Hochschule Bremen, Fachbereich 4: Elektrotechnik und Informatik

-- Projektbericht --

⋆ LDC2[1] ⋆

Honeywell Series 16
Object Program Analysis Tool

Philipp Hachtmann, Mat.-Nr. 124370

10. Dezember 2006

Labor Softwaretechnik 2
Prof. Andreas Spillner

Wintersemester 2006/2007

[1] Name may be subject of change

Contents

1 Introduction

1.1 Motivation

Software for the Honeywell Series 16 minicomputers is very rare today, most of the software ever written for them seems to be lost. But some of it still exists — spread over the world in dusty boxes on attics or in cellars, on software preservation websites and on collectors' harddisks.
The software is stored on punched paper tapes or in disk files containing images of paper tapes.
Often enough the tape labels (resp. file names) are way too vague to exactly identify and understand the contents of a tape.
Back in the times when the Series 16 machines were still in use, everybody knew *his* paper tapes' contents and *his* programs' library dependencies. Everybody had an I/O library for *his* computer's configuration.
Nowadays we have only the remains of all that: many object and library tapes with sometimes obscure names written on it.
We can neither verify the correctness of the data on the tapes nor get any information about a tape's exact contents, exported symbols, and dependencies.
So there is a need for the collector to verify, analyse, sort, split, and recombine his paper tape software.
The *LDC2* program provides a handy approach to help doing this using a newer computer with an UNIX style operating system, disk drives, and some kind of paper tape reading facility.

1.2 The Technical Background

The Honeywell Series 16 assembly program DAP-16 [1] and the FOTRAN IV compiler program [2] read source code from punched paper tape and translate it into relocatable object code which then again is punched out on paper tape. See [3] for operation details.
The object data is organised in blocks of variable layout and length consisting of 16 bit wide native Series 16 machine words. The block format is described in [3], pages 5-21ff.
To store the object blocks on punched paper tapes, each word in the block is divided into three parts: the first one is 4 bits wide, the second and third one are 6 bits wide.
Each frame is now translated so that the resulting paper tape frames don't contain any printable or control characters. Then the block is enclosed by control characters and sent out to the paper tape punch device.
The resulting "silent 4/6/6 code" can be easily processed by the ASR 33/35 printing terminal. It is even possible to mix listing and object code on one tape because the text between the silent data blocks is ignored by the loader.
See [4], page 10-1ff for more information about this concept, but be aware that the translation table in the document contains *severe* errors.

1.3 *LDC2* Functionality

LDC2 is a program which is capable of object and library tape image analysis.
It reads in the data from a file or a serial line, retranslates the mangled characters (4/6/6 code, see above), and reassembles the data blocks into an internal representation.
The block checksum is checked immediately after a block has been read in. This provides for immediate error reporting and handling. One input data error handling possibility is to immediately stop a tape reader connected to a serial port so that the user has a clue where to look for faulty regions on the paper tape. This is very useful when reading in last existing copies of old software.
After the error-free blocks are present in the program's memory, it is possible to do various operations on it. Examples are symbol name extraction and listing, splitting up a tape image's contents into several distinct files, or even reverse assembly and relocating into a virtual Series 16 main memory.

2 General Requirements

2.1 Must Criteria

- The program shall run on any computer running GNU/Linux.
- It has no graphical user interface but a help option and command line error reporting.
- The program shall support and check all block types according to [5].
- The Program must be able to list a tape image's contents.
- The program shall be capable of splitting tape images into single object files.
- The program's structure must be modular and easily extendable.

2.2 Optional features

- Possibility to split a tape image into single data block files.
- Software handshake paper tape reader control:
  A paper paper tape reader connected to the computer's serial port could be motion-controlled by use of XON and XOFF characters. This is useful when error breaks for tape inspection are enabled and the paper tape device does not support RTS/CTS hardware handshake.
- Full POSIX.1, 2004 edition, compliance.
  This enables the program to be usable under many different UNIX style operating systems.

2.3 Further additions explicitely beyond the scope of this project

- Detailed data block analysis:
  - Linking objects into a virtual memory representation.
  - Generating disassembly listings of block contents
- Support for self loading system tapes generated by *PAL-AP* (see [6]).

3 Detailed Functional Specification

3.1 Data input processing

/inp001/ – Data source select

The tape data in Honeywell silent 4/6/6 code [5] shall be read in from standard input or from an input file.

3.2 Data integrity checks

The tape data shall be checked against the Honeywell tape block specification found in [5].

/chk001/ – Block integrity check

Every data block's integrity shall be checked by the means of detecting unexpected end of file while reading in the block.

/chk002/ – Block checksum check

Every data block's checksum shall be calculated and checked according to [5].

/chk003/ – Block type check

Every data block contains its blocktype encoded in a specified location. It shall be checked that each data block's type field contains one of the type codes described in [5].

/chk004/ – Object integrity check

Every object on the tape (i.e. the output of one DAP-16 or FORTRAN IV compiler run) consists of multiple data blocks. According to [5], the last block of each object must be of one of the end block types.
It shall be checked if the last data block read before end of input or an end of tape mark is an appropriate end block.

/chk005/ – Interactive checksum error handling

The program shall be configurable to pause block processing on detection of a checksum error (/chk002/) and resume processing after user intervention.
This feature is incompatible with data input from standard input.

3.3 Additional features

/h001/ – Print help message

A help message is output to standard error.

3.4 Data analysis

/da001/ – List tape contents

Output a list containing a human-readable fixed-format description of all blocks contained in the input data.
The output line format shall be unified for all block types.

/da002/ – List exported symbols

Output a list of all exported symbols.

/da003/ – List called symbols

Output a list of all called symbols.

/da004/ – List unsatisfied dependencies

Output a list of all unsatisfied dependencies i.e. all called symbols that appear in the output of /dp003/ but not in the output of /dp002/.

/da005/ – Tape information output

It shall be possible to output a line containing at least the following information about a tape image:

- Number of bytes read from data source
- Number of blocks read in

/da006/ – Text output

Text from /da001/,/da002/,/da003/, /da004/, and /da005/ is output to standard output or to a file.

3.5 Data processing

/dp001/ – Split into object files

The tape data shall be divided into self-contained object files.
The generated file names are derived from the first symbol name defined in the current object.
Example: If an object exports the symbols SONNE, MOND and STERNE, the resulting file name will be SONNE.
EOT (end of tape) blocks shall not be output in this mode of operation.

/dp002/ – Split into numbered object files

Similar to /dp006/, but the object file names are prepended with a zero-padded three digit number and EOT blocks are not suppressed.

3.6 Error handling

/err001/ – Error types

The following distinguished error types shall exist:

- Input/output error:
  Error occuring during reading from the input file descriptor or during writing to an output file.
  This error shall always lead to an error message and immediate program termination.
- Block integrity error:
  Caused by failed check according to /chk001/.
  This error shall lead to an error message and immediate program termination or a warning message and program continuation.
- Checksum error:
  Caused by failed check according to /chk002/.
  This error shall lead to an error message and immediate program termination or a warning message and program continuation.
- Unknown block type error:
  Caused by failed check according to /chk003/.
  This error shall lead to an error message and immediate program termination or a warning message and program continuation.
- Object integrity error:
  Caused by failed check according to /chk004/.
  This error shall lead to an error message and immediate program termination or a warning message and program continuation.
- Usage error:
  Error caused by errors in the program configuration i.e. wrong parameters or impossible combinations of parameters.
  This error shall always lead to an error message and immediate program termination.
  Additionally, the help (/h001/) message is output.
- Internal program error:
  Error caused by internal program problems.
  This error should never occur.
  This error shall always lead to an error message and immediate program termination.

/err002/ – Program exit codes

Exit condition	Return code
Successfull program completion	0
File open error	1
Input/output error	2
Block integrity error	3
Checksum error	4
Unknown block type error	5
Object integrity error	6
Usage error	7
Internal program error	100

Table 1: Exit codes

/err003/ – Error and warning messages

Error messages shall consinst of "Error:" and the error reason.
Warning messages shall consist of "Warning:" and a descriptive text.
The following texts shall be used to form error and warning messages:

Error condition	**Descriptive text**
File open error	Could not open <filename> for reading/writing!
Input error	Could not read from <filename>!
Output error	Could not write to <filename>!
Block integrity error	Block integrity check failed!
Checksum error	Block checksum wrong!
Unknown block type	Unknown block type!
Object integrity	Object integrity check failed!
Usage error	<Specific message describing the problem>

Table 2: Messages

/err004/ – Error message output

Any error messages shall be output to standard error.
There shall be no way to suppress error messages.

/err005/ – Warning message output

Any warning messages shall be output to standard error.
There shall be a possibility to suppress warning messages.

3.7 Program configuration

/cfg001/ – Configuration process

The following sequence shall be used to acquire the working configuration:

1. Configuration file according to environment variable:
   If the environment variable LDC_CONFIG is set, that file shall be parsed.

2. Configuration file according to command line parameter:
   If the command line parameter specifying a configuration file is found, the specified configuration file is parsed.
   Any values in this configuration file override values found in the previously read configuration file.

3. Configuration parameters passed via the command line:
   Parameters passed on the command line will override values from previously read configuration files. If the command line mentiones a configuration file to read, this file is processed before the other parameters supplied on the command line.

/cfg002/ – Parameter types

The program shall accept two types of parameters:

- Switches:
  A switch is a binary value which can have the value true or false.

- Strings:
  A string parameter is a parameter requiring a string value.
  A typical use would be a file name for input or output.

/cfg003/ – Switch parameters

The following switch parameters shall be accepted:

Long Form	Short form	Description
help	h	Show help message (/h001/)
output_info	a	Output data info (/da005/) .
output_called	c	Output a list of called symbols (/da003/).
output_exported	e	Output a list of exported symbols (/da002/).
output_unsatisfied	u	Output a list of unsatisfied dependencies (/da004/).
split_objects	s	Split into object files (/dp001/).
split_objects_numbered	S	Split into numbered object files (/dp002/).
ignore_block_errors	b	Ignore block integrity errors (/err001/,/chk001/).
ignore_checksum_errors	k	Ignore Checksum errors (/err001//chk002/).
pause_on_checksum_error	p	Wait for user input on checksum error (/chk005/).
ignore_unknown_block_errors	n	Ignore errors causes by datablocks of unknown type (/err001//chk003/).
ignore_object_integrity_errors	g	Ignore errors caused by objects without proper end block (/err001//chk004/).

Table 3: Configuration switches

/cfg004/ – String parameters

The following string parameters shall be accepted:

Long Form	Short form	Description
in_file	i	Set input file (/inp001/).
out_file	o	Set output file for text output(/dp008/).

Table 4: Configuration strings

/cfg005/ – Configuration file syntax

The configuration file must comply two the following grammar, given in EBNF:

```
ConfigurationFile = { ConfigurationLine "<EOL>" } ;
ConfigurationLine = SwitchLine | StringLine | CommentLine ;
       SwitchLine = WSpace SwLong  WSpace "=" BoolValue ;
       StringLine = WSpace StrLong WSpace "=" String ;
      CommentLine = WSpace [ "#" String ] ;
           WSpace = { " " } ;
           SwLong = "output_info" | "ouput_called" | "output_exported"
                  | "output_unsatisfied" | "split_objects" [ _"numbered" ]
                  | "ignore_block_errors" | "ignore_checksum_errors
                  | "pause_on_checksum_error"
                  | "ignore_unknown_block_errors"
                  | "ignore_object_integrity" ;
         StrLong = "out_file" | "in_file" ;
        BoolValue = TrueStr | FalseStr ;
           String = { ? Every character except newline ?} ;
          TrueStr = "yes" | "1" | "true" ;
         FalseStr = | "no" | "0" | "false" ;
```

/cfg006/ – Command Line syntax

The command line must comply to the following grammar, given in EBNF:

```
CommandLine = ProgramName Arguments ;
ProgramName = String ;
  Arguments = { " " } { " " Argument } [ " " InfileName ] ;
   Argument = { Switch | Parameter }
 InfileName = String ;
     Switch = ShortSwitch | LongSwitch ;
  Parameter = ShortParam | LongParam ;
     String = { ? Every character except newline ? } ;
ShortSwitch = "-" SwShort ;
 LongSwitch = "--" SwLong [ "=" BoolValue ] ;
 ShortParam = "-" StrShort String ;
  LongParam = "--" StrLong "=" String ;
    SwShort = "h" | "a" | "c" | "e" | "u" | "s" | "S"
            | "b" | "k" | "p" | "n" | "g" ;
     SwLong = "help"
            | "output_info" | "ouput_called" | "output_exported"
            | "output_unsatisfied" | "split_objects" [ _"numbered" ]
            | "ignore_block_errors" | "ignore_checksum_errors"
            | "pause_on_checksum_error"
            | "ignore_unknown_block_errors"
            | "ignore_object_integrity" ;
   StrShort = "o" ;
    StrLong = "out_file" ;
```

/cfg007/ – Configuration implications

- *pause_on_checksum_errors* implies *ignore_checksum_errors*.
- *pause_on_checksum_errors* is completely ignored when input is read from standard input.
- *output_** inhibits tape information output (/da001/).
- *help* inhibits all other actions.

/cfg008/ – Default behaviour

- Data is read from standard input.
- All error conditions result in immediate program termination and an error message.
- Tape information according to /da001/ is output.
- Normal Text is output to standard output.
- Error messages are output to standard error.

4 Implementation Specification

— To be done —

5 Test Specification

— To be done —

6 Appendix

List of Tables

List of Figures

References

[1] Honeywell. *DAP-16 and DAP-16 MOD 2 Assembly Language Reference Manual*, June 1971. Doc. No.70130072442B.

[2] Honeywell. *FORTRAN IV manual*, April 1977. Doc. No.70130071364A.

[3] Honeywell. *316/516 Programmers' Reference Manual*, May 1969. Doc. No.130071585C.

[4] Honeywell. *DDP-516 Users Guide*, March 1967. Doc. No.130071627.

[5] *Object Program Format*, chapter 5, pages 21–27. In [3].

[6] Honeywell. *"Series 16 Equipment operators' manual"*, November 1973. Doc. No.70130072165F.

The mentioned Honeywell documents can be found on `http://www.series16.adrianwise.co.uk`.