New Stanford Pascal – Installation for MVS (TK4-)

First of all: please excuse possible errors in my English; I am German and not a native English speaker … I will do the best I can.

This new installation procedure was inspired by the video created by moshix (https://www.youtube.com/watch?v=aU0kGtDUa7E), who had some problems when he tried to install my compiler using the old installation guide. I tried to make it simpler and easier to use. Let‘s see if I was successful on this.

To do the installation, you should refer to my GitHub repository https://github.com/StanfordPascal/Pascal. It contains all the Pascal development stuff, including scripts and testcases and so on. You could download the whole repository as a ZIP file or simply pull it to your maschine; it is not that large.

For the MVS installation, you have to use the mvsinst subdirectory only; you can simply ignore all the rest.

The mvsinst subdirectory at the moment looks like this:

```
 Verzeichnis von c:\work\pascal\work\mvsinst

07.12.2017  23:54    <DIR>          .
07.12.2017  23:54    <DIR>          ..
07.12.2017  22:56             2.982 pasalloc.job
07.12.2017  23:17         6.468.156 pascaln.txt
07.12.2017  22:56             1.726 pasdel.job
07.12.2017  22:56             9.422 pasdownl.job
08.12.2017  00:10               780 PASINST1.JOB
07.12.2017  22:56             2.918 PASINST2.JOB
07.12.2017  23:54            39.760 PASLIBX.OBJ
08.12.2017  00:09             1.677 pasload.job
07.12.2017  23:54            31.600 PASMONN.OBJ
07.12.2017  23:54            37.680 PASSNAP.OBJ
07.12.2017  23:54            17.760 PASUTILS.OBJ
07.12.2017  23:54            10.800 SPLITMVS.OBJ
              12 Datei(en),      6.625.261 Bytes
```

Most important is the file PASCALN.TXT, which contains all the stuff coded in one file. The OBJ files are FB 80 object files, which should be transferred to the TK4- machine in binary and together make up a (Pascal) program which reads PASCALN.TXT and puts everything at the right place. The JOB files contain jobs to support the installation process. See more details on the following pages.

Step 1: Preparing the files for the upload to TK4-

This is the step where you have to take most care, and where I have no control. The problems that moshix faced occurred here.

First of all: the text files are created on Windows, because that is the environment where I do my development (it works since 12.2016, when Stanford Pascal became first operational on Windows; I have two OS/2 machines, too, but I keep them for historical reasons, only, and my Linux machines are used for networking etc. throughout the house, but not for development at the moment – I only use them from time to time to verify that the compiler works there, too – same goes for OS/2).

So, if your preferred development environment is Unix or Linux, you should probably get rid of the 0x0d0a linends on my textfiles first (using, for example, the dos2unix utility). The textfiles are, of course, PASCALN.TXT, and all the files with the JOB extension.

After that, you could load the files to TK4- to **arbitrary datasets**

All the files are **FB 80** (including PASCALN.TXT), and from the directory listing on the previous page you can see how large they are.

For the names:

- the target name of PASCALN.TXT does not matter much, because you only have to change it in one place
- the preferred name for the dataset for the object files would be PASCALN.RUNTIME.TEXT, because the JOB PASINST1 builds the SPLITMVS load module from there (the object files need binary transfer, BTW)

but wait: first of all, you should decide (carefully), how the high level qualifier (HLQ) for your New Stanford Pascal compiler installation should be.

See next page.

Step 2: Choosing a HLQ for New Stanford Pascal

Choosing HLQ = PASCALN makes life easier; the distributed jobs will work without change, and the JCL procedures which are distributed in PASCALN.COMPILER.PROCLIB even need no change etc.

Choosing another HLQ (for example HERC01) is possible, of course, but it requires you to change almost every installation job and you will have to specify the HLQ later on every compile job (or change the default in the procs).

For the rest of this paper, I will assume that you choose the standard HLQ = PASCALN.

Step 3: Upload the JOB files to an arbitrary dataset

Choose or create a FB 80 dataset and load the Jobs from the mvsinst subdirectory there. The Jobs are:

```
07.12.2017  22:56             2.982 pasalloc.job
07.12.2017  22:56             1.726 pasdel.job
07.12.2017  22:56             9.422 pasdownl.job
08.12.2017  00:10               780 PASINST1.JOB
07.12.2017  22:56             2.918 PASINST2.JOB
08.12.2017  00:09             1.677 pasload.job
```

Take care: I would suggest **NOT to put these jobs into the target dataset PASCALN.COMPILER.CNTL** (where they are located normally). Because if you do that and if you apply changes to your local copies of these jobs, these changes will later be overwritten by the installation procedure. So it is much better to load these jobs elsewhere (for example PASCALN.PRIVATE.CNTL).

Step 4: Delete an old installation using Job PASDEL

This job deletes an old installation (if present). **Take care !!**

You will have to change the HLQ, if you didn‘t choose PASCALN.

Step 5: Create datasets for the Pascal system using Job PASALLOC

This job runs IEFBR14 and creates all missing datasets with the proper attributes.

You will have to change the HLQ, if you didn‘t choose PASCALN.

When using PASCALN, you should have the following datasets after this step:

```
------------------------------  RFE DSLIST  --------------------- Row 1 of
Command ===> _____________________________________________________ Scroll ===> C
S DATA-SET-NAME------------ VOLUME ALTRK USTRK ORG FRMT % XT LRECL BLKSZ REF
' PASCALN.COMPILER.CNTL      PUB013    15     4 PO  FB  26  1    80 19040 173
' PASCALN.COMPILER.LOAD      PUB013    30    24 PO  U   80  1       19069 173
' PASCALN.COMPILER.MESSAGES  PUB010    30     2 PO  FB   6  1    80 19040 173
' PASCALN.COMPILER.PAS       PUB003   300   210 PO  FB  70  2    80 19040 173
' PASCALN.COMPILER.PROCLIB   PUB002    15     1 PO  FB   6  1    80 19040 173
' PASCALN.COMPILER.TEXT      PUB001    48    30 PO  FB  62  2    80 19040 173
' PASCALN.DBGINFO            PUB011    24    10 PO  FB  41  1    80 19040 173
' PASCALN.RUNTIME.ASM        PUB000    60    31 PO  FB  51  2    80 19040 173
' PASCALN.RUNTIME.LOAD       PUB000    60    16 PO  U   26  1       19069 173
' PASCALN.RUNTIME.TEXT       PUB001    24    13 PO  FB  54  1    80 19040 173
' PASCALN.TESTPGM.ASM        PUB000   150     2 PO  FB   1  1    80 19040 173
' PASCALN.TESTPGM.CNTL       PUB000   150     7 PO  FB   4  1    80 19040 173
' PASCALN.TESTPGM.LOAD       PUB002    75     5 PO  U    6  1       19069 173
' PASCALN.TESTPGM.PAS        PUB011    60    45 PO  FB  75  1    80 19040 173
' PASCALN.TESTPGM.TEXT       PUB002    75     1 PO  FB   1  1    80 19040
  **END**    TOTALS:    5090 TRKS ALLOC       756 TRKS USED      32 EXTENTS
```

Step 6: Upload the OBJ files to PASCALN.RUNTIME.TEXT

The OBJ files are needed to build the SPLITMVS utility that puts everything in the right place. So they first need to be FTPed (binary mode) to PASCALN.RUNTIME.TEXT.

These are the OBJ files:

```
07.12.2017  23:54            39.760 PASLIBX.OBJ
07.12.2017  23:54            31.600 PASMONN.OBJ
07.12.2017  23:54            37.680 PASSNAP.OBJ
07.12.2017  23:54            17.760 PASUTILS.OBJ
07.12.2017  23:54            10.800 SPLITMVS.OBJ
```

(I always considered binary FTP as the hard part, but from the moshix experience I learned that this in fact was no problem at all.)

PASCALN.RUNTIME.TEXT should look like this after the upload:

```
 PASCALN.RUNTIME.TEXT on PUB001 ------------------------------------ Row 1 of
 Command ===> _____________________________________________________ Scroll ===> C
   _NAME___   _TTR__ VV.MM  CREATED _____CHANGED_____  INIT _SIZE   MOD __ID__
 . PASLIBX    000101
 . PASMONN    000301
 . PASSNAP    000401
 . PASUTILS   000601
 . SPLITMVS   000603
   **END**    000604     2017-12-07 PUB001  MOD                   IBMOSVS2
```

Step 7: Build the SPLITMVS load module using Job PASINST1

The Job PASINST1 builds the SPLITMVS load module from the OBJ files in PASCALN.RUNTIME.TEXT. This should be a no-brainer and complete with RC = 0. SPLITMVS will read PASCALN.TXT (see later) and put everything at the right place. SPLITMVS is a Pascal program, BTW. If you complete the installation successfully, you will see the source code of SPLITMVS on PASCALN.TESTPGM.PAS.

Step 8: Upload PASCALN.TXT to TK4-

The file PASCALN.TXT contains all the stuff and needs to be loaded (before SPLITMVS) to an arbitrary dataset. This may be a member of a PO file or a sequential file. FB 80, in any case. You may choose any name you want.

The distributed job PASLOAD, BTW, expects a PO dataset PASCALN.LOADFILE with a member called PASCALN. So if you don‘t want to change anything, create this.

(Once again: take care of the 0x0d chars on Linux/Unix; it is no bad idea to take a look at the target file on TK4- after the upload; SET HEX ON).

Step 9: Run Job PASLOAD (SPLITMVS) to put everything at the right place

The job PASLOAD runs the Pascal program SPLITMVS, which reads the PASCALN.TXT file and puts everything at the right place, including the still missing OBJ files, which are encoded in hex in the PASCALN.TXT file.

You will (maybe) have to change the name of the input file on the INPUT DD statement:

```
//*****************************************************************
//* assign input file here, should be sequential and have lrecl 80
//* needs to be uploaded from Windows or Unix
//* take care of 0x0d linends on Unix/linux - run dos2unix first
//*****************************************************************
//INPUT    DD  DISP=SHR,DSN=PASCALN.LOADFILE(PASCALN)
```

and you maybe have to change the different output DD statements, if you chose another HLQ than PASCALN.

If PASLOAD completes successfully, the most critical part of the installation is done.

Step 10: Run Job PASINST2 to complete the installation

The job PASINST2 builds load modules from different TEXT objects, including the load modules for the two compiler passes (PASCAL1 and PASCAL2).

The first steps of PASINST2 show a return code of 8, which is OK, but the last two steps should return zero:

```
                                          J E S 2   J O B   L O G
10.17.33 JOB  158  $HASP373 PASCALNI STARTED - INIT  1 - CLASS A - SYS TK4-
10.17.33 JOB  158  IEF403I PASCALNI - STARTED - TIME=10.17.33
10.17.33 JOB  158  IEFACTRT - Stepname  Procstep  Program   Retcode
10.17.33 JOB  158  PASCALNI   LKEDA               IEWLF880  RC= 0008
10.17.33 JOB  158  PASCALNI   LKEDB               IEWLF880  RC= 0008
10.17.33 JOB  158  PASCALNI   LKEDC               IEWLF880  RC= 0008
10.17.33 JOB  158  PASCALNI   LKEDD               IEWLF880  RC= 0008
10.17.33 JOB  158  PASCALNI   LKEDE               IEWLF880  RC= 0008
10.17.33 JOB  158  PASCALNI   LKED1               IEWLF880  RC= 0000
10.17.33 JOB  158  PASCALNI   LKED2               IEWLF880  RC= 0000
10.17.33 JOB  158  IEF404I PASCALNI - ENDED - TIME=10.17.33
10.17.33 JOB  158  $HASP395 PASCALNI ENDED
```

After this final installation step, the load libraries PASCALN.COMPILER.LOAD and PASCALN.RUNTIME.LOAD should contain the executable modules (PASCALN.COMPILER.LOAD: the two compiler passes PASCAL1 and PASCAL2 and PASCALN.RUNTIME.LOAD: 5 executable objects, which are linked to the applications as needed).

Step 11: Copying Compiler procedures to SYS2.PROCLIB

There are four JCL procedures to support the work with Stanford Pascal:

- **PASNC** (compile and create FB 80 object file in TEXT dataset),
- **PASNCG** (compile and go, doesn‘t create any objects),
- **PASNCL** (compile and link, creates load module in LOAD dataset),
- **PASNCLG** (compile, link and go, creates load module in LOAD dataset).

These four procedures are distributed in file **PASCALN.COMPILER.PROCLIB**.

I strongly suggest that you copy these four members to **SYS2.PROCLIB**, so that you can call them from everywhere.

Step 12: Verifying the Pascal compiler installation

To verify the installation, I suggest that you run some example programs, using the sample jobs on **PASCALN.TESTPGM.CNTL.**

For example:

- PRIMZERL: computes a large table of primes and does some prime factor computations using this table

- FIBOK: computes some Fibonacci numbers using a very expensive recursive algorithm

- FIBDEMO: the same as FIBOK, but ends with ABEND 1002 due to a logic error (and shows the PASSNAP features, a language specific abend handler)

- TESTPAS: old sample program from the 1979 Stanford installation (still working)

You could also try the Pascal source code formatter PASFORM:

use PASFORM on PASCALN.COMPILER.CNTL to compile it and PASFORM on PASCALN.TESTPGM.CNTL to run it on the first pass of the compiler (output goes to PASCALN.TESTPGM.PAS).

Have fun with this new version of the Stanford Pascal compiler;
please send comments and suggestions to

berndoppolzer@yahoo.com

or

bernd.oppolzer@t-online.de