

New Stanford Pascal – Installation for MVS (TK4-) - and z/OS (Version of 2018.05)

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.

Once again, I reworked the MVS installation. This new installation procedure avoids large files which must be uploaded to the Hercules machine or mainframe prior to the installation. Instead it uses remote jobs started by a Windows CMD file called SUBMVS. These jobs have large instream data sets (SYSIN DD ...) and put all the files directly at the right places. I'll explain later how this is done.

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 machine; 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 should look similar to this:

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

```
08.12.2017  15:49          30.143 external_procedures.odt
08.12.2017  15:49          72.047 external_procedures.pdf
13.02.2018  01:42          30.175 installation_guide_mvs.odt
13.02.2018  01:43         268.348 installation_guide_mvs.pdf
07.10.2011  21:07          61.440 nc.exe
10.05.2018  22:45           4.218 pasalloc.job
02.03.2018  22:18           6.772 pasdecod.job
10.05.2018  22:45           2.045 pasdel.job
02.03.2018  22:20            824 PASINST1.JOB
02.03.2018  22:20           2.956 PASINST2.JOB
02.03.2018  22:19          381.455 pasload0.job
10.05.2018  22:37         7.441.842 pasload1.job
10.05.2018  22:38          991.945 pasload2.job
10.05.2018  22:38         3.147.586 pasload3.job
02.03.2018  13:14           323 submvs.cmd
          18 Datei(en),      12.444.320 Bytes
          2 Verzeichnis(se), 188.693.917.696 Bytes frei
```

Most important are the files PASLOAD1.JOB, PASLOAD2.JOB and PASLOAD3.JOB, which contain all the compiler stuff. But the other files with the JOB extension are important as well.

PASLOAD1.JOB – contains the compiler and the runtime (mandatory)
PASLOAD2.JOB – contains test programs and job control to compile them
PASLOAD3.JOB – contains older versions of the compiler (1979, 1982, 2012) and more sample programs

Inside the load jobs, there is one large input (instream) data set, which contains data and meta information. This meta information tells the program SPLITMVS which runs inside the job:

- where to put the data (dataset name and member name)
- how it is coded (normal text or hex-coded binary data)

The program SPLITMVS which runs inside the load jobs is a Pascal program. It is transferred to the target machine before the first of the 3 load jobs runs, using a job names PASLOAD0.

At PASLOAD0 time, SPLITMVS is not yet available. That's why in PASLOAD0 a program called PASDECOD does a similar job. PASDECOD is a very limited version of SPLITMVS, written in ASSEMBLER. It is transferred as ASSEMBLER source to the target machine and then assembled there, see PASDECOD.JOB.

That is:

- PASDECOD is transferred as ASSEMBLER source to the target system
- PASDECOD is then used to transfer SPLITMVS to the target system
- SPLITMVS is then used to transfer all the rest (all the compiler stuff) to the target system; SPLITMVS puts everything at the right place
- because SPLITMVS transfers only 80 byte char and object modules, some runs of the linkage editor are needed to build load modules.

See the description of the steps needed to prepare the Stanford Pascal system on Hercules (or z/OS) on the following pages.

Step 1: General considerations

From some bad experience other users had, I know a little bit about the pitfalls you probably might encounter.

What you need to accomplish in the end: run some jobs on your Hercules or z/OS installation that I prepared for you ... but I know nothing about your installation.

Known problems:

a) Codepage

My jobs and programs use the Codepage 850 (ASCII); the installation will work best if your Hercules emulator uses the CODEPAGE 850/1047 setting. I hope that any other setting will do, too. The only character with a possible problem should be the vertical bar which is used for string concatenation; the compiler accepts two different code points for this.

b) Windows dependencies

My files have been built on Windows, and that's why they have CRLF line ends (0x0d0a). If you don't like this (that is, if you are on Unix or Linux), you should change my files before you use them.

My SUBMVS.CMD script sends job control to MVS (on Hercules) using the socket interface; the Hercules configuration line looks like this:

```
000C 3505 ${RDRPORT:=3505} sockdev ascii trunc eof
```

I use a program called NC.EXE to send files from Windows to the Hercules machine (which listens to the port 3505 at localhost). The call of the NC program looks like this in the SUBMVS.CMD script:

```
nc -w1 127.0.0.1 3505 <%1
```

If you are not using Windows, you will have to find another way how to send jobs from your client machine to the Hercules target. Or: you could send the job control files to the target machine using FTP (or another file transfer mechanism). But ...

c) The jobs are large

The size of the jobs (PASLOAD1 etc.) doesn't matter much, if you send them from your client machine to the Hercules target. Because then, the large instream file will not be stored. Instead, the instream file will be separated by SPLITMVS into smaller files, which are stored directly at the desired place.

But: if you decide to copy the whole job (PASLOAD1, PASLOAD2 etc.) to the target machine, you will probably run into space problems. I don't recommend this. If you absolutely want to do this, then make sure that you have a disk which is large enough to hold the data.

d) The jobs expect the HLQ of the datasets to be PASCALN

if you want another HLQ, see next paragraph ...

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.

From this point on, I only talk about starting certain jobs, giving their name. It doesn't matter in general, if the job is started from remote using SUBMVS or a similar script, or from a TSO session (if the job has been transferred to the target machine). But there is one difference: if the job is started from remote, it needs USER and PASSWORD information, to be controlled by RAKF.

Step 3: 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 4: 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.OLDCOMP.CNTL     PUB013  300    1 PO  FB   0  1    80 19040 180
' PASCALN.OLDCOMP.SAMPLE   PUB013  300   57 PO  FB  19  1    80 19040 180
' PASCALN.OLDCOMP.SOURCE   PUB013  450  450 PO  FB 100 16    80 19040 180
' 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.MATHLIB   PUB013   30   24 PO  U   80  1    19069 173
' PASCALN.RUNTIME.MATHTEXT  PUB001   24    7 PO  FB  29  1    80 19040 180
' 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 5: Upload the PASDECOD program using the Job PASDECOD

This job compiles and links the ASSEMBLER program PASDECOD, which is used to decode the binary information on the following job (which makes up the Pascal program SPLITMVS).

Step 6: Run PASLOAD0 (Upload SPLITMVS)

PASLOAD0 transfers some OBJ files to PASCALN.RUNTIME.TEXT; these OBJ files are needed to build the SPLITMVS utility that puts everything in the right place.

These are the OBJ files (old 2017.12 version; PC directory):

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

PASCALN.RUNTIME.TEXT should look similar to this after PASLOAD0:

```
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: Run PASINST1 to build the SPLITMVS load module

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 be used in PASLOAD1, 2 and 3 (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: Run the Jobs PASLOAD1, PASLOAD2 and PASLOAD3 (SPLITMVS) to put everything at the right place

The jobs PASLOAD1, PASLOAD2 and PASLOAD3 run the Pascal program SPLITMVS. The jobs contain large instream data sets; SPLITMVS reads this data and builds (many) small members from this information and puts them at the right places.

PASLOAD1.JOB – contains the compiler and the runtime (mandatory)
PASLOAD2.JOB – contains test programs and job control to compile them
PASLOAD3.JOB – contains older versions of the compiler (1979, 1982, 2012) and more sample programs

only PASLOAD1.JOB is really mandatory to run and test the compiler.

If PASLOAD1 (PASLOAD2, PASLOAD3) completes successfully, the most critical part of the installation is done.

Step 9: 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  IEFACRT - 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 10: Copying Compiler procedures to SYS2.PROCLIB

There are five 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).
- **PASNCLO** (compile and link using the old compiler version, used in bootstrapping)

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

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

There is another procedure **PAS1982T**, which can be used to run the **1982 McGill version** of the compiler. It can be used to compare this old version to the actual version. I had to do this sometimes to look if certain errors were already present in the 1982 version, or if I have inserted them. But to use the 1982 compiler, you will have to compile it before, using the actual compiler.

The **1979 Stanford version**, which is present on the distribution, too, does not compile with the actual compiler, but it is present in executable form elsewhere on TK4-.

Step 11: Troubleshooting – Known problems

a) With z/OS, the module name of the linkage editor is **IEWL**; IEWLF880 will not work

b) The compiler needs the Fortran library **SYS1.FORTLIB** for certain subroutine calls (SIN, SQRT etc.). To reduce this Fortran dependency, I removed SYS1.FORTLIB from the compiler procedures and inserted **PASCALN.RUNTIME.MATHLIB** instead.

So: on TK4- (Hercules), you should **copy SYS1.FORTLIB to PASCALN.RUNTIME.MATHLIB**. Same goes for a z/OS installation.

If you don't have a SYS1.FORTLIB on your system, you could build one from **PASCALN.RUNTIME.TEXT(MATHLIB)**; this is distributed with PASLOAD1 and is a image of SYS1.FORTLIB in **XMI format**. There is a Job called PASFORTL on PASCALN.COMPILER.CNTL, which builds the PASCALN.RUNTIME.MATHLIB from this XMI file.

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