3 ARCHIVER

Overview

The VisualDSP++ archiver[*], `elfar.exe`, combines object files[†] from the assembler into archive (library) files, which can serve as a reusable resource for code development. The linker can rapidly search the archive files for routines (archive members) referred to in other objects and link these routines into your executable program. You can run the archiver from a command line, or produce an archive file as the output of a VisualDSP++ project.

This chapter contains the following information on the archiver:

- "Archiver Guide" on page 3-2: introduces the archiver's functions
- "Archiver Command-Line Reference" on page 3-6: reference information on archiver operations
- "Archiver Glossary" on page 3-10: a glossary of archiver related terms

---

[*] Also called "librarian."

[†] The archiver is general-purpose function; it can combine (and extract) arbitrary files. This manual refers to DSP object files because they are relevant to DSP code development.

Archiver Guide

The `elfar.exe` combines and indexes object (or any other) files, producing a searchable archive file. All software developers using the archiver should be familiar with the following operations:

- "Creating Archives in VisualDSP++ Environment" on page 3-2
- "Archiver Operations" on page 3-3
- "Making Archived Functions Usable" on page 3-4
- "Archiver Command-Line Reference" on page 3-6

Creating Archives in VisualDSP++ Environment

Within the VisualDSP++ IDDE, you can choose to create an archive file as your project's output. You can do it in the **Project Options** dialog box that appears when you create a new project, or click **Project Options** in an existing project. In the **Project Type** field, select **DSP Archive file**.

VisualDSP++ writes its output to `<projectname>.DLB`. To modify or list the contents of an archive file, or perform any other operations on it, you must use the archiver from the command line.

```
Syntax: elfar [-switch][-i filename] archive_file object_file ..
```

Refer the *VisualDSP++ User's Guide for ADSP-21xxx DSPs* for more information. Refer to "Archiver Command-Line Syntax" on page 3-6 for more information on command-line input.

Archiver Operations

The archiver can perform the following operations, as directed by options on its command line (or from VisualDSP++ IDDE):

- Append one or more object files to an existing archive file.
- Create an archive file from a list of object files.
- Delete file(s) from an archive file.
- Extract file(s) from an archive file.
- List the filename contents of an existing archive file (to `stdout`).
- Replace file(s) in an existing archive file.

The archiver can only run one of these operations at a time. However, for action switches (see in "Archiver Command-Line Switches" on page 3-8) that can take a list of files as arguments, it can input a file containing the (whitespace-separated) names of object files, which makes long lists easily manageable.

Filename Conventions

To maintain consistency within your code, it is recommended that you use the conventions in Table 3-1. (Note that VisualDSP++ always writes out `<`*`projectname`*`>.DLB` when it creates an archive.)

Table 3-1. File Name Extension Conventions

Extension	File Description
`.dlb`	Archive file
`.doj`	Object file - input to archiver
`.txt`	Archiver `-i` switch input (list file)

Making Archived Functions Usable

In order to use the archiver effectively, you must know how to write archive files and how to write code that accesses these archives. Archive usage consists of two tasks, namely:

- Writing *archive routines*, functions that can be called from other programs.
- Using archive routines. i.e. accessing archived functions in your code.

Writing Archive Routines: Creating Entry Points

An archive routine is a routine in your project that can be accessed by other programs. Each such routine must have a globally visible start label (*entry point*). Any code accessing that routine must know the entry point's name and declare it as an external variable in the calling code.

To create these entry points, use the following steps:

1. Declare the start label of each routine as a global symbol with the assembler's `.GLOBAL` directive. That is the entry point.

```
...
.global dIriir;
.var/dm/seg=seg_dmda FAE[2];
.init FAE[2]: 0x1234,0x4321;
.global FAE;
dIriir: CNTR=N-2;
I2 = ^FAE;
```

2. Assemble and archive the code containing these routines. You can do so in two ways:
   - Direct the VisualDSP++ IDDE to produce an archive. When you build the project, your object code containing the entry points is packaged in `<`*`projectname`*`>.DLB`.

Note: You can extract the object (`.DOJ`) whenever you want, for example, to incorporate it in another project.

- If you create executable or unlinked object files from the IDDE, you can archive them afterward from the command line. The result is the same.

Using Archive Routines

Programs that call an archive routine must define the routine's start label as an external label with the assembler's `.EXTERN` directive.

When linking the program, you specify the archive file (`.DLB`) to the linker with the names of the object files to link. The linker searches the library file to resolve symbols and links the appropriate routines into the executable file. Any file containing a label referenced by your program is linked into the executable output file.

The advantage of linking archives over the individual object files is that the linker can search archives faster, and you do not need to enter all the file names, just the archive name. In the following example, the archiver creates the `filter.dlb` archive, containing the object files: `taps.doj`, `coeffs.doj`, and `go_input.doj`:

```
elfar -c filter.dlb taps.doj coeffs.doj go_input.doj
```

If you then ran the linker with the following command line, the linker links the object files `main.doj`, `sum.doj`, and `graph.doj`; uses the default linker description file, `adsp-ts001m.ldf`; and creates the executable file (`main.dxe`):

```
linker -Dadsp-ts001 main.doj sum.doj graph.doj filter.dlb
```

Assuming that one or more library routines from `filter.dlb` are called from one or more of the object files, the linker searches the archive, extracts the required routines, and links the routines into the executable.

Archiver Command-Line Reference

The archiver (`elfar.exe`) processes object files into an archive file, which results in faster linking of programs that use these archive objects. The archiver's output is an archive file with the file name extension `.DLB`.

This section provides reference information on the archiver command line and linking. A list of all switches and a description for each switch appears in "Archiver Command-Line Switches" on page 3-8.

When using the archiver within the VisualDSP++ environment, the archiver can only produce an archiver as the output for a project. To use other archiver features, you must use the command line version of the archiver.

Archiver Command-Line Syntax

Use the following syntax for the archiver command line. Table 3-2 describes each switch.

```
elfar -[a|c|d|e|p|r][-v][-i filename] archive_file object_file...
```

This command line is subject to the following constraints:

- You can select only one action switch (`a`, `c`, `d`, `e`, `p`,`r`).
- The verbose operation switch `-v`, must not be in a position where it can be mistaken for an object file, meaning it cannot follow the *`archive_file`* on append or create.
- The file include switch, `-i`, must immediately precede the include file name.
- Use the archive filename first, followed by switches. The `-i` and `-v` are not action switches, and can appear later.

- Enclose long file names and file names containing whitespace or special characters within double quotation marks.
- Append the appropriate file name extension to each file.
- Do not use wildcards. To perform an archive operation on a list of member files, write the list into a text file and use it as input to the command line (with the `-i` switch).
- Specify *`archive_file`* to add, remove or replace the *`object_file.doj`* object file.

  Note that the archiver's `-i` switch lets you input a list of members from a text file, instead of listing all the members on the command line. Also note that when you use the archiver's `-p` switch, you do not need to identify any members on the command line.

Items shown in `[ ]` are optional. Items shown in *`italics`* are user-defined and are described with each switch.

The archiver's command line is case-insensitive. For example, the following command line

```
elfar -v -c my_lib.dlb fft.doj sin.doj cos.doj tan.doj
```

runs the archiver as follows:

`-v` — Selects verbose mode for the archiver

`-c my_lib.dlb` — Creates an archive file named `my_lib.dlb`

`fft.doj sin.doj cos.doj tan.doj` — Puts the named object files in the `my_lib.dlb` archive file

Table 3-1 on page 3-3 lists the relevant types of files and extensions that the archiver takes as parameters.

Archiver File Search

File searches are important in the archiver's process. The archiver supports relative and absolute directory names, default directories, and user-selected directories for file search paths. File searches occur as follows:

1. *`Default directory`*—If you do not include path information in the file name, the archiver searches for the file in the current project directory.
2. *`Specified path`*—If you include relative or absolute path information in a file name, the archiver only searches in that location for the file.

Archiver Command-Line Switches

Table 3-2 describes each archiver switch and corresponding parameter/attribute. The switches must appear before the *`archive_file`* name on the command line, except that the `-i` switch appears in place of the *`object_file`*. Items shown in *`italics`* are user-defined and are described with each switch.

Table 3-2. Archiver Command-Line Switches

Switch	Description
`archive_file`	The archive (`.DLB`) that the archiver modifies. This parameter appears after the switches.
`object_file`	One or more object files (`.DOJ`), also called *`member_files`*, that the archiver uses when modifying the archive. This parameter appears after *`archive_file`*. You can use the `-i` switch to input the object file names as a list.
`-a`	Append one or more *`object_file(s)`* to the named *`archive_file`*.

Table 3-2. Archiver Command-Line Switches (Cont'd)

Switch	Description
`-c`	Create a new *`archive_file`* containing *`object_files`* on the command line.
`-d`	Delete one or more *`object_files`* from the selected *`archive_file`*.
`-i <`*`filename`*`>`	Use a file (*`filename`*), containing member-file names, as an input. This file lists *`object_files`* to add or modify in the selected *`archive_file`* (`.DLB`).
`-p`	The `-p` (print archive contents) switch directs the archiver to print to standard output a list of *`object_files`* in the selected *`archive_file`* (`.DLB`).
`-r`	Replace the named file(s) in the library.
`-v`	The `-v` (verbose archiver messages) switch directs the archiver to output status information, as the archiver processes your files.
`-M`	`-M` (produce dependencies) prints dependencies to stdout. Used only with `-c`.
`-MM`	`-MM` (build and produce dependencies) prints dependencies to stdout and builds an archive. Used only with `-c`.

Archiver Glossary

Archives—contain one or more members. Archives are searched by the linker for routines that you call from your programs.

Entry point | Start label—is the label in the archive member that begins the routine that you call from your program. Entry points are global labels in the archive member and are external labels in your program.

Members—of an archive are object files that are processed into the archive by the archiver.