Using the Object-Maint Models

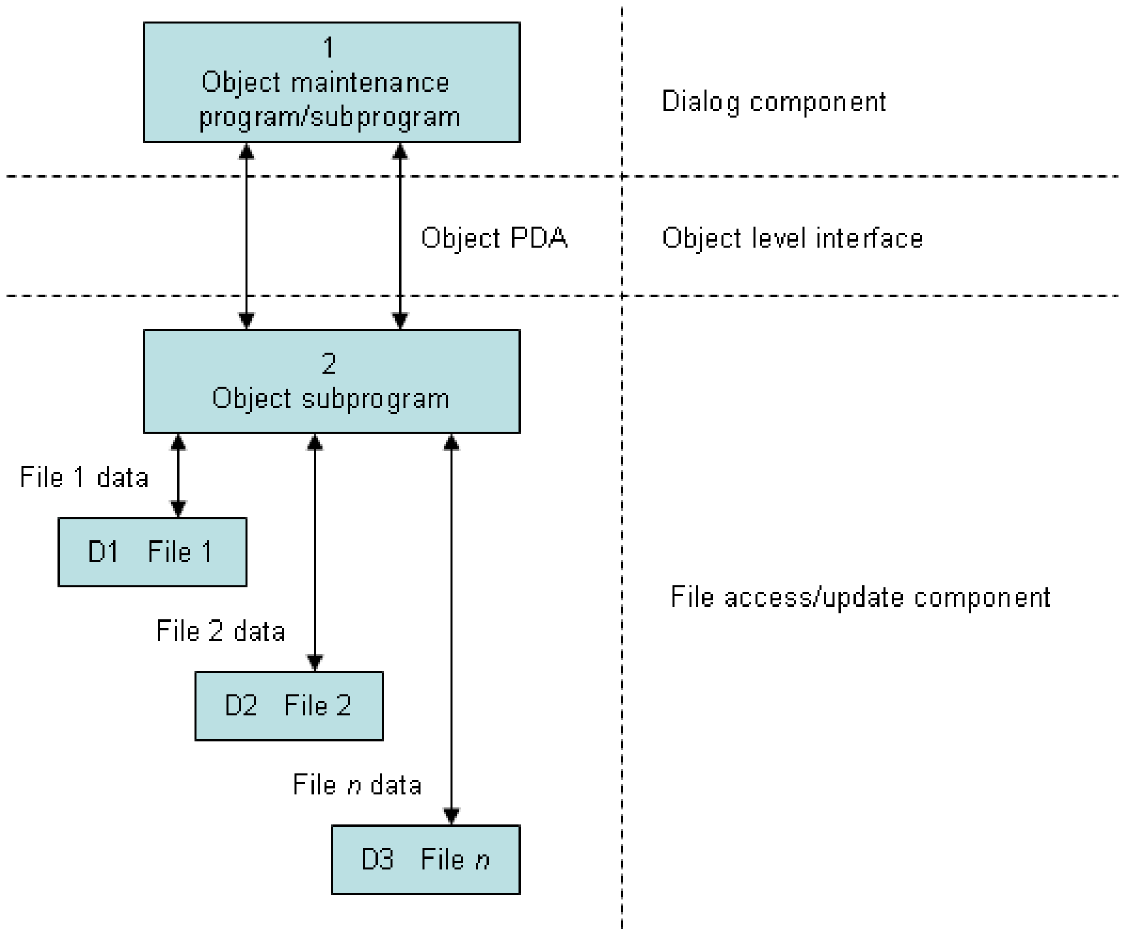
This section describes how to use the Object-Maint series of models to generate the modules required for an object-maintenance process. The following topics are covered:

- Introduction
- Object-Maint-Subp Model
- Object-Maint-Dialog Model
- Object-Maint-Dialog-Subp Model

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 model
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 Model

The Object-Maint-Subp model generates a subprogram that maintains complex data objects. The subprogram updates all entities within an object and contains 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).

This section covers the following topics:

- CDAOBJ2 Data Area
- Object PDA
- Restricted PDA
- Editing and Processing of Entities
- Additional Checks within User Exits
- Processing Order in Adabas Files
- Processing Order in Non-Adabas Files
- Pre-editing Checks
- Post-editing Checks
- Object Instance Hierarchy Tree
- Data Access Subroutines
- Store a Before Image of Data
- Parameters for the Object-Maint-Subp Model
- User Exits for the Object-Maint-Subp Model

CDAOBJ2 Data Area

The Object-Maint-Subp model uses the CDAOBJ2 data area. This data area 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.
```

The following sections describe some of the fields in this data area.

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.

To determine whether the GET-BY-ISN condition code is set, see Determine Which Condition Codes are Set.

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.

Object PDA

The Object-Maint-Subp model generates the 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  *****                   01-05-16
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.

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.

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

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-*entity-name*	Executed during the pre-editing phase, before the Predict automatic rules are checked and the children of the current entity are processed.
V1-*entity-name*	Executed during the pre-editing phase, after the Predict automatic rules are checked and before the children of the current entity are processed.
V2-*entity-name*	Executed during the post-editing phase, after the Predict automatic rules are checked and all children of the current entity are processed.

The EXTENDED-RI-CHECKS user exit contains the following validation routine:

Subroutine	**Description**
V-*relationship-name*	Executed during the pre-editing phase, after the Predict automatic rules are checked and after the V1-*entity-name* 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*.

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.

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:

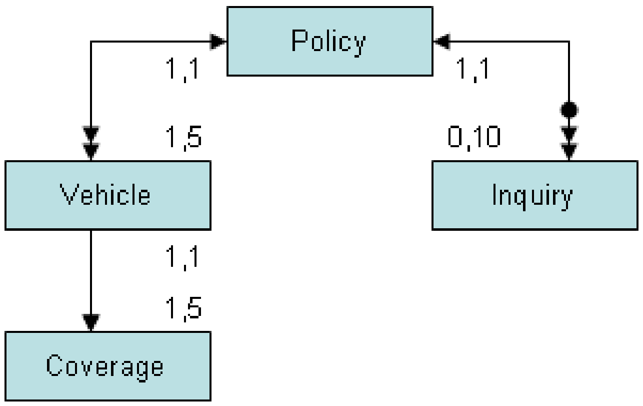
- Each non-null occurrence of an entity must correspond to a non-null occurrence of its parent entity; if an entity occurrence is set to null, so are all the occurrences of its child entities. This property of existence can be seen as a downward propagation from parent to child. If a record is set to 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.)

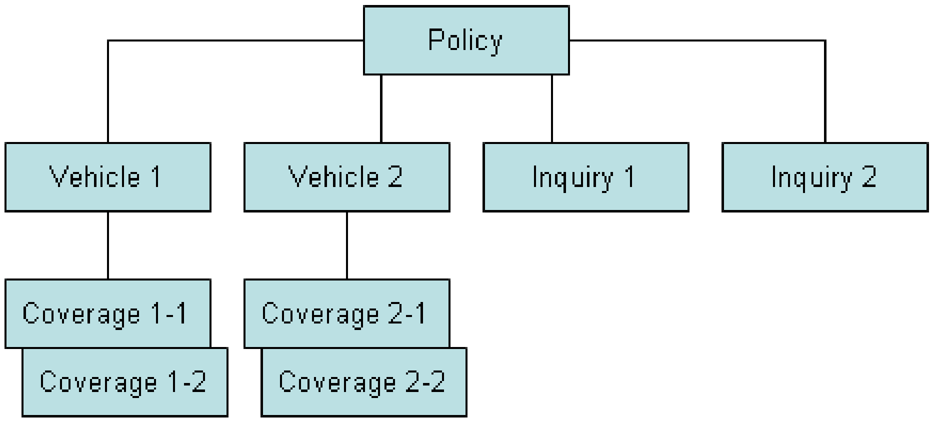
- The attributes of an entity occurrence can propagate upward to those of its parent entity. This type of propagation can be seen while traversing a post-order tree. When a node is encountered for the last time, the entity occurrence can contribute to the attributes of its parent since all of its attributes (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.

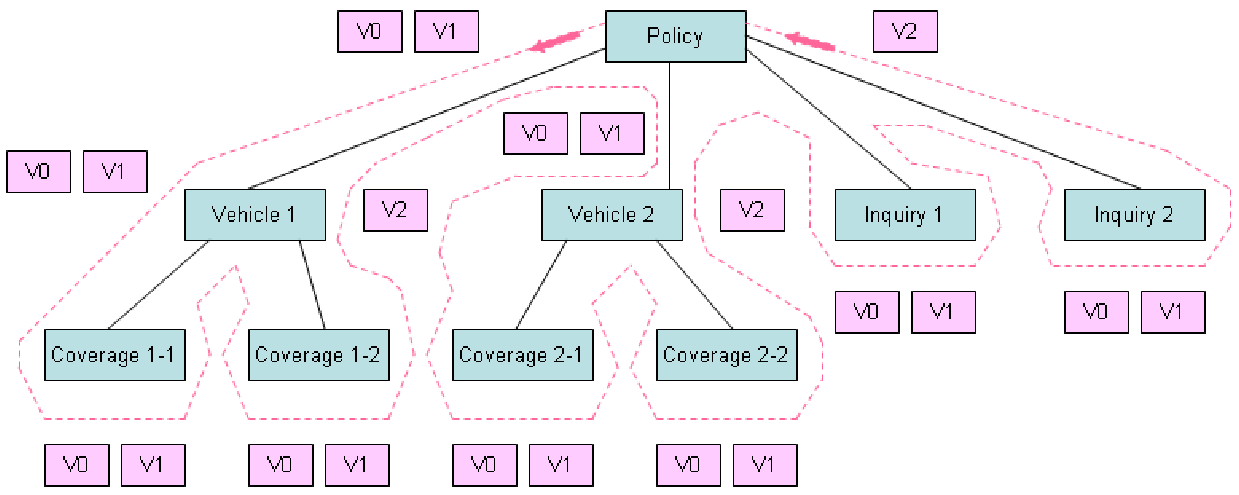
Consider an insurance policy object defined with the following entity relationships:



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
```

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 a Before Image of Data

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.

Parameters for the Object-Maint-Subp Model

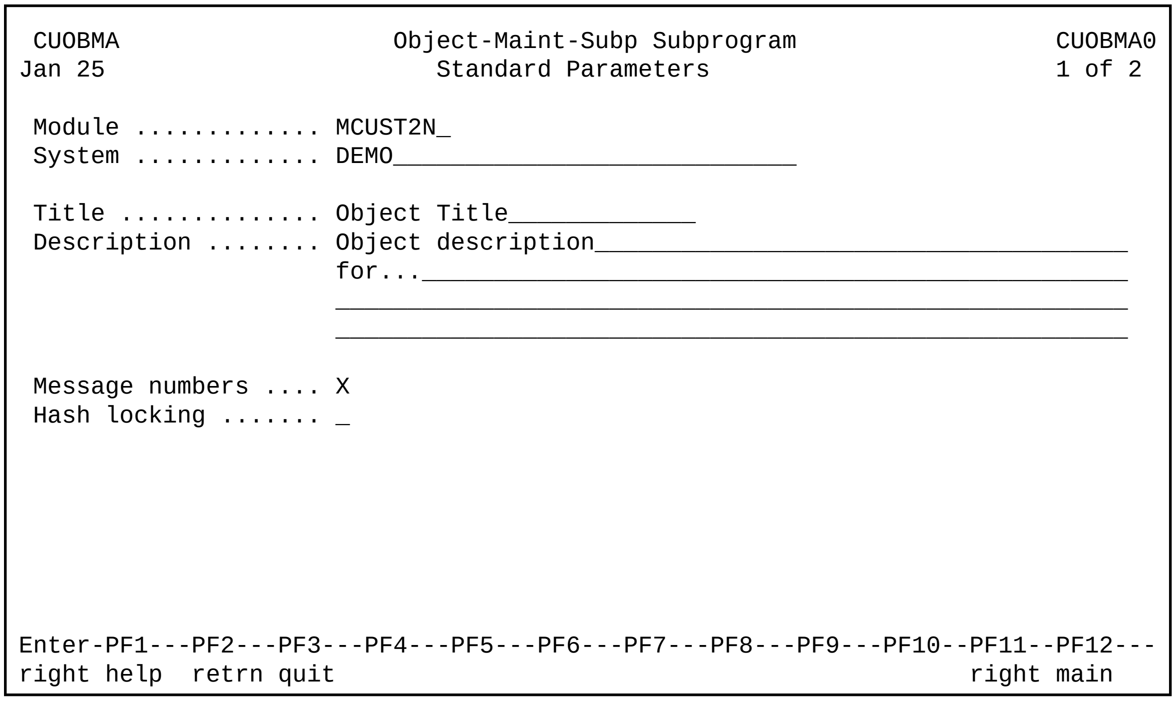
The Object-Maint-Subp model has two specification panels: Standard Parameters and Additional Parameters. This section describes these panels. The following topics are covered:

- Standard Parameters Panel
- Additional Parameters Panel

Note:
For more 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:

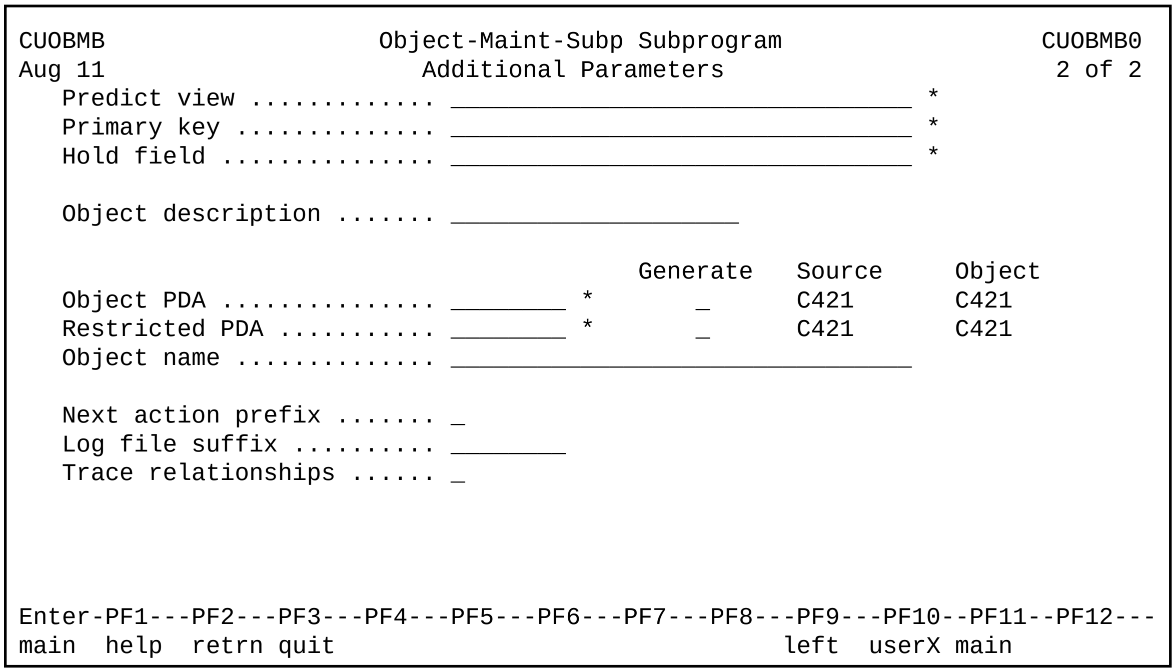


```
 CUOBMA                   Object-Maint-Subp Subprogram                  CUOBMA0
Jan 25                       Standard Parameters                        1 of 2

 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 ....... ____________________

                                            Generate   Source     Object
   Object PDA ............... ________ *        _      C421       C421
   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	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 “hold” field must exist in the primary file for the object. Valid data types are: ● T *TIMX ● A10 *TIME ● B8 *TIMESTMP ● N7 *TIMN ● A26 *TIMX (DB2 time stamp format) ● If the format is none of the above, it must be numeric. **Note:** If the hash-locking method is used, this field is not displayed. For more information, see Hash-Locking Option.
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 Object PDA.
Restricted PDA	Name of the restricted PDA used in conjunction with the object-maintenance subprogram. For more information, see Restricted PDA.
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	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. If the source code resides in the current library, regenerating it will execute a STOW command and overwrite the previous version.
Object	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. If the object code resides in the current library, regenerating it will execute a STOW command and overwrite the previous version. **Note:** 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.

Field	**Description**
Object name	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. **Note:** The object name cannot match the name of a file included in the object, nor any field in the object.
Next action prefix	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. For example, if the primary key is made up of Company + Account + Division and you do not want the Next action to span division values, specify “2”. Specify “1” if the Next action is to be limited to the current Company value.

Field	Description
Log file suffix	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. The following fields are required in the log file that corresponds to the header entity in the object: ● LOG-TIME Assigned with *TIMX for T format or *TIMN for N7 format. ● LOG-DATE 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.) ● LOG-TID Assigned with *INIT-ID. ● LOG-USER Assigned with *INIT-USER. ● LOG-ACTION 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. In the log files corresponding to the sub-entities in the object, only the LOG-ACTION field is required. **Note:** 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).
Trace relationships	If this field is marked, 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.

Hash-Locking Option

The Natural Construct administrator can change optimistic record locking from the default timestamp method to the hash-locking method. If 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
   Object PDA ............... ORDERPDA *       X      SHDEMO     SHDEMO
   Restricted PDA ........... ORDERPDR *       X      SHDEMO     SHDEMO
   Object name .............. ORDER___________________________
   ** Object LDA is generated when Object PDA is generated
   Object LDA ............... ORDERNH_ *              SHDEMO     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

Tip:
You can use this option to store a before image of data. For information, see Store a Before Image of Data.

User Exits for the Object-Maint-Subp Model

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

```
CSGSAMPL                 OBJECT-MAINT-SUBP Subprogram                   CSGSM0
Aug 27                            User Exits                            1 of 1

               User Exits             Exists    Sample   Required Conditional
    -------------------------------- -------- ---------- -------- ------------
 _  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  retrn quit                    bkwrd frwrd
```

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

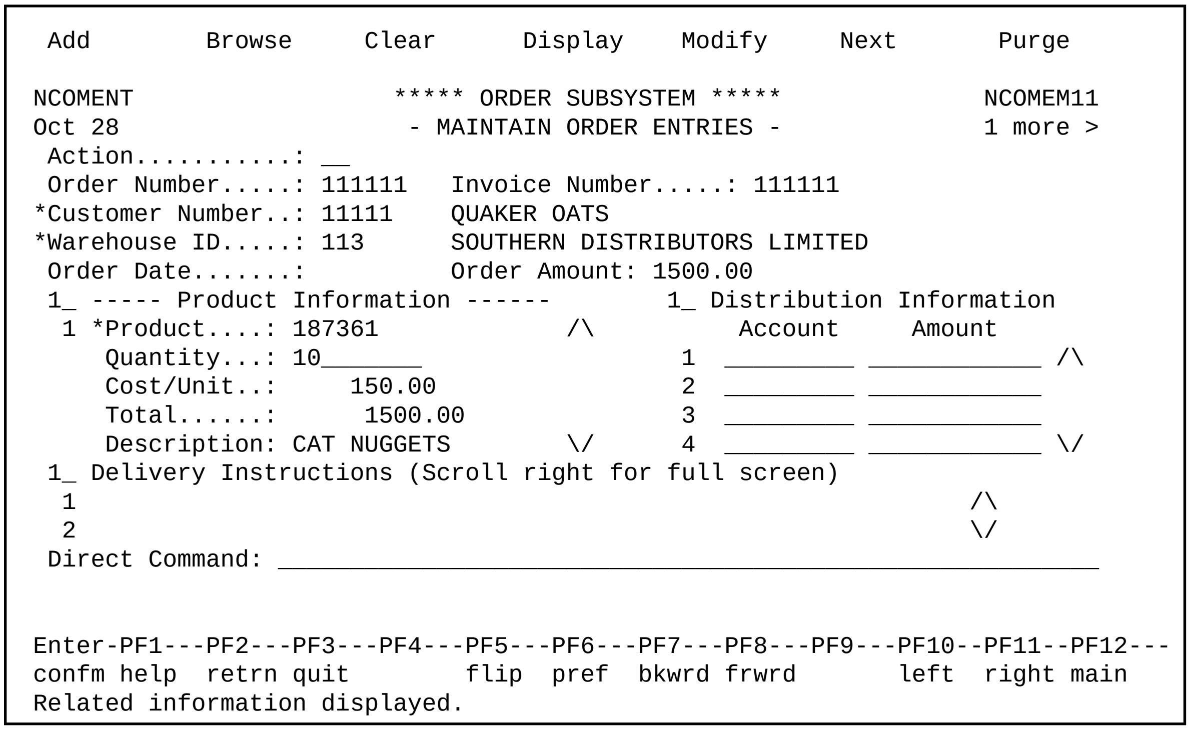
The code in this exit is executed before the STORE command is issued. Use this exit when you want to change the primary key for an object (for example, when you want to generate a unique primary key number).

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 frwrd       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 press 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 (bkwrd) or PF8 (frwrd) 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 ............. CST341S_________________________
 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.

Field	Description
#ACTION field length	Length of the action field (1, by default). By default, all action fields except Former are marked. If you do not want the program to perform a particular action, enter a blank in that action field. At least one action must be marked. The available actions are: ● Add Adds the specified object. ● Browse Name of the browse subprogram that supports the Browse action. (Use the Browse-Subp or Browse-Select-Subp model to generate browse subprograms.) ● Clear Clears the specified field values from the panel. ● Display Displays the specified object. ● Modify Modifies the specified object. ● Next Displays the contents of the record having the next higher primary key value from the current key value. If no higher value exists, the End of Data reached message is displayed. ● Purge Purges the specified object. ● Former Displays the contents of the record having the next lower primary key value from the current key value. If no lower value exists, the Start of Data reached message is displayed. **Note:** To add user-defined actions, see *Add an Action*, *Natural Construct Generation*. When using the Object-Maint-Dialog model, this feature works together with two user exits. For information about these exits, see *SELECT-ADDITIONAL-ACTIONS* and *ADD-ACTION-PROCESSING, Natural Construct Generation*.
Window support	If this field is marked, the output from the generated object-maintenance dialog is displayed in a window instead of on a panel.

Field	Description
Push button support	If this field is marked, actions can be selected by cursor or mouse.
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       bkwrd frwrd       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.

Field	Description
>> 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. **Note:** You can also scroll to another panel by pressing the frwrd or bkwrd 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. **Note:** For a description of the variables you can use on maps, see Variables You Can Use with Object-Maint-Dialog Model Maps.
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 if 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 “Starting from” 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 “Starting from” field (#ARRAY*n* where *n* = 1, 2, 3, or 4).

Field	Description
Scroll region location	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 bkwrd and frwrd PF-keys positions the scroll regions backward and forward. 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. **Note:** Press the deflt PF-key to compute these coordinates by examining the map's source.
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	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. 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.

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 used in this model 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 (bkwrd) to display the scroll region specifications (Map name, Scroll region, etc.) for the previously-defined panel.

Display Specifications for Next Panel

Press PF8 (frwrd) 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                    bkwrd frwrd       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	Name of the first relationship. You can define up to 10 relationships. After defining the first, press PF8 (frwrd) to display the specification fields for the next relationship; press PF7 (bkwrd) 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. 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. **Note:** For relational databases, type R (referential constraint) relationships are also processed.
View Generation Options	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: ● User generated To define your own view of the related file in the LOCAL-DATA user exit, mark this field. ● Use relationship name 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. ● Predict generated 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.) ● Generate from map on panel 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.)

Field	Description
Relationship Processing	To specify when relationship processing is performed by the generated program, indicate one or both of the following processing options: ● New object displayed To perform a file lookup on the related file each time a new object is displayed, mark this field. ● Control variable modified 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.
MOVE BY NAME to	To copy the lookup data to another structure, specify the name of the structure.
PERFORM subroutine	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: ● IF found 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". ● IF not found 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". ● IF null 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".

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 helproutine.
#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.
#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.

Variable	Format	Definition	Description
#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. The CDLAYOM2 layout map provides push button support.

Variable	Format	Definition	Description
#BKWRD-LAB1 #BKWRD-LAB2 #BKWRD-LAB3 #BKWRD-LAB4 #FRWRD-LAB1 #FRWRD-LAB2 #FRWRD-LAB3 #FRWRD-LAB4	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. Include reverse video display among the push button attributes.

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
    -------------------------------- -------- ---------- -------- ------------
 _  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---
frwrd help  retrn quit                    bkwrd frwrd
```

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 #SCR-CV	Locked Locked
Modify	#KEY-CV #SCR-CV	Locked Open
Add, Copy	#KEY-CV #SCR-CV	Open 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.........: 93/04/26 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        bkwrd frwrd       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.