Enhanced Source Code Format

The dialog editor supports the following source formats:

- 213. This is the format generated by Natural Version 2.1.3 (New Dimension). It is supported for input only. You cannot generate 2.1.3 format with Natural Version 2.2 and with Natural Version 3.
- 22C. This is the format generated by Natural Version 2.2.2. As from Natural for Windows, Unix Version 4.1 and OpenVMS, this format cannot be generated.
- 22D. This is the standard "enhanced" source code format. It is generated for compiling, storing, and editing dialogs in Natural Version 2.2.3 and above.

This section describes the syntax conventions for entering code in the source code window provided by the program editor.

- Syntax Conventions
- How Natural Dialogs Work
- Syntax

Syntax Conventions

Syntax is described using the following meta-notation:

```
syntax_element_name :=: description
```

Syntax Element	Represents
syntax_element_name :=:	Identifies the construct whose structure is defined by *description.*
description	Program code displayed in UPPER-CASE letters.

The Syntax Symbols used in this section are explained in the *Statements* documentation. In addition, the following symbols are used within the diagrams:

Symbol	Represents
{*element1* **|** *element2***}**	Elements contained within braces and separated by vertical bars indicate that exactly one of the elements must be specified.
< >	Pairs of angle brackets indicate that code is separated into multiple lines. Each pair of brackets represents the end of a Natural source line.
/*[and **/*]**	These bracket pairs form a collapsible block in list mode, even in user code sections.

Line-number references are explicitly *not* supported within dialogs, as the dialog editor will generate line numbers for dialog sources without consideration of any line-number references.

Line length and source size are subject to the general limits imposed by Natural.

Note:
Where syntax elements are shown separated by blanks, blanks are required, but the exact number of blanks is not significant.

How Natural Dialogs Work

This section gives you some background information so you can better understand what the various elements of the dialog source do.

A Natural dialog is an executable module. You invoke it either directly from the Natural environment or from another executing module with an `OPEN DIALOG` statement. Invoking the dialog causes the dialog's executable to be instantiated and executed with the system variables `*CONTROL` equal to `NULL-HANDLE` and `*EVENT` equal to `OPEN`. This `OPEN` event is processed by calling the inline subroutine `*DLG$SUBR$CREATE$WINDOW` which creates the dialog window. Window creation invokes the dialog with `*CONTROL` equal to `NULL-HANDLE` and `*EVENT` equal to `AFTER-OPEN`, to which the dialog responds by calling the inline subroutine `*DLG$SUBR$CREATE$CONTROLS`. This subroutine creates any dialog elements defined for the dialog, and then calls `PROCESS GUI ACTION AFTER-CREATION` to notify Natural that window creation is complete.

Depending on further application processing and user input, the dialog will repeatedly be executed with `*CONTROL` and `*EVENT` set appropriately. The dialog remains instantiated until it is explicitly unloaded as a result of the window being destroyed. This is the default reaction to the `CLOSE` event which in turn occurs as a result of either end-user interaction or the `CLOSE DIALOG` statement.

The same dialog module may be instantiated more than once, resulting in more than one dialog and its window and dialog elements being active. Each dialog has a unique numeric ID which is available as `*DIALOG-ID` during execution of that dialog, and as *window_handle*.CLIENT-DATA wherever that dialog's window handle is accessible (the window must be created with that attribute setting).

Syntax

- General Syntax
- Subsections of the General Syntax
- Subordinate Syntax Sections

General Syntax

This is the complete source of a dialog.

```
dialog_source_22D
 :=:
```

```
/** DIALOG SOURCE 22D dialog_info_section_22D
[dialog_options_section_22D]
dialog_data_section_22D [user_subroutines_section_22D]
window_definition_22D control_definitions_22D
default_handler_section_22D
error_handler_section_22D before_any_section_22D
event_handlers_22D after_any_section_22D
END /** END-DIALOG-SOURCE
```

Subsections of the General Syntax

dialog_info_section_22D

```
dialog_info_section_22D
 :=:
```

The dialog’s "info" section, consisting of a banner line, frame gallery information,

and an optional comment.

```
/*[ DEFINE DIALOG INFO
```

The following line is always generated by the dialog editor, but ignored on input.

```
/*D* Natural Dialog Description version_string / date
 time
```

One or more following lines are present in dialogs generated using the frame gallery.

If present, they are preserved by (but cannot be changed in) the dialog editor.

```
[/*DF frame_gallery_info]...
```

The empty comment line is generated by the dialog editor if no user comment is

present, but ignored on input.

```
[dialog_comment :=: {/** EMPTY DIALOG COMMENT | [user_code_line_protected_by_/**_prefix]...}]
 /*] END-DIALOG-INFO
```

dialog_options_section_22D

```
dialog_options_section_22D
 :=:
```

These option settings are only generated by the dialog editor if the "Save settings

with dialog" option is on (see *Dialog Editor Options* and *Setting the Options* in the *Using Natural Studio* documentation). When a dialog is being loaded and this section is present,

the option is turned on.

```
[/*[ DEFINE OPTION SETTINGS {/** SET option_name option_value}...
 /*] END-OPTION-SETTINGS]
```

dialog_data_section_22D

```
dialog_data_section_22D
 :=:
```

The dialog’s data section, consisting of a global, parameter, and local data area.

The sequence is fixed.

```
 DEFINE DATA gda_section_22D
 pda_section_22D lda_section_22D
 END-DEFINE
```

Subsections of the dialog_data_section_22D

gda_section_22D

```
gda_section_22D
 :=:
```

The optional global data area specification. Note that it may reference blocks.

```
/*[ DEFINE GLOBAL DATA [GLOBAL USING gda_specification] /*] END-GLOBAL-DATA
```

pda_section_22D

```
pda_section_22D
 :=:
```

The dialog’s parameter data area must always contain the parent handle as the

first field.

```
/*[ DEFINE DIALOG PARAMETERS PARAMETER 01 #DLG$PARENT HANDLE OF
 GUI BY VALUE user_data_section_22D
 /*[ DEFINE USING [PARAMETER USING pda_name]... /*] END-USING /*] END-DIALOG-PARAMETERS
```

lda_section_22D

```
lda_section_22D
 :=:
```

The dialog’s local data area. The handle declarations for all dialog elements are

necessary if the dialog is to be compiled, but are ignored by the dialog editor

on input and re-generated from the actual dialog element definitions below.

```
/*[ DEFINE LOCAL DATA LOCAL /*[ DEFINE HANDLES {01 control_name
 HANDLE OF control_class[<>]}... /*] END-HANDLES user_data_section_22D
 /*[ DEFINE USING [LOCAL USING lda_name]... /*] END-USING /*] END-LOCAL-DATA
```

user_subroutines_section_22D

```
user_subroutines_section_22D
 :=:
```

The list of user-defined subroutines. The enclosing pseudo-syntax is generated

only if there actually are subroutines.

```
/*[ DEFINE SUBROUTINES [DEFINE SUBROUTINE subroutine_name
 user_code_section_22D END-SUBROUTINE]...
 /*] END-SUBROUTINES
```

window_definition_22D

```
window_definition_22D
 :=:
```

The window is defined within a standard subroutine (this must always be present).

```
DEFINE SUBROUTINE #DLG$ SUBR$CREATE$WINDOW /** DEFINE CONTROL window_name
 non_array_control_definition_22D
  END-SUBROUTINE /** END-CONTROL
```

control_definitions_22D

```
control_definitions_22D
 :=:
```

All dialog elements are defined within one standard subroutine (this must

always be present). The sequence generated by the dialog editor is: menu bar,

tool bar, font controls, timers, any other dialog elements.

```
DEFINE SUBROUTINE #DLG$SUBR$CREATE$CONTROLS /** DEFINE DIALOG
 ELEMENTS [control_definition :=:
```

Each dialog element definition is enclosed in pseudo-comments.

```
/*[
 DEFINE CONTROL control_name[array_bounds]
```

The following optional comment appears in the source code only.

```
[control_comment
 :=: {user_code_line_protected_by_/**_prefix}...]
```

The following code creates the dialog element.

```
{non_array_control_definition_22D
 | array_control_definition_22D}
```

List box controls and selection-box controls may have items lists defined as one

array of dialog elements. These follow the dialog element creation code.

```
[control_items :=: /* DEFINE ITEMS control_name_array_bounds
 array_control_definition_22D
 /*] END-ITEMS ] /*] END-CONTROL
```

```
]...
 END-SUBROUTINE /** END-DIALOG-ELEMENTS
```

default_handler_section_22D

default_handler_section_22D
 :=:

The DEFAULT event handler, specified as subroutine. For compilation, it must be

present, as this subroutine is called from various places.

```
DEFINE SUBROUTINE
 #DLG$HANDLER$DEFAULT /** DEFINE EVENT DEFAULT user_code_section_22D
  END-SUBROUTINE /** END-EVENT
```

error_handler_section_22D

error_handler_section_22D
 :=:

The ERROR event handler, specified as ON ERROR section. Optional.

```
ON ERROR /** DEFINE
 EVENT ERROR user_code_section_22D
 END-ERROR /** END-EVENT
```

before_any_section_22D

before_any_section_22D
 :=:

The BEFORE-ANY event handler, that is, the code which precedes the DECIDE

statements which evaluate *CONTROL and *EVENT.

```
/*[ DEFINE EVENT BEFORE-ANY user_code_section_22D
 /*] END-EVENT
```

event_handlers_22D

event_handlers_22D
 :=:

The DECIDE statements which evaluate first *CONTROL, then *EVENT,

activating the appropriate event handlers.

```
DECIDE
 ON FIRST *CONTROL /** DEFINE ALL EVENTS {dialog_events :=: /*[ DEFINE
 EVENTS FOR DIALOG VALUE NULL-HANDLE
```

*CONTROL = NULL-HANDLE indicates an event associated either with the

window or with the dialog itself.

```
DECIDE ON FIRST *EVENT /*[
 DEFINE EVENT OPEN VALUE 'OPEN'
```

The OPEN event handler starts with any BEFORE-OPEN user code and ends

with the window creation call.

```
user_code_section_22D
 PERFORM #DLG$SUBR$CREATE$WINDOW /*] END-EVENT /*[ DEFINE EVENT AFTER-OPEN VALUE
 'AFTER-OPEN'
```

The AFTER-OPEN event occurs while the window creation call is being

processed, that is, it is a nested call of the dialog. It includes the creation of all

dialog elements and the assignment of those window attributes which use

dialog element handles (for example, the handle of the menu bar). User

code may follow.

```
PERFORM #DLG$SUBR$CREATE$CONTROLS [extra_window_attributes_22D]
 PROCESS GUI ACTION AFTER-CREATION WITH window_name PROCESS GUI ACTION
 RESET-ATTRIBUTES
```

The user's after-open code follows.

```
user_code_section_22D
 /*] END-EVENT /*[ DEFINE EVENT CLOSE
```

The CLOSE event occurs either because the end user directly closes the

window, or because the parent window is closed, or as the result of a CLOSE

DIALOG statement.

```
VALUE 'CLOSE'
 {dialog_close_handler_22D :=: user_code_section_22D}
```

The following call "destroys" the window and unloads the dialog.

```
PROCESS GUI ACTION DELETE-WINDOW WITH window_name
```

The following statement leaves the dialog, bypassing the AFTER-ANY

handler.

```
ESCAPE ROUTINE IMMEDIATE /*] END-EVENT
```

The following dialog event handlers are optional and include any user-defined

events.

```
[event_handler_section_22D...]
 NONE PERFORM #DLG$HANDLER$DEFAULT END-DECIDE /*] END-EVENTS
```

```
}
```

All event handlers for dialog elements defined in

*DLG$SUBR$CREATE$CONTROLS are listed below.

```
[control_events_section_22D...]
 NONE PERFORM #DLG$HANDLER$DEFAULT END-DECIDE /** END-ALL-EVENTS
```

All events that are not processed and have not been suppressed are handled in the

DEFAULT event handler. Any dialog elements created outside

DLG$SUBR$CREATE$CONTROLS are handled there.

after_any_section_22D

```
after_any_section_22D
 :=:
```

The AFTER-ANY event handler, that is, the code which follows the DECIDE

statements evaluating *CONTROL and *EVENT.

```
/*[ DEFINE EVENT AFTER-ANY user_code_section_22D
 /*] END-EVENT
```

Subordinate Syntax Sections

user_data_section_22D

```
user_data_section_22D
 :=:
```

A section with data declarations. The dialog editor generates a comment if no user

code is present. The user code layout is preserved, except that the whole section is

indented appropriately.

```
[frame_code_section_22D]
 {/** EMPTY USER CODE SECTION | user_code_line_22D...}
 [frame_code_section_22D]
```

user_code_section_22D

```
user_code_section_22D
 :=:
```

A section with executable statements. The dialog editor generates a comment if

no user code is present. The user code layout is preserved, except that the whole

section is indented appropriately.

```
[frame_code_section_22D]
 {;/** EMPTY USER CODE SECTION | user_code_line_22D...}
 [frame_code_section_22D]
```

frame_code_section_22D

```
frame_code_section_22D
 :=:
```

This protected section is code in a frame gallery dialog. Do not change this code.

```
/*[ DEFINE FRAME CODE [user_code_line_22D...]
 /*] END-FRAME-CODE
```

user_code_line_22D

```
user_code_line_22D
 :=:
```

A code line within an event section or subroutine. Indentation is preserved by the

dialog editor. However, a minimum indentation is enforced.

```
indented_code_line
```

event_handler_section_22D

```
event_handler_section_22D
 :=:
```

A single event handler section, that is, the VALUE clause for the event in a DECIDE

statement for the relevant dialog element.

```
/*[ DEFINE EVENT event_name
 VALUE 'event_name' user_code_section_22D
  /*] END-EVENT
```

control_events_section_22D

```
control_events_section_22D
 :=:
```

The collection of all event handlers for one dialog element, that is, a VALUE clause

for the dialog element’s handle containing a DECIDE statement for *EVENT. If th

dialog element is an array, all elements are handled in this section.

```
/*[
 DEFINE EVENTS FOR control_name VALUE control_name[(*[,*)] DECIDE
 ON FIRST *EVENT event_handler_section_22D...
 NONE PERFORM #DLG$HANDLER$DEFAULT END-DECIDE /*] END-EVENTS
```

non_array_control_definition_22D

```
non_array_control_definition_22D
 :=:
```

A non-array dialog element definition corresponds to the PROCESS GUI statement action that creates the dialog element. As the WITH PARAMETERS ... END-PARAMETERS clause is used, all attributes not mentioned have their default values. The dialog editor does not generate such attributes.

The first and the second attributes must be HANDLE-VARIABLE and TYPE.

The dialog editor uses the predefined attribute value names defined in the standard local data area NGULKEY1 if this data area has been included in the dialog. If it is not included, the dialog cannot be compiled. Note that variable references are allowed for only a subset of the attributes.

```
PROCESS GUI ACTION ADD WITH
 PARAMETERS {attribute_name = {constant | variable_reference}<>}...
 END-PARAMETERS [frame_code_line_22D...] /*] END-CONTROL
```

array_control_definition_22D

```
array_control_definition_22D
 :=:
```

An array definition of a dialog element consists of one PROCESS GUI statement for each array element. Only a subset of dialog elements may be defined as arrays (for example, not the dialog window or list box controls). Instead of the WITH PARAMETERS ... END-PARAMETERS clause, explicit attribute assignments and the simple WITH clause are used, as array elements will share many equal but non-default attribute values.

The dialog editor is quite restrictive with respect to dialog element arrays: a maximum of two dimensions is allowed; elements cannot be placed individually; only a subset of the attributes may vary between elements.

The dialog editor will, when scanning dialog element array definitions, take those attribute values which may not vary from the first element’s definition. Specifically,

all coordinate definitions but the first will be derived from the pseudo-attributes H-SPACING, V-SPACING, and ARRANGE-IN-COLUMNS. (Note, however, that not all dialog elements have coordinates.) All attributes not mentioned explicitly have their default values; this is enforced by the initial PROCESS GUI ACTION RESET-ALL.

```
PROCESS GUI ACTION RESET-ATTRIBUTES [spacing_info:=:
 /** H-SPACING = horiz_spacing /** V-SPACING = vert_spacing /**
 ARRANGE-IN-COLUMNS = {TRUE|FALSE}<>] ] {
```

Creation of one array element. Numeric references to a message file string must be handled in an explicit PROCESS GUI statement action for that string, as direct assignment to a number would simply convert that number to a string.

```
[attribute_assignment:=:{
  control_name_and_index.attribute_name := attribute_value|
 PROCESS GUI ACTION GET-MESSAGE-TEXT WITH number indexed_control_name
 }<>]... PROCESS GUI ACTION ADD WITH parent_name type_name control_name_and_index

}...
```

extra_control_attributes_22D

```
extra_control_attributes_22D
 :=:
```

Some attributes cannot be defined meaningfully in the PROCESS GUI ACTION ADD that creates the dialog element; an example is a non-modifiable selection-box control’s STRING, which must be one of the items’ STRING values. This attribute is therefore defined after all items have been added.

```
control_name_and_index.attribute_name
 := attribute_value
```

extra_window_attributes_22D

```
extra_window_attributes_22D
 :=:
```

Some attributes cannot be defined meaningfully in the PROCESS GUI ACTION ADD that creates the dialog, for example, the default button can only be defined after that button has been created.

```
window_name.attribute_name
 := attribute_value
```