Using the Object-Browse Models

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, business service. The modules are also ideal if access to the data is required for validation purposes.

This section covers the following topics:

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.

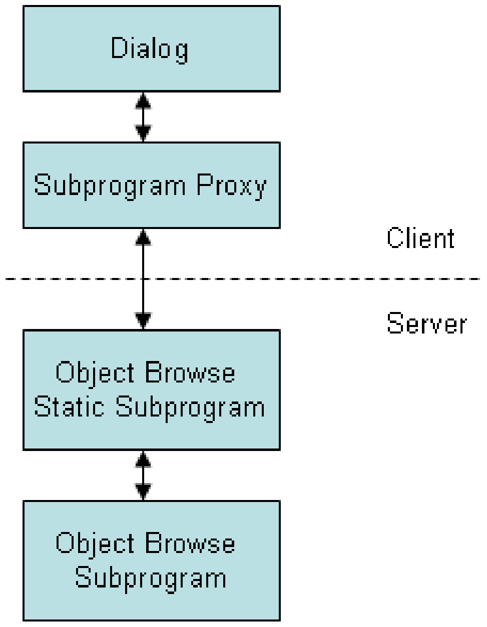
- Ability to toggle between histogram mode and normal mode, depending on whether or not the caller wants distinct key values to be returned.
- Ability to confine a search by locking portions of the key.
- Ability to determine the most efficient record selection logic based on the specified search criteria.

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

Model	Generated Modules
Object-Browse-Subp	● Object-browse subprogram ● Object parameter data area (defines returned row data) ● Key parameter data area (defines search key values) ● Restricted parameter data area (private data used internally by the browse object to maintain context)
Object-Browse-Static	● Object-browse static subprogram ● Static object parameter data area

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 Natural 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

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 CUSTOMER-WAREHOUSE-ID	Ascending 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 company on file and continues to the last company.

You can also specify two prefixes for the sort key. In this case, users must specify exact values for the COMPANY and DIVISION 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).

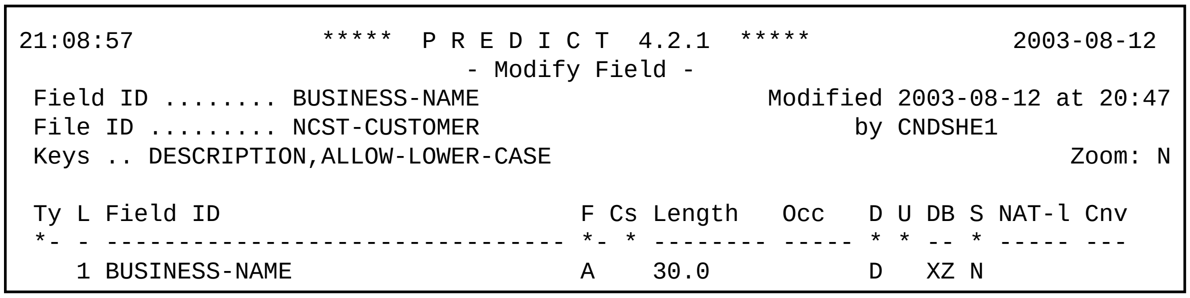
Additional minimum/maximum functionality can be coded within user exits (for example, READ-INPUT-CRITERIA).

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  4.2.1  *****              2003-08-12
                               - Modify Field -
 Field ID ........ BUSINESS-NAME                      Modified 2003-08-12 at 20:47
 File ID ......... NCST-CUSTOMER                            by CNDSHE1
 Keys .. DESCRIPTION,ALLOW-LOWER-CASE                                     Zoom: N

 Ty L Field ID                         F Cs Length   Occ   D U DB S NAT-l 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.

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 components, 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 ............. CST421S_________________________

 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 ____________________________ ________________________________ *     _

                                          Generate   Source     Object
 Object PDA ............... OBJBRROW *        X
 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	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. After selecting the view, the first six key components in the view are displayed. **Tip:** 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.

Natural (DDM)	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.
	
	The Predict definition of the primary file determines which fields are included in the DEFINE DATA section of the generated code. The format of the generated code in the DEFINE DATA section has the following structure:
	
	`1 Primary-file-name VIEW OF Data-definition-module`
	`   2 fields pulled from Predict of Primary-file-name`

Program view	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. 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.
Logical keys	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. 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. 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.

Field	Description
Key components	First key component for a logical key that maps to more than one component. You can add more key components and set options in the Optional Parameters window. To display this window, press PF5 (optns) or mark the Option field and press Enter. Using this window, you can add as many as four more components and specify options for them. For information, see Define Optional Parameters. To define key components, specify either one superdescriptor or multiple individual descriptors. **Note:** 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.
Option	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 press Enter. For more information, see Define Optional Parameters.
Object PDA	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. 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”. For more information, see Object PDA.
Key PDA	Key PDA that contains all of the components contained in the logical keys, as well as a unique ID field. 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”. For more information, see Key PDA (Search Key).
Restricted PDA	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. 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”. For more information, see Restricted PDA.

Field	Description
Generate	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).
Source	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. 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 in this window are generated and stowed when the object-browse subprogram is generated — regardless of whether the subprogram is stowed.

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 press 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                    bkwrd frwrd                   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 (frwrd) and PF7 (bkwrd).

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.
Descending	To have the key component values listed in descending sequence in the generated browse, mark this option. Otherwise, values are sorted in ascending sequence. **Note:** For Adabas and VSAM files, all components of a logical key must use the same sort order.
Histogram support	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. **Note:** This option is only allowed when the associated key has one key component.
Limit components	Number of components of a superdescriptor (compound key) used in the logical key (if the relational database table contains a superdescriptor with many components). 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". **Note:** Using fewer components in the key may make accessing the key more efficient.

Field	Description
Prefix components	Prefix used for components of a superdescriptor (compound key). 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. **Notes:** 1. For more information about prefix components, see Use Compound Browse Keys with Multiple Components. 2. When browsing Adabas or VSAM files by a single superdescriptor, efficiency is not affected by specifying prefix key components.

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 CUSTOMER-WAREHOUSE-ID	Ascending 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.

Field	**Description**
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)	• 0 = DEFAULT (embedded wildcard in key) • 1 = LESS-THAN • 2 = LESS-THAN-OR-EQUAL • 3 = EQUAL • 4 = GREATER-THAN-OR-EQUAL • 5 = GREATER-THAN • 6 = BEGINS-WITH **Note:** 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)	If this flag is set, browsing by a non-unique key begins at a specified record. 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. 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.
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.

Field	Description
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               OBJECT-BROWSE-SUBP Multi-Module                   CSGSM0
Aug 18                            User Exits                             1 of 1

               User Exits             Exists    Sample   Required Conditional
    -------------------------------- -------- ---------- -------- ------------
 _  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                    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-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..
        **SAG GENERATOR: OBJECT-BROWSE-STATIC             VERSION: 4.5.1
        **SAG TITLE: Object Browse Static ...
        **SAG SYSTEM: SYSCSTDE
        **SAG DESCS(1): Object Browse Static for ....
        **SAG SUBPROGRAM-NAME: CUSBRSUB
        **SAG STATIC-OBJECT-PDA: CUSBP020
        **SAG STATIC-OCCURRENCES: 00020
        ********************************************************
        * Program  : CUSBR020
        * System   : SYSCSTDE
        * Title    : Object Browse Static ...
        * Generated: Oct 08,03 at 01:03 PM by SAG
        * Function : Object Browse Static for ....
        *
        *
        *
        * History
        ********************************************************
        DEFINE DATA
          PARAMETER USING CUSBRKEY
          PARAMETER USING CUSBP020
          PARAMETER USING CUSBRPRI
          PARAMETER USING CDBRPDA
          PARAMETER USING CDPDA-M
        LOCAL
          01 #PROGRAM (A8)
        END-DEFINE
        PROG.    /* to allow escape from routine.
        REPEAT
        *
        PERFORM INITIALIZATIONS
          CALLNAT 'CUSBRSUB'
                      CUSBRKEY
                      CUSBRROW
                      CUSBRPRI
                      CDBRPDA
                      MSG-INFO
        *
        ********************************************************
        DEFINE SUBROUTINE INITIALIZATIONS
        ********************************************************
        *
          ASSIGN #PROGRAM = *PROGRAM
        *
        END-SUBROUTINE /* INITIALIZATIONS
        *
        ESCAPE BOTTOM(PROG.) IMMEDIATE
        END-REPEAT  /* PROG.
        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 ............. CST341S_________________________

  Title .............. Object Browse Static ..._
  Description ........ Object Browse Static for ....__________________________
                       ______________________________________________________
                       ______________________________________________________
                       ______________________________________________________

                                      Source      Object
  Object Browse Subp . ________ *
  Object PDA .........


                            PDA       Generate    Source      Object
  Static Occurrences . 10_  ________      _


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

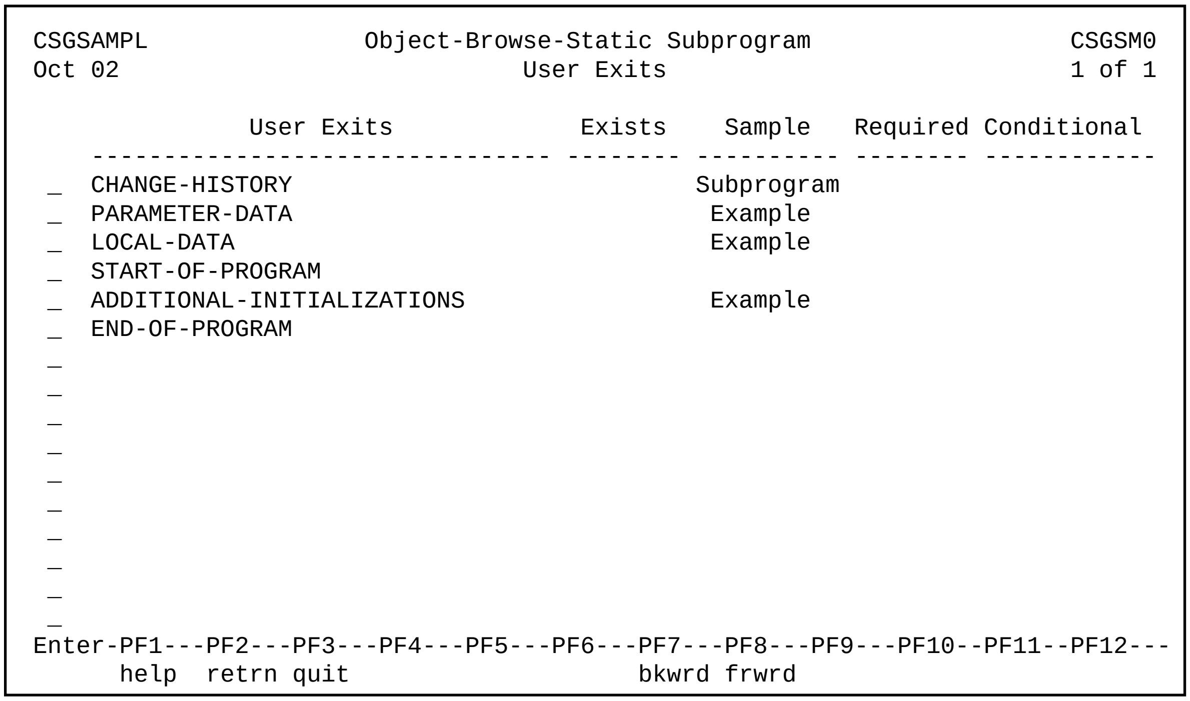
The fields in the upper portion of this panel are similar for all models. For a description of these fields, see Common Fields on the Standard Parameters Panel.

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 nnn 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
    ------------------------------- -------- ---------- -------- -----------
 _  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 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-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 helproutines — with or without selection capabilities.

In addition, the Object-Browse-Dialog model:

- Provides user exit modules to support regeneratable 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 generated object-browse dialogs.

Object-browse dialogs are useful to an application as:

- Stand-alone programs invoked from a menu by an object-browse dialog driver.
- Active helproutines 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, the user 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:

```
NCPRDOBD                  Table Subsystem
Oct 15                    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             exprt bkwrd frwrd 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 CDBRPDA.
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 for the object-browse dialog. This field provides 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 generate a title page when a map is not being used with an object-browse-dialog module. By default, a Natural Construct-generated browse module does this and an object-browse-dialog module does not. Mark this field to have the generator automatically code the page title if a map is not being used.

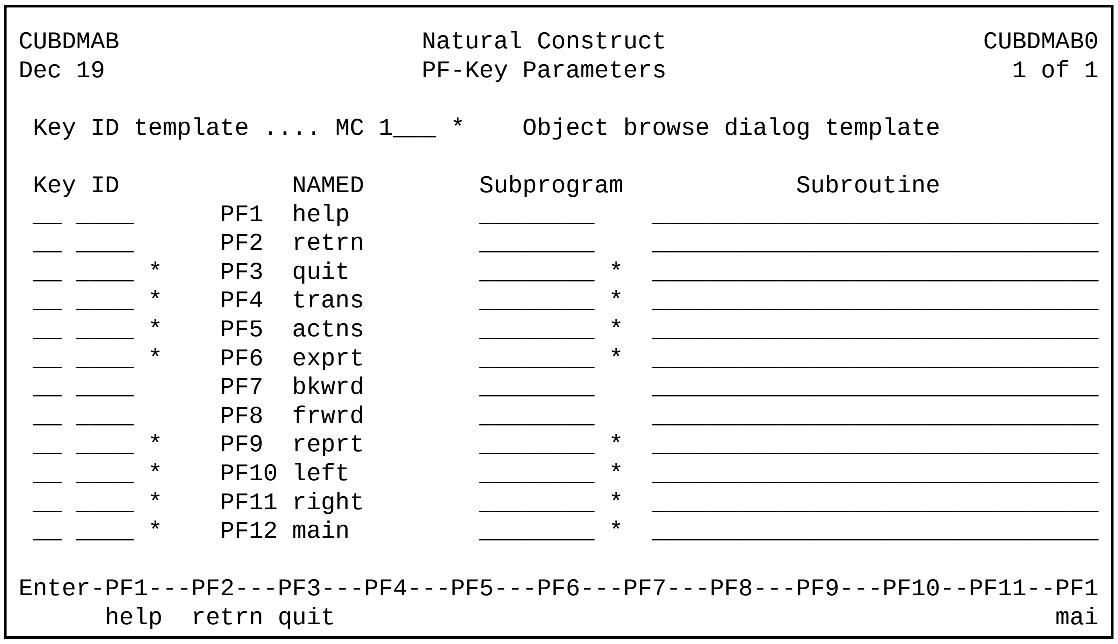
Note:
These fields are used by the Transform-Browse model to determine how to transform a Natural Construct-generated browse module into object-browse modules. For more information about transforming browse modules, see *Transform-Browse Model*, *Natural Construct Generation*.

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  bkwrd       ________    ________________________________
 __ ____      PF8  frwrd       ________    ________________________________
 __ ____ *    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. Press 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 press 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. Press Enter.

Change the Reserved PF-Keys

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

Use the Null PF-Key

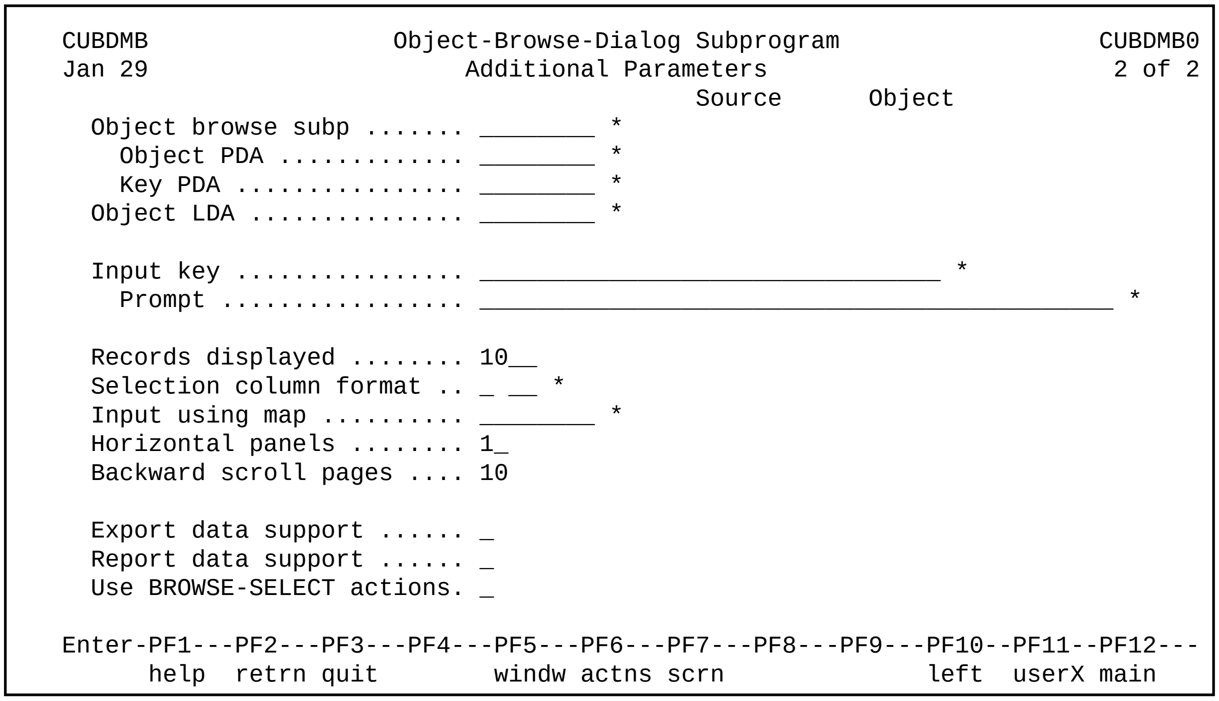
The null PF-key (PF0) disables the default PF-key in the position to which you assign it. Use this key when you want to disable the functionality of a PF-key and not replace it with another function. For example, to disable PF12 (main), enter “PF0 null” in the corresponding Key ID field in the PF-Key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

Define International Parameters

To define international parameters for your object-browse dialog, press PF9 (intnl) 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. **Note:** 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.
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 Use SYSERR References for Headings and Prompts.
Records displayed	Number of records displayed on the screen at one time. By default, the generated module displays 10 records at one time.

Field	**Description**
Selection column format	Indicates the format for the selection column. The following formats are available: ● A (alphanumeric) 1–253 units ● B (binary) 1–126 units ● C (attribute control) ● D (date) ● F (floating point) 4, 8 units ● I (integer) 1, 2, 4 units ● L (logical) ● N (numeric, unpacked) 1–29 units, 1–7 decimals ● P (numeric, packed) 1–29 units, 1–7 decimals ● T (time)
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. You can use a different layout map for each horizontal panel. If you include an asterisk (*) at the end of the map name (MYMAP*, for example), Natural Construct replaces the asterisk with the panel number in the generated module (MYMAP1, MYMAP2, MYMAP3, etc.). If you specify more than 9 panels, the map name cannot exceed 6 bytes. **Note:** If you do not specify a map name, Natural Construct places the input fields sequentially at the bottom of the panel.
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. **Note:** Natural Construct-generated modules do not allow backward scrolling through data that has not been previously scrolled through in a forward direction.

Field	Description
Export data support	If this field is marked, 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. 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: ● Use the Export-Data-Fields user exit model to generate the exit (this model is available on the mainframe only) ● Select and define the EXPORT-DATA-FIELDS user exit by right-clicking the generated the module in the Natural for Windows editor You can customize the work file number and delimiter character (character used to delimit fields on the report) for your site. **Note:** 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.
Report data support	If this field is marked, 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. 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: ● Use the Report-Data-Fields user exit model to generate the exit (this model is available on the mainframe only) ● Select and define the REPORT-DATA-FIELDS user exit by right-clicking the generated the module in the Natural for Windows editor **Note:** 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.
Use BROWSE-SELECT actions	If this field is marked, the browse dialog uses the same actions as those used by a Browse-Select model. For information, see Define or Customize Browse-Select Actions.

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 press 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. Press Enter.

Define or Customize Browse-Select Actions

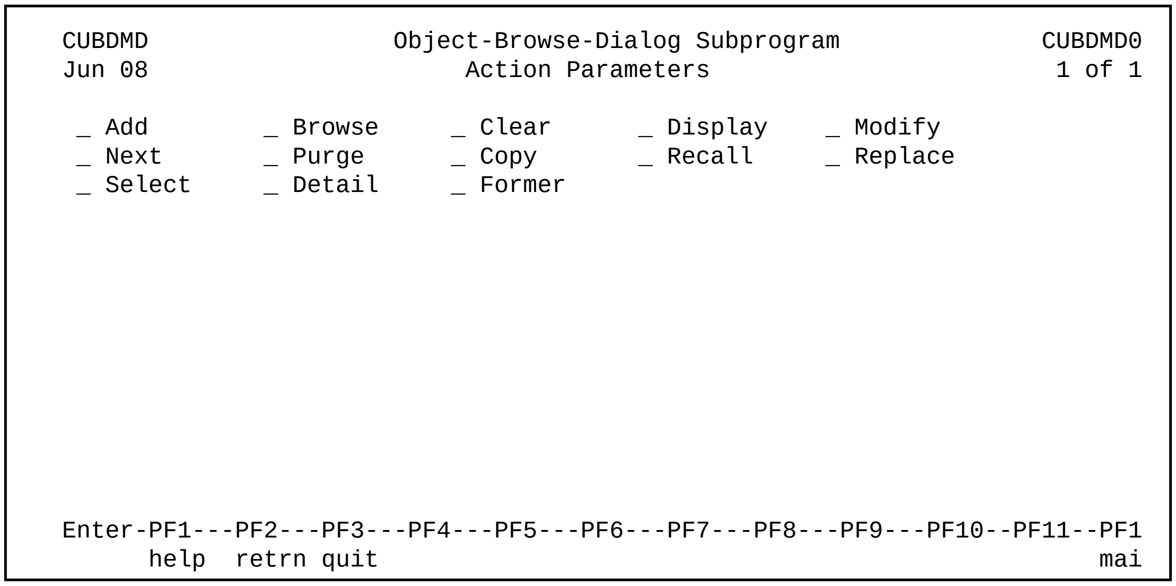
You can define Browse-Select-style actions for your object-browse dialog, as opposed to the standard actions for the object-browse dialog (for example, add, detail, display, modify, purge, and select).

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.

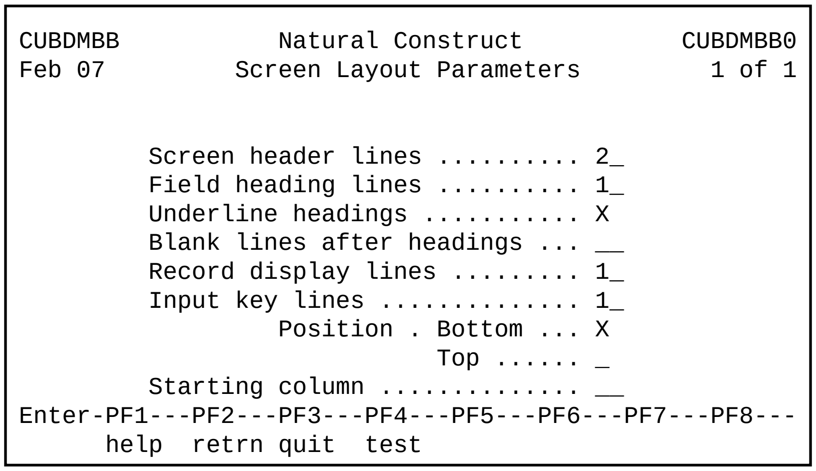
4. Press Enter to confirm the changes.

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	If this field is marked, field headings are underlined when the generated module is invoked. By default, this field is marked 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	
Bottom	If this field is marked, 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	If this field is marked, 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. Press 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: ● SC (Single Construct) ● SA (Single Application)
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. **Note:** If you use SYSERR references, text is retrieved from the CSTPFK library in SYSERR. 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.
PFKEY-NAME-LONG	Text or SYSERR reference used as a long description of a single PF-key.
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: ● C (Conditional; PF-key is reserved by the model and is conditional on the selection of options) ● O (Optional; PF-key can be overridden by the user) ● R (Required; PF-key is required by the model)

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

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/O’>
          (2)<’PF9/#PF-STOP/#STOP-NAME/*9002.1/*9002.2/O’>
```

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  *****                         00-10-16
                      - 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/O'>
          (2)<'PF9/#PF-STOP/#STOP-NAME/*9002.1/*9002.2/O'>
*
* add new single pfkey attributes
*
          (3)<'PFnn/#PF-name/#key-NAME/*9003.1/*9003.2/O'>
```

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: ● MC (Model Construct) ● MA (Model Application)
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. **Note:** 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. **Note:** 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/   /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’>
```

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  *****                   00-10-16
                      - 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  bkwrd        PF7          Scroll backward
  _  SC0004  frwrd        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                    bkwrd frwrd                   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: ● API-NAME (A8) ● INPUTS (A1/1:V) ● INPUT-OUTPUTS (A1/1:V) ● OUTPUTS (A1/1:V) 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.
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)	• METHOD (A2) • VERSION (N1) • SOURCE-ID (A2) • PFKEY-ID (N4)
INPUT-OUTPUTS (A1/113)	• PFKEY-POSITION (A4) • PFKEY-POSITION-VARIABLE (A32) • PFKEY-NAME-VARIABLE (A32) • PFKEY-NAME (A12) • PFKEY-NAME-LONG (A32) • PFKEY-STATUS (A1)
OUTPUTS (A1/68)	• METHOD-LIBRARY (A8) • METHOD-DESCRIPTION (A60)

- 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)	• METHOD (A2) • VERSION (N1) • START-SOURCE-ID (A2) • START-PFKEY-ID (N4) • MAX-PFKEY-REQUESTED (N4) • RESTRICT-SOURCE-ID (A2)
INPUT-OUTPUTS (A1/1428)	• PFKEY-ARRAY (1:12) Redefined as: ○ PFKEY-IDENTIFIER (A6) Redefined as: • PFKEY-SOURCE-ID (A2) • PFKEY-PFKEY-ID (N4) ○ PFKEY-POSITION (A4) ○ PFKEY-POSITION-VARIABLE (A32) ○ PFKEY-NAME-VARIABLE (A32) ○ PFKEY-NAME (A12) ○ PFKEY-NAME-LONG (A32) ○ PFKEY-STATUS (A1)
OUTPUTS (A1/72)	• METHOD-LIBRARY (A8) • METHOD-DESCRIPTION (A60) • MAX-PFKEY-RETURNED (N4)

- 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: ● SC (Single Construct) ● SA (Single Application)
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.
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. **Note:** If you use SYSERR references, text is retrieved from the CSTACT library in SYSERR. **Note:** 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: ● C (Conditional; action is reserved by the model and is conditional on the selection of options) ● O (Optional; action can be overridden by the user) ● R (Required; action is required by the model)

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/O'>
          (2)<'#BROWSE/*8002.1/*8002.2/*8002.3/O'>
          (3)<'#CLEAR/*8003.1/*8003.2/*8003.3/O'>
          (4)<'#COPY/*8004.1/*8004.2/*8004.3/O'>
          (5)<'#DETAIL/*8005.1/*8005.2/*8005.3/O'>
          (6)<'#DISPLAY/*8006.1/*8006.2/*8006.3/O'>
          (7)<'#MODIFY/*8007.1/*8007.2/*8007.3/O'>
          (8)<'#NEXT/*8008.1/*8008.2/*8008.3/O'>
          (9)<'#PURGE/*8009.1/*8009.2/*8009.3/O'>
         (10)<'#RECALL/*8010.1/*8010.2/*8010.3/O'>
         (11)<'#REPLACE/*8011.1/*8011.2/*8011.3/O'>
         (12)<'#SELECT/*8012.1/*8012.2/*8012.3/O'>
```

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/O'>
          (2)<'#SKIP/*9002.1/*9002.2/*9002.3/O'>
```

Note:
SYSERR references are defined in the CSTACT 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  *****                       00-10-16
                      - 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/O'>
          (2)<'#SKIP/*9002.1/*9002.2/*9002.3/O'>
*
* add new single action attributes
*
          (3)<'#NEW-ACTION/*9003.1/*9003.2/*9003.3/O'>
```

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: ● MC (Model Construct) ● MA (Model Application)
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. **Note:** If you use SYSERR references, the text is retrieved from the CSTACT library in SYSERR.
MODEL-ACTION-VALUES	Action template containing a group of single action definitions. **Note:** The maximum number of actions assigned to one action template is 12.
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 *****                        00-10-16
                      - 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:

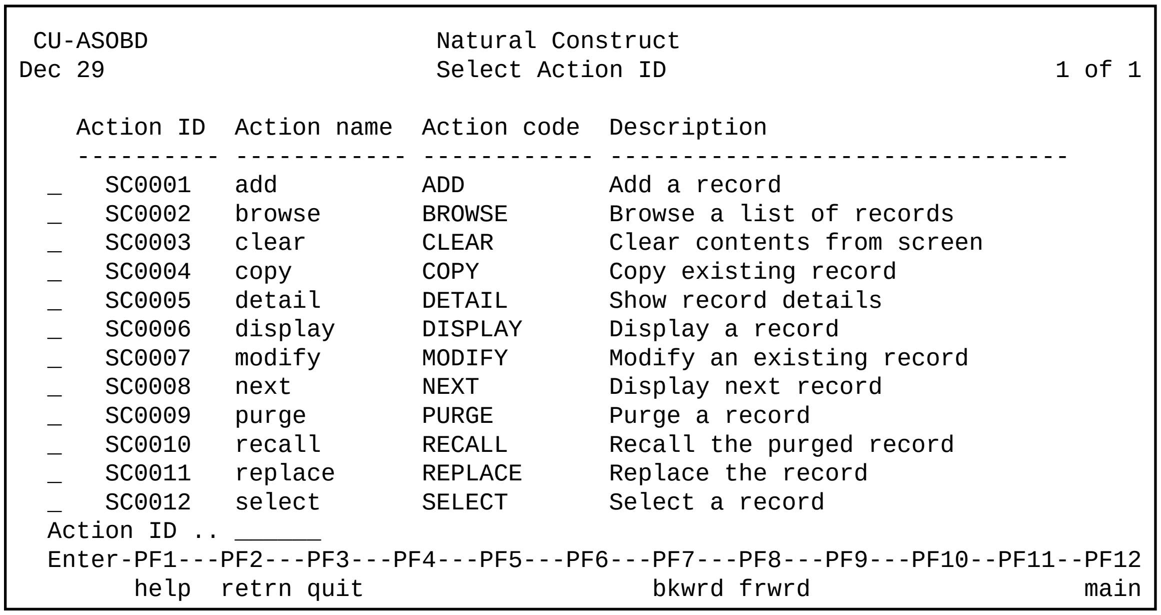
```
    02 INITIAL-MODEL-ACTION-VALUES (A15/1:MAX-MAA,1:24)
*     Model\   source ID/action ID/status override/
*           \          /     _____/               /
*            \        /    / ___________________/
*             \      /    / /
    INIT (1,1) <'SC/0012/C'>
         (1,2) <'SC/0010/ '>
         (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/O'>
         (2,4) <'SA/0001/ '>
         (2,5) <'SC/0010/C'>
```

7. Stow CD-ACT.

Select Actions and Action Templates

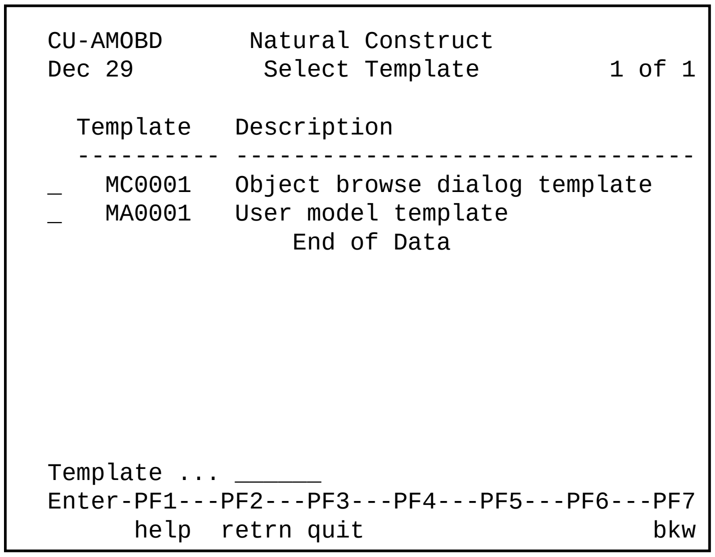
You can display helproutine windows listing valid IDs for single actions and action templates. The CU-ASOBD helproutine (listing the single actions) is invoked by the CU-ASH driver. The CU-AMOBD helproutine (listing the action templates) is invoked by the CU-AMH driver. You can also create your own helproutine 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                    bkwrd frwrd                   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                    bkw
```

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	Parameter data area (PDA) containing the following fields: ● API-NAME (A8) ● INPUTS (A1/1:V) ● INPUT-OUTPUTS (A1/1:V) ● OUTPUTS (A1/1:V) 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.
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)	• METHOD (A2) • VERSION (N1) • SOURCE-ID (A2) • ACTION-ID (N4)
INPUT-OUTPUTS (A1/89)	• ACTION-NAME-VARIABLE (A32) • ACTION-CODE(A12) • ACTION-NAME (A12) • ACTION-NAME-LONG (A32) • ACTION-STATUS (A1)
OUTPUTS (A1/68)	• METHOD-LIBRARY (A8) • METHOD-DESCRIPTION (A60)

- 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)	● METHOD (A2) ● VERSION (N1) ● START-SOURCE-ID (A2) ● START-ACTION-ID (N4) ● MAX-ACTION-REQUESTED (N4) ● RESTRICT-SOURCE-ID (A2)
INPUT-OUTPUTS (A1/1140)	● ACTION-ARRAY (1:12) Redefined as: ○ ACTION-IDENTIFIER (A6) Redefined as: ● ACTION-SOURCE-ID (A2) ● ACTION-ACTION-ID (N4) ○ ACTION-NAME-VARIABLE (A32) ○ ACTION-CODE (A12) ○ ACTION-NAME (A12) ○ ACTION-NAME-LONG (A32) ○ ACTION-STATUS (A1)
OUTPUTS (A1/72)	● METHOD-LIBRARY (A8) ● METHOD-DESCRIPTION (A60) ● MAX-ACTION-RETURNED (N4)

- 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                      Natural Construct                         CSGSM0
Nov 21                            User Exits                            1 of 1

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

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:

1. To view the specifications for this example, refer to the NCPRDEX text member in the Natural Construct demo system.
2. 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 bkwrd frwrd 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 NCPRDOBD 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:

```
 NCPRDOBD                     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 bkwrd frwrd 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 bkwrd frwrd 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)

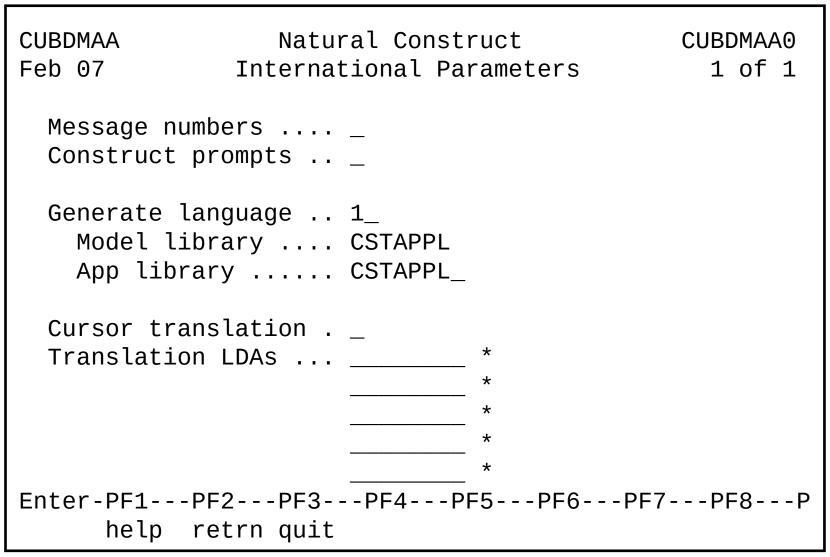
Define International Parameters

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 intnl 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. If this field is marked, the generated code uses message numbers rather than message text.
Construct prompts	Type of messages used. If 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	If this field is marked, 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. **Note:** Use the Object-LDA model to create translation LDAs. For information, see Object-LDA Model.

3. Press 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, press 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 Additional Parameters	First heading and Second heading 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             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 bkwrd frwrd
```

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 frwrd or bkwrd PF-keys to scroll to it.

3. Press 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.

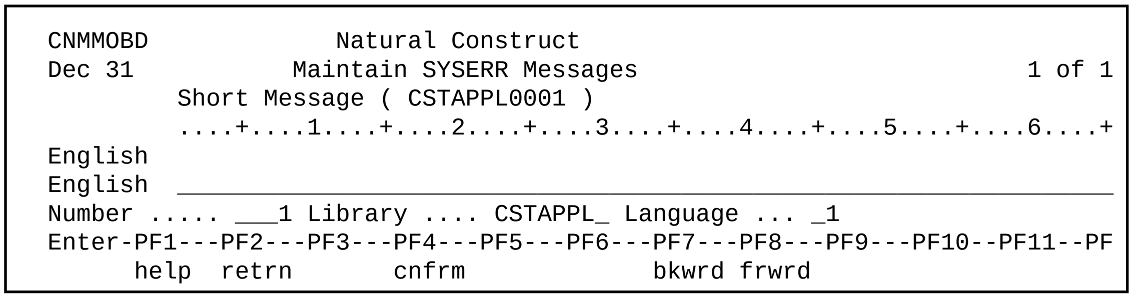
Add or Maintain a SYSERR Reference

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 31          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             bkwrd frwrd
```

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 frwrd or bkwrd 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 helproutine 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 ............. CST421S_________________________

  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.

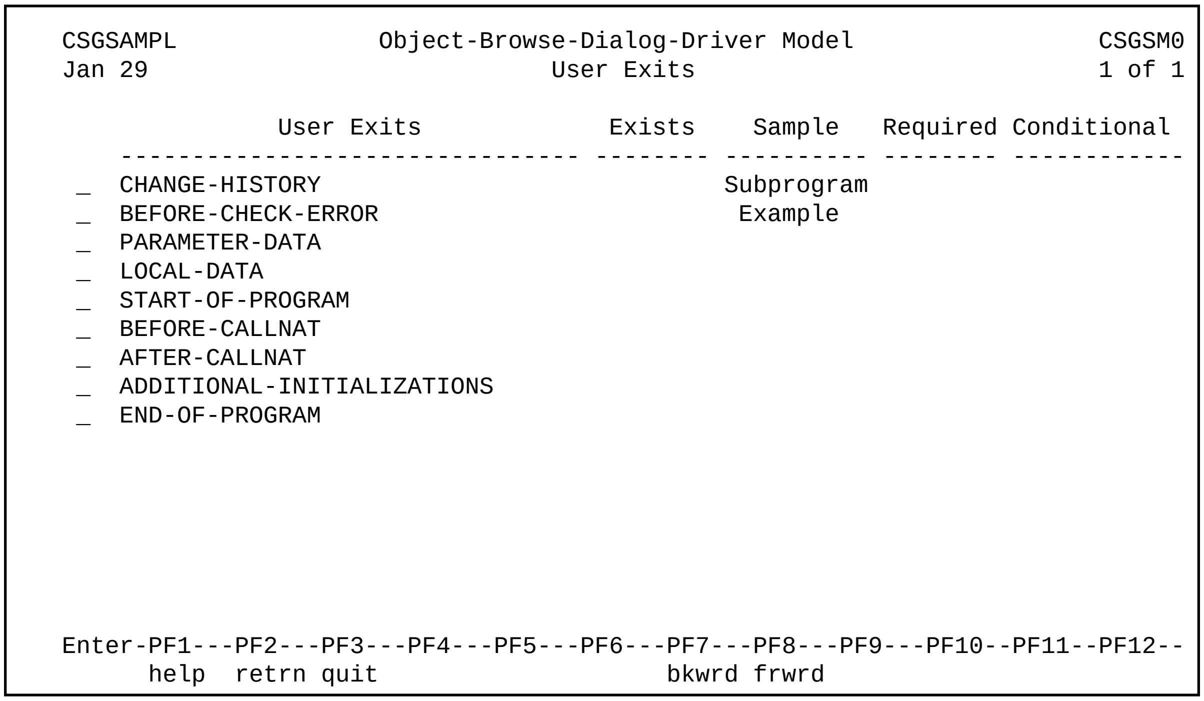
Note:
The Module type for an object-browse-dialog-driver module can be "P", "H", or "N". When the module type is "N" (subprogram), the primary key (#PDA-KEY) can be overridden (similar to a browse subprogram generated by the Browse-Subp model).

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
      -------------------------------- -------- ---------- -------- ------------
   _  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                    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-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 ............. CST421S__________________________

           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.

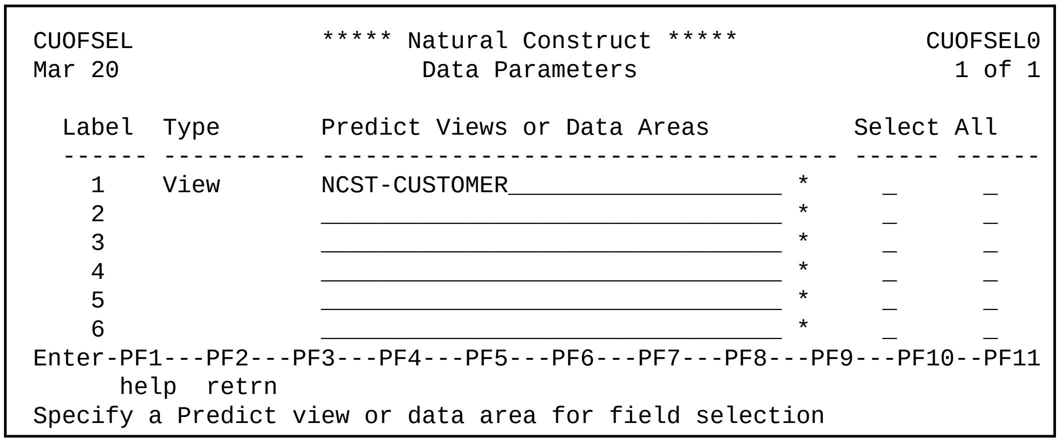
The fields in the lower portion of this panel are:

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). **Note:** 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:



```
CUOFSEL             ***** Natural Construct *****            CUOFSEL0
Mar 20                     Data Parameters                     1 of 1

  Label  Type       Predict Views or Data Areas        Select All
  ------ ---------- ---------------------------------- ------ ------
    1    View       NCST-CUSTOMER___________________ *     _      _
    2               ________________________________ *     _      _
    3               ________________________________ *     _      _
    4               ________________________________ *     _      _
    5               ________________________________ *     _      _
    6               ________________________________ *     _      _
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11
      help  retrn
Specify a Predict view or data area for field selection
```

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 CUOFPDA; you cannot re-select existing fields in CUOFPDA.

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 press Enter.
All	To select all fields from the corresponding view or data area, mark this field and press Enter. • When selecting from a view, multiple-valued fields (MUs) within a periodic group (PE) are not included. • When selecting from a data area, constants and arrays with a rank greater than 1 are not included. **Note:** If there is more than one structure in a data area, only the first structure is included when you mark this field.

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 bkwrd frwrd       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	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. **Note:** You can also display the specification lines for other fields by pressing the frwrd or bkwrd PF-keys.
Ord	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): 1. Scroll to the first empty line (you may have to press the frwrd PF-key). 2. Press the selfd PF-key and select a new field. By default, the new field is assigned the next available number. 3. Type 15 over the default number (next to the new field). 4. Press the reord PF-key. In this example, the fields are reordered so that the new field is now in the second position (position 20).

Field	Description
Lbl	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.
Field Name	Name of a field selected for the screen layout (as defined in Natural). The Object-LDA model supplies this name.
Field Heading	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. Field-level help is available to select an existing SYSERR number or create a new SYSERR entry. For more information, see Use SYSERR References for Headings and Prompts.
+ or -	To add more key components to a logical key or set additional options for the logical keys, mark this field and press 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. **Note:** You can also display the window by pressing the parms PF-key on this panel.

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 *KeyName*Count (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, FindBy*KeyName*). 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	● Speeds up the development process; code generated by this model may already exist. ● Combines data and business layers.	● Has only one method (BROWSE). ● Places lookup and derived data in a separate level 1 data area; this data cannot be assigned at the ROW level.
Object-Browse-Select	● Provides more methods; the method names are more descriptive. ● Separates data and business access layers. ● Assigns lookup and derived data at the ROW level with other data. ● Allows users to easily create tables for data; the data is handled like a dataset on the client.	● Requires an additional subprogram layer.

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 MultiMaint 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: FindBy*KeyName* and *KeyName*Count. 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 .................... _
                                PDA        Generate    Source      Object
 Static occurrences ...... 20_  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. **Note:** If you use an object-maintenance subprogram and object-browse subprogram, both subprograms must use the same primary file.
Time	If this field is marked, 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 **Note:** 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 the static occurrences hardcodes this value (which is used to identify the *V* value in the object-browse subprogram) into the object-browse-select subprogram. 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 (BCUST + S20 in the example). **Note:** If you change the number of static occurrences on this panel, you should also change this number in the default PDA name.
Message numbers	If this field is marked, message numbers are used and messages are retrieved from SYSERR at runtime. **Note:** 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.

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	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: ● ROW-STATE Contains maintenance instructions or maintenance message codes. ● ROW-ID Used internally on the client (do not change). ● ROW-ERROR-DATA Contains message information returned by the object-maintenance subprogram for the affected row (i.e., transaction states).
CDBUPDA	Subset of the CDBRPDA PDA for object-browse subprograms. CDBUPDA contains all the parameters in CDBRPDA except the following: ● METHOD Not required. ● SORT-KEY The object-browse-select subprogram sets this value for CDBRPDA based on the FindBy* methods. ● HISTOGRAM The object-browse-select subprogram sets this value for CDBRPDA based on the Count* methods). ● ROWS-REQUESTED Not required. ● LEADING-FIXED-COMPONENTS Set in the CDBUPDA2 parameter data area (for information, see Specify Leading Fixed Components for the Logical Key). ● USE-UNIQUE-ID The object-browse-select subprogram sets this value.

Parameter	Description

CDBUINFO	Similar to the CDPDA-M parameter data area for object-browse subprograms; this PDA contains messages and transaction information for all rows. For example:
	
	`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`

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
    -------------------------------- -------- ---------- -------- ------------
  _ 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
```

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*.