NM6403 Software Development Kit

# NeuroMatrix® NM6403 Programmer's Reference

**Version 1.0**

**30002-01 33 02 A**

JOINT-STOCK COMPANY
RESEARCH CENTRE

# Contents

# Figures

# Tables

# Preface

The preface describes the purpose and composition of the document, gives a short summary of the sections and chapters, and determines the style and symbolic notations used in this document.

**Note**

*This reference guide does not contain any information of the processor design. To get the data on the structure of NM6403 refer to other sources, particularly to the document of the «NeuroMatrix ®NM6403 SDK. Assembly Language Overview».*

### *About This Manual*

This Guide contains the following information:

- composition of NeuroMatrix® NM6403 Software Development Kit (SDK);
- reference information on each component of the SDK;
- calling conventions;
- descriptions of file formats used in the DSK.

### *Organization*

This manual is divided into chapters. Each chapter describes one of components included in the SDK. It contains reference information about each SDK component used for making the executable code for NM6403. For example: purpose, description manual, conventions on file extensions, command line options, additional utilities for viewing object and executable files, calling convention.

### *Typographical Conventions*

This reference guide uses the following typographical conventions:

| | |
|---|---|
| `Courier` | Denotes text that may be entered at the keyboard: commands, file and program names, and assembler and C++ source. This is most often used in syntax descriptions. |
| `-help` | It is the same as in previous case. Underlined portion of the word indicates that a short notation can be used in command line instead of a long one when calling a |

| | |
|---|---|
| | program. Instead of `-help` the short `-h` can be used, and this abbreviation will not cause the change in the program behavior. |
| *`Courier`* | Marks the text that should be replaced by user information, e.g. by a path to a particular SDK component. |
| **Text** | This way that the text is marked requires special attention. |

**Note** *Boxes like this contain information on significant notes and comments to the context.*

### *Conventions on File Names*

When selecting file names one should keep to a notation adopted in MS-DOS, i.e. no more than 8 symbols is allocated for a name and no more than 3 for an extension. The extension should be separated by a point. All files should have extensions.

**Note** *Information on what particular extensions should be used for particular types of files is presented in chapters describing corresponding components using these extensions.*

Example of a file name corresponding to above conventions:

```
d:\mydir\my_file.cpp
```

### *SDK Components*

There is a list of NM6403 SDK components and a list of its filenames, which are described in this manual.

| | | |
|---|---|---|
| `nmcc` | Compiler C++ (`nmcc` shell) | Chapter 2 |
| `asm` | Assembler | Chapter 3 |
| `linker` | Linker | Chapter 4 |
| `libr` | Object Files Librarian | Chapter 5 |
| `dump` | Decoder of Object and Executable Files | Chapter 6 |
| `emurun` | Instruction Level Simulator | Chapter 7 |
| `temu` | Accurate Cycle | Chapter 8 |

Simulator

SDK contains C run-time library that is presented both as a source code and as object library. In the second case it is named libc.lib and located in LIB directory.

## *Feedback*

### Feedback on This Manual

If you have feedback on this manual, please contact your supplier, giving:

- the manual's title;
- the manual's document number;
- the page number(s) to which your comments refer;
- a concise explanation of the comment.

General suggestions for additions and improvements are also welcome.

### Feedback on NeuroMatrix® NM6403 Software Development Kit

If you have comments or suggestions about the NeuroMatrix® NM6403 Software Development Kit, please contact your supplier, giving:

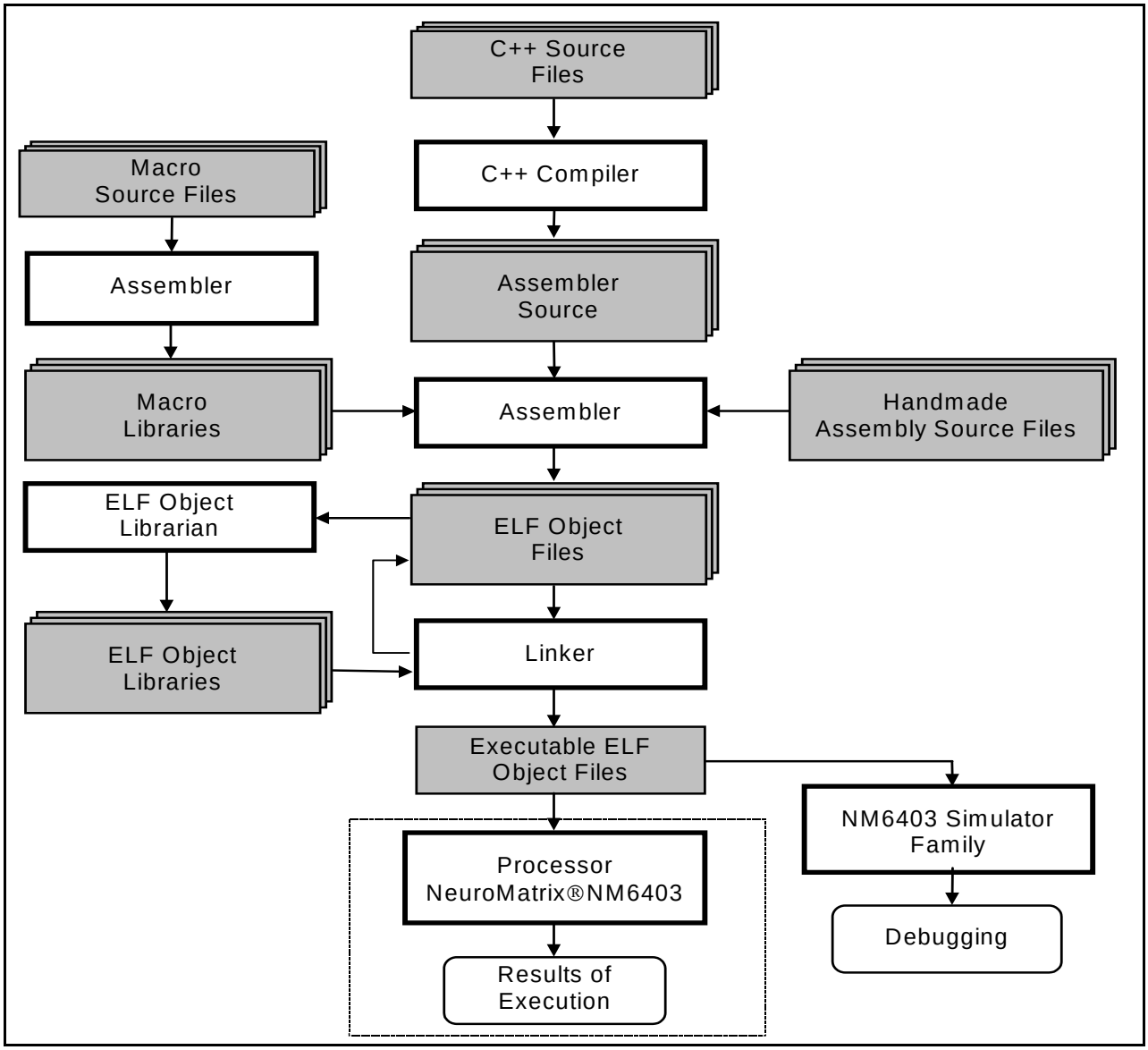
- the platform and release of the NeuroMatrix® NM6403 software tools you are using;
- a small sample code fragment which illustrates your comment;
- precise description of your comment or suggestion.

# 1 Software Development Kit Overview

### 1.1 NM6403 Software Development Flow

Figure 1-1 shows the C++ and assembly language development flow.

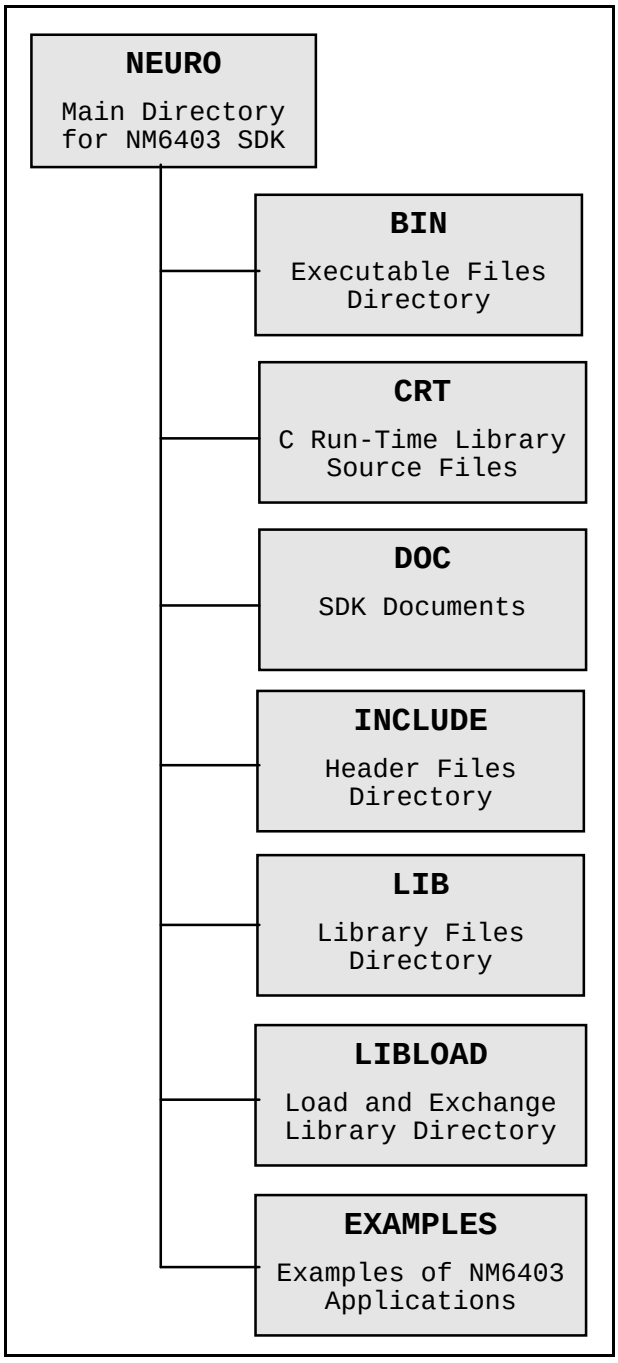
*Figure 1-1. NM6403 Software Development Flow*




### 1.2 Structure of SDK Directories

For convenience of work with NeuroMatrix® NM6403 SDK a predetermined structure of directories is used. It is described by the following directory tree:

*Figure 1-2. NM6403 SDK Directory Tree*




The content of the directories depends on the version of NM6403 SDK supplied. It is presented in the document on the SDK installation.

### *1.3 NEURO Environment Variable*

To enable SDK components to automatically find all necessary libraries and header files in the course of compiling and assembling application programs `NEURO` environment variable is introduced.

The environment variable is placed into file `autoexec.bat` in the form of the following line:

`NEURO=<path_to_the_main_SDK_directory>`

for example:

`SET NEURO=D:\NEURO`

Apart from pre-setting the environment variable to file autoexec.bat section PATH should be complemented with path to catalogue BIN, since it contains the NM6403 SDK executable programs:

`PATH=%PATH%;D:\NEURO\BIN`

Those modifications of `autoexec.bat` are made automatically at the stage of installation. The user can also make those changes later, but they must be done before start using SDK.

# 2 C++ Compiler

## 2.1 Introduction

This chapter contains information about NM6403 C++ compiler. It contains the data about the C++ language version supported by the compiler, short information of the compiler composition, options of compiler control and gives an example of compiling a simple program.

**Note** *This reference guide cannot be used as an introductory course in programming in C++ language; neither it contains reference information on C++ language.*

## 2.2 About NM6403 C++ Compiler

The NM6403 C++ Compiler is a full-featured compiler that translates C/C++ programs into NM6403 assembly language source. The following list describes key characteristics of the compiler:

- NM6403 C++ Compiler supports definition of C++ language described in draft standard ANSI X3J16/95-0029 with the exception some kind of templates.
- The compiler is designed as a Windows 95/98/NT console application and is controlled by command line options.
- The compiler package comes with the runtime library `libc.lib`. The source code of the contents of the library is available in the `CRT` directory of the installed NM6403 SDK. The library includes functions for time-keeping, dynamic memory allocation, data conversion, and floating-point arithmetic.
- The compiler supports a flat memory model. There are no any restrictions on object code size. Memory space is limited only by configuration of hardware.
- The compiler package includes a shell program, which enables the user to execute all steps of program translation into NM6403 executable code with one command.
- The compiler has straightforward calling conventions, allowing the user to easily write assembly and C/C++ functions that call each other.
- Executable and Linkable File (ELF) format allows the user to define system's memory map at a link time. This maximizes performance by enabling the user to link C/C++ code and data into specific memory areas.
- Debugging Information format DWARF provides rich support for source-level debugging.

- Data sizes of `char`, `short`, `int`, `float` types are 32 bits; data sizes of `long` and `double` types are 64 bits.

### *2.3 Getting Started With the Compiler*

The NM6403 C++ compiler produces a single assembly language source file that must be assembled and linked. The simplest way to compile, assemble, and link a C++ program is to use the `nmcc` shell program, which is included with the compiler. This section provides a quick walkthrough so that the user can get started without reading the entire programmer's guide.

1) Create a simple file called `MyApp.cpp` that contains the following code:

```
/************************************************/
/*                  MyApp.cpp                   */
/*        (Simple file for walkthrough)         */
/************************************************/

#include <time.h>


clock_t t0;

int main()

{

    t0 = clock();
    if (t0 < 1000) return t0;
    else           return -1;
}
```

2) Invoke `nmcc` to run the compiler, assembler and linker:

```
nmcc MyApp.cpp
```

The shell program runs compiler, assembler and linker as follows:

`preproc` → Preprocessor
`c0` → Front-end compilator
`codegen` → Code Generator
`asm` → Assembler
`linker` → Linker

In case no errors were found, the shell prints nothing.
By default, `nmcc` deletes the assembly language output file after translation is finished. Use `-Stemp` option to retain assembly language file and other temporary files:

```
nmcc -Stemp MyApp.cpp
```

3) By default, `nmcc` creates an ELF object files (`.elf`) and ELF absolute executable file (`.abs`). The absolute executable file may be

run on an instruction level simulator:

```
emurun MyApp.abs
```

The instruction level simulator executes the program and prints the returned value, for example:

```
MyApp.abs: : WARNING: return 21 = 0x15
```

4) To compile the program with debug information `-g` option is used.

```
nmcc -g MyApp.cpp
```

5) Invoke `emudbg` to debug the program step-by-step on the NM6403 Source Level Debugger. Load `MyApp.abs` from disk into a disassembler window. To debug the program on a source level select the menu item **VIEW|SOURCE|MyApp.cpp**.

For more information about invoking the C++ compiler and the `nmcc` shell program, refer to Section 2.4 on page 2-3.

## *2.4 Compiling C++ Code*

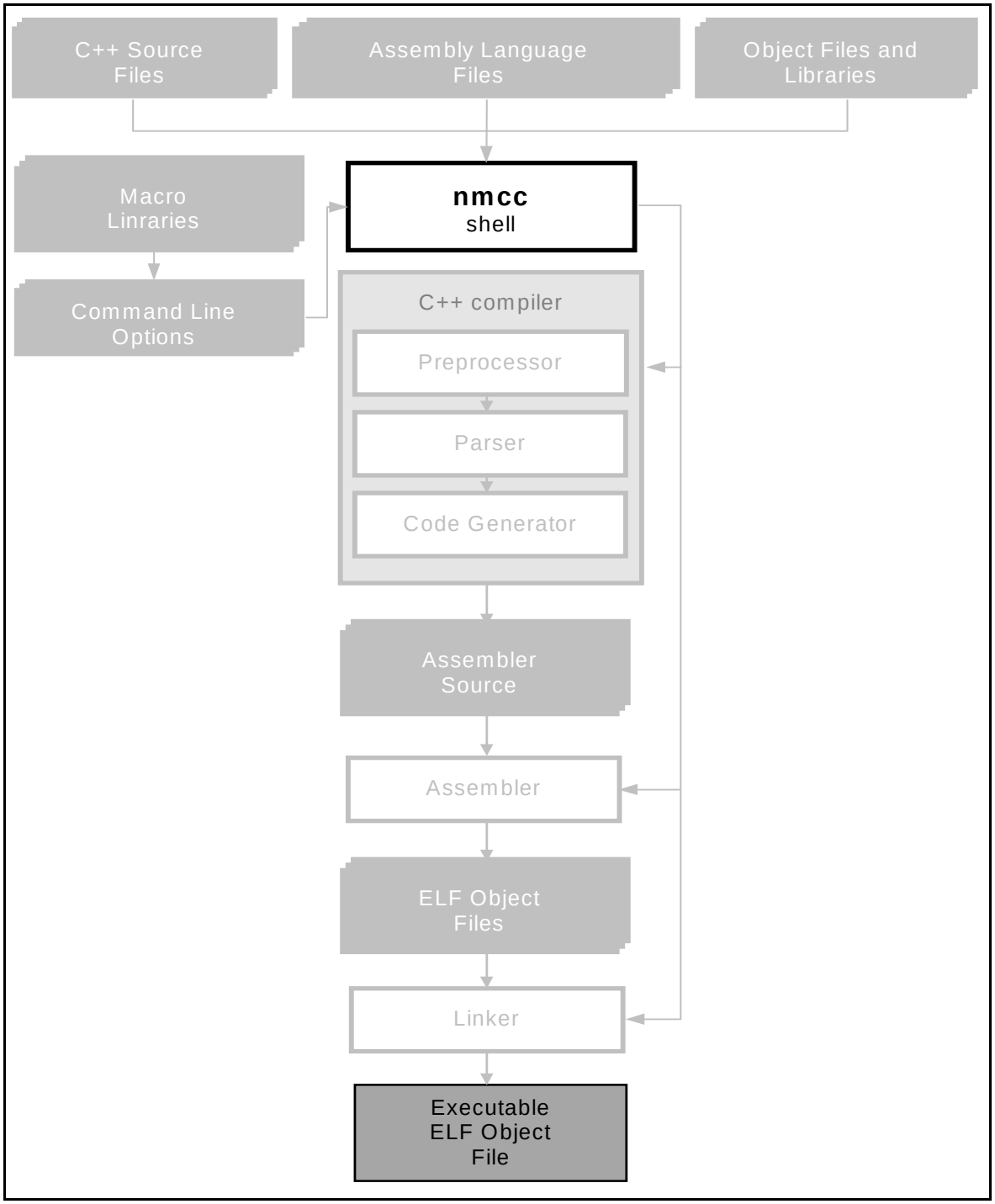
NeuroMatrix® NM6403 SDK contains a special utility, called `nmcc` shell program, which enables the user to execute all steps of program translation into NM6403 executable code with one command. This paragraph provides a complete description of how to use `nmcc` to compile, assemble and link the user programs.

The `nmcc` shell runs one or more source modules through the following:

- The **compiler** includes the preprocessor, front-end compilator and code generator.
- The **assembler** generates ELF object file.
- The **linker** links object files and object libraries to create an executable object file. It can be omitted (with `-Snolink` option) if the user needs to get only object files, for instance, to build a library.

By default, nmcc compiles, assembles and links files. Figure 2-1 illustrates the path `nmcc` shell takes.

*Figure 2-1. The nmcc Shell Overview*



## 2.5 Invoking the C++ Compiler

| **nmcc** [*-options*] [*filenames*] |
|---|

| | |
|---|---|
| **nmcc** | is the command that invokes the compiler, the assembler and the linker. |
| *options* | affect the way the compiler processes input files. |
| *filenames* | are one or more C/C++ source files, assembly source files, object files, object libraries, macro libraries. |

The options control the way the compiler processes files. The filenames provide a method of identifying source files and output files. Options and filenames can be specified in any order on the command line.

Depending on a file extension to be processed `nmcc` invokes the appropriate SDK component. The specified filenames are listed below.

## 2.6 Specifying Filenames

The NM6403 SDK specifies the following extensions for designating file types:

*Table 2-1. File Extensions Used in NM6403 SDK*

| EXTENSION | FILE TYPE |
|---|---|
| .cpp, .c | C/C++ source file. |
| .hpp, .h | C/C++ header file. |
| .asm | Assembly source file. |
| .elf,.elz | Object file. |
| .abs | Absolute executable file. |
| .rel | Relocatable executable file. |
| .lib | Object library. |
| .mlb | Macro library. |
| .lst | Listing of assembly source file. |

| | |
|---|---|
| .map | Memory map file. |

Files without extensions are rejected by `nmcc`. The conventions for filename extensions allow the user to compile C/C++ files, assemble assembly files, link object files with a single command.

## *2.7 Compiler Options*

Command line options control the operation of both `nmcc` and the programs it calls. This section provides a description of option conventions, an input summary table, and a details description of each of the options.

Options are single letters or multi-letter words, are **case sensitive**, and are preceded by a hyphen. Each option must be separated from others. Options can be specified in any order. Options that have parameters, such as `-o`*`outfilename`*, should not be separated from them. If the parameter is separated from the option with a space, the option is ignored by `nmcc`; the separated parameter is treated as an input file.

Some default paths to header files and libraries can be set up by using the NEURO environment variable. It specifies the path to the INCLUDE directory and to the LIB directory that contains C runtime library. For a detailed description of the NEURO environment variable, refer to section 1.3 on page 1-3.

*Table 2-2. The Compiler General Options*

| OPTIONS | DESCRIPTION |
|---|---|
| `-help`,<br>`-h` | Print out all options. |

*Table 2-3. The nmcc Shell Options*

| OPTIONS | DESCRIPTION |
|---|---|
| `-Stmp`,<br>`-Skeeptemps` | Do not remove intermediate files. |
| `-Snoexec` | Instead of invoking the compiler, the assembler and the linker `nmcc` prints out all information about the components invoking conditions, a list of command line options used to compile, assemble and link each source file of the user's program, and expanded paths. |
| `-Snolink` | Do not invoke the linker. |
| `-Ssyntax` | Invoke only the preprocessor and the front-end compilator to check syntax of a C++ source file. |

*Table 2-4. The nmcc Options Carried to the SDK Components*

| OPTIONS | DESCRIPTION |
|---|---|
| `-g` | Enable symbolic debugging. |
| `-I<dir>` | Add *dir* to #include search path. |
| `-L<lib_dir>` | Add *lib_dir* to libraries search path. |
| `-X<option>` | Set the options of the front-end compilator. |
| `-D<symbol_name>` | Predefine *symbol_name*. |
| `-U<symbol_name>` | Undefine *symbol_name*. |
| `-T` | Distinguish identifiers of C++ programs by first eight symbols. |
| `-C` | Do not remove initial comments when compiling C++ files. |
| `-l<filename>` | Create an assembly listing file. |
| `-x<filename>` | Create a cross-references file. |
| `-o<filename>` | Name the output file. |
| `-c<filename>` | Supply memory configuration file name. |
| `-m<filename>` | Name the map file. |
| `-@<filename>` | Supply command file for the linker. |
| `-abs` | Generate absolute file (the default option). |
| `-rel` | Generate relocable file. |
| `-elf` | Generate object file. |
| `-heap=XXXX` | Set memory heap size on a local bus to XXXX words. |
| `-heap1=XXXX` | Set memory heap size on a global bus to XXXX words. |
| `-stack=XXXX` | Set stack size to XXXX words. |
| `-start` | Define entry point. |
| `-addr=XXXX` | Set address of a segment (can be used when the memory configuration file is not supplied). |

Detailed description of the component options can be found in the correspondent chapters.

### 2.7.1 Delivery of Reference Information (-help or -? Option)

When invoking nmcc with the only -help or -? option, help information is printed out. It gives the user short reference on every option controlling nmcc behaviors. The following information will appear on the screen:

*Figure 2-2. The nmcc Reference Information*

```
NeuroMatrix(r) NM6403 C++ Compiler v1.5.4

List of available options:
Miscellaneous options:
   -help,-h,-?          print out this help.
   -Skeeptemps[=<>]     do not delete the temporary files
                        [and specify temp directory]
                        (synonym '-Stmp'),
   -Snoexec             do not execute constructed commands
                        but rather print it to stdout,
   -Snolink             only compile to object files,
   -Ssyntax             check for syntax errors only
                        (do not create any files).
Code Generation options:
   -g                   generate and keep debug info,
   -O                   turn on optimization,
   -B                   disable NM6403 silicon bugs correction,
Preprocessor options:
   -I<>                 specify include directory,
   -L<>                 specify libraries path,
   -X<>                 set frontend option,
   -D<>                 define macro,
   -U<>                 undefine macro,
   -T                   differ identifiers by 8 first symbols,
   -C                   do not delete comments,
Output files:
   -l<>                 specify assembler listing file,
   -x<>                 specify assembler cross-reference file,
   -o<>                 specify output file name,
   -c<>                 specify linker configuration file,
   -m<>                 generate map file,
   -@<>                 specify linker command file.
Explanation of the other permitted options:
 -heap -heap1 -stack -start -addr -asm -abs -rel -elf -rom,
see in the linker documentation.
```

After displaying this message nmcc operation is completed.

The same reference information is printed out when starting nmcc without command line options at all.

### 2.7.2 Service Options (prefix -S)

All `nmcc` service options begin with `-S` prefix.

#### 2.7.2.1 Keeping Intermediate Files (-S*keeptemps* and -S*tmp*)

This option is used to keep intermediate files generated by the C++ compiler components. The options `-Stmp` and `-Skeeptemps` are synonyms. The intermediate files are as follows:

- `.cc` - files resulted from the preprocessor operation;
- `.ic` - files resulting from the operation of the front-end compiler;
- `.asm` - files generated by the code generator from files .ic.

By default, when a given option is not set, `nmcc` deletes all intermediate files of types listed above.

Object files .elf as well as executable files are **never** removed by `nmcc`.

**Note** *When compiling, only those* .asm *files are removed that were created as a result of the compilation of files* .cpp. *Coincidence of the names of files without extension in this case serves as a criterion; for example, when compiling file* prog.cpp *files* prog.cc, prog.ic, prog.asm *will be created. In the absence of parameter* –Stmp *they will be removed.*

#### 2.7.2.2 Printing Out Expanded Invoking Conditions (- S*noexec* Option)

The -Snoexec option instructs `nmcc` that **instead** of actual invoking the package components, it should **print out** the list of expanded component starting options.

This option can be helpful when the user wants to know what particular command line options were delivered to the input of each of the components being started.

Example:

```
nmcc -Snoexec MyApp.cpp

preproc -F -DNM6403 "-IC:/NEURO/include" MyApp.cpp C:/WINDOWS/TEMP/MyApp.cc
c0 -o C:/WINDOWS/TEMP/MyApp.ic C:/WINDOWS/TEMP/MyApp.cc
codegen -q C:/WINDOWS/TEMP/MyApp.ic -oC:/WINDOWS/TEMP/MyApp.asm
asm -q "-IC:/NEURO/include" C:/WINDOWS/TEMP/MyApp.asm -oMyApp.elf
linker -q "-lC:/NEURO/lib" MyApp.elf libc.lib
```

#### 2.7.2.3 Disabling Linker (- S*nolink* Option)

The `-Snolink` option makes possible to disable the phase of object modules linking. In this case, compilation stops after object files were generated.

#### 2.7.2.4 Checking C++ Source Syntax (-S*syntax* Option)

The `-Ssyntax` option disables invoking of all the components of the C++ compiler except the preprocessor and the front-end compilator.

### 2.7.3 C++ Compiler Options

The most options of `nmcc` are actually the options for one or several SDK components. Detailed description of SDK component options is presented in corresponding sections of this manual.

Further follows short description of the C++ Compiler options. There is brief description for each option, the format and the list of components in whose command line this option may be found.

#### 2.7.3.1 Creating Debug Information (-g Option)

The `-g` option causes the compiler components to generate symbolic directives for use with the NeuroMatrix® NM6403 C++ source debugger.

It is used without additional arguments.

The `-g` option is used by:

- the front-end compilator;
- the code generator;
- the assembler.

When `nmcc` meets the `-g` option, it automatically passes -d1 to the linker. For more information about the `–d1` option, refer to Section 4.8.2 on page 4-16.

#### 2.7.3.2 Adding Directories for Header Files and Libraries Search (-I and -L Options)

The `-I`*`dir`* option adds *`dir`* to the list of directories to be searched for #include files. The options can be used several times to define several directories. The `-I`*`dir`* option is used by:

- the preprocessor;
- the assembler.

It allows the components to search the following:

- C++ header files;
- Macro libraries.

The directory name should not be separated from the option, for example:

-ID:\NEURO\INCLUDE

The `-L`*`dir`* option adds *`dir`* to the list of directories to be searched for libraries. The options can be used several times to define several directories. The nmcc shell passes the `-L`*`dir`* option to the linker.

The directory name should not be separated from the option, for example:

-ID:\NEURO\LIB

### 2.7.3.3 Front-end Compilator Options (-X*argument* Options)

The `-X` allows the user to set various options for the front-end compilator.

*Table 2-5. The Front-end Compilator Options*

| PARAMETERS | DESCRIPTION |
|---|---|
| `-Xinline=`*n* | This front-end compilator option defines the processing mode of inline functions defined by the user. If `n` is equal to `1` the front-end compilator will build in the code for those functions instead of function call. Otherwise, the calls of inline functions will be processed as usual calls.<br>Default value: -Xinline=1. |
| `-Xexcep=n` | This front-end compilator option defines the C++ exception generation mode. If `n` is equal to `1` the front-end compilator supports exceptions. Otherwise, the code of exception support is not generated. The use of this key makes sense only in case the user program does not use the exceptions. In this case, additional code for stack unwinding will not be generated. If `n` is equal to `0` but the C++ source code contains the *throw*-expression the compiler will return an error message.<br>Default value: -Xexcep=1. |
| `-Xrtti=n` | This option tells the front-end compilator to support mechanism of runtime type information (RTTI). If `n` is equal to `1` the front-end compilator generates such structures. Otherwise, no data for the RTTI support is generated. The use of this option makes sense only in case when the user program uses RTTI in C++ programs. It should be noted that not all expressions `typeid` and `dynamic_cast` require the generation of RTTI data. If `n` is equal to `0` but the C++ source code contains the RTTI data the compiler will return an error message.<br>Default value: -Xrtti=0. |
| `-Xold=n` | This option tells the front-end compilator to support some old regulations of ANSI C, canceled by ANSI C++. If `n` is equal to `1` the old restrictions are supported. This option permits implicit transformation from arithmetic types to enumerators (`enum` types) and an implicit transformation of `void` to an arbitrary type. The option is helpful while compiling the ANSI C code |

| | |
|---|---|
| | with the compiler.<br>Default value: `-Xold = 0`. |

#### 2.7.3.4 Preprocessor Options (-D, -U, -T, -C Options)

*Table 2-6. The Preprocessor Options*

| PARAMETERS | DESCRIPTION |
|---|---|
| `-D`*`name`*`[=`*`def`*`]` | This option predefines *`name`* for the preprocessor. This is equivalent to inserting `#define` *`name def`* at the top of each C++ source file. If optional *`def`* is omitted, `-D`*`name`* equal to 1. Examples:<br>`-DDEBUG`<br>`-UDEBUG`<br>`-DVER=100` |
| `-U`*`name`* | This option undefines the predefined constant *`name`*. Overrides any `–D` option for *`name`*. |
| `-T` | This option sets the C++ identifier identification mode. If the option is set the identifiers names are compared by the first eight symbols. Otherwise, the names are compared by all symbols. The option does not require any additional arguments. |
| `-C` | This option allows preserving comments when processing C++ source files by the preprocessor. It does not require any additional arguments. |

### 2.7.4 Assembler Options

This subsection contains short description of some assembler options. For more information about the assembler options, refer to Chapter 3.

#### 2.7.4.1 Generating an Assembly Listing File (-l Option)

The `–l` (lowcase “L”) option invokes the assembler with `–l` option to produce an assembly listing file. If the option is set up, the file **.lst** will be produced for every assembly source file, listed in `nmcc` command line, and for every assembly source file generated by the C++ compiler.

#### 2.7.4.2 Generating a Cross-Reference Listing File (-x Option)

The `–x` option invokes the assembler with –x option to produce a symbolic cross-reference in the listing file. If the option is set up, the file **.crf** will be produced for every assembly source file, listed in `nmcc` command line, and for every assembly source file generated by the C++ compiler.

### 2.7.5 Linker Options

This subsection short description of some linker options. For more information about the linker options, refer to Chapter 4.

#### 2.7.5.1 Defining Output File Name (-o Option)

The `–o`*`filename.ext`* option names the output file. If the user does not want to employ the `–o` option, the linker creates an output file with the name of the first file met in the command line. For example, in case

```
nmcc MyApp.cpp Filter.asm Mask.elf
```

the output file name will be `MyApp.abs`. If the order is changed, the out file name will be changed, too. For example, in case

```
nmcc Filter.asm MyApp.cpp Mask.elf
```

the output file name will be `Filter.abs`.

If the user defines the output file name, the order is ignored, for example:

```
nmcc MyApp.cpp Filter.asm Mask.elf –oMask.abs
```

In this case, the output file name will be `Mask.abs`.

If output file extension is not specified it will be defined by the linker and will depend on the output file type.

*Table 2-7. Default Extensions for Linker Output Files*

| EXTENSION | DESCRIPTION |
|---|---|
| `.abs` | Absolute executable files. |
| `.rel` | Relocatable executable files. |
| `.elz` | Object files. |

#### 2.7.5.2 Supplying a Memory Configuration File Name (-c Option)

The `–c`*`filename`* option provides the linker with a memory configuration file name.

For more information about this option, refer to Section 4.11 on page 4-25.

#### 2.7.5.3 Generating a Memory Map File (-m Option)

The `–m`*`filename`* option tells the linker to generate a memory configuration file.

For more information about this option, refer to Section 4.8.9 on page 4-20.

#### 2.7.5.4 Supplying a Linker Command File Name (- @ Option)

A command file is an alternative technique of setting the linker options and input filenames. The command file can be used to set up default or common options and filenames. Using the command file is especially

convenient when nmcc is run consecutive times with the same set of options or/and input files.

The name of the command file should follow immediately after the -@ option:

*-@filename*

The preceded hyphen is set only in case -@ is the nmcc shell option. The same linker option is used without the hyphen. For more information, refer to Section 4.6.6 on page 4-12.

## *2.8 The nmcc Default Configuration*

### 2.8.1 List of Components Default Options

The following default options and/or filenames are set for the correspondent compiler components every time the user runs nmcc:

- -F (exclude /usr/include path from the #include search path) and -DNM6403 (define NM6403 macro, which defines the target processor). These are the default options for the preprocessor,
- -q (suppress progress messages) for the code generator, assembler and for the linker,
- -I%NEURO%\include for the preprocessor and for the assembler. The pathnames are directories that contain #include files and assembler macro libraries,
- -l%NEURO%\lib for the linker. The pathnames are directories that contain object libraries,
- libc.lib – C runtime library containing the processor startup code. This file is added to a command line if at least one C++ source file name is specified. In case nmcc creates an output file from only assembly source files, the library is not added.

### 2.8.2 Default Output File Name

In case when compilation and the assembly of an application task consisting of several files are realized, and in this case the name of the output file is not indicated, the following default convention is operative:

The name of the output file coincides with the name of the first in list file being compiled, and the extension corresponds to the type of the output file, e.g. the result of successful compilation:

```
nmcc aaa.cpp bbb.cpp ccc.cpp
```

will be file aaa.abs.

The default result of the compilation is the absolute executable file (extension .abs).

The linker determines all conventions described in this point and used by default since this is it that completes the compilation stage. For more detailed information about default values, refer to Section 2.8 on page 2-14.

### *2.9 Example of Invoking nmcc*

This example demonstrates the process of assembling the absolute file from two initial files:

```
>nmcc t2.cpp templ.cpp -otempl.abs
"templ.cpp", line 5: (warning): return type for ‘main’ to integer type
>
```

File `t2.cpp` is the first one in the command line. However, the output file with the aid of key `-o` is named after the main file of the project: `templ.abs`. The library of execution time C++ is used for assembling the output file, the library is connected automatically on condition that the variable of environment NEURO is set.

The warning was issued by the front-end compilator since the type of function `main` in file `templ.cpp` was described as `void` that does not conform to the requirements of the new standard of C++ language.

It is possible to see what components were invoked by `nmcc`:

```
> nmcc t2.cpp templ.cpp -otempl.abs -Snoexec

preproc -F -DNM6403 "-IC:/NEURO/include" t2.cpp C:/WINDOWS/TEMP/t2.cc
c0 -o C:/WINDOWS/TEMP/t2.ic C:/WINDOWS/TEMP/t2.cc
preproc -F -DNM6403 "-IC:/NEURO/include" templ.cpp C:/WINDOWS/TEMP/templ.cc
c0 -o C:/WINDOWS/TEMP/templ.ic C:/WINDOWS/TEMP/templ.cc
codegen -q C:/WINDOWS/TEMP/t2.ic -oC:/WINDOWS/TEMP/t2.asm
codegen -q C:/WINDOWS/TEMP/templ.ic -oC:/WINDOWS/TEMP/templ.asm
asm -q "-IC:/NEURO/include" C:/WINDOWS/TEMP/t2.asm -ot2.elf
asm -q "-IC:/NEURO/include" C:/WINDOWS/TEMP/templ.asm -otempl.elf
linker -q -otempl.abs "-lC:/NEURO/lib" t2.elf templ.elf libc.lib
```

It is obvious that the `nmcc` has made a substitute for the `NEURO` variable environment; standard files of headers (`.h`) and standard macro-libraries are located on the disc in catalogues substituted automatically. It is also obvious what intermediate files are created in the course of components operation.

`t2.cc t2.ic t2.asm templ.cc templ.ic templ.asm`

Since the parameter `-Stmp` was not found in the command line, the `nmcc` shell has removed all these files on completion of the work. Only object and absolute files remained (the linker creates absolute executable files by default):

```
t2.elf, templ.elf and templ.abs
```

### *2.10 The nmcc Shell Error Messages*

During the operation of `nmcc` erroneous situations may arise. If an error appeared in the course of the work of some component, the component itself will issue the message of this error. In this case `nmcc` ceases further work and completes with the result 1.

Error message may be generated by `nmcc` itself.

The `nmcc` shell messages are, as a rule, the result of an error in the parameters of its call. All `nmcc` messages start with prefix `" cc :"`

The `nmcc` shell generates the following messages:

"Unrecognised option '<used_name>'"

The `nmcc` shell was called with a key unknown to it (all `nmcc` options are preceded by hyphen).

"Unrecognised input file <used_name>'"

The `nmcc` shell was called with a file type unknown to it (known types of files:`.c .cpp .asm .elf .elz .lib`).

"Multiple source files forbids implicit specification of list file name."

The name of listing file was indicated simultaneously with the indication of some initial files `.c`, `.cpp` and `.asm`. When starting with many initial files it is forbidden to use key `-l` with a name indicated. However, the `-l` option may be used without an additional argument, in this case for each assembler file (initial or created as a result of compilation of file C++) the listing file with a name is created, the name formed from its name by replacing the extension by `.lst`.

"Multiple source files forbids implicit specification of cross-ref file name."

The reason is the same as with the previous error. `-x` can be used without an additional argument. Extension `.crf`. is used for the files of cross-references created in this case.

"No such environment variable: <used_name>"

A non-existing environment variable was used in the command line of the `nmcc` call. Used name of the variable is displayed.

"Unexpected environment variable substitution error"

An unclear error has occurred while substituting the values of the environment variables. Under the normal operation of the `nmcc` and at normal condition of the system this error cannot appear. In case of its appearance, one should apply to programmers with a detailed description of conditions at which this error has occurred.

## *2.11 Characteristics of NM6403 C++*

This section contains information about the aspects of compiler realization and the system library of supporting C++ language that have been defined in the standard of C++ language, as defined by realization.

### 2.11.1 Standard Data Types

By virtue of the architecture of processor NM6403 and of the set of supported commands of memory access all standard elements of the data are expressed in 32- or 64-bit words depending on the number of bits required for their presentation.

In the processor a 32-bits word is least accessible, therefore, regardless of the fact that type char or short can be presented by 8 or 16 bits respectively, they occupy 32 bits in the memory.

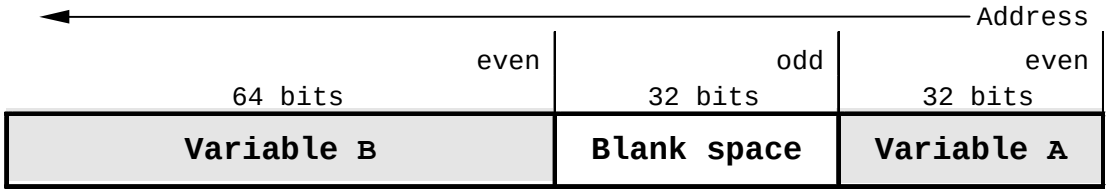
Base types of the data of C++ language are presented in the Table, as well as their size in 32-bits words and the alignment of the data type in the memory:

*Table 2-8. NM6403 C++ Data Size*

| BASE TYPES OF DATA | SIZE | QUANTITY OF SIGNIFICANT BITS | ALIGNMENT OF DATA TYPE IN MEMORY |
|---|---|---|---|
| `char` | `32 bits` | `8 bits` | `1` |
| `short` | `32 bits` | `16 bits` | `1` |
| `int` | `32 bits` | `32 bits` | `1` |
| `long` | `64 bits` | `64 bits` | `2` |
| `indicators` | `32 bits` | `32 bits` | `1` |
| `float` | `32 bits` | `32 bits` | `1` |
| `double` | `64 bits` | `64 bits` | `2` |
| `long double` | `64 bits` | `64 bits` | `2` |

Term «Alignment of data type in memory» is used to emphasize those variables occupying 64 bits cannot be arranged by memory odd addresses (see Figure 2-3). Therefore, if two variables are located alongside each other, one of which is 32 bits in size and lies in the memory at an even address and the variable following it is 64 bits, an unused space is located between these two variables which are 32 bits in size.

*Figure 2-3. Alignment of Data Types in Memory*




### 2.11.2 Identifiers and Character Set

An identifier may have any length, however compiler C++ ignores symbols after the 256$^{th}$ (the standard requires not less than 31 symbols). All the symbols up to 256$^{th}$ one are significant.

Set of symbols found in the initial program is essentially a 7-bit ASCII set. In the comments all symbols from the 8-bit ASCII Table can be found.

Uppercase and lowercase letters are distinguished for internal and external identifiers.

### 2.11.3 Data Types Range (limits.h and float.h)

Two header files are defined in the standard of ANSI C++ language: `limits.h` and `float.h`, that describe the ranges of meanings of constants defined by the data types.

The standard also defines the minimum size and availability/absence of a sign bit for the data types being described.

The minimum size of a variable which is not a bit field, is equal to 32 bits:

```
CHAR_BIT 32
```

The maximum number of bytes in a multibyte symbol:

```
MB_LEN_MAX 1
```

For the following integer-valued numeric ranges the medium column contains the numerical presentation of a limit being defined and the right column reflects its bit presentation in a hexadecimal format.

| SYMBOLIC DESIGNATION | LIMIT VALUE | BIT PRESENTATION |
|---|---|---|
| CHAR_MAX | 255 | 0xFF |

| | | |
|---|---|---|
| CHAR_MIN | 0 | 0x00 |
| SCHAR_MAX | 127 | 0x7F |
| SCHAR_MIN | -128 | 0x80 |
| UCHAR_MAX | 255 | 0xFF |
| SHRT_MAX | 32767 | 0x7FFF |
| SHRT_MIN | -32768 | 0x8000 |
| USHRT_MAX | 65535 | 0xFFFF |
| INT_MAX | 2147483647 | 0x7FFFFFFF |
| INT_MIN | -2147483648 | 0x80000000 |
| LONG_MAX | 9223372036854775807 | 0x7FFFFFFFFFFFFFFF |
| LONG_MIN | –9223372036854775808 | 0x8000000000000000 |
| ULONG_MIN | 184467440737709551616 | 0xFFFFFFFFFFFFFFFF |

# 3 Assembler

## 3.1 Introduction

This chapter contains information about NM6403 assembler. It contains the data about the interface of the compiler, command line options, modes of operation and about assembler error and warning messages.

## 3.2 About Assembler

Assembler for NM6403 - is a one-pass compiler translating a program written in assembly language into an ELF object file. The assembler does not resolve internal references and does not resolve undefined external symbols. Its main task is to construct a Table of symbols and to conduct transformation of assembler lines into processor instructions.

The assembler allows to create the libraries of macros and to add new macros into already existing libraries.

## 3.3 Assembler Development Flow

The assembler processes files in assembler language received from three sources:

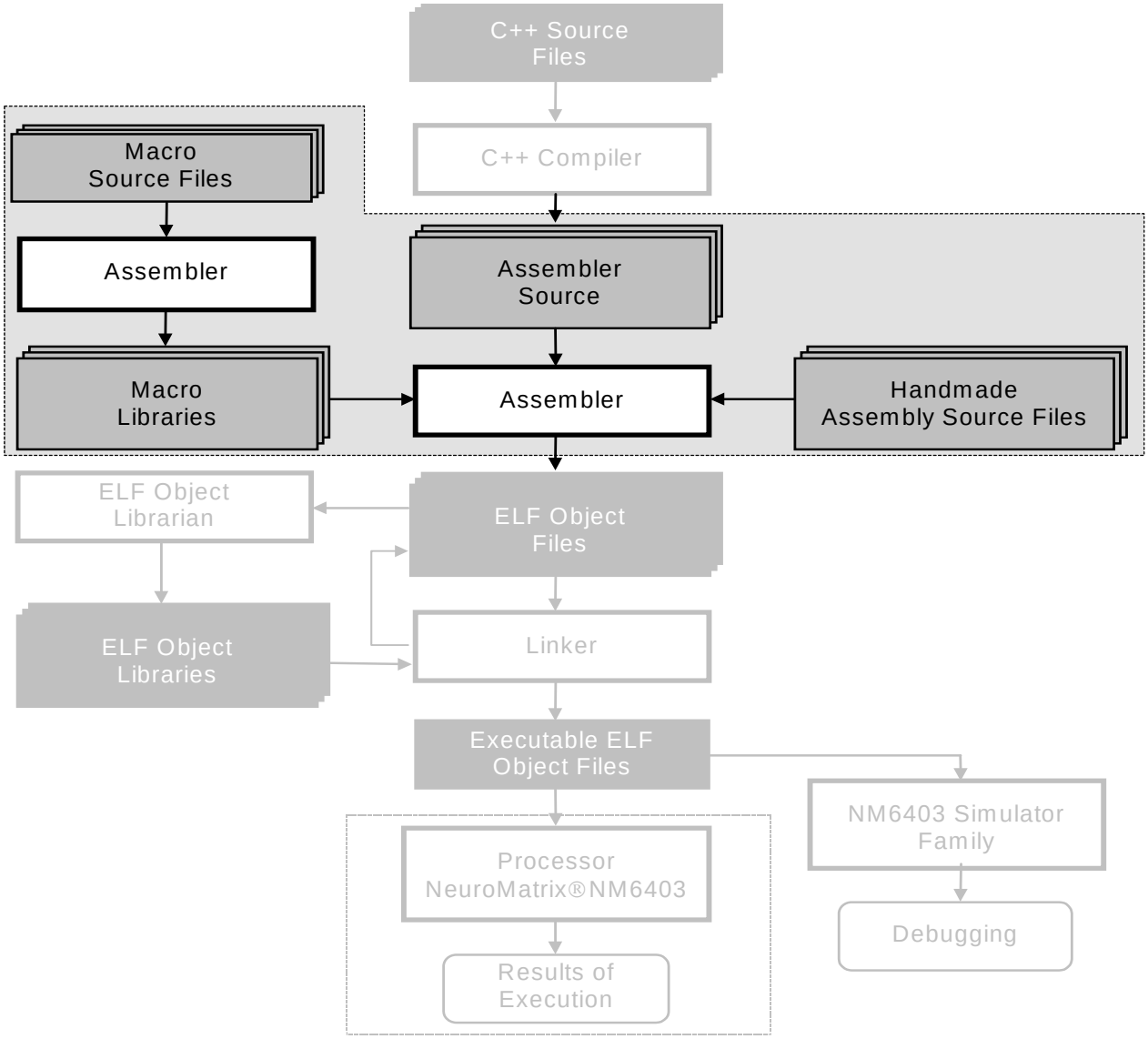
- files generated by compiler C++;
- files in assembler language developed manually;
- macro libraries generated by the assembler itself during the preceding sessions of operation.

A result of assembler operation can be either an object file or a macro library.

The above input and output data of the assembler define its place in the structure of NM6403 SDK.

Position of the assembler within the general path of obtaining a program being executed from various types of input data is not shown.

*Figure 3-1. Assembly Language Development Flow*




### *3.4 Invoking the Assembler*

**asm** [*options*] [ *input file* ]

| | |
|---|---|
| **asm** | Name of file containing assembler executable code for NM6403. |
| *input file* | Input file containing the program in assembler language. |
| *options* | Assembler parameters (with prefix «-»). They can be arranged in an arbitrary place of a command line and in arbitrary sequence. (More details are discussed below). |

Non-compulsory parameters are put into square brackets.

If there is no input file indicated or no parameter is set, the assembler will not realize any actions but will only display the message about its own name, version and copyright.

In the end of the work, the assembler, being a return code, returns the total quantity of errors in the text of the initial program, if any, or 0 in case of their absence.

### *3.5 Assembler Options Summary*

To control the assembler operation a set of parameters (keys) is used.

All parameters are preceded with prefix «-».

They should be indicated in a command line.

The sequence of parameter arrangement is of no importance. Parameters are divided among themselves with spaces.

Presented below are the Tables of the assembler parameters.

*Table 3-1. General Options*

| PARAMETERS | DESCRIPTION |
|---|---|
| `-q` | The assembler does not issue any messages on the work process except error messages |
| `-i` | The assembler does not issue any messages on its work |
| `-h or -?` | The assembler displays short information on call parameters. |
| `-t` | The assembler displays its title defining its full name, version and copyright data. |
| `-p` | At the beginning of its work the assembler displays its full name containing the path to a catalogue where it is located. |
| `-I`*`macrolib`* | Indicates a catalogue where the macros libraries should be searched. |

Parameters -p, -t -h, -? are reference parameters and therefore cannot be used simultaneously. If they appear at the same time the assembler issues an error and informs that only one of the parameters can be used as an input one.

For example, if the -t option is set up:

asm -t

the output message is displayed as follows:

```
Assembler for NM6403 1.32.(c)RC Module 1996-1999. All rights reserved
```

Whereas with the simultaneous appearance of any other parameter alongside it in the command line

For example, when calling the assembler with parameters:

asm -t -h, or asm -t -q

the display will show:

Invoke error ASM702: -p,-h,-? or -t must stand alone

*Table 3-2. Output Filenames*

| PARAMETERS | DESCRIPTION |
|---|---|
| -o*filename* | Name of the output file of a component is preset. If an option is not preset, the output file name coincides with the input file name. If the option is preset, the output file receives the name: *filename*. |
| `-l` | Generates file with the program listing. |
| `-x` | Creates file with a list of cross-references. |

*Table 3-3. Macro Librarian Mode Options*

| PARAMETERS | DESCRIPTION |
|---|---|
| `-m[`*`macrolib`*`]` | With the appearance of this parameter in the command line, the assembler jumps to the macro-librarian mode. Name *macrolib* indicates a library name. By default, it coincides with the input file name that is complemented with extension `.mlb`. |
| `-a` | This parameter provides the addition of macros contained in the input file to the macro-library whose name is defined with a previous key. |

**Note** *In case among the keys of the command line supplied to the assembler input there are parameters of transition to the macro-librarian mode, no object file is generated.*

*Table 3-4. Handling Output Messages*

| PARAMETERS | DESCRIPTION |
|---|---|
| `-W[+\|-]`*`num`* | This parameter enables/disables the display of an individual message by its number. |
| `-W[+\|-]`*`group`* | This parameter, combined with the abbreviation defining various groups of messages, enables prohibition / permission to display the messages of the preset group. |

### *3.6 General Options*

Related to general-purpose parameters are those that have the same meaning and execute similar actions for the most of the SDK components of processor NM6403.

#### 3.6.1 Printing Out Reference Information (-h, -? Options)

When starting the assembler with parameter -h, or -? reference information is displayed on all the keys defining the assembler behavior and the result of its work. The display will show the following:

*Figure 3-2. The Assembler Reference Information*

```
rogram usage: asm [-[i|q]plxda] [-I<>] [-o<>] [-m<>] [-W<>] in_file

in_file       - input asm file name

ommon switches:
-q             - disable program header message.
-i             - force to produce no messages (include -q switch).
-I<macrodir>   - set path for the macro libraries.
-p             - print out assembler location.
-t             - print out assembler header.
-h,-?          - print out this help page.
Each of the -p,-t,-h,-? switch can't be combined with other switches.

utput production switches:
-o<out_file>   - specify output file name;
                 if omitted, out_file assigns to <in_file> with extension
                 changed to '.elf'.
         -l             - generate listing file named <out_file> with extension
                          changed to '.lst'.
         -x             - generate cross-reference file named <out_file> with
                          extension changed to '.crf'.
         Output switches can't be used with macro liblarian switches.

        Macro librarian switches:
         -m[<macrolib>] - turn on librarian mode;
                          if <macrolib> omitted, filename.mlb will be created.
                          No elf file will be generated.
         -a             - append to library. Library file must exist.
         Macro liblarian switches can't be used with output switches.

        Debug switches:
         -d             - turn on verbose syntax parsing

        Warning control switches:
         -W[+|-]<num>   - enable/disable warning number <num>;
                          if sign omitted, warning is disabled.
         -W[+|-]<group> - enable/disable entire <group> of warnings;
                          legal groups are
                              all       - all warnings,
                              debug     - warnings issued by the debug commands,
                              object    - object generating warnings (e.g. unused label),
                              compile   - all translation warnings,
                              librarian - librarian warnings.
```

With the appearance of `-h` or `-?` in the command line the assembler work is completed, input file is not created.

### 3.6.2 Disabling Output Information (-q and -i Options)

Depending on parameter the assembler produces information in the following way:

- In case of absence of keys -q or -i the assembler delivers to a user all the information generated, namely, title header, warnings, error messages.
- In case of -q the user only obtains information on errors and warnings. This mode is often used during a package start of the assembler or at calling from the integrated environment.
- In case of -i the assembler produces nothing, i.e. in this mode of assembler operation even error messages are suppressed.

### 3.6.3 Printing Out the Banner (-t Option)

Parameter -t is of exceptionally informative nature. It is used for getting information on the name of a given SDK component, on version number and on copyright. With the appearance of this key in the command line, the assembler issues an information message:

```
Assembler for NM6403 1.32.(c)RC Module 1996-1999. All rights reserved
```

and completes the work. The assembler work is completed without creating an output file.

### 3.6.4 Displaying the Assembler Pathname (-p Option)

Parameter -p is of purely informative nature. It is used for searching the location of an assembler called. Often a situation arises that a user does not know where a program is called from. It is only obvious that this is one of catalogues from the list of general-access catalogues in file autoexec.bat (`PATH=...`). In this situation, key `-p` can be used. The assembler, when faced by this key in the command line, will send the following message to the display:

`D:\NEURO\BIN\ASM.EXE,`

and complete the work. No output file is created.

## *3.7 Output File Types*

Depending on keys used as input parameters the assembler generates the following types of output parameters:

- `.elf` - object file;
- `.lst` - file-listing of initial text;
- `.crf` - file with a list of cross-references;
- `.mlb` - macro-library.

### 3.7.1 Setting the Output File (-o*filename* Option)

The result of the assembler work in translation mode is an output object file. Its name is defined by means of setting a key `-ofilename` in the command line. Between key `-o` and the file name there should be no space. The file name stands for a full name i.e. the full path to the file. If only a name is preset and the path is omitted the file will be created in the current catalogue. If a file with this name existed already in this directory, it will be replaced by a new one without any additional warnings.

If the assembler is not encountered by a key defining the name of an output file, the file name is used with the addition of extension `.elf`, and this becomes the name of the output file.

### 3.7.2 Creating an Assembly Listing File (-l Option)

With the appearance of key `-l` among the input parameters of the assembler an output file is generated that contains the program listing. It has extension `.lst`.

The listing file is created additionally to the object file and it keeps reference information on the following:

- shift relative to segment beginning;
- code image of each command;
- lines of the program initial text.

A small fragment of the program listing is given as an illustration:

*Figure 3-3. Fragment of an Assembly Listing File*

```
                          * begin ".text"
                          * <_Mirror>
                          *
00000000 4e000000 ffffffff * nb1 = 0FFFFFFFFh; // Neuron boundaries
00000002 4e400000 ffffffff * sb = 0FFFFFFFFh; // Sinaps boundaries
                          *
00000004 3fe40000          * push ar4, gr4;
00000005 3fe10000          * push ar1, gr1;
00000006 3fe20000          * push ar2, gr2;
                          *
00000007 50100000
00000008 47d40000 fffffff8 * ar4 = ar7 - 8;
```

| offset | machine code | assembly source code |
|---|---|---|

### 3.7.3 Creating a Cross-References File (-x Option)

When adding parameter `-x` to the assembler calling command line a file with cross-references is formed. It has extension `.crf` and is generated additionally to the object file.

The cross-reference file contains the list of symbols and their definitions in the format as follows:

- symbol name;
- its meaning;
- line number in the file with initial text where it is defined;
- line numbers of the initial text where references to this symbol are available.

For illustration, we herein present a small fragment of a cross-reference file.

*Figure 3-4. Fragment of a Cross-Reference File*

```
****** List of sections ******
Section Name              Size
 .values                  200
 .bss.values              0
 .text                    70
****** List of objects ******
Object Name     Type    Bind     Defined in    Offset
_Mirror         LABEL   Global   .text         0
Matrix0         VAR     Local    .values       0
Matrix          VAR     Local    .values       100
```

## *3.8 Macro Librarian Mode*

Apart from its main function of translating assembly source files into object files, the assembler allows to operate in the mode of a macro-librarian. In this case, instead of an object file it generates a macro-library where all the macros found in the input assembler program are collected.

### 3.8.1 How to Use the Assembler as the Macro Librarian (-m[*macrolib*] Option)

Parameter `-m` transfers the assembler to the macro-librarian mode. In this case, the input file undergoes a syntax check-up, this followed by transferring macros contained in it to a special intermediate presentation and are entered to the output file.

Parameter `-m` can be used both with an additional parameter and without it. The additional parameter contains the file name where the macro-library will be written. There should be no space between `-m` and file name.

If the file name is absent, a file whose name coincides with that of the input assembler file and the extension is `-.mlb`, will be used as a macro-library.

If parameter -m is used without additional parameter `-a` (see below), a new file is created. If a file with this name already exists it will be replaced by the new one without any additional warning messages.

### 3.8.2 Adding Macros to a Macro Library (-a Option)

Parameter `–a` is used to show that macros found in the input file should be added to the already existing macro-library.

Thus, if only key `-m`, is used, the macro-library is created anew regardless of the fact if a file with this name had existed. If a pair of keys `-m -a` is used, the macros will be added to the existing library.

**Note** *In case a combination of keys -m -a, is used but the macro-library in this case does not exist, an error will be displayed. This is done to avoid generation of erroneous libraries. Normally, user, when using key -a, bears in mind that the library is kept on the disc. If it is not found, it is most likely that a path was indicated wrongly.*

## *3.9 Controlling the Assembler Warning Messages*

The assembler has developed possibilities of control of warnings. All warnings are divided into several groups related to various nodes of operation or types of data processed. The division by groups is shown below in Subsection 3.9.2 on page 3-11.

### 3.9.1 Controlling Warnings Output by Their Numbers (-W[+|-]<num> Option)

Parameter `-W` that is followed by a warning number provides enabling/disabling a warning with a preset number.

If sign "+" is set, a warning, in case of its appearance, is displayed.

With sign "`-`" set, a warning is ignored and not displayed.

For example, there is an error in the program in the assembler language, this error probably does not lead to wrong program operation:

```
aaa: word = 8080808080808080h;
```

The error consists in that a 64-bit constant is assigned to a variable declared as a 32-bit word.

When compiling an assembler file containing a given line a message will be displayed as follows:

```
Assembler for NM6403 v1.32. (c) RC Module Inc. 1996-1999. All rights reserved.
"mirror.asm",19 : Warning ASM003: Initializer too big, truncated
```

The same will happen if parameter `-W+3` will be added to the command line. However, if parameter `-W-3` will be used as an input parameter the message will be ignored and will not appear on the screen.

By default, all warnings are in active state (+) except for a warning with number `ASM200`, for it `-W-200` is set.

### 3.9.2 Controlling Group of Warnings (-W[+|-]*group* Option)

Parameter `-W` in combination with abbreviation defining various groups of messages allows to control the display of messages.

All warnings displayed in the course of assembler work can be divided into groups as follows:

- `all` - all warnings of assembler;
- `debug` - all warnings generated when processing debugging information;
- `object` - all warnings generated when creating assembler objects such as unused marks;
- compile - warnings arising in the course of code generation;
- librarian - warnings generated by the assembler in the macro-librarian operation mode.

The description relating to types of messages making particular groups is presented in subsection 3.10.1 on page 3-12.

Example:

All the warnings but one can be disabled in the following way:

```
asm.exe mytest.asm -W-<all> -W+3
```

**Note** *Actions of W-parameters depend on priorities of their appearance in the command line, i.e. a parameter next in turn cancels a previous one. For example, sequence* `-W-<all> -W+3` *disables all the warnings except the third one whereas sequence* `-W+3 -W-<all>` *disables all warnings including number 3 warning with as well.*

## *3.10 Error Messages*

In the course of operation the assembler may issue three types of messages:

- warnings;
- errors;
- internal and fatal errors.

With the appearance of warnings the work of the assembler is not ceased since the problems arising in this situation still make possible to generate an object code, though, probably, operating wrongly.

In case of appearance of any type of error no output file is generated since either there is no possibility for code generation, or the generated code will be known to be wrong and impossible for implementation. Errors are divided into usual ones, connected with wrong program writing, and fatal, when a program, even free of errors, cannot be compiled because, for instance, a file with macro-libraries is not available, or there is a shortage of disc space.

To send a message to the display the assembler uses a formatted line. Its format is unified for the entire set of the SDK and looks as follows:

**"file_name", [line_number] error_type  error_number: error_message.**

Line number is not indicated for internal and fatal errors.

Three figures are allocated for error number, i.e. it lies within an interval between 0 and 999. Among error types numbers are distributed in the way as follows:

Table 2-1 Ranges of numbers for various types of messages.

| ERROR NUMBERS | DESCRIPTION |
|---|---|
| 0 – 399 | Warnings. |
| 400 – 799 | Errors. |
| 800 – 949 | Internal errors. |
| 950 - 999 | Fatal errors. |

Presented below is a list of messages issued by the assembler when various types of erroneous situations occur.

### 3.10.1 Warnings

Messages of this type indicate to the probable availability of logical or semantic inaccuracies available in a program processed. The display of this message does not affect the correctness of output files formed by the assembler.

**Compilation messages (group `compile`).**

| | |
|---|---|
| `ASM001` | "Section name in construction end differs from a name when opening :" |
| | A section in the assembler is always put in special brackets. Opening brackets are set by pseudocommands: .begin, .data, |

.nobits. After the opening bracket a section name is located. Pseudocommand .end. is used as a bracket closing the section. After it the section name should be located as well. It should coincide with the name when opening. If section names with opening and closing brackets do not coincide, this warning arises (disciplinary construction). In case of mismatching the closing bracket section name is ignored.

| | |
|---|---|
| ASM002 | "Use pseudofunctions 'float' or 'double'" |
| | Variables of float or double type are presented in the processor memory in IEEE 754 format. The Assembler translates automatically constants of 1.5 или 2.7E-3 type to a required form. However, to provide for a correct translation it is necessary to use pseudofunctions float or double, which, being recognized by the assembler, would be transformed as required. In other words, a correct record of the constants with floating point looks as follows: float(1.5) or double(2.7E-3). Otherwise the constants of this type will be replaced by zero constants. |
| ASM003 | "Value of initialization constant is to great, truncated" |
| | Such a message usually appears when trying to initialize a 32-bits variable by a 64-bit constant. After truncating only the junior portion of the constant remains, the senior one is ignored, for example: 0123456789ABCDEFhl -> 89ABCDEFh. |
| ASM004 | "Value of initialization constant is too small, extended" |
| | This message usually appears when trying to initialize a 64-bits variable by a 32-bit constant. Extension takes place due to zero filling of the constant senior portion, for example : 12345678h -> 0000000012345678hl. |
| ASM005 | "Pseudofunctions loword/hiword cannot be applied to address expressions" |
| | Capacity of address expressions does not exceed 32 bits whereas pseudofunctions loword/hiword are intended for access to the junior/senior portion of a 64-bits word respectively. |
| ASM006 | "Name of a macros in construction end differs from the name when opening :" |

Macros in assembler are always put in special brackets .end. Opening brackets are always set by a pseudocommand: .macro. After the opening bracket macros name is located. Pseudocommand .end is used as a bracket closing the section. It should coincide with the name when opening. If there is no coincidence between the macros names at opening and closing brackets this warning appears (a disciplinary construct). In case of mismatching the macros name at the closing bracket is ignored.

| | |
|---|---|
| ASM100 | "To receive a correct result the result register should not coincide with registers of multiplier and multiplicand" |

For a multiplying operation executed hardwarily a requirement exists according to which the operation result cannot be written neither to a register where the multiplicand was located, nor to a register where the multiplier was contained (see Description of assembler language for NM6403). The message warns that the multiplying will be executed incorrectly.

| | |
|---|---|
| ASM101 | "Multiplying operation, when executing, uses the result register, by default we substitute gr0" |

If the product register at multiplying was not indicated, for example, there is an entry in the assembler program line: gr2*:gr7, the result by default will be saved in register gr0.

| | |
|---|---|
| ASM102 | "Only 1 is permissible, we use 1" |

In assembler language there are some constructs where only «one» can be used, for instance: gr2 = gr0 - 1. A constant cannot participate in scalar or vector arithmetic operations. And a record of gr0 - 1 type is considered not as an operation with a constant but as the unary arithmetic operation of the general-purpose register modification and corresponds to a certain processor instruction. If a different number is erroneously substituted to such construct it is automatically replaced by 1.

| | |
|---|---|
| ASM103 | "Only register gr7 can be a multiplier, we use gr7" |

At multiplying operation executed hardwarily there is a requirement according to which a multiplier should always be in register gr7. If in a command being processed instead of gr7 some other register is located it will be replaced by

gr7.

| | |
|---|---|
| ASM104 | "Only 0 is acceptable, we use 0" |
| | In assembler language there are some constructs where only 0 can be used, for example vector constructs where 0 comes as a zero operand: with 0 + data. In this case zero is a sign that a zero vector is automatically generated in the processor, this vector allows to pass `data` to afifo without changes. If another number is erroneously substituted to such construct, it automatically replaced by 0. |

**Messages of debugging information (group debug).**

| | |
|---|---|
| ASM200 | "The second argument is not used" |
| | This message is disabled by default. It can be issued when analyzing the debugging information of DWARF format, when instead of one required parameter two parameters are supplied to the pseudocommand. In this case the second parameter is ignored. |
| ASM201 | "Index cie should coincide with an ordinal number of a given '.debug_frame_cie'" |
| | The message arises at the mismatching of index cie with the ordinal number of the debugging pseudocommand .debug_frame_cie. |
| ASM202 | "The directory index should coincide with the ordinal number of the debugging pseudocommand '.debug_source_directory'" |
| | The message arises at the mismatching of the catalogue index with the ordinal number of the debugging pseudocommand .debug_source_directory. |
| ASM203 | "The file index should coincide with the ordinal number of command '.debug_source_file'" |
| | The message arises at the mismatching of the file index with the ordinal number of the debugging pseudocommand .debug_source_file. |
| ASM204 | "In this version command '.debug_macro_def' is not |

| | |
|---|---|
| | realized, we skip it" |
| | Support of debugging information for macroses in this SDK version is not realized. |
| ASM205 | "In this version command '.debug_macro_undef' is not realized, we skip it" |
| | Support of debugging information for macroses in this SDK version is not realized. |
| ASM206 | "In this version command'.debug_macro_start_file' is not realized, we skip" |
| | Support of debugging information for macroses in this SDK version is not realized. |
| ASM207 | "In this version command '.debug_macro_end_file' is not realizes, we skip" |
| | Support of debugging information for macroses in this SDK version is not realized. |
| ASM208 | "Attribute identifier should be a number" |
| | Symbolic information is used as an attribute identifier. |
| ASM209 | "Attribute value should be an address" |
| | Wrong type of the debugging pseudocommand parameter. |
| ASM210 | "Abstract command should consist of three elements" |
| | Pseudocommand structure is preset wrongly. |
| ASM211 | "Only one '.debug_root_die' is possible" |
| | More than one pseudo command is found in the program text `.debug_root_die`. The second and subsequent pseudo commands `.debug_root_die` are ignored. |
| ASM212 | "Sequence of commands '.debug_*die*' should start with '.debug_root_die'" |
| | Pseudo command `.debug_root_die`, that defines the beginning of commands sequence `.debug_*die*.` is |

absent.

| | |
|---|---|
| ASM213 | "Ter die should a number" |
| | Ter die is probably defined as a symbolic constant. |
| ASM214 | "The first argument of operation should be an address" |
| | The first argument of pseudocommand is of a type differing from an address. |
| ASM215 | "Both arguments should be numbers" |
| | One or both arguments of the pseudocommand are defined as symbolic constants. |
| ASM216 | "Operation code should be a number" |
| | Symbolic expression is used as an operation code. |
| ASM217 | "Both operation code and two arguments should be numbers" |
| | Symbolic expression is used as an operation and/or argument code. |
| ASM218 | "Wrong number of arguments" |
| | The number of arguments in the line exceeds the required one or falls short of it. |
| ASM219 | "Wrong type of arguments" |
| | Data of wrong type are supplied as an argument. |
| ASM220 | "Sequences '.debug_line' cannot be embeddded" |
| | Pseudooperation sequences describing numbers of initial text lines cannot be embedded. |
| ASM221 | "In the first '.debug_line' index of initial file should be indicated" |
| | The first pseudooperation .debug_line, describing the number of line should store the index of initial file where it |

takes place.

| | |
|---|---|
| ASM222 | "Command '.debug_start_sequence' is omitted." |
| | Pseudocommands of sequence processing are found though the initial pseudocommand .debug_start_sequence was not preset. |

| | |
|---|---|
| ASM223 | "Bytes of the block should be numbers" |
| | Probably, bytes preset by symbolic constants were found in the block structure. |

**Messages of object file composition (group object)**

| | |
|---|---|
| ASM300 | "Unused label" |
| | A label was declared in the program in assembler language, however no reference to it was found, therefore it will be ignored. |

| | |
|---|---|
| ASM301 | "Generation of debugging information is impossible" |
| | This warning appears in place where the first error is found at parsing the debugging information. Starting from this place the generation of the object code will continue, however all debugging information will be ignored. |

**Messages of macro-librarian (group librarian)**

| | |
|---|---|
| ASM350 | "Macros already exists, we skip :" |
| | The macros with this name already exists in the macrolibrary, therefore a newly found macros will be skipped. |

### 3.10.2 Errors

This type of messages is an indication of some error in a file processed. This error does not cause immediate stoppage of the assembler operation but in case of appearance of this category of messages output files are not generated.

| | |
|---|---|
| ASM400 | "Too long constant" |

This error appears in case a declared constant does not go into 64 bits.

| | |
|---|---|
| ASM401 | "Wrong format of decimal number" |

Wrong format of decimal number means that it comprises symbols not belonging to digit series 0-9.

| | |
|---|---|
| ASM402 | "Wrong format of binary number" |

Wrong format of binary number means that it comprises symbols differing from 0 and 1.

| | |
|---|---|
| ASM403 | "Wrong format of octal number" |

Wrong format of octal number means that it comprises symbols not belonging to digit series 0-7.

| | |
|---|---|
| ASM404 | "Wrong format of hexadecimal number" |

Wrong format of decimal number means that it comprises symbols not belonging to digit series 0-9, and not corresponding to letters A, B, C, D, E, F.

| | |
|---|---|
| ASM405 | "Impermissible symbol" |

Impermissible symbol in assembler language implies a symbol not belonging to the following subset of symbols: lowercase and uppercase Latin letters, decimal numbers and also symbols as follows:

. , / \ : ; " ’ ` [ ] - + = & ! < > ( ) * { } _.

| | |
|---|---|
| ASM406 | "Impermissible condition" |

An impermissible condition in assembler language means the following:

if after symbols u> without space there is any other symbol except =. That is, there should be u>=.

if after symbols <> without space there is any other symbol except 0. That is, there should be: <>0.

| | |
|---|---|
| ASM407 | "After quoting operation there should be an identifier" |

Symbol '``' (reverse1 apostrophe) can be used in assembler

language only in quoting operations, i.e. when a user wants to use an auxiliary word as an identifier; it must be necessarily preceded by symbol '`'. If there is a space between a given symbol and identifier or the identifier is absent, this error occurs.

| ASM408 | "Line terminators do not coincide" |
|---|---|

If a line is put in double quotation-marks the assembler is right to expect that since a sign of opening quotation-marks " was found, closing double quotation-marks should appear before the end of the line. If the closing quotation-marks are not found or single quotation-marks are found instead of them, this error occurs.

| ASM409 | "Wrong declaration of a variable" |
|---|---|

This error arises if a variable is declared outside some program section and at the same time it is not a variable of common or extern type.

| ASM410 | "Wrong initialization" |
|---|---|

This error occurs when the assembler discovers an attempt of initializing a variable of common or extern type. Variables of these types cannot be initialized by virtue of their nature.

| ASM411 | "Wrong type of variable - WORD is supposed" |
|---|---|

.

| ASM412 | "Wrong declaration of structure" |
|---|---|

This error arises when the name of the structure defined alongside an opening bracket `struct` does not coincide with the name at a closing bracket `end`, for instance:

```
struct AAA
  A: word;
  B: long;
end AA;
```

| ASM413 | "Structure field is not found" |
|---|---|

This error arises when trying to appeal to the non-existing field of a structure used, for example:

ar0 = offset(AAA, C);,

though structure AAA only has fields A and B.

| | |
|---|---|
| ASM415 | "Wrong use of word OWN" |

This error appears when using key word «own» outside the macros body that is impermissible since this word is used for labels declared only inside the macros.

| | |
|---|---|
| ASM416 | "Wrong name of library" |

This error occurs in the case when after key word «import» the name of macro-library is not preset or the name of non-existing library is preset.

| | |
|---|---|
| ASM417 | "Wrong size of array" |

The reason for this error is wrong determination of the array size when it is defined as equal to zero or to a negative number.

| | |
|---|---|
| ASM418 | "Wrong counter in duplicator" |

This error occurs when zero or a negative number is an argument of a duplicator dup defining the quantity of pattern repetitions, for instance: 80808080h dup -10.

| | |
|---|---|
| ASM419 | "Wrong array index" |

This error arises when trying to get access to the outside of the array borders. The error can be found only at an obvious setting of the array element index. For example, if the access to the array element is preset in a marked line with index -1 or 100 and, at the same time, correct values of the indexes are within the range from 0 to 99.

| | |
|---|---|
| ASM422 | "Wrong register pair" |
| | When a register pair is used, for instance, for reading a 64-bits word from the memory, the address register and the general-purpose register should have similar indexes ar0, gr0 or [ar3+=gr3]. |
| ASM423 | "Wrong constant expression" |
| | Regulation of composing a constant expression is violated. The assembler is unable to calculate it, consequently, it issues an error. |
| ASM424 | "Using '.endif' without '.if'" |
| | Pseudocommands '.endif' and '.if' should always be found in pairs '. Absence of '.if' with '.endif' available causes erroneous situation. |
| ASM425 | "Recursive use of '.repeat'" |
| | The assembler does not tolerate embedded pseudocommands .repeat. |
| ASM426 | "Wrong method of addressing" |
| | Non-existing method of addressing was used. The list of addressing methods being used is given in document **Software NeuroMatrix® NM6403. Description of assembler language** |
| ASM427 | "Address registers from various groups" |
| | Address registers from various groups cannot be used in this assembler instruction. Registers ar0 - ar3 compose one group, registers ar4 - ar7 - the second one. See more details in **Software NeuroMatrix® NM6403. Description of assembler language.** |
| ASM428 | "Assignment register should be an address register" |
| | This error arises at an attempt to use an address register for executing arithmetic operations, for instance: gr4 = ar5+gr5. Such operation is impossible, instead of it the operation of address register modification can be used: ar4 = ar5+gr5. |

| | |
|---|---|
| ASM429 | "Wrong operand of a shift command" |
| | This error arises when a shift operator is preset improperly, e.g. in operations of cyclic shift a number differing from 1 is used as a shift value. |
| ASM430 | "Shift through carry should be executed to one unit only" |
| | This error appears when the size of a shift in shift operations through flag `carry` is set improperly when a number other than 1 is used as a shift value. |
| ASM431 | "It should be a number" |
| | In a repeat counter defined by pseudocommand .repeat. the quantity of repetitions is preset by a constant digital value., for instance repeat 32 or .repeat 100. The same relates to the repeat counter used in vector commands whose range of values is within 1-32. |
| ASM432 | "Counter is not within the range of (1-32)" |
| | The vector command repeat counter should be within the range from 1 to 32. This range is defined by the depth of queues FIFO and by the vector processor internal structure. |
| ASM433 | "Syntax error" |
| | Most widely spread error that informs that in a line where it occurred a syntax construct exists that cannot be processed by the assembler. |
| ASM434 | "Redefining a label" |
| | A label defined in a given line had been previously defined in another place, e.g. above in the text. This error may also occur when a label is used in a macros before which key word own is missing. . |
| ASM435 | "Recursive use of macros" |
| | The assembler tolerates the embeddedness of macroses i.e.when in the body of one macros a call of another one is contained and so on. However, if in this call chain the name of a macros already mentioned is found, an error will be generated since this assembler version does not support |

recurrent macros calling.

| | |
|---|---|
| ASM436 | "Quantity of parameters does not coincide in a macros call" |
| | When developing a macros a programmer defines a number of parameters transferred to him. If, while calling the macros, the number of parameters does not correspond to what was defined during the development, this error appears. |
| ASM437 | "Repeated definition" |
| | Repeated definition of identifier is found in this line. |
| ASM438 | "Wrong type of a field" |
| | Internal error |
| ASM439 | "Wrong type of a variable" |
| | Internal error. |
| ASM440 | "Undefined label" |
| | If in the program a transfer to a label is found that, though declared, remained undefined, this error is generated. Syntactically the label definition looks as follows: <MyLabel>. |
| ASM441 | "Initialization of a variable in section .nobits" |
| | Section .nobits is entered into the assembler with the purpose of saving non-initialized variables. If an initialization is found in this type of section, an error arises. The variable being initialized should be transferred to a section of a corresponding type. |
| ASM442 | "Repeated use of an identifier" |
| | This error arises when trying to declare a local variable that has a name which coincides with the name of a variable declared before. |
| ASM443 | "Improper field definition" |
| | . |

| | |
|---|---|
| ASM444 | "Macros redefinition" |
| | In this line a macros is being defined whose name coincides with a macros already defined before. |
| ASM445 | "Undefined macros" |
| | In this line macros is called whose definition location has not been found. Probably the macros is defined in another file. Use command import<file_name>, where a file name is substituted that defines a macros sought for. |
| ASM446 | "Undefined constant" |
| | A constant that was not previously defined is used in this line. |
| ASM447 | "Undefined variable" |
| | A variable that was not previously defined is used in this line. |
| ASM448 | "A label should not be of ‘common’ type" |
| | Error arises in case key word common is found that cannot be used with labels but with variables only. |
| ASM449 | "Improper format of a number with floating point" |
| | Only several record formats of real numbers are supported that are as follows:<br>float(5.6)- single accuracy real number,<br>float(123.00e12)- floating-point number (32 bits),<br>double(123.00e12)- long floating-point number (64 bits). |
| ASM450 | "Undefined die :" |
| | Error is related to the fact that die, used when recording debugging information commands, has not been defined previously. |
| ASM451 | " ‘;’is lost after .endrepeat" |

| | |
|---|---|
| | " ‘;’ is lost after .endif" |
| | Symbol ‘;’ is often lost after pseudocommands .endrepeat, or .endif. This error is a reminder. |

### 3.10.3 Internal and Fatal Errors

The message of internal error testifies to an internal failure of the assembler that is followed by an immediate stoppage of the assembler operation. When such an error arises do not fail to apply to the assembler program builders (see section «How to execute notes and suggestions» in «Foreword».)

The message of a fatal error is an indication of a serious error that appeared in the course of assembler operation. After displaying such message the assembler completes the operation by emergency. Even in case when a file being compiled does not contain syntax and semantic errors, a fatal error may arise. It may, for example, be related to the shortage of disc space or memory.

| | |
|---|---|
| ASM950 | "Cannot open a file :" |
| | Probably, the file carries a flag read-only, or is used by another application. |

| | |
|---|---|
| ASM951 | "Error of stack of lexical analyzer" |
| | The stack of the lexical analyzer has turned out to be overfilled due to the area shortage in the main storage in the course of compilation. Increase of the main storage is required. |

| | |
|---|---|
| ASM952 | "Memory shortage" |
| | Memory shortage on the disc and in the main storage required for compiling a particular file. |

| | |
|---|---|
| ASM953 | "Internal error" |
| | With the appearance of this error apply to the SDK programmers (see section «How to execute notes and suggestions» in Preface. |

# 4 Linker

## 4.1 Introduction

This chapter describes the linker interface, its input control parameters and various modes of operation. It presents detailed information on each key, the default key set used at the linker start-up, the data on work organization with the aid of the command file.

The chapter contains the description of the configuration structure of the file that makes possible to control the arrangement of various parts of a program being executed in the processor storage at arbitrary versions of the physical memory configurations.

A full list of errors displayed in the course of the linker operation is presented as well as the reasons for their appearance and methods of eliminating them.

Apart from this, the chapter comprises the examples of operation with the linker starting from the command file and configuration file creation and up to the analysis of the obtained memory map file.

## 4.2 Linker Overview

The linker for NeuroMatrix® NM6403 is a console application Windows95/NT.

The linker is oriented to operation with the files of ELF format only. It fully supports this format with the exception of creation and processing of dynamic libraries. However, the linker is intended for processing object files for processor NeuroMatrix® NM6403 only since the ELF format does not specify the algorithms of relocation calculations. This algorithm is machine-dependent i.e. is specially defined for each processor type and depends on its data dimension and on the addressing methods supported by it.

The linker commands some optimization capabilities. His optimization is only related to the optimization of the size of object files. A special algorithm embedded in it makes possible to remove sections and symbols unused by the program, to replace references to local symbols by references to sections origins where they are defined. That makes it possible to reduce the volume of the program code and, thus, the time of processing it.

Means of self-control are nested into the linker. At the appearance of a failure situation it sends the diagnostics to the computer screen or to a file transferred as a parameter. Along with user errors the linker also informs an application programmer of internal errors that might occur during the operation. In case of appearance of an internal error it is necessary to apply to linker program builders and to transfer them the files where in the course of work the error occurred and also the data on the combination of input parameters.

### 4.2.1 Main Features

When processing the input object data the linker implements functions as follows:

- integrates sections with similar names and creates for them their own relocation tables required for the readjustment of references for a certain memory configuration of a computation device;
- in the course of constructing executable files that are adjusted for a certain configuration of a computation device, it calculates the addresses of symbols and sections, adjusts all references stored in the relocation tables;
- combines the sections into program segments for accelerating and simplifying loading a program into the computer memory;
- resolves undefined external references among input files;
- removes sections and symbols not used by a program from the output file and also removes debugging information;
- delivers information on errors found in the course of linkage editing.

### 4.2.2 Adjustment to Various Memory Configurations

The linker supports various versions of computer memory configuration. For this purpose a C-like language has been developed with the aid of which in a special file, named a configuration file, ranges of operating addresses accessible to the processor are described, the addresses of program segments loading are set, the distribution of sections being loaded among the segments, their relative location. This language contains three main directives MEMORY, SECTIONS and SEGMENTS that make possible to form the data for the linker, enabling it to correctly adjust addresses and references in an executable file. For more details see Section 4.11 on page 4-25.

### 4.2.3 Output File Types

The linker allows creating three types of output files of ELF format:

- **absolute executable file** - is essentially a set of program segments. Each segment is an image of a certain bucket of the processor memory. It has the loading address, the size of the bucket being imaged and the sequence of codes that should be placed in a particular memory area. And all references to symbols are replaced by the absolute addresses of these symbols;
- **executable relocatable file** - differs from the absolute executable file in that for it there are no previously set addresses of location in the processor memory. The address of each section is defined only at the stage of program loading, therefore in a file considered the concept of a segment being loaded is absent. Each loaded section has its own relocation table and all references to code symbols and slots are

resolved at the moment of loading to the processor when the address of memory location for the section is already defined. Therefore the relocatable executable file, in contrast to an absolute one, contains the table of symbols and tables of relocations;

- **object file** - is a result of combining input files. It contains an integrated table of symbols, generalized sections, recalculated relocation tables. An object file, in contrast to an executable one, may comprise undefined symbols.

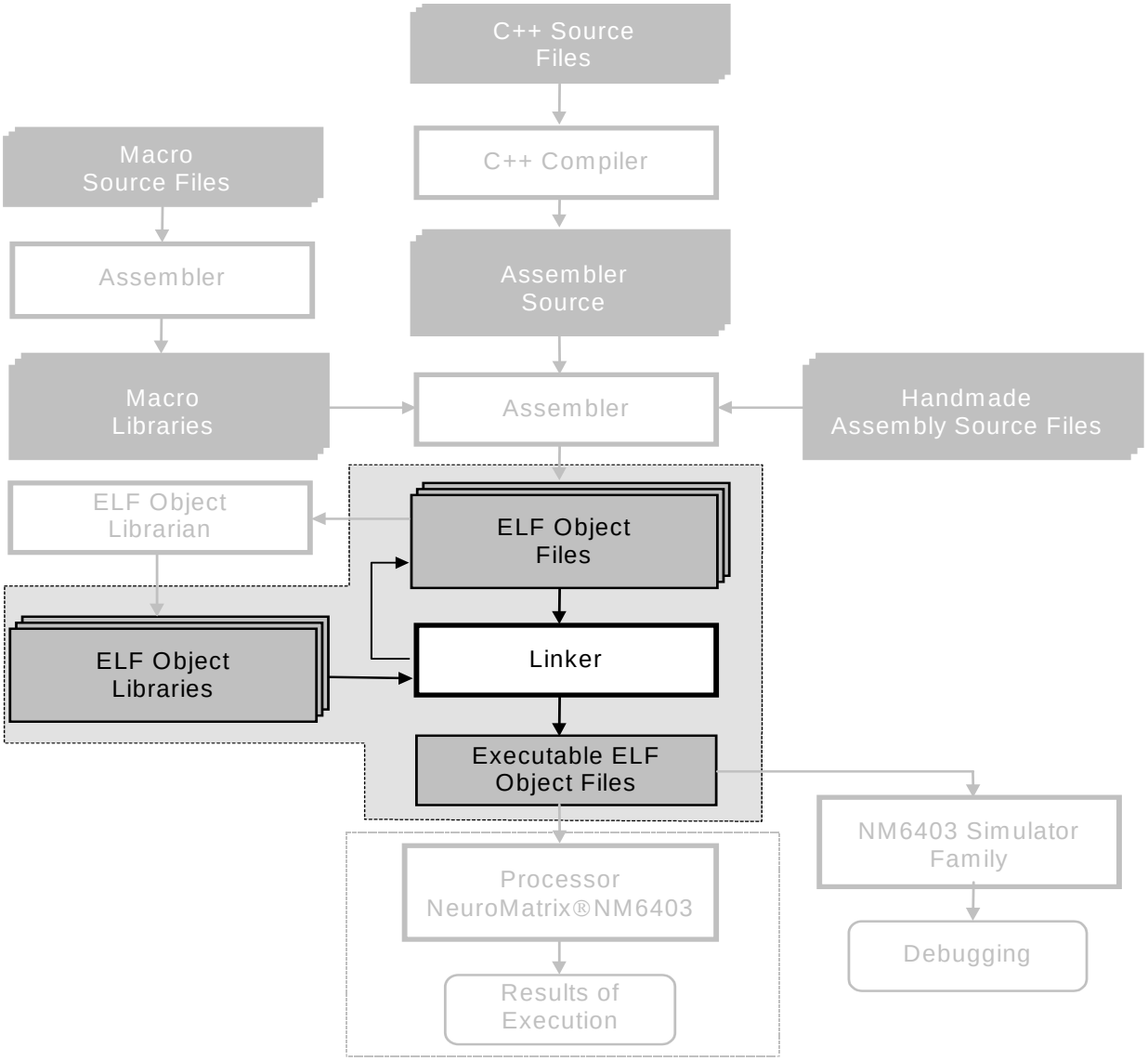
### 4.2.4 Linker Return Value

The linker, in case of successful work termination, issues a corresponding message and returns the value 0. If an error took place in the course of linker operation a message of error appears on the screen and the linker operation is completed as under abnormal conditions with a returned value of 1 or more, depending on the quantity of errors detected.

## *4.3 Linker Development Flow*

Figure 4-1 illustrates the role of the linker in the course of developing application programs for processor NeuroMatrix® NM6403. The linker processes several types of input files including object files, libraries, command and configuration files. The linker creates an absolute executable file or an executable relocatable file intended for loading into the processor memory or into the program emulator.

*Figure 4-1. Linker Development Flow*




## *4.4 Invoking the Linker*

To invoke the linker, enter

**linker** [*options*] *filenames* [*options*]

| | |
|---|---|
| **linker** | Name of file containing the executable code of linker |
| *filenames* | Input object files and libraries. Can be arranged in arbitrary order. Extension in the file name is of no importance. The linker defines object files by a magic number. By the same number single object files differ from libraries. |

*options* — parameters of linker control (with prefix "-"). Can be arranged in any place of a command line, in arbitrary sequence. (More detailed discussion is presented below).

There are two ways of starting the linker

- Linker start-up with the setting of parameters and file names in the command line. The next example contains the linker call for editing the linkage of two object files file1.elf и file2.elf. As a result of the linker work an absolute executable file will be produced. Parameter -o establishes the name of the output file result.abs.

```
linker file1.obj file2.obj -oresult.abs;
```

- The linker starts-up with indication of the command file containing the parameters and names of input object files. The command file is essentially a textual file whose each line contains one or several parameters of the linker. To instruct the linker that the parameters should be taken from the command file, parameter "@file" is added to the command line where file - is the command file name. The command file may look as follows:

```
-oresult.abs
file1.elf
file2.elf
```

In this case the start of the linker with the command file in the quality of an input parameter will look in the following way:

```
linker @file.cmd
```

The command file is a convenient form of recording the input parameters of the linker. It is especially necessary when the linker needs to bring together a great quantity of object files, and the command line is not capable of enclosing all input data. For example, the command file may look as follows:

```
file1.elf file2.elf
```

And the linker start-up in this case will look in the following way:

```
linker @file.cmd -oresult.abs, or
linker -oresult.abs @file.cmd
```

### *4.5 Linker Options Summary*

To control the work of the linker a set of parameters (keys) is used

**All parameters start with symbol "-".**

**In parameter names uppercase and lowercase letters are distinguished.**

They can be indicated in a command line or command file. The sequence where the parameters are arranged is of no significance. The parameters are separated among themselves by spaces. If an argument follows a parameter it is recorded without space, for instance -ofile.abs. Otherwise the linker will perceive the given name as the name of an input object file and, being unable to find it or trying to open, will display an error. In this event the linker operation will end as under abnormal conditions.

Presented below is a summary table of the parameters of linker control:

*Table 4-1. Linker Genertal Options*

| OPTIONS | DESCRIPTION |
|---|---|
| `-h, -?` | Reference data issuing |
| `-q[n][=filename]` | Silence mode. `n = 0..2`. Depending on n, delivered to the flow is all information, error messages only or nothing. If the file name is indicated, the information will be reentered to it. |
| `-t` | Issuing the linker header, information on the version. |
| `-p` | Issuing the complete path of its own. |
| `-ofilename` | Presetting the name of the output file (if there is no extension it is defined in accordance with the type of the output file). |
| `@filename` | Presetting the command file name. |

*Table 4-2. Output File Fypes*

| OPTIONS | DESCRIPTION |
|---|---|
| `-abs` or `–a` | Absolute executable file. |
| `-rel` or `–r` | Executable relocatable file. |
| `-elf` or `–e` | Object file. |

*Table 4-3. Specific Options*

| OPTIONS | DESCRIPTION |
|---|---|
| `-cfilename` | Assignment of configuration file. |
| `-mfilename` | Creating memory map of an absolute executable file. |
| `-lpathname` | Path to the catalogue of library files. |
| `-d0` | Ban on deleting unused sections. |
| `-d1..3` | Saving debugging information in the mode of unused sections deleting. |
| `-d4` | Deleting all debugging information and unused sections. |
| `-heap=size` | Size of a heap in a local storage (Kwords) |
| `-heap1=size` | Size of a heap in a global storage (Kwords) |

| | |
|---|---|
| `-stack=`*`size`* | Size of a stack (Kwords) |
| `-start=`*`entrypoint`* | Point of input to the program. |
| `-asm` | Enabling the initialization of static global variables in C++ language. |
| `-addr=`*`address`* | Default address of the segment being created in absence of the configuration file. |

## *4.6 General Options*

Related to the general options are the keys having similar meanings and executing similar actions for the most of NM6403 SDK components.

### 4.6.1 Printing Out Reference Information (-h, -? Options)

When starting the linker without input parameters or with parameter `-h`, or `-?` reference information on all the keys defining the linker behavior and the result of its work is sent to the monitor screen. The information displayed on the screen is as follows:

*Figure 4-2. The Linker Reference Information*

```
Linker for NeuroMatrix NM6403 * v1.07 * (c)1996-99 * RC Module.
Usage:  nlink [options] <file1> ... <fileN>
Options:
-abs (-a)        - absolute file for output
-rel (-r)        - movable file for output
-elf (-e)        - object file for output
-o<filename>     - name of output file
-c<filename>     - name of config file
-m<filename>     - name of map file
-start=<name>    - name of entry point
-stack=<size>    - stack size
-heap=<size>     - local heap size
-heap1=<size>    - global heap size
-d<n>            - 'cleaning level'; n = 0..4
                   0 - don't remove anything
                   1 - mark DEBUG-sections before analysis
                   2 - standard, then mark all DEBUG-sections
                   3 - standard, then copy InUse value from base sections
                   4 - standard algorithm (maximum removing)
                     When <n> is omitted, 4 assumed
                     Default is also 4 (full clear)
-asm             - to process file generated via assembler (not C)
-q[n][=filename] - set up linker diagnostic; n = 0..2
                   0 - all information
                   1 - only error messages
                   2 - silence
                   if <filename> specified redirect output
-addr=<address>  - default segment address definition
```

With the appearance of key `-h` or `-?` in the command line the work of the linker is completed before editing regardless of the availability of other parameters. Output file in this case is not created.

### 4.6.2 Suppressing Progress Information (-q[n][=*filename*] Option)

Depending on parameter n the linker issues information in the way as follows:

- in the case `n = 0` the linker delivers to the user all generated information, namely, a title header, warnings, error messages,

- in the case `n = 1` the user only receives the information on errors and warnings. This mode is often used at the linker package start or at the call from an integrated environment,
- in the case `n = 2` the linker does not issue anything, i.e. at the linker operation in this mode even error messages are suppressed.

The case when `n` is absent is equivalent to `n = 1` i.e. the record of parameter `-q` or `-q1` leads to a similar result.

If after key `-q[n]` through sign «equal» without space a file name is preset, the linker creates the file with this name and sends all the information to it. If a file with this name already exists a new file will be recorded on top of the old one without any additional warning.

### 4.6.3 Displaying the Banner (-t Option)

Parameter -t is of exceptionally informative nature. It is used for getting the data on the name of this SDK component, on the version number and on the copyright. With the appearance of this key in a command line the linker issues an information message:

```
Linker for NeuroMatrix® NM6403 *v1.0* (c)1996,98 * RC Module
```

and completes the work. The work of the linker will be completed regardless of the availability of other parameters in the command line. No output file is created.

### 4.6.4 Printing Out the Linker's Location (-p Option)

Parameter `-p` is of purely informative nature. It is used for locating a linker called. Often a situation arises that the user does not know where the linker is being called from. It is only known that this is one of the catalogues presented in the list of general-access catalogues in file autoexec.bat (`PATH=...`). In this situation key `-p` can be used. The linker, having found this option in the command line will display a message, for instance:

```
I am located in C:\MODULE\NMCSDK\BIN\LINKER.EXE
```

and completes the work. The work of the linker will be completed regardless of the availability of other parameters in the command line. No output file is created.

### 4.6.5 Setting Output File Name (-o*filename* Option)

Output file is the result of the linker operation. Its name is defined by way of setting key `-o`*`filename`* in the command line. Between -o and the file name there should be no space. A full name is meant by the file name, i.e. a full path to the file. If only the name is set and the path is omitted, the file will be created in a current catalogue. If a file with such name already existed in the given directory it will be replaced by a new one without any additional warnings.

If the linker does not encounter in the command line a key defining the name of an output file, the name of the first file from the input file list will be used as a name of the output file. In this case the extension of the new file will depend on what type of a file it is:

*Table 4-4. The Linker Output File Extentions*

| EXTENSION | DESCRIPTION |
|---|---|
| `.abs` | For absolute executable files |
| `.rel` | For executable relocatable files. |
| `.elz` | For object files. |

In the base NM6403 SDK the main extension for object files is "`.elf`". However the linker assigns extension "`.elz`" to the output object file. This is done to avoid replacement of the old (input) object file.

### 4.6.6 Using a Command File (*@filename* Option)

The command file is an alternative way of setting parameters to the linker. When a great number of keys and file names is needed to be listed in the command line, situation of overfilling often occurs since the length of the command line is restricted by 128 symbols. Therefore it is more convenient to arrange all the parameters in the command file and to transfer its name, fitted with prefix `@` as the command line parameter, to the linker. There should be no space between sign `@` and the name of the command file. The command line in this case will look much easier. It will approximately look as follows:

```
linker @cmdfile.cmd
```

All the parameters of the linker can be arranged in the command line both each in individual lines and several parameters in one line. For example, none of the records presented below will be erroneous:

```
-r
-d2 -stack=128
test1.elf -oresfile.rel
test2.elf test3.elf
```

It is convenient to place the constant part of the command line to the command file and to set the rest, frequently changing parameters, in the command line directly. For example, if there is a need to transfer the following set of parameters to the linker as an input file:

```
-a -stack=4 -heap=1024 -heap1=1024 -d4 main.elf
test1a.elf test1b.elf rtl.lib,
```

and at the same time to trace the result of the linker operation for various types of an output file and for different algorithms of unused sections deleting. The work with such set of parameters can be organized in the way as follows:

- To place all the keys, not changeable during testing, to the command file myfile.cmd. Then the command file may look as follows:

```
-stack=4 -heap=1024 -heap1=1024
main.elf test1a.elf
test1b.elf rtl.lib
```

- To write changeable parameters in the command line along with the command file. Then the process of starting up the linker will be reduced to the following:

```
linker -a @myfile.cmd -d4
```

At the same time there will be no need to rewrite each time the long command line and be worried about the shortage of place in it for the parameters that may add to the general list.

### *4.7 Output File Types Description*

The linker works with object files of ELF format intended for execution on processor NM6403. In the title of such files a field is located defining a type of machine for which they have been created. In the given case this field has a special meaning corresponding to the processor. With any other meanings of this field the linker will issue a corresponding error and cease the work.

The linker generates three types of output files:

- **absolute executable file** is intended for execution on processor NM6403. It has the name «absolute» because for each structure incorporated into it, an absolute (accurate) address of its location in the memory accessible to the processor, is known. Its internal features are structured in such a way that promotes the maximum acceleration and simplification of a process of loading into the memory of a computation device. The file is divided into special fields named segments, each of them representing a dump of a certain field of the computer memory. The loading of such file is reduced to copying the segments by addresses preset in their headers.

- **executable relocatable file** is intended for execution on processor NM6403. In contrast to the previous file type the addresses of loading its structures are not known in advance and are defined only at the moment of loading. Therefore a file of this type contains a table of symbols and relocation tables for sections where references to the symbols are found. All symbols from the symbol table should be defined. That means that each symbol should contain information of a section where a location was allotted for it. The layout of this type of file in the computer memory is realized by a special loader that should assign the addresses of section layout, calculate symbol addresses and resolve all references located in the section relocation tables.

- **object file** is not intended for executing on a processor and is an intermediate step in creating a program being implemented. The linker allows to make one object file from several ones, a file that comprises all input information. Such a file is intended for further assembling with other object files. The linker realizes a set of internal conversions of object files, therefore object files produced from under the assembler cross-compiler will be changed and simplified, though they still will remain object files. These changes are connected with the resolution of individual types of references within sections and with the replacement of the references to all local symbols by references to the origin of sections where they have been defined.

In the course of creating executable files the linker, if a user wishes, may enable the algorithm of the deletion of unused sections (see 4.8.1 on page 4-16). In the mode of object file creation this algorithm is not provided for, and a corresponding warning is issued about it.

### 4.7.1 Creating an Absolute Executable File (-abs or -a Option)

Parameter -a instructs the linker that the output file should be represented by an absolute executable file. This key is set by default, therefore if the linker does not find an input parameter defining the type of an output file, it will create an absolute file. An absolute file should necessarily contain an input point. By default the input point is named "start". Normally it is contained in the startup code and constitutes a global label. The user may enter his name of an input point (see 3.8.6 Defining the name of an input point (key `-start=<label_name>`) on p. 3-19.

The following examples bring together files `file1.elf` and `file2.elf`, and create an absolute executable file:

```
linker -abs file1.elf file2.elf -oresult.abs
linker -a file1.elf file2.elf -oresult.abs
linker file1.elf file2.elf -oresult.abs
```

### 4.7.2 Creating a Relocatable Executable File (-rel или -r Option)

Parameter `-r` instructs the linker that the output file should be represented by an executable relocatable file. To place this file in the memory of a neuro-computer a special loader should be used that receives from the user the computer memory map and the address of each section loaded. The loader should compute the addresses of all symbols and to resolve all references stored in the file of this type, to define an input point. With this purpose the loader processes the symbol table and relocation tables of corresponding sections. In contrast to an absolute file wherein sections being loaded are combined into segments and are loaded to the memory simultaneously, the load process of an executable relocatable file requires each section to be processed individually. However, relocatable files make possible to more flexibly organize the load process depending on the contextual conditions of the program operation.

The following examples bring together files `file1.elf` and `file2.elf`, and create an executable relocatable file:

```
linker -rel file1.elf file2.elf -oresult.rel
linker -r file1.elf file2.elf
```

### 4.7.3 Creating an Object File (-elf or -e Option)

Parameter -e instructs the linker that the output file should be represented by an object file. The object file generated by the linker generalizes and stores the information from input files. It realizes the following conversions:

- *bonds together sections of similar names.* «Bonding» means adding an input section next in turn to the end of an output section of the same name; after that such sections are perceived as an integral whole. The case in point is not all the sections contained in object files but only those that store initialized and non-initialized data, program codes and debugging information. Apart from these sections object files contain auxiliary information on symbols, on references to symbols and on the symbol names. The procedures of processing such information differ from bonding. They will be described in paragraphs given below,
- *creates a summary table of symbols.* In so doing references to defined external symbols are partially resolved. In contrast to the two preceding types of files, object files may contain unresolved external references,
- *constructs relocation tables* for those output sections that contain references to symbols. It resolves a part of the references, namely, relative references from the sections to symbols that are also defined in these sections (such references often arise when using skip command in programs in assembler language),
- *replaces references to local symbols* by references to the origin of the sections in which these local symbols were defined. This allows to unload the symbol table, to delete a great number of dead local symbols that affects also the size of an output file.

In the mode of object file creation the linker enables the mode of dead sections deletion.

It should be noted the mechanism of the merging of initialized and non-initialized data sections has a number of peculiarities. If in various object files brought together section of the same name are found, i.e. one section contains initialized data and the second one - non-initialized, their merging leads to the formation of the initialized data section. The part that was not previously initialized is zeroed.

The following examples bring together files `file1.elf` и `file2.elf`, and create an object file:

```
linker -elf file1.elf file2.elf -oresult.elf
linker -e file1.elf file2.elf -oresult.elf
```

## *4.8 Specific Options*

Related to specific parameters of the linker is the set of keys not found in other SDK components or having a different meaning there.

### 4.8.1 Removing an Unused Sections and Debug Information (-d4 Option)

When creating an output file the linker provides the user with a means for excluding non-usable information from it. Such information includes debugging information and sections for which there are no references from other sections (except special auxiliary sections that are processed separately).

The linker uses this mode by default when creating an executable file. Another record of this mode, equivalent to it, is `-d` where a figure is not indicated.

Complete removal of unnecessary information is possible only at creating an absolute or relocatable executable file. On attempted indication of this mode while creating an object file the linker ignores key `-d4` and issues a corresponding warning.

The following example brings together files `file1.elf` and `file2.elf`, and creates an absolute executable file from where all dead sections and symbols, including debugging information, are removed:

```
linker -d4 file1.elf file2.elf -oresult.abs
linker -d file1.elf file2.elf -oresult.abs
```

### 4.8.2 Keeping Debug Information (-d{1..3} Option)

Modes of partial preservation of the debugging information serve the purpose of minimizing the output file output when there is a need to save the debugging information. Under all these modes dead sections and the debugging information sections coupled with them are partially removed (should be removed at least).

These three modes differ in algorithms and they have appeared, mostly, due to the lack of full clarity with regard to interrelations between the sections of data and sections with debugging information. In further versions, probably, only one mode of three will remain.

Similar to the full removal, partial removal of unnecessary information is possible only at creating an absolute or relocatable executable file. On attempted indication of this mode while creating an object file the linker ignores key `-d1..3` and issues a warning.

The next example creates an absolute executable file `alone.abs`, from where all dead data sections are removed and from the section with

debugging information only those are saved which refer to the remaining data sections.

```
linker alone.elf -d3
```

### 4.8.3 Keeping All Data in an Optput File (-d0 Option)

Parameter `-d0` is used when a user wishes to save in the output file all information contained in input files, including that which, probably, does not participate in the program.

The following example puts together files `file1.elf` and s, and creates an absolute executable file saving all input information.

```
linker -d0 file1.elf file2.elf -oresult.abs
```

When creating object files this key is a default key and in addition the only possible one among keys `-d{0..4}` since at this stage there is no information on which data and code will be used in future and which will be not.

When creating executable files **key -d0** in certain cases **may impede** the generation of an output file. This will occur in situations when in one or several input modules given to the linker as parameters, undefined global symbols are saved with no references made to them.

Let us, for example, consider a file in C++ language which contains the declaration of a function as follows:

```
extern int MyUndefFunc();
```

However, there is not a single reference to it. Such a construct may cause a warning of compiler C++, however it will get to the assembler file and, through it, to the object one as well. When the linker begins to combine this file with other ones for creating an executable file, it will find that this label is not defined but there are no references to it. If any of the keys `-d{1..4}` is used the symbol under consideration will be ignored and will not get into the output file. However key `-d0` stipulates that all symbols and sections of input object files will be transferred to the output file. The main property of the executable file is the absence of unresolved external references. Therefore the linker will issue error «undefined global symbol» and complete the work as under abnormal conditions.

### 4.8.4 Defining Memoty Heap Size (-heap и -heap1 Options)

Compiler C++ uses for processor NM6403 sections of non-initialized data `.heap` (local heap) and `.heap1` (global heap) for creating a dynamic execution time heap used by function `malloc()`. With the aid of keys -heap and `-heap1` it is possible to define the heap size in a global and local memory of a computer. The magnitude of the heap is preset after sign «equal» and measured in 32-bits words:

```
linker -heap=0x40000 -heap1=0x100000,
```

allotted for local heap are 256K of words and for global one 1Mb of words.

Between keys `-heap` and `-heap1`, sign "=" and the heap size there should be no space.

By default the sizes of the local and global heaps are equal to 1K of words. If the heap size is not preset by a user, the linker will issue a reminder that only 1K of words are allocated for the heap.

The linker creates sections `.heap` and `.heap1` only in case they are available in input files, in particular, in the library for the work with dynamic memory. In all other cases these keys are ignored.

The linker also creates global symbols `__HEAP_SIZE` and `__HEAP1_SIZE`. They are assigned with meanings equal to the sizes of local and global heaps of the execution time memory.

### 4.8.5 Defining System Stack Size ( -stack=*size* Option)

For the work of programs on processor NM6403, function call, saving registers and return addresses a stack is used. Stack is a section of non-initialized data with name `.stack`, whose size is preset at the stage of creating an executable file, i.e during linkage editing. The stack size is preset with the aid of parameter `-stack=`*`size`* and measured in 32-bits words.

The following example illustrates the process of stack defining (sections `.stack`) measuring 4K of words:

```
linker -stack=0x1000
```

By default, the stack size is equal to 1K of words.

Between key `-stack`, sign "=" and the stack size there should be no spaces.

If not indicated in an opposite way in the configuration file, section .stack is isolated in the memory among other sections. However, to increase fast-action it is recommended to place this section to a fastest from accessible memories of the processor. The mechanism of sections layout by predefined addresses is presented in the description of the configuration file (see 3.11.3 Section SECTIONS on p. 3-29).

### 4.8.6 Defining the Entry Point (-start=*label* Option)

The memory address from where the program starts to be executed is named **an input point**. When a loader places the program into a computer memory, the command counter should be initialized, and the address of the program origin should be written.

By default, when there is no input point name preset in the command line, the linker thinks that it has name **`start`**. If the global symbol with this name is not found in the symbol table or found but not defined, an error "input point start is not defined" is issued to the user.

Key `-start=`*`label`* makes possible to preset its label name from where the program is to be started. The label should be necessarily defined in

one of the sections and have a global type of bonding. The linker computes its address and saves it as the input point address.

The following example illustrates the process of defining a user input point that is marked in the program by label BEGIN:

```
linker file1.elf file2.elf -start=BEGIN
```

Among key `-start`, sign "=" and label name there should be no space.

Usually the input point with name start is defined in the start-up code that is preserved in the execution time library. When this library is connected the start-up code in the process of linkage editing is automatically added to the user program.

**Note** *If a standard library is used and the input point is defined the standard startup code will not be used and with the additional key presetting* -d{1..4} *it can be removed from the executable file at all. Then the user takes the responsibility on correct setting stack register* ar7 *when starting the program and when processing the return from the program.*

### 4.8.7 Disabling Initialization of Static Global Objects (-asm Option)

Parameter `-asm` permits the linker the disabling of static global objects required for initialization in C++.

The following example puts together files `file1.elf` and `file2.elf`, and creates an absolute executable file disabling the processing and creation of specialized sections `.init` and `.fini`:

```
linker file1.elf file2.elf -asm -oresult.abs
```

If a user does not use the standard start-up code for starting the program he may delete the said sections from the executable file. Since sections `.init` и `.fini` are processed in a special way, they cannot be deleted by means of using key `-d{1..4}`, this is why a special key `-asm` is entered.

The start-up code contains two function calls:

```
call ctor;
call dtor;
```

They are called respectively before and after the function __main - the main function of the user program. Labels `ctor` and `dtor` are defined at the beginning of sections `.init` и `.fini`. If in the program static global objects were found whose fields need to be initialized, in section `.init` и `.fini` the calls of corresponding functions of initializing and removing these functions are added. If there are no fields that need to be initialized before the function __main functions `ctor` and `dtor` contain only return instructions.

If a user uses his input point and his program is written in assembler and does not require early initialization of static fields, he may use this key and delete sections `.init` и `.fini` from the executable file.

### 4.8.8 Defining Library Search Path (-l (lowercase "L") Option)

Parameter `-l` presets the path to the catalogue of library files. If during the assembly of the executable file the linker has found an undefined global symbol to which a reference exists, then, to define this symbol, it scans the libraries that are presented in the command line and contained in the catalogue indicated by a user with the aid of key `-l`.

If a user wishes to connect a library that is located not in the current catalogue, he may add its full name to the command line or indicate a catalogue where it is located by means of key `-l`.

The following example puts together files file1.elf and file2.elf, and realizes the retrieval of undefined global symbols in the library `mylib.lib`, located in catalogue *c:\lib*:

```
linker file1.elf file2.elf mylib.lib -lc:\lib -oresult.abs
```

There should be no spaces between key `-l` and the path to the catalogue of libraries.

The linker realizes the retrieval of libraries in the following order:

- considers libraries whose names are preset in the command line;
- realizes search of libraries with preset names in a current catalogue;
- scans catalogues preset by key -l in the same sequence in which they are listed in the command line or command file.

If a user wishes to preset several catalogues for searching libraries he should enter parameter -l to each catalogue:

```
linker file1.elf file2.elf mylib.lib -lc:\rtl -lc:\lib -oresult.abs
```

**Note** *Each library consists of a certain number of object files. If a symbol sought-for is defined in a given library, this means that it is defined in one of object files being part of the library. Therefore only this file will be loaded by the linker for further processing. If it contains much outside information with no relation to defining a symbol considered, it will be deleted from the final executable file if key -d{1..4} is available.*

### 4.8.9 Name the Memory Map File (-m*filename* Option)

Parameter `-m` presets the file name where the linker will write down the information on memory distribution map for a given output file. The memory map is created only for absolute executable files. It describes the following:

- distribution of processor NM6403 memory (addresses and sizes of banks);
- position of segments in memory banks,
- distribution of output sections by segments,
- position of input sections in output sections,
- absolute addresses of all global symbols.

The map file contains the name of the output file and the input point.

In addition it comprises the following tables:

- Table describing the addresses and sizes of processor memory banks, sizes of the memory page of each bank,
- Table describing the distribution of segments by memory banks. Each segment has several attributes: attribution to the memory bank, absolute address of origin and size,
- Table describing the distribution of sections by segments. Ech section has several attributes such as an attribution to a segment, absolute address of origin and size.
- Apart from this the table comprises information on input sections, on output components, namely, to what input file a particular section belongs and on its size.
- Table describing names and addresses of all global symbols of the program.

The following example brings together files `file1.elf` and `file2.elf`, creates an absolute executable file and makes a memory map for it saving it in file `mapfile.map`:

```
linker file1.elf file2.elf -mmapfile.map -oresult.abs
```

There should be no space between key `-m` and the name of memory file map.

The name of the memory file map is put in case it differs from the output file name. By default the name of the file map coincides with the name of the output file but has extension "`.map`". Therefore, the same example but with the default name of the file map will look as follows:

```
linker file1.elf file2.elf -m -oresult.abs
```

and the resulted memory file map will have name `result.map`.

### 4.8.10 Supplying the Configuration File Name (key -c<file_name>)

Parameter `-c` defines the name of configuration file for creating an absolute executable file. The configuration file contains all information required for the correct layout of the program in the processor memory. More detailed description of the configuration file is presented in section 3.11 Configuration file on p. 3-24.

The following example brings together files `file1.elf` and `file2.elf`, creates an absolute executable file and uses for it the configuration file `cfgfile.cfg`:

```
linker file1.elf file2.elf -ccfgfile.cfg -oresult.abs
```

There should be no space between key `-c` and the name of the configuration file. The configuration file is used only for creating the absolute executable file.

If during the creation of the absolute file key `-c` with the configuration file name is not set, the linker creates an output file with the only program segment, whose origin address is equal to `0x00000000` and the size is not limited. The limitless size of the segment means that it encloses all the sections loaded with consideration for the requirements on their alignment. To change a segment address key `-addr` described below should be used.

### 4.8.11 Setting the Default Segment Address ( -addr=*address* Option)

In the mode of creating an absolute executable file, when the configuration file is not preset, the linker creates a unified data and code segment which comprises all sections loaded found by the linker in the input files. To preset the location address of this segment in the processor memory key `-addr=`*`address`* is used. The absolute segment origin address written in hexadecimal form is entered into field «address», for example:

```
linker file1.elf file2.elf -addr=0x00000080;
linker file1.elf file2.elf -addr=0x80000080;
```

## *4.9 Default Options*

When starting the linker with the set of input object files the following default input parameters are set:

*Table 4-5. Linker Default Options*

| OPTION | STATUS | DESCRIPTION |
|---|---|---|
| `-a` | Set | As a result of linker operation an absolute executable file will be created. |
| `-d4` | Set | All debugging data and unused sections and symbols are deleted. |
| `-heap=0x400` | Set/Not set | Initial size of the heap in the local memory is |

| | | |
|---|---|---|
| | | equal to 1K. The parameter is set be default in case of including the library of control of the dynamic memory into the input files list. |
| `-heap1=0x400` | Set/Not set | Initial size of the heap in the local memory is equal to 1K. The parameter is set be default in case of including the library of control of the dynamic memory into the input files list. |
| `-stack=0x400` | Set | Initial stack size is equal to 1K |
| `start=start` | Set | Label `start` with a global bonding type is regarded as a default input point. |
| `-asm` | Not set | Support of primary initialization of global variables in C++ is enabled. |
| `-l""` | Set | With undefined global symbols found in the operational catalogue the search of libraries is performed where these symbols could be defined. |
| `-addr=0x00000000` | Set | In the absence of the configuration file one loadable segment with the load address 0x00000000 is created. |

### *4.10 Correct and Incorrect Option Combinations*

A part of the linker parameters called information parameters (-h, -?, -t, -p) cannot be used in combination with others, not included in this group. The main purpose of these parameters is the presentation of information to a user regarding the procedure of linker startup, input parameters used, version, location. In so doing, when the linker encounters an information parameter, it ceases the operation. For example, being faced with the input parameters combination

```
-q=file.log test.elf -h
```

the linker will not start the procedure of input object file processing but will only supply reference information with a list of control keys to `file.log`.

For each type of an output file its own set of input parameters exists. Presented below are sets of input parameters for each type of an output file generated by the linker.

Set of input parameters of an absolute executable file.

*Table 4-6. The Absolute Executable File Options*

| Option | Description |
|---|---|
| `-a` | absolute executable file. |
| `-c`*`filename`* | configuration file assignment. |
| `-heap=`*`size`* | heap size in the local memory (kilobytes) |

| | |
|---|---|
| -heap1=*size* | heap size in the global memory (kilobytes). |
| -stack=*size* | stack size (kilobytes) |
| -start=*label* | point of input to the program |
| -asm | disabling the support of initialization of global static objects in C++ language. |
| -l*pathname* | Path to the catalogue of library files. |
| -d0 | Ban on the deletion of dead sections. |
| -d1..3 | Saving of debugging information in the mode of dead sections deletion. |
| -d4 | Deletion of all debugging information and dead sections. |
| -m*filename* | creation of the memory map of an absolute relocatable file |

*Table 4-7.The Executable Relocable File Options*

| OPTIONS | DESCRIPTION |
|---|---|
| -r | Executable relocatable file. |
| -heap=*size* | heap size in the local memory (kilobytes) |
| -heap1=*size* | heap size in the global memory (kilobytes). |
| -stack=*size* | stack size (kilobytes) |
| -start=*label* | point of input to the program |
| -asm | disabling the support of initialization of global static objects in C++ language. |
| -l*pathname* | Path to the catalogue of library files. |
| -d0 | Ban on the deletion of dead sections. |
| -d1..3 | Saving of debugging information in the mode of unused sections deletion. |
| -d4 | Deletion of all debugging information and unused sections. |

*Table 4-8. The Object File Options*

| OPTION | DESCRIPTION |
|---|---|
| -e | Object file. |

Attempts to input parameters included in various sets may at best lead to ignoring a part of the parameters with warnings displayed, and at worst -

the process of creating an output file will be stopped with an error message displayed. In fact, the attempt to simultaneously create output file (`-e`) and input point (`-start=label`) should lead to ignoring the second parameter, and the simultaneous use of keys -e and -a - to an error.

### *4.11 Configuration file*

The configuration file is used by the linker **only at creating absolute executable files**. It contains the information as follows:

- on the ranges of accessible physical addresses and hardware characteristics of the memory banks of processor NM6403 (physical memory configuration);
- on the layout of the application program segments in the memory and their sizes (logic memory configuration);
- on the attribution of code sections, data and other elements of the program to particular segments;
- on the alignment of sections and segments to the border of memory pages or to other values.

A configuration file is a text file written in some formal Си-similar language of description.

The file consists of an arbitrary quantity of description blocks of three types:

- memory descriptions (MEMORY),
- segment descriptions (SEGMENTS),
- section descriptions (SECTIONS).

Example of a configuration file:

*Figure 4-3. An Example of the Linker Configuration File*

```
MEMORY /* physical memory configuration */
{
 LOCAL0 at 0x00000000, len = 0x00100000, page = 0x10000;
 LOCAL1 none;

 GLOBAL0 at 0x80000000, len = 0x00100000;
 GLOBAL1 at 0x80100000, len = 0x00100000;
}

SEGMENTS /* logic memory configuration (segments) */
{
  name1 in GLOBAL0, length = 0x1000;
  name2 in GLOBAL1, len = 1024;
  name3 at 0x30000000, l = 2048;
  name4 in LOCAL0;
}

SECTIONS /* Layout of sections by segments */
{
  .text   in name2, align page;
  .data   in name1;
  .text1  in name1, align page;
  .nobits in name3;
  .heap   at 0x10020000;
  .stack  at 0x00010000;
}
```

If sections MEMORY and SECTIONS are absent the linker supposes that a default model of memory is used. In this case all sections of the object file are located by addresses starting from `0x00000000`, in the order of their coming.

Numeric values can be written in decimal and hexadecimal formats, in the same way as in languages C/C++: a decimal value consists of decimal digits and starts not from zero, hexadecimal values have prefix 0x, that is followed by a sequence of hexadecimal digits 0..9, A..F, a..f.

### 4.11.1 MEMORY Section

Section MEMORY contains the description of the physical configuration of memory accessible to the computer processor. The whole memory is broken down by address fields named as banks.

For each memory bank parameters are put as follows:

LOCAL0
LOCAL1
GLOBAL0
GLOBAL1 - reserved names of memory banks;

len - memory bank size

page - memory bank page size

#### 4.11.1.1 Reserved Names for Memory Banks

Memory banks may have only one of four names: LOCAL0, LOCAL1, GLOBAL0, GLOBAL1.

Section MEMORY should contain the characteristics of each memory bank. If one or several banks are not available in a given configuration, this should be reflected by means of using key word none opposite the name of a corresponding bank.

For each bank its own page size can be preset. A user may control only the page size of the memory bank. If the value of this field is not put by the user, a default meaning is assigned to it that corresponds to the maximum possible size of the page. All the remaining parameters of the section are defined by a physical configuration of a computation device.

Example of section MEMORY:

```
MEMORY /* Physical configuration of memory */
{
 LOCAL0 at 0x0, len = 0x100000, page = 0x10000;
 LOCAL1 none;/*Bank1 of the local memory is absent */

 GLOBAL0 at 0x80000000, len = 0x00100000;
 GLOBAL1 at 0x80100000, len = 0x00100000;
}
```

#### 4.11.1.2 Memory Default Pattern

When a configuration file is not preset or section MEMORY is absent from it, the linker uses a default memory pattern. This patter is based on the architecture of processor NM6403 and it implies that all 32-bit space is accessible to a user. The initial address of the executable file layout in the processor memory is supposed as equal to 0x00000000. For loading absolute executable files a single one-size segment is created. The sections are arranged into the segment in the order as they come, however there are two priorities of the layout. Sections of initialized data come under the first priority, those of non-initialized - under the second one. Such an approach makes it possible first to arrange initialized sections in the segment, and after that - non-initialized ones.

### 4.11.2 SEGMENTS Section

In section SEGMENTS the layout of the segments of the application program by memory banks is preset.

Each segment has the following attributes:

name - segment name, no more than 32 symbols. Symbols A-Z, a-z, ., _ can be used for presetting a name. Spaces inside

the name are not permissible. The segment name is not included into any of the name tables, it is preset only for the convenience of reading the configuration file and for issuing the information of errors.

| | |
|---|---|
| length<br>or<br>len<br>or<br>l | - segment size. It may be omitted. In this case the segment may swell to the limits of the bank wherein it is defined. Segment size limits its maximum volume. Actually the segment may be of lesser size since this size is defined by the sizes of sections contained in it. |

Segment cannot cross the border of the memory bank, i.e. it always belong to one bank only. To avoid confusion similar names should not be used for different segments.

Belonging of a segment to a particular bank is defined by using key word in. If several segments are located in the bank, they are arranged one after the other, in an order defined in the configuration file. In this case, the address of each segment is calculated only after arranging all segments, however it is within the border limits of the addresses of the bank which said address belongs to.

Exact address in the memory can be assigned to a segment with the aid of key word at. It should belong to the range of accessible physical addresses of the processor. Thus the address of this segment is preset in advance and the distribution of other segments is realized with consideration for this fact.

Example of section SEGMENTS:

```
SEGMENTS /* Logic configuration of memory (segments) */
{
  name1 in GLOBAL0, length = 0x1000;
  name2 in GLOBAL1, len = 1024; // segment with top-limited size
  name3 at 0x30000000, l = 2048;// segment with defined address
  name4 in LOCAL0;              // segment with floating size
}
```

A segment may have no size preset, then the linker will calculate it by itself. The segment size is defined as a sum of sizes of sections included into it with consideration of their alignment. Practically the segment may contain both the sections defined in the configuration file and those, whose memory layout is not regulated there though these sections are loadable. The segment floating size implies that sections can be added to it not described in the configuration file, without any restrictions within the limits of this memory bank.

Alignment of the segment is equal to the maximum alignment of sections included into it. That is, if a segment consists of three sections, and one of them is aligned to the border of a memory page and the rest - to the

border of a 64-bits word, the entire segment will be aligned to the border of a memory page.

#### 4.11.2.1 Distribution of Segments Within the Limits of Memory Bank.

Several versions of segment distribution within the memory bank exist:

- distribution of segments of preset sizes. In this case the linker arranges segments in the order as they follow that is presented in the configuration file. The segments are arranged one after the other with consideration for alignment constants preset for each segment. If the space in the bank is insufficient the linker will issue a corresponding error. Real addresses of the segments will be calculated after the layout process is finished,
- distribution of segments when one of them has a predefined size and address. If this segment is not the first in the list the bank is divided into two parts. The first part is complemented with the segments in an order determined in the configuration file. A segment with a defined address is skipped. If there is no space for a segment next in turn, it is placed after a segment with a defined address. The formed free space remains unfilled,
- distribution of segments when one of them has a floating size. Regardless of anything, first placed in the bank are all segments whose sizes are determined only by sections described in the configuration file. Further, the segments are complemented with sections being loaded that are not mentioned in the configuration file. If a segment with a floating size is the first in the list, all these sections are added to it. In this case a check for the excess of the memory bank size is realized. If a preset size segment is the first, sections are added to it until its real size exceeds a preset threshold. Then the remaining sections will be added to the following segments according to the same procedure,
- distribution of segments when two of them have a floating size. The situation almost does not differ from the previous paragraph. All additional sections will be added to the first segment with a floating size, therefore the availability of a floating size with the second segment is of no importance, actually, it will not differ in anything from segments with a limited maximum size,
- distribution of segments when a segment with a floating size is located before that with a defined address. In this case the first segment may be increased to a certain limit, i.e. its size is limited by an address space from the bank beginning to the beginning of the defined address segment. Further procedure of adding sections remains unchanged.

### 4.11.3 SECTIONS Section

This section contains the description of sections loaded that constitute the object file. Each section may have attributes as follows:

| | |
|---|---|
| name | - section name. For presetting a name symbols: A-Z, a-z, _can be used. Spaces inside the names are impermissible. As in the assembler language a section name should not be preceded by a point, i.e. the names of sections in assembler and in the configuration file coincide. |
| align page | - alignment with the memory page beginning. This means that a section should start with an address multiple to the size of this bank memory page where it comes from (by default all sections loaded are aligned with the border of a 64-bits word). |

A section always is included in a particular segment (here we are speaking about absolute executable files only). If several sections belong to one and the same segment they are arranged in an order defined in the configuration file. Sections not mentioned in the configuration file which, though, are loadable sections, are also added to the segments in compliance with the rules described in subsection 4.11.1 on page 4-26.

Manual setting of the absolute address of a section: at `0x80000000`; this is an address that starts the layout of a section being described. In this case for a given section a special segment is set that has a definite address and size equal to the size of a section.

Presented herein is an example of SECTIONS section:

```
SECTIONS /* Layout of sections by segments */
{
  text   in name2;
  data   in name1;
  text1  in name1, align page;
  nobits in name2, align page;
  heap   at 0x10020000;
  stack  at 0x00010000;
}
```

The sequence of sections in a segment is determined by several factors, as follows:

- common order of the layout of sections in a segment, when the sections of initialized data are arranged first and then those of non-initialized data,
- at first sections presented in the configuration file are arranged in a segment, and then the remaining ones with a load flag.

The following example shows how the linker will arrange the sections in a segment.

- object file contains the following sections to be loaded:

```
textProc;       (section of initialized data)
bss.dataVector; (section of non-initialized data)
dataVector;     (section of initialized data)
textMain;       (section of initialized data)
dataProc;       (section of initialized data)
bss.dataProc;   (section of non-initialized data)
textProc1;      (section of initialized data)
```

- the configuration file presets the arrangement of several of them in segment VECTORS:

```
SECTIONS
{
  dataVector     in VECTORS;
  bss.dataVector in VECTORS;
  textMain       in VECTORS, align page;
  textProc1      in VECTORS;
}
```

- actual arrangement of sections in the segment set by the linker:

```
//   sections of initialized data.
dataVector;  // from configuration file.
textProc;    // not marked in configuration file.
textMain;    // from configuration file.
textProc1;   // from configuration file.
dataProc;    // not marked in configuration file.
// sections of non-initialized data.
bss.dataVector; // from configuration file.
bss.dataProc1;  // not marked in configuration file.
```

However, the actual arrangement of sections in the segment may differ from the one presented above. What can affect this arrangement is the alignment of section with the border of the memory page. For example, if as a result of sections alignment between sections `dataVector` and `textMain` an unused bucket is formed, the linker may place to it one of the sections not mentioned in the configuration file. This depends on the size of a section inserted and on the requirement of its alignment in the memory. If section `textProc` with consideration for alignment has a size less than that of the unused bucket, final arrangement of the sections in the segment will be as follows:

```
//   sections of initialized data.
dataVector;  // from configuration file.
textProc;    // not marked in configuration file.
textMain;    // from configuration file.
textProc1;   // from configuration file.
dataProc;    // not marked in configuration file.
```

#### 4.11.3.1 How to Name Data Sections

When filing the configuration of data sections names the following circumstance should be taken into account: for each section of initialized data (defined in the assembler by entry word data) the cross-assembler creates a pair section of non-initialized data where all non-initialized data declared in the initialized data section, are transferred (more detailed information see in the document: **NeuroMatrix® NM6403 SDK. Assembly Language Overview**.

If the section of non-initialized data has appeared as a pair of a corresponding initialized section, its name is generated by means of adding prefix "`bss.`" to the beginning of the pair section name. That is, if the section name was "`dataVector`", the name of the pair section of non-initialized data: "`bss.dataVector`". This fact needs to be considered when forming the configuration file.

One of such section pairs may happen to be empty and, even if they both are mentioned in the configuration file, a user, controlling the linker, decides what should be done with the empty section: either leave it in the output file, or remove using key `-d[n]` (see subsections 3.8.1, 3.8.2,.3.8.3).

## *4.12 An Example of Using the Linker*

This example demonstrates the process of assembling object files `demo.elf`, `matrix.elf` and a number of libraries used by them, into a single program - absolute executable file. The input point start is defined in file `startup.elf`.

Assume that the configuration of processor NM6403 physical memory is as follows:

Global Memory:

- Bank 0 from address `0x80000000` to address `0x8003FFFF`
- Bank 1 from address `0x80040000` to address `0x8007FFFF`

Local Memory

- Bank 0 from address `0x00000000` to address `0x0003FFFF`
- Bank 0 from address `0x00040000` to address `0x0007FFFF`

Output sections are formed from the following input sections:

- pairs of sections `MatrixArray` and `bss.MatrixArray` from `matrix.elf` should be arranged in the local memory, and sections Matrix1Array и bss.Matrix1Array in the global memory,
- executable code contained in sections `.text` of files `demo.elf` and `matrix.elf`, is gathered into one output section which needs to be

arranged (by virtue of the task internal reasons) in the local memory in address `0x00000200`,

- from section `.text` function `UDIV32` is called that realizes signless division of 32-bits numbers. The function body is stored in library module `div.elf`, which being part of the time execution library `libc.lib` is located in catalogue *d:\neuro\lib*,
- from section `.text` function `MulVects` is called that realizes scalar multiplying of matrixes. The function body is stored in the library of vector-matrix computations `matvect.lib` located in the current catalogue,
- the program uses the operation with dynamically isolated arrays. With this view functions of execution time libraries `libc.lib` are connected located in library modules `malloc.elf`, `calloc.elf` and `free.elf`.

The configuration file for the linker *demo.cfg* looks in the following way:

```
MEMORY /* physical memory of neuroprocessor */
{
    LOCAL0  at 0x00000000, len = 0x40000, page = 0x4000;
    LOCAL1  at 0x00040000, len = 0x40000;
    GLOBAL0 at 0x80000000, len = 0x40000;
    GLOBAL1 at 0x80040000, len = 0x40000;
}
SEGMENTS
{
    GlobalSeg in GLOBAL0;
    LocalSeg  in LOCAL0;
}
SECTIONS
{
    text at 0x00000200;
    MatrixArray      in LocalSeg;
    bss.MatrixArray  in LocalSeg, align page;
    Matrix1Array     in GlobalSeg;
    bss.Matrix1Array in GlobalSeg, align page;
    stack in LocalSeg, align page;
    heap  in LocalSeg, align page;
    heap1 in GlobalSeg, align page;
}
```

Command file demo.cmd stores all the keys of the linker start-up, the names of input object files, of the configuration file:

```
-a -oOUTFILE.ABS -cdemo.cmd -ld:\system\libs -
mdemo.map
-stack=4096
-heap=65536
-heap1=65536
demo.elf
matrix.elf
```

The command line of the linker startup looks as follows:

```
linker.exe @file.cmd
```

The information of the program arrangement of the in the processor memory can be received by scanning the content of the memory map file `demo.map`:

### *4.13 Linker Error Messages*

In the course of operation, the linker may encounter incorrect data. In this case, it will send a message to the screen. For displaying messages, the linker uses a formatted line. Its format is unified for the whole SDK complex and looks as follows:

**«[file_name]»: error_type  error_number: message_of_error**

Where:

file_name - name of file where an error occurred at the parsing of its internal structure. Field `file_name` can be absent when the error took place outside the block of parsing of input files.

error_type - one of error types presented below in this Section.

error_code - symbolic designation of an error. It is a alphanumeric code whose first three symbols present the abbreviation of the name of component that generated the error message and the rest represent its number. Each SDK component has an independent numbering of errors. Example of the error code: `LNK412`.

message_of_error - short reference information on the reason of the erroneous situation in the course of the particular SDK component operation.

Example:

"file.elf" ERROR LNK412: "Several symbol tables in one file !"

All incorrect situations are divided by the linker into four groups according to four types of errors:

- **Warnings.** A warnings appears when an incorrect situation occurred may affect the final result of the librarian operation, however the librarian has enough information for creating a correct output file. For example, when one of the input directives cannot be used in a particular mode of the librarian operation, the linker issues corresponding warnings. In this case the work on parsing the remaining input parameters continues.
- **Errors.** An error arises when because of incorrect input data a librarian is not capable to create a correct output file. For example, ... . In a situation like this error message is displayed, and the librarian ceases the work and returns a non-zero value.
- **Internal errors.** An internal error may arise due to the incorrect work of a librarian itself. At the appearance of a situation like this all initial data are recommended to be transferred to programmers for eliminating the defects. Under this situation the message of internal error is displayed, and the librarian ceases the work and returns a non-zero value.
- **Fatal errors.** A fatal error appears in case of memory shortage for librarian's work. It testifies to the fact that in a situation turned-out the work of the librarian with given input parameters cannot be continued. The system parameters need to be changed.

Three figures are allocated for an error number, i.e. it is within the range of 0 to 999. Among the types of errors numbers were distributed in the following way:

| | |
|---|---|
| 0 - 399 | Warnings. |
| 400 - 799 | Errors. |
| 800 - 949 | Internal errors. |
| 950 - 999 | Fatal errors. |

### 4.13.1 Warnings

| | |
|---|---|
| LNK001 | "Section"..." should not have a relocation table. Table is ignored." |
| | - a section should not have a relocation table but yet it has it. In this situation the relocation table is ignored. The error has occurred in the process of formation of a particular object file. If later on, the linker will not manage to create an output file, a corresponding error will arise. At this particular stage only a warning is displayed that does not lead to the linker work stoppage. |
| LNK003 | "Default size of heap "..." is equal to 1K of words (32-bit)" |
| | - a warning that a user has not preset the size of a corresponding heap of |

| | |
|---|---|
| | dynamic memory although he used in his program operation functions from the dynamic memory. Since the time execution library does not trace the output outside the heap, the user should himself take care of it. Therefore the warning reminds that the user has not preset a new heap size and its default size is 1K of words. |
| LNK004 | "Wrong parameter "...". It is ignored." |
| | - a warning that a user has preset an incorrect parameter in the command line or command file. This parameter in no way affects the process of execution of its work by the linker. |
| LNK005 | "Repeated key setting "...". It is ignored." |
| | - a warning that a user has repeatedly preset a parameter in the command line or command file. This parameter is ignored.. |
| LNK006 | "To avoid the removal of input file output file has extension ".elz"." |
| | - a warning that a user may spoil the content of an input object file since its name coincides with that of an output one. Such a message is issued in the object file assembly mode (key -elf или -e) with the name of output file not preset. Since by default all object files have extensions ".elf" there is a risk of losing the input file. To eliminate this problem the linker will create an output file with extension ".elz". |
| LNK007 | "In the command file a call of another command file is found. It is ignored." |
| | - a warning that a user is trying to realize a recursive call of the command file. That is to say, in the command file the call of another command file cannot be contained. The linker ignores this directive. |
| LNK008 | "Cannot open command file "...". It is ignored." |
| | - a warning that the linker cannot open the command file specified as a parameter of the command line. This instruction is ignored. |
| LNK010 | "When creating object file key "-d" is ignored." |
| | - a warning that in the mode of object file creation key -d designating the deletion of dead sections is ignored. |

### 4.13.2 Errors

| | |
|---|---|
| LNK401 | "Reference to undefined local symbol "..." from section "..."." |
| | - this error may arise due to the appearance of an error in the input object file whose name is presented at the beginning of the message formatted line. |
| LNK402 | "Section "..." has different flags in different files." |
| | - that means that in one of input object files the header of this section contains the flag of its load to the computer memory whereas in the other file the section does not contain this flag, i.e. is not loadable. The error can be generated by improper operation of the assembler compiler or by a failure in the object file. |

| | |
|---|---|
| LNK403 | "Shortage of space in segment "..." for locating sections." |
| | - this error arises when it is written in the configuration file that segment «...» has a preset size and contains some set of sections, and in the course of the computation of sections size it occurs that there is a shortage of space for them in the segment. The error can be corrected by means of either increasing the segment size or relocating one or several sections to another segment. |
| LNK404 | "Unknown type of reference in section "..."." |
| | - this error arises when in a corresponding field of an object file a meaning different from the expected one is stored. In all there are three types of reference to a symbol: absolute, relative and byte reference. If in a field defining the reference type another meaning is put, this causes an error. The error can result from improper work of a cross-assembler or a failure in the object file. |
| LNK405 | "Input point is not preset "..."." |
| | - this error arises when the linker cannot define the address of the global label defining the input point. By default the input point is named "start". Therefore the linker defines the address of symbol "start" as an input point. With the change of the input point name the linker uses the address of a new global symbol. If the symbol is not defined this error arises. (see par. 3.8.6 Defining input point name (key -start=<label_name>) on p. 3-17) |
| LNK406 | "Structure of this file does not correspond to format ELF." |
| | - this error arises when the linker tries to open a file that is not an object file, or its format does not conform with the one adopted within the base Software. A check is necessary whether it is that file that is supplied to the input of the linker. |
| LNK407 | "This processor type is not supported." |
| | - this error arises when trying to supply to the linker input an object file of ELF format intended for a processor type not supported by this base Software (for anyone except processor NM6403). When this error arises it is necessary to recompile the initial text of the program from where this object file was received. |
| LNK408 | "This file is created with the aid of obsolete version of library of access to ELF." |
| | - when this error arises it is necessary to recompile the initial text of the program from where this object file was received. Probably, a new version of the linker generates object files that turn out to be inconsistent in their internal structure with files received with the aid of the previous linker version. |
| LNK409 | "Several symbol tables are not supported in one file." |
| | - this error may appear when an object file received by the linker as an input parameter contains several symbol tables. This configuration is not forbidden by ELF format, however the base Software of processor NM6403 is |

| | |
|---|---|
| | designed for storing only one symbol table in the object files generated within the processor framework. With the appearance of this error it is necessary to recompile the initial text of the program from where this object file was received. |
| LNK410 | "Unknown type of section ("...")." |
| | - this error may arise when in the structure of an input object file a section was found whose type the linker could not define. In all there are five types of sections encountered in the object files of ELF format: section of initialized and non-initialized data, symbol table, line table and references table. If in the field of a section header a meaning is written that does not correspond to any section type, this error will arise. Should this error appear, it is necessary to recompile the program initial text from where this object file was obtained. |
| LNK411 | "Symbol table is not found." |
| | - this error appears when the linker is unable to find a symbol table in an input object file. The symbol table should exist even in case the initial program contains none of symbols. In this case it consists of one zero element. The appearance of this error requires recompiling the initial text of the program from where this object file was derived. |
| LNK412 | "Table of symbol names is not found." |
| | - this error arises when the linker is unable to find the table of symbol names in the input object file. The table of symbol names should exist even in case the initial program contains none of symbols. In this case it consists of one zero element. The appearance of this error requires recompiling the initial text of the program from where this object file was derived. |
| LNK413 | "Wrong field in the symbol table header." |
| | - this error arises when a field storing the size of a symbol table element is equal to zero. The error may be generated due to the improper operation of the assembler compiler or a failure in the object file. If it arises, the initial text of the program from where this object file was derived, should be recompiled. |
| LNK414 | "Wrong field in header of relocation table "..."." |
| | - this error arises when a field storing the size of a reference table element is equal to zero. The message contains information in what particular table a failure occurred. The error may be generated due to the improper operation of the assembler compiler or a failure in the object file. If it arises, the initial text of the program from where this object file was derived, should be recompiled. |
| LNK415 | "Symbol "..." has different types in different files ." |
| | - this error arises when in one of input object files symbol "..." is declared as a data object and in another file it is referred to as a label (and vice versa, declared as a label and referred to as an object of data). To correct the error a programmer should straighten out the work with this symbol, to bring its |

| | |
|---|---|
| | type into adequacy in various initial files in assembler or C++ language.. |
| LNK416 | "Redefining global symbol "..."." |
| | - this error arises when a global symbol is defined in several object files. Normally global symbols are defined in one file only and in other files it is declared as external. The task of the linker is to identify all references to this global symbol with the symbol itself and to correctly calculate its address in the computer memory. If the global symbol is declared in several locations, the linker is unable to define its address. To correct the error, symbol declaration should be left in one file only (with the use of word global), and in other files to declare it as an external symbol (with the use of key words weak or extern). |
| LNK417 | "Global symbol "..." is not defined." |
| | - this error occurs when in all input object files this global symbol is declared as external (with the use of key words weak or extern), and there is no place where it would be declared global with a location allotted for it. Key word global leads to the allotment of location for the global symbol whereas extern only declares the symbol as external, with no place allotted for it. To correct the error this symbol should be declared as global in one of the files of initial texts. |
| LNK418 | "None of input files is preset." |
| | - this error arises when there is none of input object files in the command line or command file of the linker. |
| LNK419 | "Absolute file cannot be an input parameter." |
| | - this error arises when in the linker command line or command file an absolute executable file is transferred as an input object file. |
| LNK420 | "This type of object file of ELF format is not supported." |
| | - this error arises when an object file created outside the base Software of processor NM6403 is supplied to the linker input. If it appears, the initial text of the program from where this object file was obtained, should be recompiled. |
| LNK421 | "common-symbol "..." has type "label"." |
| | - this error arises when a common-symbol of «label» type is declared in the program in assembler language (example: common AAA: label; ). Common-symbol may only have «data» type. In case of this error a correction should be introduced to the file in assembler language. |
| LNK422 | "Symbol "..." stores wrong type of a section." |
| | - error in an input object file. Each symbol of the symbol table apart from the information of a file where it is defined contains the information on an input section where a location was allotted for it. This information is stored in the form of an index in the table of input file section. If this index indicates a section that is not a data section, this error arises (function issuing the error - CalcSymAddr). Probably, the error occurred due to a failure that took place at creating an input object file. It is necessary to create the object file anew |

| | |
|---|---|
| | with the aid of the assembler. If this does not lead to the error elimination this problem should be passed for solution to the programmers of the assembler compiler. |
| LNK423 | "Reference from section "..." to a symbol defined in auxiliary section." |
| | - error in an input object file. In the relocation table of the said data section a reference was found to the symbol defined in the auxiliary section, i.e. in the table of symbol names. |
| LNK424 | "Quantity of undefined symbols " |
| | - this information line sums up the search of definitions of undefined global symbols in library files. It arises only in case there is at least one undefined global symbol left. |

### 4.13.3 Fatal Errors

| | |
|---|---|
| LNK951 | "Error at request of ... memory bytes." |
| | - this error occurs when there is not enough space for the linker operation. Attempt to allocate the following array in the field of dynamic memory for locating internal structures ends in a failure. As a result the work cannot be continued. To correct the situation it is necessary to somehow deallocate the dynamic memory, for example, to unload some resident programs or to increase the free space on the disc where a system temporary file is created. |
| LNK952 | "Cannot create output file "..."." |
| | - this error arises due to some failures in the operational system when creating a file. For instance, this error may be caused by a space shortage on the disc or the absence of a catalogue where a user wishes to record a file, or a ban on filing in certain catalogues imposed by the operational system. Before continuing the work with the linker the reason for the error should be found or another place on the disc should be chosen to write down the file. |
| LNK953 | "Cannot open file "..."." |
| | - this error arises due to the problems generated by an operational system. For example, at an attempt to open a file used at the same time by other applications. Before going on the work with the linker one should realize what the reason for the error was. |
| LNK954 | "File "..." is not found." |
| | - this error arises due to the absence of an input file with this name. |

# 5 Librarian

## 5.1 Introduction

This chapter describes the interface, control parameters and operation modes of a librarian of object files from NM6403 SDK. Full description of librarian capabilities and the methods of work with is presented with including information on all keys and on using command files. A list of errors is also presented here that is found or diagnosed by the librarian. With the purpose of better comprehension examples of operation with the object file librarian are cited.

## 5.2 Librarian Features

The librarian of object files is an auxiliary means in the set of NeuroMatrix® NM6403. This program is intended for the work with object file libraries of ELF format, these files being obtained with the aid of assembler and linker of NeuroMatrix® NM6403 SDK.

The librarian is not designed for the work files of ELF format created by other program means. The program allows the following:

- to create libraries from the sets of object files,
- to list the content of libraries,
- to add, delete and replace files in libraries,
- to extract object files from libraries.

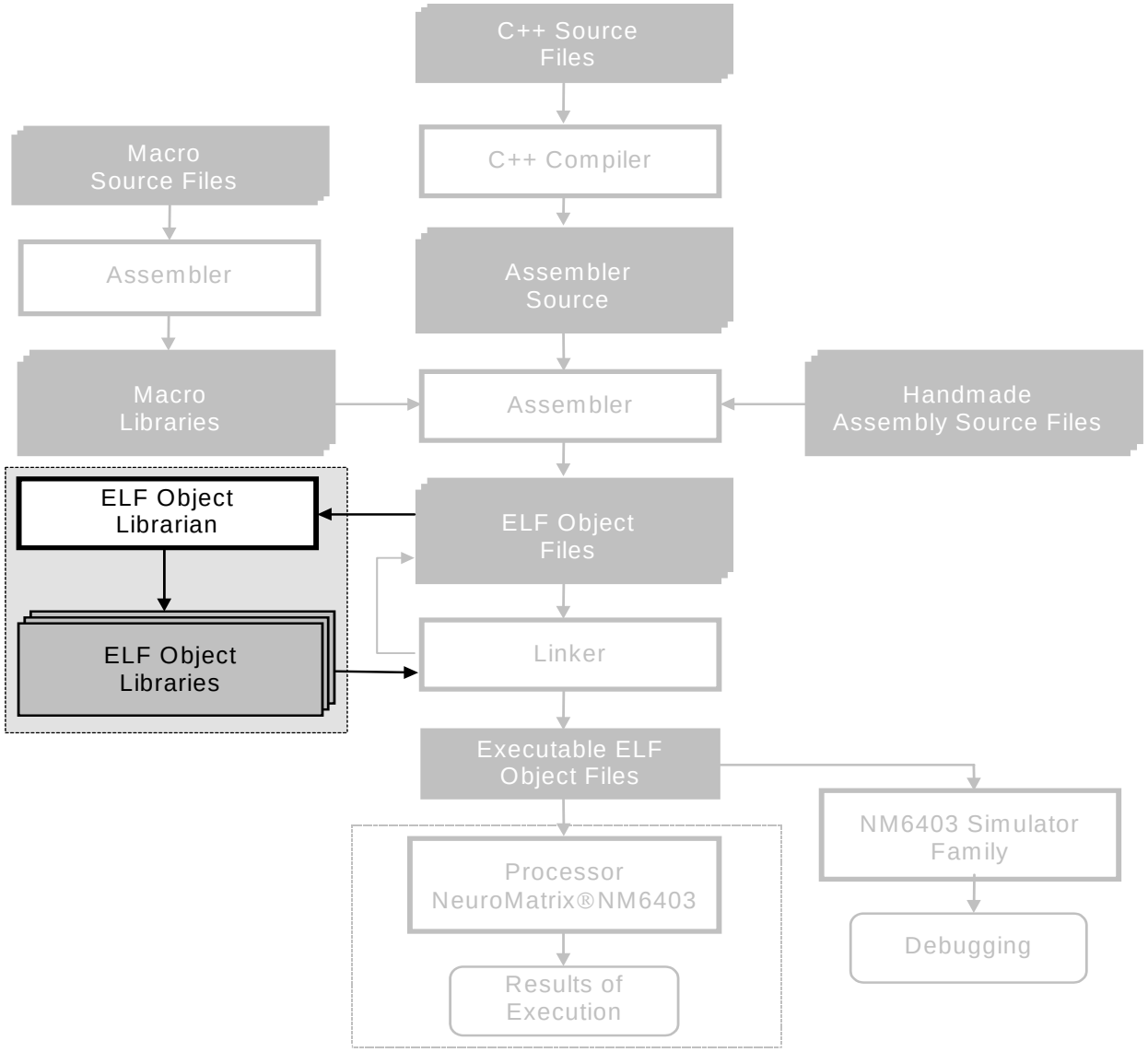
The librarian of object files is console application Windows95/NT.

The librarian of object files is oriented to operation only with files of ELF format. It fully supports this format with the exception of creating and processing dynamic libraries.

## 5.3 Librarian Development Flow

Figure 5-1 illustrates the position of the librarian in the SDK structure and its place in the process of developing application programs for processor NeuroMatrix® NM6403.

*Figure 5-1. Librarian Development Flow*




### *5.4 Invoking the Librarian*

To invoke the librarian, enter:

**libr** [*command*] [*archive_name* [*files_list*]]

where:

**libr** Name of librarian executable file.

*command* Parameter controlling the librarian operation mode.

*archive_name* Name of library the librarian works with.

*files_list* List of object files added to or extracted from the library.

## *5.5 Librarian Options*

To control the librarian operation a set of parameters (keys) is used.

**All parameters are preceded with prefix "-".**

They can be specified in a command line or command file (see 5.6 on page 5-7).

Table 5-1 presents the list of librarian control parameters.

*Table 5-1. List of Librarian Options*

| OPTION | DESCRIPTION |
|---|---|
| `-c` [create] | Creating library of object files. |
| `-a` [add] | Adding files to the object file library. |
| `-r` [replace] | Replacing files in the object file library. |
| `-d` [delete] | Deleting files from the object file library. |
| `-e` [extract] | Extracting files from the object file library. |
| `-l` [list] | Viewing the content of the object file library. |
| `-h` [help]/`-?` | Short information of the program use. |

### 5.5.1 Creating the Library (-c Option)

Under this mode the name of a library created and names of object files enabled should be specified. When specifying the names of object files the use of wildcards: '*' and '?' is permitted. If the name of an existing file is used as a library name, this file first will be removed.

### 5.5.2 Adding Files to the Library (-a Option)

This mode is similar to the previous one except that in case of the existence of a library with a specified name this library will not first be eliminated, the such object files will be added to the library. If, when trying to add a file to the library, it occurs that the file with such name already exists in the library, the addition will not take place.

### 5.5.3 Replacing Files in the Library (–r Option)

The mode is similar to mode add except that the object files specified in the command file will be added to the library regardless of the availability or non-availability of files of the same name in the library.

### 5.5.4 Deleting Files from the Library (-d Option)

In the deleting mode the name of the library and the files being deleted should necessarily be specified in the command line. If there is no file with the specified name in the library, the librarian issues a warning and continues operation.

### 5.5.5 Extracting Files from the Library (-e Option)

This mode makes it possible to extract modules from the library and to place them to individual object files. The library in this case is not changed. When extracting, the library name should be specified in the command line. If the names of the object files are not specified, all modules will be extracted from the library. When extracted, the files created receive current time.

### 5.5.6 Viewing the Content of the Library (-l Option)

The list of library elements is displayed with the indication of name, time and size. The key itself may be not specified in the command line, for example:

```
libr library.lib
```

### 5.5.7 Printing Out Reference Information (-h/-? Option)

Short information is also given at the program start-up without parameters or with keys `-h`, `-?`. Its view is as follows:

*Figure 5-2. The Librarian Reference Information*

```
Librarian * v1.02 * (c) 1997-99 * RC Module.

Usage: libr [command] [libname] [filelist]
   command  - running mode
   libname  - library name
   filelist - object files list
Options list:
   -h or -?   - short help (this one)
   -l         - list library files
   -c         - cteate library
   -a         - add files in library
   -d         - delete files from library
   -e         - extract files from library
   -r         - update files in library
Notes:
   1. Default running command is '-l'.
   2. '-c' mode usage will delete old file with the same name.
   3. In '-e' mode all files will be extracted if filelist is not specified.
   4. Extracting file deletes old one with the same name.
   5. In '-d' or '-r' mode it creates a temporary copy of library.
```

### 5.6 Using the Command File

A part of librarian parameters can be placed to the command file. In this case the command line looks as follows:

```
libr @command_file_name.
```

Several command files may exist:

```
libr @command_file1 @command_file2.
```

In addition, only a part of the command line can be placed into the command file, for example:

```
libr  @command_file_name.
```

Nested command files are illegal and are not processed.

A command file is a text file containing admissible parameters divided by an arbitrary number of dividers. Spaces, symbols of tabulation and line transfer can be used as dividers.

Example of a command file.

```
-c mylib.lib
file1.elf file2.elf
file3.elf file4.elf
```

When calling a librarian with a particular command file a library `mylib.lib` will be created that will incorporate the above-listed object files.

### 5.7 Using Wildcards

The use of wildcards as the names of object files is allowed in the commands of creating (`-c`), adding (`-a`) and replacing (`-r`). In the extraction command the use of wildcards is not allowed, however the names of object files can be omitted, in this case, all files are extracted. In the rest of cases the use of wildcards is impermissible.

### 5.8 Examples of Invoking the Librarian

Table 5-2 presents various versions of invoking the librarian.

*Table 5-2. Examples of Invoking the Librarian*

| COMMAND LINE | DESCRIPTION |
|---|---|
| libr | Short information of program use |
| libr -h | Short information of program use |
| libr -? | Short information of program use |
| libr libname [filelist] | Viewing the list of library files. |
| libr -l libname [filelist] | Viewing the list of library files. |

| | |
|---|---|
| libr -c libname [filelist] | Creating the library. |
| libr -a libname [filelist] | Adding files to the library. |
| libr -d libname [filelist] | Deleting files from the library. |
| libr -e libname [filelist] | Extracting files from the library. |
| libr -r libname [filelist] | Replacing files in the library. |

## *5.9 An Example of Using the Librarian*

### 5.9.1 Creating the Library

```
libr -c first.lib *.elf
```

This command will create the library with name `first.lib` and include in it all the files from the current catalogue that have extension `elf`. If the file with name `first.lib` already exists in the current catalogue, it will first be removed.

The cited command can be also written in the form as follows:

```
libr -c first *.elf,
```

since extension `.lib` is used by the librarian for default libraries. If there is a need to create a library without extension, a point should be put after the library name:

```
libr -c first. *.elf
```

This command will create library with name `first`.

### 5.9.2 Adding Object Files to the Library

```
libr -a first.lib *.elf
```

If there is file `first.lib` in the current catalogue, it will be checked for its being a library of established format, and then all object files with a corresponding extension and of ELF format from the current catalogue will be added to it.

If there is no file `first.lib` in the current directory, the action of this command does not differ in anything from that considered in subsection 5.9.1 on page 5-8.

### 5.9.3 Extracting Object Files from the Library

```
libr -e library.lib single.elf
```

As a result of action of this command a file with name `single.elf` will be extracted from library `library.lib`.

The library will not be changed at this operation.

#### 5.9.3.1 Extracting All Files from the Library

```
libr -e library.lib *.elf
```

As a result of action of this command all files will be extracted from library `library.lib`.

The library will not be changed at this operation.

Despite the fact that the files can be placed into the library with the memorizing of a relative or absolute path, the path, when extracting, is cut off from the file name, and all the files are placed in the current catalogue. If two files with similar names (at different paths) were contained in the library, when extracting these two files simultaneously, the second file will override the first one. In this particular situation files should be extracted one at a time specifying the full name of the file (including the path).

### 5.9.4 Replacing a File in the Library

```
libr -r lib a1.elf
```

As a result of action of this command file `a1.elf` delivered as an input parameter will replace the file with the same name in library `lib`. If there is no such file in the library, a corresponding warning will be displayed, and the file will be added to the library.

### 5.9.5 Deleting a File from the Library

```
libr -d lib a1.elf
```

As a result, of action of this command file `a1.elf` deliverd as an input parameter will be deleted from the library lib. If there is no such file in the library, a corresponding warning will be displayed, and an output file with name a1.elf will not be created. To delete several files from the library, they all should be listed in the command line.

## *5.10 Librarian Error Messages*

In the course of operation the librarian may encounter incorrect data. . In this case it will display a message to the screen. For displaying messages the librarian will use a formatted line. Its format is unified for the entire SDK complex and looks as follows:

**«[filename]»: errortype  errornumber: error_message.**

Where:

`filename` - file name at whose internal structure parsing an error occured. Field `filename` can be absent when the error occured outside the block of input files parsing.

`errortype` - one of the types of errors given below in this section.

`errorcode` - symbolic designation of error. It is an alpha-numeric code whose first three symbols represent the name abbreviation of a component that generated the error

message, and the rest are its number. Each SDK component has an independent error numbering. Example of the error code: `LBR412`.

`error_message` - short reference information on the reason for an erroneous situation in the course of operation of this SDK component.

For example:

```
ERROR LBR407: Library "AAA.ELF" is not found.
```

All incorrect situations are divided by the librarian into four groups that correspond to four types of errors:

- **Warnings.** A warnings appears when an incorrect situation occurred may affect the final result of the librarian operation, however the librarian has enough information for creating a correct output file. For example, when one of the input directives cannot be used in a particular mode of the librarian operation. The work in this case does not stop.
- **Errors.** An error arises when because of incorrect input data a librarian is not capable to create a correct output file. In a situation like this error message is displayed, and the librarian ceases the work and returns a non-zero value.
- **Internal errors.** An internal error may arise due to the incorrect work of a librarian itself. At the appearance of a situation like this all initial data are recommended to be transferred to programmers for eliminating the defects. Under this situation the message of internal error is displayed, and the librarian ceases the work and returns a non-zero value.
- **Fatal errors.** A fatal error appears in case of memory shortage for the the librarian work. It is an indication that in a situation occurred the work of the librarian with given input parameters cannot be continued. The system parameters need to be changed.

### 5.10.1 Warnings

| | |
|---|---|
| `LBR001` | "Error at receiving time of file ... creation, current time is set. " |
| | - impossible to calculate the time of file creation. When recording, current time is put in the library. |

### 5.10.2 Errors

| | |
|---|---|
| `LBR401` | "Controlling input parameter is missing." |
| | - the command of librarian control is absent (first parameter). |
| `LBR402` | "Library name not defined." |
| | - library name is missing. |

| | |
|---|---|
| LBR403 | "Cannot read input parameter number: ..." |
| | - parameter is absent. Parameter numbering is conducted from the librarian control command that is defined as the first librarian parameter. |
| LBR404 | "Unknown command ..." |
| | - wrong command. |
| LBR405 | "File ... is not a library." |
| | - wrong format of the library |
| LBR406 | "File ... is not a library (...)." |
| | - wrong format of the library; In contrast to LBR403 in this case original message of library LIBELF is displayed. |
| LBR407 | "File... already exists in the library." |
| | - file with this name is already in the library. |
| LBR408 | "Symbol ... (file ...") is already defined in the library." |
| | - impossible to add file to the library since the file contains a global symbol with a name that already exists in the library symbol table. The names of the symbol and of the file are indicated that could not be added to the library. |
| LBR409 | "Library ... not found." |
| | - library with preset name is not found in such catalogue |

### 5.10.3 Fatal Errors

| | |
|---|---|
| LBR950 | "Error at request ... bytes of memory" |
| | - this error occurs when there is not enough memory for the librarian operation. Attempt to allocate a following array in the field of dynamic memory for locating internal structures ends in a failure. As a result the work cannot be continued. To correct this situation it is necessary to deallocate the dynamic memory, for example, to unload some resident programs or to increase the free space on the disc where a system temporary file is created. |

# 6 Decoder of Object and Executable Files

## 6.1 Introduction

This chapter describes NM6403 Decoder of Object and Executable Files, its interface, its command line options and execution modes. Below in this manual the NM6403 Decoder of Object and Executable Files is referred to as the dumper.

## 6.2 The Dumper Overview

The dumper is intended to decode the contents of object and executable ELF files. It allows the user to view object ELF files, object libraries and executable ELF files.

The dumper is a command line application.

## 6.3 Invoking the Dumper

To invoke the dumper, enter:

**dump** [*options*] *input_file* [*>output_file*]

| | |
|---|---|
| **Dump** | This is the command that invokes the dumper |
| *options* | This is a parameter list of dumper control. Can be arranged in any place of a command line, in arbitrary sequence. (More detailed discussion is presented below). |
| *input_file* | This is a name of the binary object file that the dumper uses as input. |
| *>output_file* | This is a name of the text file where the dumper output information will be redirected. If the file is not specified, all data will be printed out on screen. |

The user is allowed to specify one or more binary input files and command line options in order to determine the dumper operation.

The dumper returns information to the standard output. Its output information can be redirected to a file, for instance:

```
dump MyApp.abs >MyApp.dmp
```

If the dumper is invoked without any command line options or input filenames, it provides the list of options and all necessary reference information.

If a filename is only one command line option, the dumper prints all information about the file structure (as shown in the example above).

### 6.4 The Dumper Options

The following options control the dumper's behavior:

| Option | Description |
|---|---|
| `-h` | The dumper omits the program header. |
| `-e` | The dumper omits the ELF file header. |
| `-p` | The dumper omits the contents of the program segment table. |
| `-s` | The dumper omits the contents of the section header table. |
| `-d` | The dumper hides the contents of sections. |
| `-a` | Address unit equals to 32-bit word (by the default for executable files). |
| `-b` | Address unit equals to 8-bit words (by the default for object files). |
| `-f` | The dumper hides the contents of library files. Only for libraries. |
| `-w` | The dumper prints DWARF sections as data sections, not as symbols. |

### 6.5 Processing Special Sections

The sections that begin with "`text`" or "`.text`" are treated as code sections. The dumper disassembles them and presents them in the following form:

*Figure 6-1. Fragment of Disassembled Code Section*

```
Section .text
0000000c: 0000e63f             *[ar7++]=gr6,ar6 with  nul
0000000d: 00008771             *ar6=ar7 set with  nul
0000000e: 0000d747 02000000    *ar7+=00000002 with  nul
00000010: 0000e03f             *[ar7++]=gr0,ar0 with  nul
00000011: 0000e13f             *[ar7++]=gr1,ar1 with  nul
00000012: 00000040 9c000000    *ar0=0000009c set with  nul
00000014: 0000274a 98000000    * call 00000098 with  nul
00000016: 00001050             *nul with  nul
00000017: 00001050             *nul with  nul
  ...       ...                        ...
```

| address | mashine code | disassembled source |
|---|---|---|

If the user needs to disassemble a code section, it is necessary to call it the following way: “.text*MySectionName*”, for instance: “.textLocal” or simply “.text”. The other code sections are not disassembled and they are presented in an output file as an array of binary code. All codes and addresses are stored in hex format.

The sections with the names beginning with “.dwarf” are treated as the sections of debugging information. If the –w option is not set they will be interpreted by dumper and output in symbolic form. Otherwise, the debugging information will be presented in form of a binary array.

## 6.6 An Example of the Decoded ELF File

Here is an example of the decoded executable file:

```
--------------------------
File: test0.abs

ELF Header:
Identification bytes:
File class:   32 bits class
Data encoding:       low significant byte first
ELF version:  Current version

File type:    Executable absolute file
Target machine:      NeuroMatrix
ELF version:  Current version

Entry address:                              0xc0000000
Program Segment Header Table offset:        0x34
Section Header Table offset:                0x238
```

```
User define flags:                          0x0
ELF header size:                            0x34
Program Segment Header size:                0x20
Number of Program Segments:                 3
Section Header size:                        0x28
Number of Sections:                         5
Index of .shstrtab sections:                1

Program Segment 0
---------------------------------
Type:  Loadable
File offset:                0xb8
Begin virtual address:      0x0
Begin physical address:     0xc0000000
Size in file:               0xe0
Size in memory:             0xe0
Segment flags:              0x0
Alignment:                   8

Program Segment 1
---------------------------------
Type:  Loadable
File offset:         0x198
Begin virtual address:      0x0
Begin physical address:     0xc0000038
Size in file:               0x18
Size in memory:             0x18
Segment flags:              0x0
Alignment:                   8

Program Segment 2
---------------------------------
Type:  Loadable
File offset:                0x1b0
Begin virtual address:      0x0
Begin physical address:     0xc000003e
Size in file:               0x88
Size in memory:             0x88
Segment flags:              0x0
Alignment:                  8

Section: .shstrtab index:1
-----------------------------
Type:  String table
Flags:
Load address:        0x0
Offset in file:      0x94
```

```
Size:                0x22
Link:                0
Info:                0
Align address:       1
Entity size:         0

Section: .text_init index:2
----------------------------
Type:  Program defined bytes
Flags:        Allocate memory in process image
Load address:        0xc0000000
Offset in file:      0xb8
Size:                0xe0
Link:                0
Info:                0
Align address:       8
Entity size:         0

Section: .text index:3
----------------------------
Type:  Program defined bytes
Flags:        Allocate memory in process image
Load address:        0xc0000038
Offset in file:      0x198
Size:         0x18
Link:         0
Info:         0
Align address: 8
Entity size:  0

Section: .data index:4
----------------------------
Type:  Program defined bytes
Flags:        Allocate memory in process image
Load address:        0xc000003e
Offset in file:      0x1b0
Size:                0x88
Link:                0
Info:                0
Align address:       8
Entity size:         0

Section .shstrtab
.shstrtab
.text_init
.text
.data
```

```
Section .text_init
c0000000: 00002748 0e0000c0    * goto c000000e with  nul
c0000002: 00001050             *nul with  nul
c0000003: 00001050             *nul with  nul
c0000004: 00001050             *nul with  nul
c0000005: 00001050             *nul with  nul
c0000006: 0000274c fefffffe    * skip fffffffe with  nul
c0000008: 00001050             *nul with  nul
c0000009: 00001050             *nul with  nul
c000000a: 00000000             * nul
c000000b: addeadde              illegal instruction with gr5 =  true
c000000c: 00000000             * nul
c000000d: 00001050             *nul with  nul
c000000e: 00008043 fff780e8    *gmicr=e880f7ff set with  nul
c0000010: 00f50300             *rep 32  vnul with 0
c0000011: 00001050             *nul with  nul
c0000012: 00001040 20000000    *nul 00000020 with  nul
c0000014: 00004043 93d625e2    *lmicr=e225d693 set with  nul
c0000016: 00000043 f4fcffff    *t0=fffffcf4 set with  nul
c0000018: 0000404f 00c00cc0    *pswr set c00cc000 with  nul
c000001a: 00004047 00c00000    *pswr reset 0000c000 with  nul
c000001c: 00001050             *nul with  nul
c000001d: 00001050             *nul with  nul
c000001e: 00001050             *nul with  nul
c000001f: 00001050             *nul with  nul
c0000020: 00001050             *nul with  nul
c0000021: 00001050             *nul with  nul
c0000022: 00001050             *nul with  nul
c0000023: 00001050             *nul with  nul
c0000024: 00001050             *nul with  nul
c0000025: 00001050             *nul with  nul
c0000026: 00000044 defadefa    *gr0=fadefade set with  nul
c0000028: 0000c041 3e0000c0    *ar7=c000003e set with  nul
c000002a: 0000274a 380000c0    * call c0000038 with  nul
c000002c: 00001050             *nul with  nul
c000002d: 00001050             *nul with  nul
c000002e: 00001062 0a0000c0    *[c000000a]=gr0 with  nul
c0000030: 00004044 17000000    *gr1=00000017 set with  nul
c0000032: 00001162 0c0000c0    *[c000000c]=gr1 with  nul
c0000034: 00002748 060000c0    * goto c0000006 with  nul
c0000036: 00001050             *nul with  nul
c0000037: 00001050             *nul with  nul

Section .text
c0000038: 00000044 00000000    *gr0=00000000 set with  nul
c000003a: 0000f703             * return with  nul
```

```
c000003b: 00001050             *nul with  nul
c000003c: 00001050             *nul with  nul
c000003d: 00001050             *nul with  nul

Section .data
c000003e: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00................
c0000042: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00................
c0000046: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00................
c000004a: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00................
c000004e: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00................
c0000052: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00................
c0000056: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00................
c000005a: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00................
c000005e: 00 00 00 00 00 00 00 00                        ........
```

# 7 Instruction Level Simulator

## 7.1 Introduction

This chapter describes NM6403 Instruction Level Simulator, its interface, its command line options and execution modes. Below in this manual NM6403 Instruction Level Simulator is referred to as the simulator.

## 7.2 About NM6403 Simulator

The simulator is used to simulate the operation of NM6403. The simulator executes the user program code likewise NM6403 but it doesn't simulate run-time characteristics. A result of an execution any user program by simulator is the same as by NM6403. The simulator is the command line application.

## 7.3 Invocation of Simulator

To invoke the simulator, enter:

```
emurun [options] file_name
```

The user is allowed to specify one executive file with command line options. If the simulator is invoked without input filename, it returns the list of options and all necessary reference information. The simulator outputs information to the standard output.

Example: `emurun Myapp.abs`

## 7.4 Simulator's Options

The following options control the operation of the simulator:

**Table 7-1. List of Simulator's Options**

| Option | Description |
|---|---|
| `-S` | The simulator checks parity of the stack pointer (`ar7`). |
| `-B` | The simulator checks silicon bugs. |
| `-m<value>` | This option set a memory bank size (64000x32 bits by default). |

### 7.4.1 Checking Parity of the Stack Pointer (option -S)

The option `-S` turns on checking parity of the stack pointer (register `ar7`). One of the NM6403 features is that variables occupying 64 bits cannot be arranged by memory odd addresses. At interruption processing register pair `PC,PSW` is written into the stack therefore **the stack pointer in user program when interrupt is enabled must be aligned to the border of a 64-bits word and be even.**

At delayed jump, the stack pointer may be odd actually when interrupt is enabled because interrupt processing occurs just before the execution of all delayed instructions.

### 7.4.2 Checking Silicon Bugs (option -B)

The option `-B` sets the checking of the silicon bugs which have been arisen from realization of processor. If the simulator happens on a silicon bug, it types a message.

Only the fourth bug of the scalar processor is diagnosed. The silicon bugs of the vector processor are not detected.

### 7.4.3 Memory Size Option -m

The option `-m` sets the memory bank size. It dimensions in short words. The minimum size equals 64000 of 32-bit words (it sets by default). The key `-m` defines the size of one simulated memory bank, but the size is the same for all four banks. Therefore, a total memory size is four times lager.

Example: `emurun Myapp.abs –m128000`

## *7.5 Memory Configuration*

The simulator has four memory banks of the equal size, which are arranged from next initial address:

- 00000000 – bank 0 of local memory;
- 40000000 – bank 1 of local memory;
- 80000000 – bank 0 of global memory;
- c0000000 – bank 1 of global memory.

The size of banks may be changed by option ‘-m‘.

**The memory access time memory equals to one cycle.**

## *7.6 User Program Requirements*

User program must conform to the memory configuration of the simulator.

### 7.6.1 Breakpoint

By convention for the simulator, the address of the breakpoint is equal to <entry point>+6. The returned value considered the register gr0 value. C++ program uses the start-up code from the run-time library and answers to the requirements automatically.

## *7.7 Processing Speed*

The processing speed depends on user hardware and its capacity. On PC with Pentium II 300 MHz by control Windows'95 without background user processes the processing speed amounts to 300000 scalar instructions per second and 100-10000 vector instructions per second. The processing rate of the vector instructions strongly depends on vector commands itself.

# 8 Accurate Cycle Simulator

## 8.1 Introduction

This chapter describes NM6403 Cycle Accurate Simulator, its interface, its command line options and execution modes. Below in this manual NM6403 Cycle Accurate Simulator is referred to as the cycle simulator.

## 8.2 About NM6403 Cycle Simulator

The cycle simulator is intended to simulate the operation of NM6403 and characterization the run time of user programs. The cycle simulator executes the user program code likewise NM6403. Moreover it simulates run-time characteristics such as varies delays. However the memory access time is fixed, and equals to one cycle. A result of the execution any user program by cycle simulator is the same as by NM6403. A difference between the run time on simulator and NM6403 maybe exists only for a fragment of the user program, which depends on the pipeline and the memory access time. The cycle simulator is the command line application.

## 8.3 Invocation of Cycle Simulator

To invoke the cycle simulator, enter:

```
temu [options] file_name
```

The user is specifying one executive file with command line options. If the cycle simulator is invoked without input filename, it provides the list of options and all necessary reference information.

Example: `temu –l –s -b Myapp.abs`

## *8.4 Cycle Simulator's Options*

The following options control the operation of the cycle simulator:

**Table 8-1. List of Cycle Simulator's Options**

| Option | Description |
|---|---|
| `-l<file_name>` | This option specifies a file name of execution trace. |
| `-s<file_name>` | This option specifies a file name of execution statistics. |
| `-b<file_name>` | This option specifies a file name of global bus event trace. |
| `-m<value>` | This option set a memory bank size (256Kx32 by default). |

### 8.4.1 Output Option: -l, -s, -b

- `-l` option sets a file name of execution trace;
- `-s` option sets a file name of execution statistics;
- `-b` option sets a file name of global bus event trace.

If the option is absent, the respective information returned to the standard output. If the option placed without a file name, the information doesn't output.

### 8.4.2 Memory Size Option -m

The option '-m' sets the memory bank size. It dimensions in short words. The minimum size equals 262144 of 32-bit words (set by default). The key '-m' defines the size of one simulated memory bank, but the size is the same for all four banks. Therefore a total memory size is four times lager.

## *8.5 Memory Configuration*

The cycle simulator has four memory banks of the equal size, which are arranged from next initial address:

- 00000000 – bank 0 of local memory;
- 40000000 – bank 1 of local memory;
- 80000000 – bank 0 of global memory;
- c0000000 – bank 1 of global memory.

The size of bank may be changed by option '-m'.

**The memory access time for memory equals to one cycle.**

## *8.6 User Program Requirements*

User program must conform to the memory configuration of the cycle simulator.

### 8.6.1 Breakpoint

By convention for the cycle simulator the address of the breakpoint is equal to <entry point>+6. The returned value considered the register gr0 value. C++ program uses the start-up code from the run-time library and answers to the requirements automatically.

## *8.7 Processing Speed*

The processing speed depends on user hardware and its capacity. On PC with Pentium II 300 MHz by control Windows'95 without background user processes the processing speed amounts to 300000 scalar instructions per second and 100-10000 vector instructions per second. The processing rate of the vector instructions strongly depends on vector commands itself.

## *8.8 Examples of Trace Output*

### 8.8.1 Trace Analysis of Events on Peripheral Buses of NM6403

**Table 8-2. Trace Output of Events on Peripheral Bus**

| Tick Number | Local Bus 00000000-7FFFFFFF | | | Address on Local Bus | Global Bus 80000000-FFFFFFFF | | | Address on Global Bus | Processor Internal Bus Occupation Type |
|---|---|---|---|---|---|---|---|---|---|
| | High Word | Low Word | Flag | | High Word | Low Word | Flag | | |
| 1 | 00000008 | 48270000, | r | 00000000 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 2 | 50100000 | 50100000, | r | 00000002 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 3 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz | |
| 4 | 000001a2 | 4bd70000, | r | 00000008 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 5 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz | |
| 6 | 00000016 | 4a270000, | r | 0000000a | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 7 | 50100000 | 50100000, | r | 0000000c | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 8 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz | |
| 9 | 00000302 | 4a100000, | r | 00000016 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 10 | ffffffff | ffffffff, | w | 000001a2 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | OUTPUT |
| 11 | 00000000 | 0000000c, | w | 000001a2 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | OUTPUT |
| 12 | 80000000 | 4b140000, | r | 00000018 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 13 | 9f13e000 | 9c13f518, | r | 0000001a | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 14 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz | |
| 15 | 9f13e000 | 9c13f518, | r | 0000001c | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 16 | e8dbcec1 | b4a79a8d, | r | 00000302 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 17 | 9f13e000 | 9c13f518, | r | 0000001e | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 18 | 50433629 | 1c0f02f5, | r | 00000304 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 19 | 50361c02 | e8ceb49a, | r | 00000306 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 20 | 2006ecd2 | b89e846a, | r | 00000308 | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 21 | b8916a43 | 1cf5cea7, | r | 0000030a | zzzzzzzz | zzzzzzzz, | z | zzzzzzzz, | INPUT |
| 22 | f0c9a27b | 542d06df, | r | 0000030c | e8dbcec1 | b4a79a8d, | w | 80000000, | INPUT, OUTPUT |
| 23 | 20ecb884 | 501ce8b4, | r | 0000030e | 50433629 | 1c0f02f5, | w | 80000002, | INPUT, OUTPUT |
| 24 | c08c5824 | f0bc8854, | r | 00000310 | 50361c02 | e8ceb49a, | w | 80000004, | INPUT, OUTPUT |
| 25 | 884706c5 | 844302c1, | r | 00000312 | 2006ecd2 | b89e846a, | w | 80000006, | INPUT, OUTPUT |
| 26 | 904f0ecd | 8c4b0ac9, | r | 00000314 | b8916a43 | 1cf5cea7, | w | 80000008, | INPUT, OUTPUT |
| 27 | f0a25406 | b86a1cce, | r | 00000316 | f0c9a27b | 542d06df, | w | 8000000a, | INPUT, OUTPUT |
| 28 | 6012c476 | 28da8c3e, | r | 00000318 | 20ecb884 | 501ce8b4, | w | 8000000c, | INPUT, OUTPUT |
| 29 | 58fda247 | ec9136db, | r | 0000031a | c08c5824 | f0bc8854, | w | 8000000e, | INPUT, OUTPUT |
| 30 | 30d57a1f | c4690eb3, | r | 0000031c | 884706c5 | 844302c1, | w | 80000010, | INPUT, OUTPUT |
| 31 | c058f088 | 20b850e8, | r | 0000031e | 904f0ecd | 8c4b0ac9, | w | 80000012, | INPUT, OUTPUT |
| 32 | 009830c8 | 60f89028, | r | 00000320 | f0a25406 | b86a1cce, | w | 80000014, | INPUT, OUTPUT |
| 33 | 28b33ec9 | 54df6af5, | r | 00000322 | 6012c476 | 28da8c3e, | w | 80000016, | INPUT, OUTPUT |
| 34 | d05be671 | fc87129d, | r | 00000324 | 58fda247 | ec9136db, | w | 80000018, | INPUT, OUTPUT |

When there are not events on the bus (bus is in the third state) ‘zzzzzzzz’ is output into the trace. It is used next notation convention in trace for flags:

- `r` – reading from memory;
- `w` – writing into memory;
- `z` – no operation (bus is in the third state).

Names of occupied buses are printed in last column of trace.

### 8.8.2 Trace Analysis of Execution of User Program

The example of an execution trace and comments are presented below.

**Example of execution trace**

```
193 tick, Addr:80004467  ;rep  1  nul ,wtw with nul
194 tick, Addr:80004462  ;rep 32 data = [ar4++gr4] ,ftw with vsum 0,
data, afifo
195 tick, Addr:80004463  ;rep  8 wfifo = [ar0++] with nul
197 tick,       Addr:800044f8 Delay executing for 1 tick(s)
8 ticks ar0-ar3 address unit are busy
196 tick,       Addr:80004464 Delay decoding for 8 tick(s)
204 tick, Addr:80004464  if !(N|Z) goto gr2;gr0 = gr0-1
205 tick, Addr:80004465  nul;gr6 = gr6+gr5 noflags
20 ticks        ar4 used in long operation
206 tick,       Addr:80004466 Delay decoding for 20 tick(s)
226 tick, Addr:80004466  ar4=gr6 set; nul
227 tick, Addr:80004467  ;rep  1  nul ,wtw with nul
228 tick, Addr:80004468  ar0=8000 4488 set; nul
230 tick,       Addr:00000000 Delay executing for 1 tick(s)
229 tick,       Addr:8000446a Delay loading for 1 tick
230 tick, Addr:8000446a  ;rep 32 data = [ar4++gr4] ,ftw with vsum 0,
data, afifo
231 tick, Addr:8000446b  ;rep  8 wfifo = [ar0++] with nul
233 tick,       Addr:80004488 Delay executing for 1 tick(s)
232 tick, Addr:8000446c  nul;gr6 = gr6+gr5 noflags
7 ticks weight bus is busy
22 ticks        ar4 used in long operation
233 tick,       Addr:8000446d Delay decoding for 29 tick(s)
262 tick, Addr:8000446d  ar4=gr6 set; nul
```

***Normal Instruction***

| Tick Number | Instruction Address | Disassembled Instruction. |
|---|---|---|

```
193 tick, Addr:80004467  ;rep  1  nul ,wtw with nul
```

***Delays at Execution***

| Tick Number | Address of Delayed Instruction | Delay Class and Delay Time |
|---|---|---|
| 206 tick, | Addr:80004466 | Delay decoding for 20 tick(s) |

***Interpretation of Reason of Delay at Execution***

| Amounts of Delayed Tick | Reason of Delay |
|---|---|
| 20 ticks | ar4 used in long operation |

***Classes of Delays***

| Message which is output in execution trace | Class of Delay | Address Meaning |
|---|---|---|
| `delay loading` | Delay at fetching the instruction | Address of the fetched instruction |
| `delay jump` | Waiting for the fetching and(or) the execution of the delay slot instructions before the jump | Address of the jump instruction |
| `delay decoding` | Delay of the instruction decoding. Instruction can not be execute because of the resources of the processor is busy. | Address of the delayed instruction |
| `delay executing` | Blocking of command in pipeline because of conflict with others command. Mostly conflict appears at memory access. | Address of memory access, which delays because of blocking<br>Address equals to 0 if the reason of blocking isn't concerning with memory access |


**Research Centre Module**
**Box: 166, Moscow, 125190, Russia**
**Tel: +7 (095) 152-9335**
**Fax: +7 (095) 152-4661**
**E-Mail: nm-support@module.ru**
**WWW: http://www.module.ru**



Printed in Russia Data of issue: February 21, 2006