

Customizing a Generated Application

Once the dialogs and other modules have been generated using the frame gallery, you use the Natural dialog editor to modify the dialog layout and to customize the generated code to conform to application-specific requirements.

Modules generated with the frame gallery include:

- Internal frame logic. The internal frame logic, or frame code, covers a large amount of the processing required to create a working module.
- Application specific code. Based on the DDM and fields you select, appropriate code is generated to define data structures, access and update database information, and produce dialog layout. This generated code meets most simple requirements, although you will need to modify the generated code to handle binary data, keys with unusual data types, or to improve dialog layouts.
- Customizable components which contain generated code and suggested code. Suggested code is included as a Natural comment. Suggested code helps you add further applications-specific code for your own requirements, for example to: link dialogs, handle validation, or provide active help. The customizable components can be modified and/or extended.

In modules other than dialogs, the components above are simply included together and are all modifiable using the appropriate editor.

In dialogs, the internal frame logic is included in protected code segments which are not visible in the dialog editor. The protected code segments make extensive use of copycode.

The following topics are covered below:

- Customizable Components
 - Generated Code
 - Suggested Code
 - Integrating a Dialog in the Application Shell
-

Customizable Components

The generated application-specific code and the suggested code in dialogs are included in customizable components. These are inline subroutines which are visible in the dialog editor, and which you can modify to meet your own requirements. Help is available in the dialog editor to explain how each customizable component can be used. The section Application Frames explains which customizable components are used by each frame. The section Customizable Components explains each customizable component in detail.

When you add application-specific code or modify the generated code, it is important to understand how the internal frame logic works and how you can control its behavior.

The following topics are covered below:

- Commands
- Frame Logic Control Variables
- Skeleton Objects

Commands

The processing in dialogs is triggered by events. These events are handled initially by the internal frame logic which translates them into commands. The standard commands are described in section Standard Commands.

The frame control logic calls different customizable components depending on the command being processed. See section Application Frames for a list of the components activated when a command is processed and the sequence in which these components are activated.

The frame control logic is controlled by numerous variables, which have a default setting, as described in section Application Frames. By changing these variables, it is possible to modify the internal behavior of the frame control logic contained in the protected code segments of each frame.

Frame Logic Control Variables

You can influence how the internal frame logic works by setting frame logic control variables. Some of these variables are set in the generated code included in customizable components and you can change their settings. Additional variables have defaults and are not explicitly set in the generated code. However you can add code to change the default (usually in the initialize subroutine). See section Application Frames for a list of variables, their usage and default settings.

Reusable Components

When you customize modules produced using the frame gallery, you can call various reusable components from the generated code. The reusable components are subroutines, subprograms, and other objects). These components are described in the section Reusable Components.

Skeleton Objects

Additional skeleton objects are provided to create some advanced components, for example in list box processing.

To implement a module

1. Copy the skeleton object into a new object.
Do **not** modify the skeleton itself.
2. Fill in the suggested code (data definition and coding) for the new object.

Generated Code

Prototype frames contain virtually no generated code.

Production frames include the basic processing required for simple data maintenance functions based on the DDM and fields selected by the user when creating the object view and dialog. It includes:

- a basic dialog layout;
- inclusion of references to the LDAs, PDAs and subroutines associated with the selected object view;
- logic to handle database access and transfer of data between the user interface and the database; and
- logic handling some standard navigation links between dialogs (for example, the linking of a key dialog with a maintain dialog).

This generated code is adequate for very simple functions. It is normally necessary to make changes to the generated dialog for the following reasons:

- to improve the dialog layout;
- to add code to handle requirements for data validation, improve handling of numeric fields, etc.; and
- to link dialogs, for instance to link a maintain dialog with an associated subdialog.

Suggested Code

The suggested code can be regarded as providing a model for the implementation of functions of medium complexity.

Suggested code statements are contained in the customizable components as comment lines. They consist of syntactically correct Natural statements. Values which require customizing, for example context-specific variables, are in lower case.

- Naming Conventions in the Suggested Code
- Skeleton Data Definitions

Naming Conventions in the Suggested Code

To assist in the use of SCAN/REPLACE in suggested code, certain conventions have been followed:

Placeholder	Description
view	indicates view names
xxx	indicates a view or area prefix
#xxx	prefix for user-defined variables

There are also other placeholders that you must modify manually. For example, "field1" must be replaced by a field name.

Depending on program-specific requirements, you can also add new code.

Skeleton Data Definitions

In prototype frames, there is suggested code for data definitions, for example placeholder local data areas or parameter data areas for view fields that are present in the form:

```
xxxAS00A /* Single-object PDA
```

You must replace the placeholder "xxx" with the appropriate view prefix.

Integrating a Dialog in the Application Shell

If you are using the frame gallery, dialogs are automatically integrated into the application shell. In some cases, however, you must manually integrate the dialog into the application shell.

To integrate a dialog into the application shell

1. Define a new object type (command Object type, action New) in the application shell.
2. Define a new function (command Function, action New) in the application shell.
Depending on the dialog type you create, we suggest you use the following function IDs, where "xxx" is the prefix you defined for the object view:
 - xxx_BRW (browse dialog)
 - xxx_MNT (maintain dialog)
 - xxx_DEL (delete subprogram)
 - xxx_MASS (mass processing dialog)
 - xxx_NST (nonstandard dialog)
3. Select the menu command Options and choose Refresh Initialized Data.
Once the function is defined, it can be called via the "Direct Call" dialog.
To integrate the new function into the graphical navigation, see the Natural Application Shell documentation.