNAT'
    IF EXECUTE-BEFORE THEN
        PERFORM BEFORE
    END-IF
    CALLNAT 'nnnn1' ...
    IF EXECUTE-AFTER THEN
        PERFORM AFTER
    END-IF

```

```

    RESET EXECUTE-BEFORE EXECUTE-AFTER
END-SUBROUTINE
*
DEFINE SUBROUTINE BEFORE
    EXECUTE-BEFORE := FALSE
** User Exit BEFORE Code
* Note that +METHOD can also be used in this logic
    DECIDE ON FIRST VALUE OF SUBROUTINE-NAME
        VALUE 'nnnn1-CALLNAT'
            IGNORE
        VALUE 'nnnn2-CALLNAT'
            IGNORE
        NONE
            IGNORE
    END-DECIDE
** User Exit End code
    ESCAPE ROUTINE
END-SUBROUTINE
*
DEFINE SUBROUTINE AFTER
    EXECUTE-AFTER := FALSE
** User Exit AFTER Begin Code
* Note that +METHOD can also be used in this logic
    DECIDE ON FIRST VALUE OF SUBROUTINE-NAME
        VALUE 'nnnn1-CALLNAT'
            IGNORE
        VALUE 'nnnn2-CALLNAT'
            IGNORE
        NONE
            IGNORE
    END-DECIDE
** User Exit End code
    ESCAPE ROUTINE
END-SUBROUTINE

```

### PARAMETER-DATA-UNCATEGORIZED User Exit

Use this exit if you want to expose more parameters to the client than are found in the specified subprograms and you did not specify the Categorize Parameters option on the Standard Parameters panel (for example, you can use this exit to expose a message field if the specified subprograms do not have one). This exit is optional and is used in conjunction with the MOVE-TO-UNCATEGORIZED and MOVE-BACK-UNCATEGORIZED user exits. The MOVE-TO-UNCATEGORIZED and MOVE-FROM-UNCATEGORIZED user exits are similar to the MOVE-TO and MOVE-FROM exits for the PARAMETER-DATA user exit except they are used when the Categorize Parameters option is not selected.

If you decide not to categorize parameters, every PDA from the specified subprograms will be exposed to the client. The subprogram created by the object generic subprogram will have two types of variables: parameters that will become the parameters of the business service and local data that will become the parameters to the supplied subprograms.



Initially, code is automatically generated into the MOVE-TO and MOVE-FROM exits when the Categorize Parameters option is specified. This does not happen when the option is not selected, as more code can be generated outside of user exits.

### **RESET-TEMP-MATERIALIZED User Exit**

The code in this exit temporarily resizes X-arrays before performing the MOVE BY NAME to the LDA or PDA. Use this exit when you add X-array fields to the object generic PDA.

This exit is used in conjunction with the MATERIALIZE-XARRAY-LDA-TO-PDA and MATERIALIZE-XARRAY-PDA-TO-LDA user exits. For more information, see [MATERIALIZE-XARRAY-LDA-TO-PDA User Exit](#).

### **UNDEFINED-METHOD User Exit**

The code in this exit determines what happens when an undefined method is added to the object generic subprogram and has not been included in the specifications.



**Note:** In general, the repository should access the same methods as the object generic code. If not, use this exit to define the new methods.



# 4 Using the Object-Maint Models

---

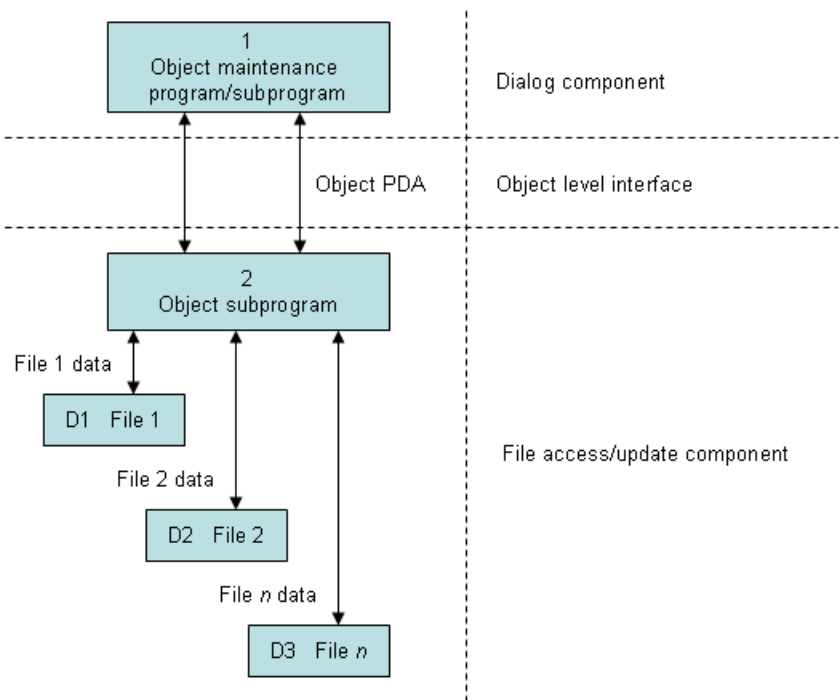
- Introduction ..... 126
- Object-Maint-Subp and Object-Maint-Enhanced-Subp Models ..... 128
- Object-Maint-Dialog Model ..... 152
- Object-Maint-Dialog-Subp Model ..... 167

This section describes how to use the Object-Maint series of models to generate the modules required for an object-maintenance process. The following topics are covered:

 **Note:** For more information on object-oriented development, see [Overview of Object-Oriented Development](#).

## Introduction

The following diagram shows the components of an object-maintenance process:



Within this hierarchy, the object-maintenance dialog program is not concerned with the internal structure of the files (which is hidden by the PDA), nor the implementation of the data actions (which is hidden by the object subprogram).

The following table lists the modules required to maintain a maintenance object and the models used to generate each module:

Module	Model used to Generate
Object-maintenance subprogram	Object-Maint-Subp or Object-Maint-Enhanced-Subp model  <b>Note:</b> The Object-Maint-Enhanced-Subp model is similar to the Object-Maint-Subp model, except it supports the generation of fields in the object PDA as dynamic fields.
Object-maintenance dialog program	Object-Maint-Dialog model
Object-maintenance dialog subprogram	Object-Maint-Dialog-Subp model

To maintain an object, such as add a new record or modify an existing one, an object-maintenance dialog invokes an object-maintenance subprogram.

► **To implement an object-maintenance process using the Object-Maint models:**

- 1 Define the files and relationships in the Predict data dictionary.

Identify the object, the integrity between objects, and the automatic rules that apply to each object. For more information, see [Define Natural Construct Objects](#).

- 2 Create the subprogram using the Object-Maint-Subp model.

This subprogram updates all entities within the object. Subprograms generated by the Object-Maint-Subp model contain the full range of integrity checks (as defined by the Predict relationships) and object semantics, whether they are in the form of Predict automatic rules or object manipulation within user exits.

The Object-Maint-Subp model also creates the parameter data areas (PDAs) for the object: the object PDA and the restricted PDA. The object PDA contains fields to store all occurrences of attributes defined in the object. This PDA is the only part of the object that is exposed to the rest of the application (for update purposes only). The restricted PDA stores information that is used internally by the subprogram. The values in this PDA must only be altered by the subprogram.

- 3 Create maps to input values for the object (if a dialog program invokes the object).

Use the Map model or Natural Map editor to create the maps, which extract fields from the object PDA.

- 4 Create the object-maintenance dialog program or subprogram using the Object-Maint-Dialog or Object-Maint-Dialog-Subp model.

This module provides the user interface to the object.

The following sections describe the Object-Maint models in the order they are implemented.

## Object-Maint-Subp and Object-Maint-Enhanced-Subp Models

---

This section describes the Object-Maint-Subp and Object-Maint-Enhanced-Subp models, which generate object-maintenance subprograms and corresponding PDAs. The generated subprograms update all entities within an object and contain a full range of integrity checks (as defined by Predict relationships) and object semantics (in the form of Predict automatic rules or object manipulation within user exits). The main difference between these models is that the Object-Maint-Enhanced-Subp model can generate large fields in the object PDA as dynamic fields. This allows long fields to occupy only the space required to pass the data to the database view. For example, one customer may require 1000 characters for delivery instructions and another customer only requires 50 characters. In the first case, 1000 characters will be placed in the parameter data area (PDA) and in the second case only 50 characters will be placed in the PDA.

The Object-Maint-Enhanced-Subp model allows you to take advantage of larger field sizes available in Natural and in the databases. In the past, an alphanumeric field in Natural was restricted to a length of 253 characters. To accommodate larger fields, you had to create an array of strings with a length of less than 254 characters each. This meant that words in a note, for example, may have been split across strings. Using this model, you can specify larger string sizes in the files and in Natural to allow the entire note to fit in one string. The model can also generate code to truncate trailing blanks, which can needlessly increase the amount of data going into the PDA, and generate an error message when a user enters data into a field that is longer than what the database expects.

This section covers the following topics:

- [Object Instance Hierarchy Tree](#)
- [CDAOBJ2 Data Area](#)
- [Generated Data Areas](#)
- [Data Access Subroutines](#)
- [Store "Before" Images](#)
- [Editing and Processing of Entities](#)
- [Additional Checks within User Exits](#)
- [Parameters for the Object-Maint-Subp Model](#)
- [User Exits for the Object-Maint-Subp Model](#)
- [Parameters for the Object-Maint-Enhanced-Subp Model](#)

- [User Exits for the Object-Maint-Enhanced-Subp Model](#)

## Object Instance Hierarchy Tree

An object is built from a primary entity and its child entities (sub-entities), which are defined in Predict with entity relationships. An instance (object value) of the object consists of occurrences (records) of the constituent entities. Depending on the update constraint type specified, the following interpretations of null occurrence are adopted:

- For update constraint type C (Cascade), an entity record is set to null if its key suffix value is set to null. An exception to this occurs when the length of the key for the child entity is equal to the length of the key for its parent entity (there is no suffix, for example). In this case, a record is set to null if all non-key attributes are null.
- For update constraint type L (suffix is a line number) and type N (renumbered suffix), an entity record is set to null if all the non-key attributes are set to null.

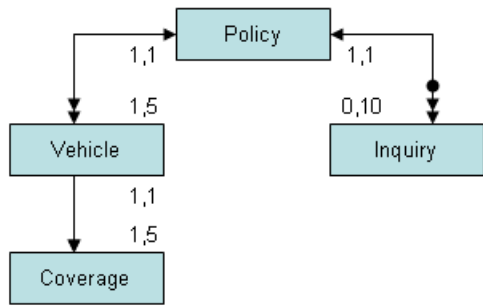
Each instance of the object can be represented by an object instance hierarchy tree, where the occurrence of the primary entity forms the node at the root of the tree and each occurrence of its child entities forms a node at a lower level. A null occurrence of an entity within the object does not correspond to any node of the hierarchy tree. With this representation, the following properties can be observed:

- Each non-null occurrence of an entity must correspond to a non-null occurrence of its parent entity; if an entity occurrence is set to null, so are all the occurrences of its child entities. This property of existence can be seen as a downward propagation from parent to child. If a record is set to null during an update, that record and all its child records are deleted.
- The attributes (field values) of an entity occurrence can propagate downward to those of its child entities. This type of propagation can be seen while traversing a pre-order tree. When a node is encountered for the first time, it can take on the attributes of its parent node. (To implement this propagation, refer to the *V0-entity-name* and *V1-entity-name* subroutines in the UPDATE-EDITS user exit.)
- The attributes of an entity occurrence can propagate upward to those of its parent entity. This type of propagation can be seen while traversing a post-order tree. When a node is encountered for the last time, the entity occurrence can contribute to the attributes of its parent since all of its attributes (and its child attributes) are already processed. (To implement this propagation, refer to the *V2-entity-name* subroutine in the UPDATE-EDITS user exit.)

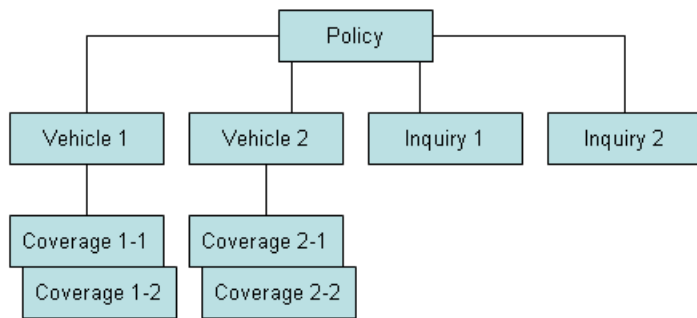


**Tip:** An instance of an object can be referred to as an instance of a class.

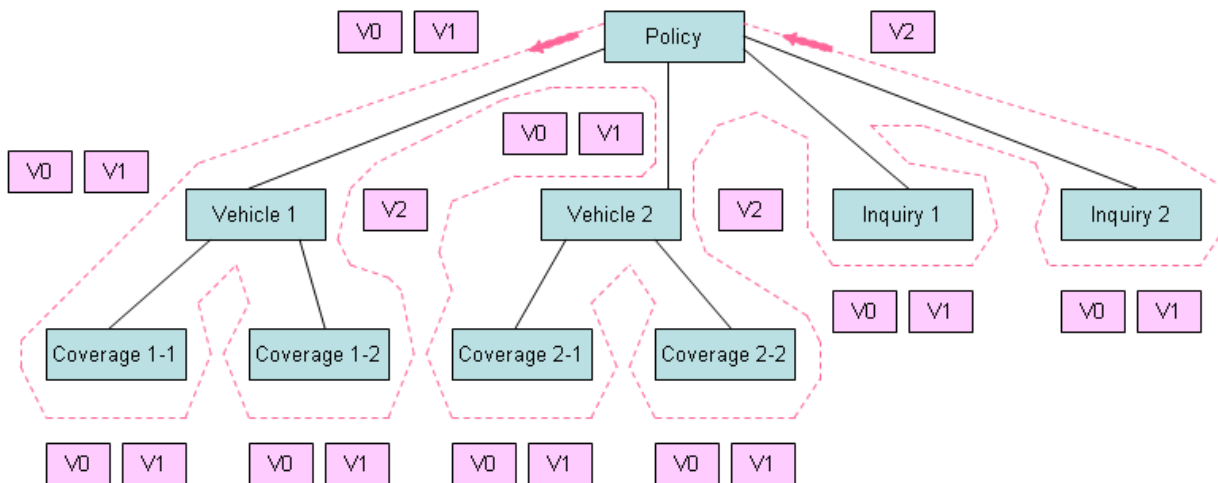
Consider an insurance policy object defined with the following entity relationships:



An object instance consisting of a policy with two inquiries and two vehicles, where the first vehicle has two coverages and the second has three, can be represented by the following object hierarchy tree:



This object hierarchy tree can be equally represented by the following diagram, which illustrates the pre-order and post-order traversing of a tree:





In this diagram, the V0 and V1 on the left side or bottom of each node represent the V0-*entity-name* and V1-*entity-name* subroutines; the V2 on the right side of the node represents the V2-*entity-name* subroutine for the corresponding entity. In this example, the following subroutines are involved:

```
V0-INS-POLICY and V1-INS-POLICY
V2-INS-POLICY
V0-INS-VEHICLE and V1-INS-VEHICLE
V2-INS-VEHICLE
V0-INS-COVERAGE and V1-INS-COVERAGE
V0-INS-INQUIRY and V1-INS-INQUIRY
```

Each node corresponding to an occurrence of the INS-COVERAGE and INS-INQUIRY entities does not have any child nodes and is called a leaf (of the tree). While traversing the object instance hierarchy tree, the first time a leaf is encountered is also the last time. Therefore, a leaf does not have a V2-*entity-name* subroutine.

The Object-Maint-Subp model generates PERFORM-*subroutine* statements that allow attributes to propagate with the V0-, V1-, or V2-*entity-name* subroutines.

### Example of PERFORM Statements

The following example shows PERFORM statements generated by the Object-Maint-Subp model:

```
*PROCESS INS-POLICY
  *FOR EACH POLICY:
    PERFORM V0-INS-POLICY
    PERFORM INS-POLICY-PREDICT-VERIFICATIONS
    (Check Predict automatic rules for INS-POLICY)
    PERFORM V1-INS-POLICY
  *PROCESS INS-VEHICLE
  *PROCESS INS-INQUIRY
  PERFORM V2-INS-POLICY
  *PROCESSING FOR THE INS-POLICY

*PROCESS INS-VEHICLE
  *FOR EACH VEHICLE:
    PERFORM V0-INS-VEHICLE
    PERFORM INS-VEHICLE-PREDICT-VERIFICATIONS
    (Check Predict automatic rules for INS-VEHICLE)
    PERFORM V1-INS-VEHICLE
  *PROCESS INS-COVERAGE
  PERFORM V2-INS-VEHICLE
  *PROCESSING FOR THE INS-VEHICLE

*PROCESS INS-COVERAGE
  *FOR EACH COVERAGE:
    PERFORM V0-INS-COVERAGE
    PERFORM INS-COVERAGE-PREDICT-VERIFICATIONS
    (Check Predict automatic rules for INS-COVERAGE)
    PERFORM V1-INS-COVERAGE
```

```
*PROCESSING FOR THE INS-COVERAGE

*PROCESS INS-INQUIRY
  *FOR EACH INQUIRY:
    PERFORM V0-INS-INQUIRY
    PERFORM INS-INQUIRY-PREDICT-VERIFICATIONS
    (Check Predict automatic rules for INS-INQUIRY)
    PERFORM V1-INS-INQUIRY
  *PROCESSING FOR THE INS-INQUIRY
```

## CDAOBJ2 Data Area

The Object-Maint-Subp model uses the CDAOBJ2 data area, which contains parameters that are common to all object-maintenance subprograms. For example:

```
1 CDAOBJ2
*
* This data area contains all
* parameters that are common to
* OBJECT-MAINT-SUBPrograms.
*
2 INPUTS
3 #FUNCTION (A15) /* GET, NEXT, UPDATE, DELETE,
*                               /* STORE, EXISTS, INITIALIZE
*                               /* Other User Defined Functions.
4 #CLEAR-AFTER-UPDATE (L) /* Initialize object variables
*                               /* after a successful UPDATE,
*                               /* DELETE or STORE.
4 #RETURN-OBJECT (L)
4 #ET-IF-SUCCESSFUL (L) /* Commit the record updates

4 #USE-ISN (L)
/* If the OBJECT was
/* generated with the
/* condition code
/* GET-BY-ISN and
/* this flag is true the
/* GET-OBJECT and
/* HOLD-OBJECT subroutines
/* will retrieve the
/* record by ISN

4 #IGNORE-HELD-ID-CHECK (L) /* Can be used to ignore
*                               /* the HELD-ID check;
*                               /* This check can be
*                               /* ignored if hash locking
*                               /* is used
4 #BACKOUT-ISSUED (L) /*when true the Object Maint issued a backout transaction
2 OUTPUTS
4 #OBJECT-CONTAINS-DERIVED-DATA (L)
4 #EXISTS (L) /* Requested object exists.
```

### Conditional END OF TRANSACTION (ET) Statement

The Object-Maint-Subp model supports a conditional END OF TRANSACTION (ET) statement. When client and server components are on different platforms, the ET logic is not easily transmitted across the network. To make this process simpler and more automated, the Object-Maint-Subp model generates a conditional ET statement that is controlled by two logical variables: #UPDATE-PERFORMED and #ET-IF-SUCCESSFUL.

Both variables must be set to True before an ET is performed. The #UPDATE-PERFORMED variable is internally set in the object-maintenance subprogram (depending on the method that was requested). The #ET-IF-SUCCESSFUL variable is set by the callers of the subprogram and passed across different platforms via the CDAOBJ2 data area.

If both components reside on the same platform:


- The object-maintenance dialog modules can continue to issue the ET as before (by default, the dialog module issues the ET)

or

- The object-maintenance subprogram can perform the ET

The following conditional statement is generated:

```
IF #UPDATE-PERFORMED AND CDAOBJ2.#ET-IF-SUCCESSFUL THEN
  END OF TRANSACTION
END-IF
```

-  **Note:** If upgrading a generated object-maintenance subprogram from Natural Construct V3 to version 4 or higher, you must also regenerate the calling dialogs. The order of generation is important; first regenerate the object-maintenance subprograms and then regenerate the dialog modules.

### GET-BY-ISN Option

The Natural Construct administrator can set up the Object-Maint-Subp model to generate code that retrieves data by ISN for Adabas files. This functionality allows a GET BY ISN statement to be converted into a CALLNAT to the object-maintenance subprogram. The GET BY ISN code is only executed if:

- CDAOBJ2.#USE-ISN is set to True
- ObjectName.OBJECT-ISN has a value
- the NEXT or FORMER actions are not being used

If an object is generated with the GET-BY-ISN condition code and the #USE-ISN field set to True, the GET-OBJECT and HOLD-OBJECT subroutines retrieve the record by ISN.

-  **Notes:**

1. To determine whether the GET-BY-ISBN condition code is set, see *Determine Which Condition Codes are Set*.
2. For information on using the Adabas ISBN as a unique primary key for maintenance, see *Use \*ISBN as the Unique Primary Key for Maintenance*.

### **Generated Data Areas**

The Object-Maint-Subp model generates the object and restricted PDAs:

#### **Object PDA**

This PDA allows data to be transferred between an object-maintenance subprogram and the object-maintenance dialog program or subprogram, and/or any other programs that invoke the object-maintenance subprogram.

The following example shows a PDA generated by the Object-Maint-Subp model:

```

15:06:58          ***** E D I T DATA *****          13-10-16
Library: SYSCSTDE      Name: ORDERPDA PARAMETER          DBID: 18 FNR: 4
Command:                                     > +
I T L Name                    F Leng  Index/Init/EM/Name/Comment
-----
  1 ORDER                      /* Object Name
  2 ORDER-NUMBER                N    6 /*
  2 ORDER-AMOUNT                P 13.2 /*
  2 ORDER-DATE                  N    8 /*
  2 ORDER-CUSTOMER-NUMBER      N    5 /*
  2 ORDER-WAREHOUSE-ID         A    3 /*
  2 INVOICE-NUMBER              N    6 /*
  2 ORDER-TIMESTAMP            T     /*
  2 C#DELIVERY-INSTRUCTIONS     N    3 /* Counter Field
  2 DELIVERY-INSTRUCTIONS      A   60 (1:20)
*
  2 C#NCST-ORDER-HAS-LINES     N    3 /* Counter field
  2 NCST-ORDER-HAS-LINES       (1:30) /* NCST-ORDER-LINES
  3 LINE-NUMBER                 N    2 /*
  3 ORDER-PRODUCT-ID           A    6 /*
  3 LINE-DESCRIPTION            A   40 /*
  3 QUANTITY                    P    9 /*
  3 UNIT-COST                   P   7.2 /*
  3 TOTAL-COST                  P   9.2 /*
*
  3 C#NCST-LINE-HAS-DISTRIBUTION N    3 /* Counter field
  3 NCST-LINE-HAS-DISTRIBUTION (1:10) /* NCST-ORDER-DISTRIBUTION
  4 DIST-LINE-NUMBER            N    2 /*
  4 DIST-NUMBER                 N    2 /*
  4 ACCOUNT                     A    9 /*
R 4 ACCOUNT                     /* REDEF. BEGIN : ACCOUNT
  5 COST-CENTER                 A    2 /*
  5 ACCT                         A    4 /*
  5 PROJECT                     A    3 /*
  4 DIST-AMOUNT                 P   9.2 /*
*
  1 ORDERPDA-ID                 N    6 /* Object identifier
R 1 ORDERPDA-ID                 /* REDEF. BEGIN : ORDERPDA-ID
  2 STRUCTURE                   /* To allow MOVE BY NAME
  3 ORDER-NUMBER                 N    6 /*

```

## Restricted PDA

The Object-Maint-Subp model also generates the restricted object PDA. The generated object-maintenance subprogram uses the restricted object PDA to store information that is used across multiple applications. An example of such information is the Adabas ISNs (Internal Sequence Numbers) of all entities within an object when the object is read. In this way, the entities can be easily retrieved for an Update action. The actual contents of the restricted PDA are only used internally by the generated object-maintenance subprograms.

For more information, see [Define Object Relationships in Predict](#) and [Support for Predict Automatic Rules](#).



**Note:** An object-maintenance subprogram has no user-interface component. For more information, see [Parameters for the Object-Maint-Subp Model](#). To see a sample subprogram, refer to ORDERN in the Natural Construct demo system.

## Data Access Subroutines

If a Natural object contains a FIND statement that must be converted to an object-maintenance subprogram, you can create a new data access function and code the FIND statement in user exits. To accommodate this functionality, and the GET BY ISN data access statement, certain code has been placed in subroutines. This allows the same code to be executed — regardless of the access method. These subroutines are:

- HOLD-PRIMARY-RECORDS-FOUND (when the data is accessed with a hold)
- GET-PRIMARY-RECORDS-FOUND (when the data is accessed without a hold)
- NO-PRIMARY-RECORDS-FOUND

For examples of these subroutines, refer to the GET-OBJECT and HOLD-OBJECT routines.

## Store "Before" Images

An object-maintenance subprogram generated by the Object-Maint-Subp model stores a "before" image of data (for example, what an order looked like before a user made changes). The before image is kept on the database and is re-requested before an update is performed.

To do this, the code must be generated with the hash-locking feature. Logical variables are then stored as Alpha format in the local data area to process the hashed values. All data has to hash to the same value as when the data was requested. If it does, then the data has not changed.



**Note:** For information about hash locking, see [Hash-Locking Option](#).



**Tip:** As the local data area is populated with the original data, you can use this data in your own logic.

## Editing and Processing of Entities

An object consists of a primary entity and all its child entities (sub-entities). Each entity is processed in the following order:

1. Pre-editing checks, which consist of all the edit checks done before the child or children of the current entity are processed.
2. Processing, during which the current entity is updated, added, or deleted.
3. Post-editing checks, which consist of all the edit checks done after the child or children of the current entity are processed.

This section covers the following topics:

- [Automatic Validation](#)
- [Processing Order in Adabas Files](#)
- [Processing Order in Non-Adabas Files](#)
- [Pre-editing Checks](#)
- [Post-editing Checks](#)

### Automatic Validation

The generated subprogram performs automatic validation using information stored in Predict. It checks for:

- The uniqueness of a key, if required
- Foreign referential constraints (inter-object relationships)
- Predict automatic rules
- Cardinality constraints for Predict relationships

### Processing Order in Adabas Files

In Adabas files, Natural Construct processes each entity in the following order:

1. Performs pre-edit checks.
2. Processes all children.
3. Performs post-editing checks.
4. Adds, updates, or deletes the entity.

## Processing Order in Non-Adabas Files

In non-Adabas files (VSAM, DB2, DL1/IMS), Natural Construct processes each entity in the following order:

### Add or Update Action

1. Performs pre-edit checks on the current entity.
2. Adds or updates the entity.
3. Processes all children of the entity.
4. Performs post-edit checks.



**Note:** For VSAM, DB2, or DL1/IMS files with a primary key, if the key for an entity is updated to a new value, the record with the new key value is added before the child records are processed and the record with the old key value is deleted after the child records are processed. Otherwise, the key is updated as usual.

### Delete Action

1. Performs pre-edit checks on the current entity.
2. Processes all children of the entity.
3. Deletes the entity.



**Note:** For relational database and DL1/IMS files with referential integrity rules (Predict type R relationships) defined for intra-object relationships with type C (Cascade) constraint type, the DELETE statement is generated only at the primary level. The DBMS handles the cascading delete through all child records.

### Pre-editing Checks

This editing is performed before the children of the current entity are processed. Natural Construct creates pre-editing subroutines (called EDIT-OBJECT for the primary entity and *E-entity-name* for sub-entities) and executes them in the following order:



### Add or Update Action

1. Builds the key for the current entity.
2. Ensures the uniqueness of the key (if required).
3. Executes the *V0-entity-name* subroutine within the UPDATE-EDITS user exit.
4. Enforces the Predict automatic rules.
5. Executes the *V1-entity-name* subroutine within the UPDATE-EDITS user exit.
6. Enforces the Restricted Update for Insertion (RUI) rules.
7. Enforces the Restricted Update (RU) rules (if the entity is greater than level 1).

### Delete Action

1. Executes the *D-entity-name* subroutine within the DELETE-EDITS user exit.
2. Enforces the Restricted Delete (RD) rules.

### Post-editing Checks

This editing is performed by the *V2-entity-name* subroutine after the children of the current entity are processed. Natural Construct generates the `PERFORM V2-entity-name` statement in the following subroutines:

- CHECK-AND-UPDATE-OBJECT (for the primary entity)
- *C-entity-name* (for the sub-entity)

Post-editing allows the upper level to maintain some desired redundancy. For example, an insurance policy requires a premium for each vehicle insured under the policy. For performance reasons, the policy has a redundant field called POLICY-TOTAL-PREMIUM. You can determine the total premium by looking at the primary entity for the object; you do not have to go through all the vehicle entities.

### Additional Checks within User Exits

This section describes different uses for user exits supplied for the Object-Maint-Subp model. The following topics are covered:

- [Provide Conditional ET Statements within User Exits](#)

- Specify Validation Subroutines

### Provide Conditional ET Statements within User Exits

In addition to a conditional END OF TRANSACTION (ET) statement, the Object-Maint-Subp model offers user exits called BEFORE-ET, BEFORE-ET-PROCESSING and AFTER-ET-PROCESSING. These exits provide the same capabilities for the ET statement in the object-maintenance subprogram as are available in dialog modules. The following conditional statement and user exits (if requested) are generated:

```

**SAG DEFINE EXIT BEFORE-ET
* Any special processing before an ET, where this code will be executed
* whether an ET is issued or not.
**SAG END-EXIT
    IF #UPDATE-PERFORMED AND CDAOBJ2.#ET-IF-SUCCESSFUL THEN

IF #UPDATE-PERFORMED AND CDAOBJ2.#ET-IF-SUCCESSFUL THEN
**SAG DEFINE EXIT BEFORE-ET-PROCESSING
    /* Any special processing before an ET.
**SAG END-EXIT
    END OF TRANSACTION

**SAG DEFINE EXIT AFTER-ET-PROCESSING
    /* Any special processing after an ET.
**SAG END-EXIT
END-IF

```

For information about these user exits, see *BEFORE-ET*, *BEFORE-ET-PROCESSING* and *AFTER-ET-PROCESSING*, *Natural Construct Generation*.

### Specify Validation Subroutines

You can specify additional edit checks within the UPDATE-EDITS user exit and additional referential integrity checks within the EXTENDED-RI-CHECKS user exit. The UPDATE-EDITS user exit contains validation subroutines that execute edit checks at different points during the processing of an entity. You can create subroutines for each entity within an object.

The UPDATE-EDITS user exit contains the following validation subroutines:

Subroutine	Description
V0- <i>entity-name</i>	Executed during the pre-editing phase, before the Predict automatic rules are checked and the children of the current entity are processed.
V1- <i>entity-name</i>	Executed during the pre-editing phase, after the Predict automatic rules are checked and before the children of the current entity are processed.
V2- <i>entity-name</i>	Executed during the post-editing phase, after the Predict automatic rules are checked and all children of the current entity are processed.

The EXTENDED-RI-CHECKS user exit contains the following validation routine:

Subroutine	Description
V- <i>relationship-name</i>	Executed during the pre-editing phase, after the Predict automatic rules are checked and after the V1- <i>entity-name</i> subroutine for the current entity is executed.

For more information about these validation subroutines and user exits, see [Object Instance Hierarchy Tree](#) and *UPDATE-EDITS* and *EXTENDED-RI-VIEWS*, *Natural Construct Generation*.

### Parameters for the Object-Maint-Subp Model

The Object-Maint-Subp model has two specification panels: Standard Parameters and Additional Parameters. This section describes these panels. The following topics are covered:

- [Standard Parameters Panel](#)
- [Additional Parameters Panel](#)



**Note:** For information about creating an object-maintenance process, see *Design Methodology*, *Natural Construct Generation*.

### Standard Parameters Panel

The following example shows the first specification panel, the Standard Parameters panel:

```

CUOBMA          Object-Maint-Subp Subprogram          CUOBMA0
Jan 25          Standard Parameters                  1 of 2

Module ..... MCUST2N_
System ..... DEMO_____

Title ..... Object Title_____
Description ..... Object description_____
                    for..._____
                    _____
                    _____

Message numbers .... X
Hash locking ..... _

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

```

The fields in the upper portion of this panel are similar for all models. For a description of these fields, see *Common Fields on the Standard Parameters Panel*. For information on the hash-locking option, see *Hash-Locking Option*.

### Additional Parameters Panel

The following example shows the second specification panel, the Additional Parameters panel:

```


CUOBMB          Object-Maint-Subp Subprogram          CUOBMB0
Aug 11          Additional Parameters                  2 of 2
Predict view ..... _____ *
Primary key ..... _____ *
Hold field ..... _____ *

Object description ..... _____

Object PDA ..... _____ *      Generate   Source   Object
Restricted PDA ..... _____ *      _         C421    C421
Object name ..... _____

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

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

 **Tip:** If the Predict view is blank and there is a value in Object PDA, you can enter the name of another PDA (which must be generated by Natural Construct and available in the current library) in Object PDA to populate the Predict view, Primary key and Object name fields with the values from this PDA.

The fields on this panel are:

Field	Description
Predict view	Name of the Predict view. A file definition for this view must exist in Predict. Predict type N (Natural Construct) relationships relating to the primary file are processed by the generated object-maintenance subprogram. Relationships defined with a cascading delete constraint are maintained as part of the object; relationships defined with a restricted delete constraint are used by the object-maintenance subprogram to implement referential constraints.
Primary key	Name of the key in Predict for the primary file. This key becomes the primary key to access the view for maintenance. The key can be a descriptor, superdescriptor, or subdescriptor. If the key does not exist in the specified Predict file, an error message is displayed.

Field	Description
Hold field	<p>Name of the field used to logically protect the record against intervening update or delete actions. Because an object-maintenance subprogram does not use the record-holding facilities of the DBMS to lock records during a GET operation, a <i>hold</i> field must exist in the primary file for the object. Valid data types are:</p> <ul style="list-style-type: none"> <li>■ T *TIMX</li> <li>■ A10 *TIME</li> <li>■ B8 *TIMESTAMP</li> <li>■ N7 *TIMN</li> <li>■ A26 *TIMX (DB2 time stamp format)</li> <li>■ If the format is none of the above, it must be numeric.</li> </ul> <p><b>Note:</b> If the hash-locking method is used, this field is not displayed. For more information, see <a href="#">Hash-Locking Option</a>.</p>
Object description	Object description used in messages. If you specify "Person", for example, messages are displayed as "Person not found" and "Person displayed".
Object PDA	Name of the parameter data area (PDA) used in conjunction with the object-maintenance subprogram. For more information, see <a href="#">Object PDA</a> .
Restricted PDA	Name of the restricted PDA used in conjunction with the object-maintenance subprogram. For more information, see <a href="#">Restricted PDA</a> .
Generate	If a generated PDA is not found in the steplib chain, this field is marked and protected. Natural Construct will generate the PDA.
Source	<p>Name of the first library in which the source code for the module is found. The source code for the module may exist in multiple libraries in the Natural steplib chain.</p> <p>If the source code resides in the current library, regenerating it will execute a STOW command and overwrite the previous version.</p>
Object	<p>Name of the first library in which the object code for the module is found. The object code for the module may exist in multiple libraries in the Natural steplib chain.</p> <p>If the object code resides in the current library, regenerating it will execute a STOW command and overwrite the previous version.</p> <p><b>Note:</b> If the Generate field is marked, the PDAs specified on this panel are generated and stowed when the object-maintenance subprogram is generated — regardless of whether the subprogram is stowed.</p>
Object name	<p>Name of the level 1 structure used to qualify the fields in the object PDA. (It is easier to identify the source of these attributes if the PDA name is used for this purpose.) The object name should be kept to a reasonable length.</p> <p><b>Note:</b> The object name cannot match the name of a file included in the object, nor any field in the object.</p>

Field	Description
Next action prefix	<p>If the primary key is compound or redefined into various components, supply a value to limit the number of prefixed components confined on the Next action. This allows the subprogram to maintain objects with a common prefix value.</p> <p>For example, if the primary key is made up of Company + Account + Division and you do not want the Next action to span the Division values, specify "2". Specify "1" if the Next action is to be limited to the current Company value.</p>
Log file suffix	<p>If you want to log objects, you have to create a log file corresponding to each entity within the object. The name of the log file is the name of the object file concatenated with the suffix specified here. For example, if the object consists of the NCST-ORDER-HEADER and NCST-ORDER-LINES entities and you specify "-LOG", the log file names are NCST-ORDER-HEADER-LOG and NCST-ORDER-LINES-LOG.</p> <p>The following fields are required in the log file that corresponds to the header entity in the object:</p> <ul style="list-style-type: none"> <li>■ LOG-TIME <p>Assigned with *TIMX for T format or *TIMN for N7 format.</p> </li> <li>■ LOG-DATE <p>Assigned with *DATX for D format or *DATN for N8 format. (If LOG-TIME has an embedded date, such as *TIMX, this field is not required.)</p> </li> <li>■ LOG-TID <p>Assigned with *INIT-ID.</p> </li> <li>■ LOG-USER <p>Assigned with *INIT-USER.</p> </li> <li>■ LOG-ACTION <p>Assigned with the #ADD, #MODIFY, or #PURGE log action codes, which are defined in the CDACTLOG local data area. You can initialize the values for these log action codes within CDACTLOG to suit your environment.</p> </li> </ul> <p>In the log files corresponding to the sub-entities in the object, only the LOG-ACTION field is required.</p> <p><b>Note:</b> For relational databases, use the underscore (_) character instead of the dash (-) for the log field names (LOG_TIME, LOG_DATE, LOG_TID, LOG_USER, LOG_ACTION).</p>
Trace relationships	<p>When this field is selected, Natural Construct displays the relationships it has accepted or rejected. During the generation process, all accepted and rejected relationships are displayed with a message indicating the type of relationship.</p>

## Hash-Locking Option

The Natural Construct administrator can change optimistic record locking from the default timestamp method to the hash-locking method. When the hash-locking method of record locking is specified, the Hold field is not available on the Additional Parameters panel. Instead, an Object LDA field is displayed, showing the name of the object local data area generated for the object-maintenance subprogram. For example:

CUOBMB	Object-Maint-Subp	Subprogram	CUOBMB0
Jan 13	Additional Parameters		2 of 2
Predict view .....	NCST-ORDER-HEADER_____	*	
Primary key .....	ORDER-NUMBER_____	*	
Object description .....	ORDER_____		
		Generate	Source
Object PDA .....	ORDERPDA *	X	SHDEMO
Restricted PDA .....	ORDERPDR *	X	SHDEMO
Object name .....	ORDER_____		
** Object LDA is generated when Object PDA is generated			
Object LDA .....	ORDERNH_ *		SHDEMO
Next action prefix .....	_		
Log file suffix .....	_____		
Trace relationships .....	_		

The hash-locking method retains the functionality of the object-maintenance subprogram. The only difference is that it checks all the object data, not just the timestamp, to ensure there have been no intervening modifications.

If the hash-locking method was specified:

- An object LDA is generated
- The generated code contains the #HASH-RETRIEVE and #HASH-DATABASE fields in the restricted PDA and will reference a Natural user exit called USR4011N

## User Exits for the Object-Maint-Subp Model

The following example shows the User Exits panel for the Object-Maint-Subp model:

CSGSAMPL Aug 27		OBJECT-MAINT-SUBP Subprogram User Exits	Exists	Sample	Required	Conditional
—	NAT-DOCS					X
—	CHANGE-HISTORY			Subprogram		
—	PARAMETER-DATA			Example		
—	EXTENDED-RI-VIEWS					
—	LOCAL-DATA			Example		
—	START-OF-PROGRAM			Example		
—	SELECT-STATEMENT			Subprogram		X
—	USER-DEFINED-FUNCTIONS			Example		
—	BEFORE-ET			Example		X
—	BEFORE-ET-PROCESSING			Example		
—	AFTER-ET-PROCESSING			Example		
—	PROCESS-ERROR-MESSAGE					
—	ERROR-MESSAGE-PDAS					
—	END-OF-PROGRAM			Example		
—	BEFORE-STORE			Example		
—	AFTER-STORE					
—	AFTER-GET			Example		
—	BEFORE-DELETE					X
—	AFTER-INIT			Example		
—	UPDATE-EDITS			Subprogram		
—	DELETE-EDITS			Subprogram		
—	AFTER-GET-EDITS			Subprogram		
—	EXTENDED-RI-CHECKS			Subprogram		
—	ADJUST-OBJECT-ID-IN-MSG			Example		
—	AFTER-UPDATE					
—	OVERRIDE-MINIMUM			Example		
—	OVERRIDE-MAXIMUM			Example		
—	MISCELLANEOUS-SUBROUTINES			Example		

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---  
 frwrd help retn quit bkwrd frwr

 **Notes:**

1. For information about the standard user exits, see *User Exits for the Generation Models, Natural Construct Generation*.
2. For information about the User Exit editor, see *User Exit Editor, Natural Construct Generation*.

The following user exits are specific to the Object-Maint-Subp model:

- [AFTER-STORE User Exit](#)
- [AFTER-UPDATE User Exit](#)
- [BEFORE-DELETE User Exit](#)



- **BEFORE-STORE User Exit**

### **AFTER-STORE User Exit**

The code in this exit is executed after the data is stored, but before the END TRANSACTION is issued. For example, it can be used in conjunction with Adabas TRS (Text Retrieval System). As TRS cannot invert a document index unless the document record exists, the code in this exit calls TRS to invert the document. In that way, the transaction can be backed out if there are any problems with TRS.

### **AFTER-UPDATE User Exit**

The code in this exit is executed after the data is updated, but before the END TRANSACTION is issued. For example, it can be used in conjunction with Adabas TRS (Text Retrieval System). As TRS does not have an update document index function, the code in this exit calls TRS to delete the document index and then calls TRS again to invert the document.

### **BEFORE-DELETE User Exit**

The code in this exit is executed before the data is deleted. For example, it can be used in conjunction with Adabas TRS (Text Retrieval System). TRS requires the document index to be deleted before the document record is deleted. The code in this exit can call TRS to delete the document index so the document record can be deleted.

### **BEFORE-STORE User Exit**

The code in this exit is executed before the STORE command is issued. Use this exit when you want to change the primary key for an object (for example, when you want to generate a unique primary key number).

## **Parameters for the Object-Maint-Enhanced-Subp Model**

The Object-Maint-Enhanced-Subp model is similar to the Object-Maint-Subp model, except it supports the generation of large fields into the object PDA as dynamic fields. As with the Object-Maint-Subp model, this model has two specification panels: Standard Parameters and Additional Parameters. The majority of the fields on these panels are the same. This section describes the fields that are specific to the Object-Maint-Enhanced Subp model. The following topics are covered:

- [Standard Parameters Panel](#)
- [Additional Parameters Panel](#)



#### **Notes:**

1. For information about the parameters for the Object-Maint-Subp model, see [Parameters for the Object-Maint-Subp Model](#).

2. For information about creating an object-maintenance process, see *Design Methodology, Natural Construct Generation*.
3. For information about the standard user exits, refer to *User Exits for the Generation Models, Natural Construct Generation*.
4. For information about the User Exit editor, refer to *User Exit Editor, Natural Construct Generation*.

**Standard Parameters Panel**

The following example shows the first specification panel, the Standard Parameters panel:

```

CUDYMA          OBJECT-MAINT-ENHANCED-SUBP Subprogram          CUDYMA0
Feb 05          Standard Parameters                          1 of 2

Module ..... SS_____
System ..... CNDPRO_____

Title ..... Object ...._____
Description ..... This subprogram is used to perform object maintenance__
                    for..._____
                    _____
                    _____

Message numbers .... _
Hash locking ..... _

Generate dynamic fields when length is greater than .... 25_
Return errors when data is truncated ..... X
Generate with large object (LO) fields ..... _

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

The fields in the upper portion of this panel are similar for all models. For a description of these fields, see *Common Fields on the Standard Parameters Panel*. For information on the hash-locking option, see [Hash-Locking Option](#).

The following fields are specific to the Object-Maint-Enhanced-Subp model:

Field	Description
Generate dynamic fields when length is greater than <i>nnn</i> (where <i>nnn</i> is a number less than 1000 and 0 indicates the data PDA contains the same lengths as the DDM)	Generates dynamic fields into the object PDA when the size of the source field is larger than the number of characters specified in this field. For example, if the specified cutoff length is 50 and a field is defined in the DDM as alphanumeric 100, an (A) DYNAMIC field will be generated into the object PDA instead of an (A100).  <b>Note:</b>

Field	Description
	<ol style="list-style-type: none"> <li>1. If the cutoff length is "0", the field sizes in the PDA will be the same as those in the DDM.</li> <li>2. If a field is affected by the cutoff length, it may not be part of a redefined field.</li> <li>3. If the cutoff length is a negative number, the length is converted to a positive value (for example, "-10" is converted to "10").</li> </ol>
Return errors when data is truncated	<p>Indicates whether the generated subprogram returns an error message when an alpha dynamic field is larger than its source database field and data in the field will be truncated.</p> <p>For example, you may have a text field with a variable length for descriptive information that you want to set to 1000 characters. Since an alpha dynamic field can handle more than 1000 characters, you must decide what happens if the user enters more. One option is to let the user enter whatever they want and the subprogram will truncate any data over the limit when it stores it in the database. Another option is to generate error messages when the user exceeds the limit and/or to stop the processing.</p> <p>When the subprogram is generated with the truncation option, Construct will provide error messages and a user exit to define how to handle the error. Within this user exit, Construct generates a list of all affected fields (i.e., are alpha dynamic fields in the PDA but not in the file view) and allows you to change the value for <code>##RETURN-CODE</code> or add an <code>ESCAPE ROUTINE</code> to continue with processing when an error occurs.</p> <p><b>Note:</b> Truncation errors and messages are processed in the <code>PROCESS-TRUNCATION-ROUTINE</code> user exit. For information, see <a href="#">PROCESS-TRUNCATION-ROUTINE User Exit</a>.</p>
Generate with large object (LO) fields	Indicates whether large object (LO) fields are maintained by the generated subprogram.

### Additional Parameters Panel

The following example shows the second specification panel, the Additional Parameters panel:

```

CUDYMB          OBJECT-MAINT-ENHANCED-SUBP Subprogram          CUDYMB0
Sep 07          Additional Parameters                          2 of 2

Predict view ..... _____ *
Primary key ..... _____ *
Hold field ..... _____ *

Object description ..... _____

Object PDA ..... NEWOMSA_ *      Generate   Source   Object
Restricted PDA ..... NEWOMSR_ *      X
Object name ..... _____

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

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

The fields on this panel are identical to the fields on the Additional Parameters panel for the Object-Maint-Subp model. The only difference is that the Generate field is always marked for the Object PDA and Restricted PDA fields. These parameter data areas will always be generated or regenerated with the Object-Maint-Enhanced-Subp model, since the field definitions may change when dynamic fields are processed.

After generating the subprogram, you can edit generated source code that is not within user exits. For information about the protected lines in the generated source code, see *Using the Source Editor* in *Using NaturalONE*.

### User Exits for the Object-Maint-Enhanced-Subp Model

The following example shows the User Exits panel for the Object-Maint-Enhanced-Subp model:

CSGSAMPL	OBJECT-MAINT-ENHANCED-SUBP	Subprogram	CSGSM0
Aug 27	User Exits		1 of 1
	User Exits	Exists	Sample
			Required Conditional
—	NAT-DOCS		X
—	CHANGE-HISTORY	Subprogram	
—	PARAMETER-DATA	Example	
—	EXTENDED-RI-VIEWS		
—	LOCAL-DATA	Example	
—	START-OF-PROGRAM	Example	
—	SELECT-STATEMENT	Subprogram	X
—	USER-DEFINED-FUNCTIONS	Example	
—	BEFORE-ET	Example	X
—	BEFORE-ET-PROCESSING	Example	
—	AFTER-ET-PROCESSING	Example	
—	PROCESS-ERROR-MESSAGE		
—	ERROR-MESSAGE-PDAS		
—	END-OF-PROGRAM	Example	
—	BEFORE-STORE	Example	
—	AFTER-STORE		
—	AFTER-GET	Example	
—	BEFORE-DELETE		X
—	AFTER-INIT	Example	
—	UPDATE-EDITS	Subprogram	
—	DELETE-EDITS	Subprogram	
—	AFTER-GET-EDITS	Subprogram	
—	EXTENDED-RI-CHECKS	Subprogram	
—	ADJUST-OBJECT-ID-IN-MSG	Example	
—	AFTER-UPDATE		
—	OVERRIDE-MINIMUM	Example	
—	OVERRIDE-MAXIMUM	Example	
—	PROCESS-TRUNCATION-ROUTINE	Example	X
—	MISCELLANEOUS-SUBROUTINES	Example	

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

Most of the user exits for this model are identical to those for the Object-Maint-Subp model.

 **Notes:**

1. For information about the standard user exits, see *User Exits for the Generation Models, Natural Construct Generation*.
2. For information about user exits that are specific to the Object-Maint-Subp model, see [User Exits for the Object-Maint-Subp Model](#).
3. For information about the User Exit editor, see *User Exit Editor, Natural Construct Generation*.

The following user exit is specific to the Object-Maint-Enhanced-Subp model:

- [PROCESS-TRUNCATION-ROUTINE User Exit](#)

### PROCESS-TRUNCATION-ROUTINE User Exit

This user exit can be used in the generated subprogram to define truncation routines and error messages for alpha dynamic fields. It is a Conditional exit and available when the PDA for the subprogram contains alpha dynamic fields in the object PDA that represent fixed-length fields in the database.

When you select the PROCESS-TRUNCATION-ROUTINE user exit, the following code is generated into the exit:

```
Module ..... ModName
Title ..... Object ....
>                                     > + ABS: X X-Y: _ S    5 L    1
A11  ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
0010 DEFINE EXIT PROCESS-TRUNCATION-ROUTINE
0020 /* Start of PROCESS-TRUNCATION-ROUTINE user exit
0030 /* note that the ##RETURN-CODE can be changed or
0040 /* ESCAPE ROUTINE can be added so that one doesn't stop the program.
0050 END-EXIT
```

To allow processing to continue when a truncation error occurs, you can change the value for ##RETURN-CODE or add an ESCAPE ROUTINE.

## Object-Maint-Dialog Model

---

The Object-Maint-Dialog model generates the dialog component (Natural program) of an object-maintenance process. The dialog component communicates with the user and invokes methods (data actions) implemented by the object-maintenance subprogram. To generate a complete maintenance process using Natural Construct’s object-oriented approach, the Object-Maint-Dialog model must be used in conjunction with the Object-Maint-Subp model (which also generates the object PDA and restricted PDA). The dialog program performs the following functions:

- Executes all INPUT/OUTPUT functions:
  - input object data and actions executed on the object
  - mark fields in error and display error messages
- Invokes the object-maintenance subprogram and passes it the object and action to be executed.
- Supports left/right scrolling for multiple panels.

- Controls up to four scroll regions on each panel. A region can also be scrolled simultaneously on two panels.
- Displays information for related entities outside the object (sub-entities).

The following example shows a generated object-maintenance dialog (only the first panel is displayed):

```

Add      Browse      Clear      Display      Modify      Next      Purge

NCOMENT                                ***** ORDER SUBSYSTEM *****                                NCOMEM11
Oct 28                                - MAINTAIN ORDER ENTRIES -                                1 more >
Action.....: ___
Order Number.....: 111111      Invoice Number.....: 111111
*Customer Number..: 11111      QUAKER OATS
*Warehouse ID.....: 113        SOUTHERN DISTRIBUTORS LIMITED
Order Date.....:                Order Amount: 1500.00
1_ ----- Product Information -----      1_ Distribution Information
1 *Product.....: 187361                /\                Account      Amount
  Quantity...: 10_____                1 _____                /\
  Cost/Unit...: 150.00                2 _____                _____
  Total.....: 1500.00                3 _____                _____
  Description: CAT NUGGETS                \/                4 _____                \/
1_ Delivery Instructions (Scroll right for full screen)
1                                           /\
2                                           \/
Direct Command: _____

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
confm help  retrn quit                flip pref  bkwrd frwr                left  right main
Related information displayed.

```



#### Notes:

1. PF5 (flip) and PF6 (pref) are available on the panel. For a description of these PF-keys, see *Defining PF-Keys for Generated Applications, Natural Construct Generation*.
2. By default, this program prompts users to select Enter to confirm a Purge action. If you choose a confirmation key other than Enter, users must confirm Add, Modify, and Purge actions. For a description of how to change the confirmation key, see *Confirmation Key Setup, Natural Construct Generation*.
3. To see the specifications for this example, refer to the NCOMENT program in the Natural Construct demo system.

This section covers the following topics:

- [Multiple Scroll Regions](#)
- [Parameters for the Object-Maint-Dialog Model](#)

- [User Exits for the Object-Maint-Dialog Model](#)

## Multiple Scroll Regions

In the Object-Maint-Dialog model example, there are three scroll regions:

- Product Information (order lines entity).
- Distribution Information (distribution entity).
- Delivery Instructions (array within the primary entity).

Depending on where the user places the cursor, pressing PF7 (bkwrđ) or PF8 (frwrđ) scrolls through the data in each of the regions.

## Parameters for the Object-Maint-Dialog Model

The Object-Maint-Dialog model has four specification panels: Standard Parameters, Additional Parameters, Scroll Region Parameters, and Related File Parameters. This section describes these panels. The following topics are covered:

- [Standard Parameters Panel](#)
- [Additional Parameters Panel](#)
- [Scroll Region Parameters Panel](#)
- [Related File Parameters Panel](#)
- [Variables You Can Use with Object-Maint-Dialog Model Maps](#)



**Note:** For information about creating an object-maintenance process, see *Design Methodology, Natural Construct Generation*.

## Standard Parameters Panel

The following example shows the first specification panel, the Standard Parameters panel:



```

CUOMMA          Object-Maint-Dialog Program          CU--MA0
Sep 16          Standard Parameters                  1 of 4

Module ..... _____
System ..... CST821S_____
Global data area ... CDGDA____ *
With block ..... _____

Title ..... Object Dialog..._____
Description ..... This program is used to maintain the..._____
_____
_____
_____

First header ..... _____
Second header ..... _____

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

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

```

The fields on this panel are similar for all models. For a description of these fields, see *Common Fields on the Standard Parameters Panel*.

### Additional Parameters Panel

The following example shows the second specification panel, the Additional Parameters panel:

```

CUOMMB                               Object-Maint-Dialog Program          CUOMMB0
Nov 19                               Additional Parameters                2 of 4

Object maint subprogram .. _____ *

#ACTION field length ..... 1  Add ..... X  Browse ... _____ *
                                Clear .... X  Display .. X
                                Modify ... X  Next ..... X
                                Purge .... X  Former ... _

Window support ..... _
Push-button support ..... _
Mark cursor field ..... _____

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

The fields on this panel are:

Field	Description
Object maint subprogram	Name of the subprogram invoked by the generated module. (Use the Object-Maint-Subp model to generate the subprogram.) The specified subprogram must exist in the current library.
#ACTION field length	<p>Length of the action field. By default, the length is "1" and all action fields except Former are marked. If you do not want the generated dialog to perform a particular action, deselect the corresponding action field. At least one action must be selected.</p> <p>The available actions are:</p> <ul style="list-style-type: none"> <li>■ Add (adds the specified object)</li> <li>■ Browse (name of the generated subprogram that supports the Browse action)</li> <li>■ Clear (clears the specified field values from the panel)</li> <li>■ Display (displays the specified object)</li> <li>■ Modify (modifies the specified object)</li> <li>■ Next (displays the contents of the record having the next higher primary key value from the current key value)</li> <li>■ Purge (removes the specified object)</li> <li>■ Former (displays the contents of the record having the next lower primary key value from the current key value)</li> </ul>

Field	Description
	<p><b>Note:</b></p> <ol style="list-style-type: none"> <li>1. To add user-defined actions, see <i>Add an Action, Natural Construct Generation</i>.</li> <li>2. When generating an object-maintenance dialog, this feature works together with two user exits. For information about these exits, see <i>SELECT-ADDITIONAL-ACTIONS</i> and <i>ADD-ACTION-PROCESSING, Natural Construct Generation</i>.</li> </ol>
Window support	Indicates whether the output from the generated object-maintenance dialog is displayed in a window or on a panel (the default).
Push button support	Indicates whether actions on the generated dialog can be selected by cursor.
Mark cursor field	Name of the field on the map where the cursor is automatically placed by the generated dialog program.

### Change the Default Window Settings

► **To change the default window settings for your object-maintenance dialog:**

- Press PF5 (windw) on the Additional Parameters panel.

The Window Parameters window is displayed. For a description of this window, see *Change the Default Window Settings*.

### Scroll Region Parameters Panel

The following example shows the third specification panel, the Scroll Region Parameters panel:

```

CUOMMC          Object-Maint-Dialog Program          CUOMMC0
Sep 16          Scroll Region Parameters              3 of 4

Horizontal panels ..... 1

>> 1 Input using map ..... CDLAY___ *

Scrollable Regions          1          2          3          4
Total occurrences .....    ___    ___    ___    ___
Screen occurrences .....    ___    ___    ___    ___
Starting from ..... #ARRAY1 #ARRAY2 #ARRAY3 #ARRAY4
Scroll with panel .....    -    -    -    -

Top left ..... Line .....    ___    ___    ___    ___
                Column .....    ___    ___    ___    ___
Bottom right .. Line .....    ___    ___    ___    ___
                Column .....    ___    ___    ___    ___

Depth occurrences .....    ___

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

The fields on this panel are:

Field	Description
Horizontal panels	Number of horizontal panels. If the generated program requires more than one input panel to accept all values that are being maintained, specify the total number of panels in this field. By default, "1" is displayed. If you specify more than one panel, Natural Construct activates the left and right PF-keys in the generated program to allow left and right scrolling between panels.
>> 1	If you specify more than one panel, you can display the map specification fields for another panel by entering that panel number in this field. By default, "1" is displayed. The number specified in this field cannot exceed the number specified in the Horizontal panels field.  <b>Note:</b> You can also scroll to another panel by pressing the frwrdr or bkwrdr PF-keys. The number of the current panel is automatically displayed in this field.
Input using map	Name of the map for the current panel. If you enter scroll region information, the specified map should contain array fields that match the specified values.  You can create the maps using the Map model or the Natural Map editor. If you require scrolling regions, you can use the CDLAYMP1 layout map with the Map model. The Map model generates all of the required indexes to control scrolling. If you create the map in the Map editor, use the CDLAYOM1 layout map.  <b>Note:</b> For a description of the variables you can use on maps, see <a href="#">Variables You Can Use with Object-Maint-Dialog Model Maps</a> .

Field	Description
Scrollable Regions	Number of the scroll region for the corresponding scroll specifications. You can define the specifications for up to four vertical scroll regions (consisting of vertical arrays) for each panel.
Total occurrences	Total number of scrollable lines required for the scroll region. The total occurrences value applies when the generated program includes a line scroll feature to scroll records in a secondary or tertiary file, or multiple-valued fields (MUs), or periodic groups (PEs). The program ensures that the values assigned to the array index values (#ARRAY1 through #ARRAY4) do not exceed the total occurrences value for each array.
Screen occurrences	If you specify a total occurrences value, specify the total number of lines displayed on the panel at one time.
Starting from	Starting index for each scroll region. Repeating fields in a scroll region must be indexed by #ARRAY $n$ for scroll region $n$ (where $n = 1, 2, 3,$ or $4$ ).
Scroll with panel	If you want to force a particular <i>starting from</i> value for a panel (so it has the same value as another panel), specify the panel number in this field. Each panel maintains its own current values for the <i>Starting from</i> field (#ARRAY $n$ where $n = 1, 2, 3,$ or $4$ ).
Scroll region location	<p>Location of the corresponding scroll region. A scroll region is always rectangular and is defined by specifying the panel coordinates of the top left and bottom right corners. In the generated dialog, pressing the bkwrld and frwrld PF-keys positions the scroll regions backward and forward.</p> <p>When you specify the location of each scroll region, you make the generated program sensitive to the position of the cursor in an active scroll region. If the cursor is inside a defined region, pressing these keys moves the cursor to the base of the active scroll region and only that region is scrolled. If the cursor is not inside a defined region, pressing these keys scrolls all regions.</p> <p><b>Note:</b> Press the deflt PF-key to compute these coordinates by examining the map's source.</p>
Top left Line	Starting line number (vertical axis) for the scroll region.
Top left Column	Starting column number (horizontal axis) for the scroll region.
Bottom right Line	Ending line number (vertical axis) for the scroll region.
Bottom right Column	Ending column number (horizontal axis) for the scroll region.
Depth occurrences	<p>To create scroll region with a third dimension, specify the maximum depth occurrences value. For a calendar with the months and days forming the first two dimensions (horizontal and vertical) and the year forming the third dimension (depth), for example, you can specify "3" to scroll up to three yearly tables of calendar months and days, and within each yearly table, scroll vertically through the days.</p> <p>To allow the value of the #DEPTH variable to be changed, you can either place the #NEXT-DEPTH (P3) variable on the specified map or use PF-keys that you process in the AFTER-INPUT user exit.</p>



**Tip:** You can think of a two-dimensional (2D) array as a collection of many one-dimensional (1D) arrays. And you can think of a fixed instance of a third dimension of a three-dimensional (3D) array as a 2D array. Therefore, a vertical scroll region on the generated dialog can consist of 1D, 2D, or 3D arrays.

This section covers the following topics:

- [Retrieve Default Values for Scroll Region Parameters](#)
- [Display Specifications for Previous Panel](#)
- [Display Specifications for Next Panel](#)

### **Retrieve Default Values for Scroll Region Parameters**

If the object-maintenance map contains scrolling regions with one-dimensional arrays, you can retrieve the default values for the scroll region parameters by pressing PF5 (deflt). The values for the scroll region parameters are read from the specified map. The scroll regions must be indexed by #ARRAY1 through #ARRAY4.

### **Display Specifications for Previous Panel**

Press PF7 (bkwrđ) to display the scroll region specifications (Map name, Scroll region, etc.) for the previously-defined panel.

### **Display Specifications for Next Panel**

Press PF8 (frwrđ) to display the scroll region specifications (Map name, Scroll region, etc.) for the next panel.

### **Related File Parameters Panel**

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

```

CUOMMD                      Object-Maint-Dialog Program                      CUOMMD0
Jun 21                      Related File Parameters                          4 of 4

>> _1 Predict Relationships ..... _____ *

View Generation Options
User generated ..... _   Use relationship name .... _
Predict generated ..... _
Generate from map on panel _

Relationship Processing
New object displayed ..... _
Control variable modified _____

Related File Processing
MOVE BY NAME to ..... _____
PERFORM subroutine ..... _____
    IF found ... _
    IF not found _
    IF null .... _

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

```

Use this panel to retrieve additional panel information. Specify the Predict relationships that relate foreign keys within the object to other tables and then define how you want Natural Construct to process the specified relationship.

The fields on this panel are:

Field	Description
Predict Relationships	<p>Name of the first relationship. You can define up to 10 relationships. After defining the first, press PF8 (frwrdr) to display the specification fields for the next relationship; press PF7 (bkwrdr) to return to the previous relationship. If you specify a relationship, the program performs file lookups (joins) on the file related to the object file in Predict.</p> <p>In each relationship, the cardinality of the object file must be N or CN, while the cardinality of the related file must be 1 or C. The update constraint type must be R (restricted update) and the delete constraint type can be blank or R. Only type N (Natural Construct) relationships are processed.</p> <p><b>Note:</b> For relational databases, type R (referential constraint) relationships are also processed.</p>
View Generation Options	<p>If you specify a relationship in the Predict relationships field, indicate which fields in the related file are placed in the local view used to retrieve foreign file information. Indicate one of the following view creation options:</p> <ul style="list-style-type: none"> <li>■ User generated</li> </ul>

Field	Description
	<p>To define your own view of the related file in the LOCAL-DATA user exit, mark this field.</p> <ul style="list-style-type: none"> <li>■ Use relationship name</li> </ul> <p>To use the relationship name as the user view name for the related file, mark this field. To avoid generating multiple views with the same name, you should specify this option when the related file can be involved in multiple lookup relationships. If you are using a map and/or defining the related file view in the LOCAL-DATA user exit, you must also use the relationship name as the related file view.</p> <ul style="list-style-type: none"> <li>■ Predict generated</li> </ul> <p>To have Natural Construct generate a view for you, based on the specified relationship, mark this field. (All fields in the related file are generated into the view.)</p> <ul style="list-style-type: none"> <li>■ Generate from map on panel</li> </ul> <p>To use the fields in a view used on another panel, specify the panel number. Natural Construct generates a view with those fields prefixed by the name of the related file on the map used for the specified panel. (The file is determined through the specified relationship name.)</p>
Relationship Processing	<p>To specify when relationship processing is performed by the generated program, indicate one or both of the following processing options:</p> <ul style="list-style-type: none"> <li>■ New object displayed</li> </ul> <p>To perform a file lookup on the related file each time a new object is displayed, mark this field.</p> <ul style="list-style-type: none"> <li>■ Control variable modified</li> </ul> <p>To perform a file lookup whenever a field associated with a control variable is modified, specify the name of the control variable. If the field associated with the control variable is an array, the control variable must be defined with an asterisk (*) on the map.</p>
MOVE BY NAME to	<p>To copy the lookup data to another structure, specify the name of the structure.</p>
PERFORM subroutine	<p>To perform other processing for each file lookup, specify the name of the subroutine in this field (the subroutine is defined in the AFTER-LOOKUP-SUBROUTINE user exit) and mark one of the following options:</p> <ul style="list-style-type: none"> <li>■ IF found</li> </ul> <p>If you want the subroutine performed whenever the object's foreign key is updated with a value that exists in the related foreign table, mark this field. Before the subroutine is performed, the #LOOKUP-STATUS variable is assigned the value "FOUND".</p> <ul style="list-style-type: none"> <li>■ IF not found</li> </ul>



Field	Description
	<p>If you want the subroutine performed whenever the object's foreign key is updated with a value that does not exist in the related foreign table, mark this field. Before the subroutine is performed, the #LOOKUP-STATUS variable is assigned the value "NOT FOUND".</p> <ul style="list-style-type: none"> <li>■ IF null</li> </ul> <p>If you want the subroutine performed whenever the object's foreign key is updated to a null value (blank for alphanumeric and 0 for numeric), mark this field. Before the subroutine is performed, the #LOOKUP-STATUS variable is assigned the value "NULL".</p>



**Note:** If the object field defined by the relationship is within an array, the control variable must be defined with an asterisk (\*) notation on the map; the occurrence that triggered the subroutine is given in the #I1 variable.

### Variables You Can Use with Object-Maint-Dialog Model Maps

You can use the following variables with maps for object-maintenance programs or subprograms:

Variable	Format	Definition	Description
#PROGRAM	A8	Output	Name of the program that invoked the map.
#HEADER1	A60	Output	First heading for the program.
#HEADER2	A58	Output	Second heading for the program.
#LEFT- PROMPT	A9	Output	For programs with more than one panel, this variable indicates the number of panels to the left of the current panel. If the current panel is the leftmost panel, this variable contains the current date.
#RIGHT- PROMPT	A9	Output	For programs with more than one panel, this variable indicates the number of panels to the right of the current panel. If the current panel is the rightmost panel, this variable contains the current time.
#ACTION	A1	Modifiable	Action applied to the current object occurrence.
#VAL-ACT	A18	Output	List of available actions.
#DIRECT-COMMAND	A60	Modifiable	Indicates support for direct command processing.
#HPARM	A65	Output/Nondisplay	Key to Natural Construct's passive help file for the current program (system name concatenated with program name). Place this variable on the map and pass it to the CD-HELPR help routine.
#NEXT-ARRAY	P5	Modifiable	Current panel number. Users can change the value in this field to reposition to the specified panel. This field is used for programs with more than one panel.

Variable	Format	Definition	Description
#ARRAY1	N7	Array Index	Index for fields in scroll region 1 that are scrolled by pressing PF7 (subtract lines-per-panel from #ARRAY1) or PF8 (add lines-per-panel to #ARRAY1). Also see #NEXT-ARRAY1.
#NEXT- ARRAY1	P5	Modifiable	Index of the first displayed occurrence of the fields in scroll region 1. Users can change the value in this field to reposition scroll region 1 to the specified panel number. This field is used for panels that support scroll region 1.
#ARRAY2	N7	Array Index	Similar to #ARRAY1, except it indexes the fields in scroll region 2.
#NEXT- ARRAY2	P5	Modifiable	Similar to #NEXT-ARRAY1, except it indexes scroll region 2.
#ARRAY3	N7	Array Index	Similar to #ARRAY1, except it indexes the fields in scroll region 3.
#NEXT- ARRAY3	P5	Modifiable	Similar to #NEXT-ARRAY1, except it indexes scroll region 3.
#ARRAY4	N7	Array Index	Similar to #ARRAY1, except it indexes the fields in scroll region 4.
#NEXT- ARRAY4	P5	Modifiable	Similar to #NEXT-ARRAY1, except it indexes scroll region 4.
#DEPTH	N7	Array Index	Index for fields that are scrolled whenever the #NEXT-DEPTH value changes. By default, no PF-key is assigned to alter the value of #DEPTH. However, this can be achieved through user exit processing.
#NEXT- DEPTH	P3	Modifiable	Indicates support for third-dimension scrolling. By default, this field indicates the current depth level. Users can change this value to reposition to a different depth level.
#LIN	P3	Output	Single-dimension array containing sequential numbers (starting from 1). The occurrences of this array match the value of the highest upper bounds specified for any scroll region. This array can be placed on the panel whenever you want to show the current scroll index value beside a scroll region.
#KD-LINE1/ #KD-LINE2/ #KD-LINES(*)	A79	Output	Names of the available actions or alternate PF-key display formats (supplied in CDDIALDA). Place them either at the top of the map or at the bottom immediately above the standard PF-key lines.  To use the Push Button feature, include the #KD-LINES-CV control variable and the '00V(NP'02 dynamic attribute. To display the push buttons in red, use '00VRE(NP'02.

Variable	Format	Definition	Description
			The CDLAYOM2 layout map provides push button support.
#BKWRD-LAB1	A2	Output	Enable backward/forward scrolling push buttons (supplied in CDKEYLDA). Place them on the map(s) next to the fields where you want to enable backward/forward scrolling.
#BKWRD-LAB2			
#BKWRD-LAB3			
#BKWRD-LAB4			Include reverse video display among the push button attributes.
#FRWRD-LAB1			
#FRWRD-LAB2			
#FRWRD-LAB3			
#FRWRD-LAB4			

### User Exits for the Object-Maint-Dialog Model

The following examples show the User Exits panel for the Object-Maint-Dialog model:

CSGSAMPL	Natural Construct			CSGSM0
Dec 19	User Exits			1 of 1
User Exit	Exists	Sample	Required	Conditional
— NAT-DOCS				X
— CHANGE-HISTORY		Subprogram		
— PARAMETER-DATA		Example		X
— LOCAL-DATA				
— START-OF-PROGRAM				
— BEFORE-INPUT				
— BEFORE-STANDARD-KEY-CHECK		Example		
— AFTER-INPUT				
— AFTER-OBJECT-CALL		Example		
— AFTER-GET		Example		
— AFTER-SCREEN-CLEAR		Example		
— END-OF-PROGRAM		Example		
— SELECT-ADDITIONAL-ACTIONS		Example		
— SET-PF-KEYS		Example		
— ADD-ACTION-PROCESSING				X
— BROWSE-ACTION-PROCESSING				X
— BEFORE-BROWSE-CALLNAT				X
— AFTER-BROWSE-CALLNAT				X
— CLEAR-ACTION-PROCESSING				X
— DISPLAY-ACTION-PROCESSING				X
— MODIFY-ACTION-PROCESSING				X
— NEXT-ACTION-PROCESSING				X
— FORMER-ACTION-PROCESSING				X
— PURGE-ACTION-PROCESSING				X
— COPY-ACTION-PROCESSING				X
— ADDITIONAL-ACTIONS-PROCESSING				
— BEFORE-ET-PROCESSING		Example		
— AFTER-ET-PROCESSING		Example		
— REINPUT-SCREEN				
— AFTER-LOOKUP-SUBROUTINES		Subprogram		
— MISCELLANEOUS-SUBROUTINES		Example		

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



**Notes:**

1. For information about these user exits, see *User Exits for the Generation Models, Natural Construct Generation*.
2. For information about the User Exit editor, see *User Exit Editor, Natural Construct Generation*.

## Object-Maint-Dialog-Subp Model

The Object-Maint-Dialog-Subp model generates a dialog component (Natural subprogram) of a maintenance process, similar to the object-maintenance dialog program described in the preceding section. The only difference between the two is that the action in the object-maintenance dialog subprogram is controlled by the calling program.

A browse-select program, for example, can call an object-maintenance dialog subprogram and pass it the #ACTION parameter (specifying the action to be performed). When this happens, certain attributes on the map used by the object-maintenance dialog subprogram are modified by two control variables the model generates. These variables define the display attributes of the map, according to the #ACTION parameter:

#ACTION Parameter	Control Variable	Attribute
Display, Purge	#KEY-CV	Locked
	#SCR-CV	Locked
Modify	#KEY-CV	Locked
	#SCR-CV	Open
Add, Copy	#KEY-CV	Open
	#SCR-CV	Open

The control variables generated by the Object-Maint-Dialog-Subp model are:

Variable	Associated With
#KEY-CV	Key fields for the object data used on maps.
#SCR-CV	All other fields for the object data used on maps.
#PROTECT-CV	Action field. Since programs generated with the Object-Maint-Dialog and Object-Maint-Dialog-Subp models generally use the same map, this control variable protects the Action field for the subprogram.

The following example shows a generated object-maintenance dialog subprogram:

```

NCOSELN          ***** ORDER SUBSYSTEM *****          NCOSEM11
May 30           - MAINTAIN ORDER ENTRIES -                1 more >

*Action (A,D,M,P,C): D Order Number: 90008_
*Customer Number....: 22222 KENT VETERINARY CLINIC
*Warehouse ID.....: 638 WATERLOO WAREHOUSING LTD.
Invoice Number.....: 333331
Order Date.....: 13/10/21 Order Amount: 229898.50

1_ ----- Product Information ----- 1_ -- Distribution Information --
1 *Product....: 333333                      Account          Amount
  Quantity...: 500_____                    1  676767676    3233.00_____
  Cost/Unit...: 50.00                          2  676767678    90.00_____
  Total.....: 25000.00                         3  989898989    80.00_____
  Description: OATS AND BARLEY CE              4  789078900    89.00_____
1_ Delivery Instructions (Scroll right for full screen)
  1 TO BE DELIVERED TO SHIPPING/RECEIVING IF BEFORE 5:00 PM,
  2 ELSE TO NIGHT DROP-OFF.SSS
Direct Command: _____
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF
      help retrn quit          flip          bkwrdr frwrdr          left right ma
Order 90008 displayed successfully.

```

This section covers the following topics:

- [Parameters for the Object-Maint-Dialog-Subp Model](#)
- [User Exits for the Object-Maint-Dialog-Subp Model](#)



**Note:** To see the specifications for this example, refer to the NCOSELN subprogram in the Natural Construct demo system.

### Parameters for the Object-Maint-Dialog-Subp Model

The specification panels for the Object-Maint-Dialog-Subp model are similar to the panels for the Object-Maint-Dialog model, with one exception. The Additional Parameters panel for the Object-Maint-Dialog-Subp model contains the Multiple action support field. If this field is specified, the generated subprogram allows users to perform multiple actions in succession.



**Note:** For information about the parameters on these panels, see [Parameters for the Object-Maint-Dialog Model](#).

### **User Exits for the Object-Maint-Dialog-Subp Model**

The User Exits panels for the Object-Maint-Dialog-Subp model are identical to the User Exits panels for the Object-Maint-Dialog model. For information about these panels, see [\*User Exits for the Object-Maint-Dialog Model\*](#).

---