Reference Manual

PATCHY

Version 5.04

November 1995

Julius Zoll

The address of this document is:
ftp://asisftp.cern.ch/cernlib/doc/ps.dir/p5refman.ps.gz
or in URL http://wwwcn.cern.ch/asdoc/ as P5 Reference

Please mail error corrections
to zoll@cern.ch

November 13, 1995

CERN
Geneva, Switzerland

Table of Contents

Chapter 1: Principles

Patchy is a family of programs, consisting of the principal program Nypatchy and its Auxiliary programs Nyindex, Nylist, Nysynopt, Nycheck, Nytidy, Nydiff, Nymerge, and Nyshell.

1.1 Program Nypatchy

Nypatchy is a pre-processor primarily intended to extract for compilation the routines of a particular program version from a file, the so-called Patchy Master file or `PAM` file, which is a file holding simultaneously all programmed-for versions of that program.

.Patchy Master files

Such a `PAM` file is an ordinary text file containing the source code of the program, interleaved with Patchy control lines supplying context information. It can be edited with normal text editors.

A `PAM` file is subdivided into "decks", one deck normally being one routine. The text of a deck is headed by a control line `+DECK,dname.` For the deck name `dname` one usually chooses the name of the routine.

Related decks are grouped into a higher level unit, which for historical reasons is called a "patch" (a somewhat misleading use of this word). The text of a patch is headed by a control line `+PATCH,pname.` This line and any following, until excluding the first line `+DECK` in this patch, if any, are called the "blank" deck of the patch.

.Program versions

These higher level units, the "patches", are the instrument to specify program versions. A given version is defined by the list of the names of all the patches which together make up this version. Patches on the `PAM` file whose names do not appear in this list are simply ignored. A patch name is added to this list by a control line `+USE,pname.` occuring in a patch which is itself already selected for `USE`. This selection has its origin in the "cradle", the initial input to Nypatchy, whose begining is pre-selected for `USE`.

Almost all Patchy control lines, and thereby sections of source code, can be made conditional on the `USE` status of some patches. A patch name used for this purpose does not necessarily have to be the name of a patch physically present on the `PAM` file.

.Cradle

The "cradle", read by Nypatchy from standard input by default (unless a file is substituted), controls the particular execution of Nypatchy through the control lines necessary to select the program version and what to do with it. At least conceptually, the cradle is a file and can be structured into patches and decks, just like a `PAM` file, except that its beginning is considered to belong to the patch `CRA*`, which is pre-selected for `USE`. A control line `+PAM,...` given in the cradle instructs Nypatchy to continue input from a file, whose name has to be given. Having reached the end of the `PAM` file, input continues from the cradle with the next line after. One excecution of Nypatchy may involve zero, one, or many, `PAM` files. `PAM` files can only be called from the cradle, not from `PAM` files.

.Sequences

A "sequence" is a set of text lines of which identical copies have to be delivered to maybe many decks. A sequence is defined by heading the set of lines with a line `+KEEP,sname.` Any deck which needs it pulls in a copy with the control line `+SEQ,sname.`

.Actions

The Patchy Auxiliary program Nydiff allows to compare two versions of the same `PAM` file at different stages of evolution. The differences are delivered as a set of "actions" which permit the older to be updated to the newer file. Action control lines are `+DELETE`, and

+ADD or +REPLACE which head the addition or replacement material. Lines on the PAM file are addressed by the names of the patch and deck to which they belong, plus their ordinal line numbers within the deck.

.Deck contents

There are thus three different kinds of material which a deck may contain:

"Control material" on a PAM file is made up essentially of +USE lines (and just possibly +DIVERT); for the cradle there are a number of other control lines of this kind.

"Foreign material" of a deck is material given in this deck, but for use elsewhere: sequence definitions with +KEEP, and actions with +ADD, +REPL, etc., which influence the assembly result of other decks.

"Self material" is material really belonging to the deck where it occurs, not affecting the assembly result elsewhere. Most decks contain just that.

Most of the time a deck contains just one Fortran or C routine. But there will also be decks containing only control material, or there may be decks holding shell-scripts, or other material not meant to be compiled. Also, there may be decks which contain several related small routines together in same same deck.

All the control lines in a deck (except a few run-initializing control lines of the cradle) can be made conditional; moreover ordinary text lines can be made conditional by surrounding them with control lines +IF, +ELSE, +ENDIF.

.Processing modes

Having defined the wanted program version with +USE lines, one has to tell Nypatchy, by means of control lines given in the cradle, what it is supposed to do with the material of this version. This is done by attaching "processing modes" to some or all of this material.

The most obvious is EXE mode, which causes transfer of the assembled material to the output file(s), ready for compilation (or whatever it may be needed for). The control line +EXE. (without parameters) will cause delivery of all the material belonging to the selected program version. With control lines +EXE,pname. one can select individual parts.

LIST mode can be assigned to all or some patches or decks of the program version, to obtain a listing showing how each selected deck is assembled.

There are two other modes, DIVERT and XDIVERT, diversion and extra diversion, which can be used to divide the material between different output streams. DIVERT mode is mainly used to designate routines which have to be compiled with down-graded optimization.

.Nypatchy operating in normal mode

The task of a normal run is the assembly of the EXE selected parts of the wanted program version onto the so-called Assembled Material or ASM output files, ready to be compiled, or to be used in some other way.

Operation is a *single-pass scan* through the homogenous sequential input stream:

- input starts with the cradle, whose beginning is the "blank deck" of the cradle patch CRA*. With control lines given here one specifies the wanted operation:
  - +USE lines select the program version;
  - +EXE etc. lines select processing modes;
  - +ASM lines select the output streaming;
  - +OPT etc. lines may select processing options.
- input continues with the cradle which may contain more decks of P=CRA* and other patches, until the first line +PAM.
- a line +PAM effectively includes the contents of the named file at this point. Normally the file is rewound before processing, processing runs until the end-of-file, and the file is again rewound.

- input reverts to the cradle just after the line `+PAM` when processing of the `PAM` file is complete.
- several `PAM` files may be called up in succession; any number of decks or patches may be given in the cradle. Input is stopped either by a control line `+QUIT` or by End-of-File for the cradle.

This single-pass sequential processing has important consequences: it permits rules for resolving ambiguities by the principle that upstream material over-rules downstream material, for example for multiple definitions of the same sequence with `+KEEP`. It requires that any action, activation, or definition, be given upstream to the material concerned. Thus a sequence has to be defined by the time it is called, a patch has to be `USE` selected (or not) by the time its processing starts.

The processing unit of Nypatchy is the deck. To process an accepted deck it is read complete into memory. The accumulated actions into this deck, if any, are inserted, sequence calls are resolved, and processing modes are evaluated.

The handling of the material contained in the deck depends on its nature:

a) self material, suitably interleaved with foreign material coming into this deck, is copied to the `ASM` file if `EXE` mode is on. If `LIST` mode is on a listing is produced on standard output of all the details of the assembly process, showing every line going into the assembled result, with its line number and patch/deck origin.

b) foreign material, *i.e.* sequences and actions, going out from this deck is held in memory, waiting until it is needed, with processing mode flags attached as appropriate.

c) `+USE` lines cause setting of flags, waiting for patches and decks to come, or to be inspected for conditional control lines.

.Nypatchy operating in Update mode

The task of an `UPDATE` run is to apply a set of corrections to a `PAM` file, delivering again a `PAM` file (rather than compilable material). See para. 4.10 for details.

1.2 The Patchy Auxiliary programs

With the exception of Nyshell, these programs are straight-forward utilities handling `PAM` files:

Nyindex produces a sorted index of the patches, decks, and sequence definitions, contained on a `PAM` file, along with a table-of-content.

Nylist and *Nysynopt* list `PAM` files, with patch and deck page headers, and with local and global line numbers. They list a `PAM` file as is, line by line; with Nypatchy one can list a particular program version, showing how each routine is put together.

Nycheck runs through a `PAM` file to check the syntax of all control lines contained.

Nytidy makes a cleaned-up copy of a `PAM` file.

Nydiff compares two versions of the same `PAM` file at different stages of evolution, delivering a correction patch with which the older version could be updated to the newer version.

Nymerge updates `PAM` file titles and replaces decks by new versions.

Nyshell receives from Nypatchy a "log file" describing all the routines which have been created through a particular physical output stream in `SPLIT` or `MODIFY` mode, writing each routine to a file of its own. Its purpose is to produce a shell-script to compile all the routines, with Fortran, `CC`, or the assembler, with or without optimization, depending on the properties of each routine. It can be made to compare the dates of the source file of each routine, of its corresponding .o file, and of any include file used, to decide whether re-compilation of the routine can be avoided, thereby saving run time.

Chapter 2: Control lines to structure PAM files

2.1 Control-line format

.Examples:

```
+PATCH, P=LILIUM, T=DATA, IF=SIX, IF=WHITE.
+USE, P=CONVALLARIA, POLYGONATUM, MAIANTHEMUM.
+KEEP, Z=HUMUS.
+SEQ, Z=HUMUS, AQUA, LUX.
+ADD, P=ROSA, D=ALPINA, L=24.

+__IF, WHITE.
+__ELSE, IF=GREEN.
+__ENDIF.
```

.Control line key

Control-lines are flagged by "+" in column 1 and are normally identified by the first 3 letters of the key word in columns 2-4, but for the control-lines `+_IF`, `+_IFNOT`, `+_ELSE`, `+_ENDIF` (without or with leading underscores) where the full key word should be given.

Trailing characters of the key word are allowed and ignored, the key is separated from the first parameter by comma:

```
+REPLACE, P=...
+REPL, P=...
+REP, P=...
```

Lines with "+" in column 1 but unknown key are treated as ordinary lines. The significant part of the key word may not contain blanks.

.Environment variable substitution

Before parsing a Patchy control line given *in the cradle*, identified as such by its key-word, Nypatchy will substitute environment variables specified à la Unix with the curly brackets required, for example:

```
+USE, ${machine}.
```

where the variable might have been set in the shell-script like:

```
setenv machine "SUN, SOLARIS"    (C shell)
```

.Comment field

Normally Patchy processes the information on a control-line up to and excluding the first character ".", case-insensitive.

The columns starting with the first after the terminating "." until the end-of-line are called the comment field of the control-line. Normally this is freely available for any comment:

```
+DEL, P=ROSA, D=ALPINA, L=258.    Suppress temporary test
```

But with some control-lines the comment field is used to carry case-sensitive long-text information, with leading blanks being significant:

```
+PAM, T=ATTACH  ./cern/new/src/car/zebra.car
+SEQ, QDATE   .      PARAMETER (IPDATE=?)
```

To protect the user against mistyping a dot for a comma Nypatchy will print a warning if the terminating dot is "obscured". It is not obscured if the comment field is empty, or if the terminating dot is preceded or followed by at least 2 blanks, or surrounded by at least one blank, like so:

```
+PAM, T=ATTACH  ./cern/new/src/car/zebra.car
+USE, QS_UNIX.  running Unix
+USE, QS_UNIX . running Unix
```

This check is switched off if `BACK` compatible operation mode is "on" (the default for the time being).

.Parameter key

A parameter consists of the parameter key and the parameter value(s) separated by "=". Of the parameter key only the first letter is significant:

```
+ADD, PATCH=ROSA, DECK=ALPINA, LINE=24
+ADD, P=ROSA, D=ALPINA, L=24
```

The parameter key of the *first* parameter may be omitted, in which case a default key is assumed; the default depends on the kind of the control-line:

```
+PATCH, LILIUM, T=DATA.
+USE, CONVALLARIA, POLYGONATUM, MAIANTHEMUM.
+EXE, CONVALLARIA, D=MAJALIS.
```

The default key for any control line is indicated in the specifications by including it in round brackets, for example:

```
+USE,(P=)pname, D=dname,...
```

Special handling has been provided for further omitted keys only with the action lines `+ADB`, `+ADD`, `+REPL`, `+DEL` (example: `+ADD,ROSA,ALPINA,24.`) This is described in para. 2.5.

.Parameter value

Parameter values are, depending on the key:

names for keys `P=`, `D=`, `Z=`, `IF=`; `S=`, `R=`, `F=`:
character value of up to 32 characters, excess characters are permitted but ignored; normally all characters are alphanumeric, but other `FORTRAN` characters are allowed, except the separators "=,.-". The number of significant characters can be reduced with the control line `+NAMES`,n.

numbers for keys `L=`; `C=`: unsigned integer or pair of integers separated by "-".
for key `N=`: integer or pair of integers separated by "-".

options for key `T=`:
a word, normally alphanumeric. The first 1 or 3 characters (depending on the contex) must be given, further characters must be correct if present. In the specifications the optional extra characters are indicated in lower case.

In many circumstances multiple parameter values are accepted:

```
+USE, CONVALLARIA, POLYGONATUM, MAIANTHEMUM.
+DEL, ASPARAGUS, L=12, 36-38, 54-56, 75, 91.
+SEQ, HUMUS, AQUA, LUX.
+EXE, POLYGONATUM, D=ODORATUM, VERTICILLATUM.
```

Not every intuitively meaningful form is actually programmed for; only the explicitly allowed forms will operate correctly.

For backward compatibility, the parameter key may be repeated with every parameter value:

```
+SEQ, Z=HUMUS, AQUA, LUX.
+SEQ, Z=HUMUS, Z=AQUA, Z=LUX.
```

Except in the case of the `IF` parameter, there is no difference between the two forms.

.Conditions

Any Patchy control-line (except **+UPDATE**, **+NAMES**, **+GAP**, **+MORE**, **+TITLE**, **+ENDIF**) may be made conditional on the **USE** status of some patches by adding **IF** parameters.

If present, the **IF** parameters must be the last parameters on the line:

```
+PATCH, LILIUM, T=DATA, IF=SIX.
```

Logical operations **OR**, **AND** and **NOT** are possible with the **IF** parameters:

```
+EXE, GEUM, D=RIVALE, IF=APR,MAY,JUN, IF=HUMID, IF=-FROST.
```

The **IF** parameters are written in general as follows:

```
..., IF=p11,p12,..., IF=p21,p22,..., IF=p31,p32,..., ...
```

The pij are patch names, possibly preceded by "-" to indicate the logical **NOT**, used as variables in the above truth-function. The truth-value of a particular pij given as "pname" is "true" if patch pname is **USE** selected, and "false" otherwise; if given as "-pname" the value is "true" if patch pname is *not* **USE** selected. The elements pij of the **IF** group "i" combine by the logical **OR**; all **IF** groups combine by the logical **AND**.

Hence, a line with **IF** parameters is accepted if each **IF** group is "true" (logical **AND**); the **IF** group "i" is "true" if at least one of its pij is "true" (logical **OR**). The truth-value is evaluated with the **USE** selection status of the patches involved at the moment in time when the action requested by the control-line is actually performed for the first time.

Associated material belonging to a rejected control-line is skipped:

```
+ADD, HYPERICUM, PERFORATUM, 24, IF=-COWS,-SUNSHINE.
 <action material>

+PATCH, EPIPACTIS, IF=EUROPA.
 <patch material>

+KEEP, AQUA, IF=WINTER.
 <sequence material>
```

Patch names quoted in **IF** parameters need not be actual patches physically present on the **PAM** file.

.Changed features since Patchy version 4:

1) the maximum line length is now 512 characters for all lines;

2) control-lines are no longer cut after column 72;

3) a terminating dot can no longer be cancelled by following it with a comma;

4) the **IF** status of a control line is no longer evaluated at the time the line is read, but only when it is used.

2.2 Header lines: +TITLE, +PATCH, +DECK

A PAM file should start with some text identifying or describing the file. This material, up to and excluding the first line +PATCH we call the "title patch". Its very first line is the PAM title in the restricted sense, whose standard format is, for example:

```
PATCHY  5.00 /55  1994/02/18 11.20   CERN library L400
```

Its first word is the PAM file identifier, consisting of up to 32 characters like other Patchy names, (leading blanks or an initial "C" followed by blanks ignored) followed by the version number and maybe the optional update level, followed by the date and the time; the rest of the line may contain arbitrary text.

The name of the title patch is the PAM file identifier prefixed with the "commercial at", for this example it would be @PATCHY. A title patch is not normally USE selected (except in UPDATE mode).

The line

```
          +TITLE.
    or    +TITLE:  PATCHY  5.00 /55  1994/02/08 11.20  anything
```

may be used to separate 2 PAM files residing on the same file, it starts a new title patch which must give a new PAM title. The line +TITLE and the first line of the title patch may be joined, as shown above. Note the use of the colon in this case.

The line

```
          +PATCH, (P=)pname, T=type, IF=...
```

starts patch "pname" and also the "blank deck" of this patch; it has line-number zero.

The parameter T=Repeat on a +PATCH line indicates that another patch of the same name follows downstream. This causes Nypatchy to hang on to any non-consumed material for this patch after the processing of the present patch has finished.

The line

```
          +DECK, (D=)dname, P=pname, T=type, IF=...
```

starts deck "dname"; it has line-number zero.

The parameter P=pname can be used to indicate the patch to which this deck belongs; Nypatchy ignores it.

The type parameter T=Join on a line +PATCH or +DECK can be used to suppress the page eject which normally occurs with Nylist in non-compact mode just before listing this patch or deck.

As far as Patchy is concerned, deck names need not be unique on a PAM file, but there must not be two decks of the same name belonging to the same patch; if this happens the second deck cannot be addressed. But as far as the result from Nypatchy is concerned, duplicate deck names are acceptable only for mutually exclusive decks, otherwise the sources of the two decks may destroy each other.

.Data types

The T parameter is also used to specify the "data type" of the patch or the deck, that is the nature of the material contained in the patch or the deck.

There are 7 standard data types:

```
FORT    for Fortran
CC      for C
INCL    for C include files
AS      for assembler
DATA    for a data type with no implied actions
SHELL   for shell scripts (VAX: .com files)
CRAD    for Patchy cradles to be used in a subsequent run
```

For back-compatibility the data-type designators used with version 4

```
COmpile, Xcc, As*, Data
```

are also recognized (As* means: A or anything starting with `AS`).

Further data types may be defined by the user, for each type he chooses a name and sticks to it. Data type names must be given exact, they cannot be abbreviated or have trailing characters, except for the back-compatible handling mentioned just above. For details on the use of data types, see the description of control line `+ASM`, para. 4.8.

The default value for a patch is `T=FORT`, the default value for a deck is the type of the patch to which it belongs.

.Conditions

The whole patch or deck is skipped if the `IF` result is "false" or if it is not `USE` selected at the time its processing is started. The contents of a skipped patch or deck are not analysed in any way, except that the `T=REPEAT` parameter on the header line of a `USE` selected but `IF` deselected patch is recognised.

.Examples:

```
+PATCH, LILIUM.
+DECK, CANDIDUM.
```

normal header lines of patch and deck.

```
+PATCH, LILIUM, T=DATA,REPEAT, IF=SIX.
```

header line of patch `LILIUM`, containing material of type `DATA` (for some user program, not for the compiler or assembler); another patch with the same name will come later in the `PAM` stream; the material of the present patch is to be ignored if patch `SIX` is not `USE` selected.

```
+PATCH, OPHRYS, T=LATEX.
```

header line of patch `OPHRYS` with material for processing with Latex.

2.3 Usage of sequences: +KEEP, +SEQ, +CDE

"Sequences" are strings of zero, one, or several lines, which once defined with `+KEEP,name.` can be called up any number of times for inclusion in any downstream deck with `+SEQ,name.` Two distinct kinds of usage are made of sequences: they are used to provide Common-Dimension-Equivalence declarative statements for the Fortran decks of a program and they are used to provide optional code for places where it is demanded.

.Defining

A sequence is defined thus:

```
global:           +KEEP, (Z=)sname, T=type, IF=...
patch-directed:   +KEEP, (Z=)sname, P=pname, T=type, IF=...
deck-directed:    +KEEP, (Z=)sname, P=pname, D=dname, T=type, IF=...
                   <lines of sequence material>
```

A global sequence is available to any downstream deck; a patch-directed sequence is available only to the decks of patch "pname"; a deck-directed sequence is only available to deck "dname" of patch "pname". The memory occupied by non-global sequences is liberated as soon as the processing of patch "pname" or of deck "dname" is complete.

The sequence material may contain, apart from ordinary text lines, Patchy control-lines `+SEQ`, `+IF` etc, `+INCL`, `+NIL`, `+KILL`; – any other control-line, or the end-of-deck, terminates the sequence definition. Thus the definition of a sequence may "quote" other sequences, and it may contain conditional material.

Because of this property we call `+SEQ`, `+IF` etc, `+INCL`, `+NIL`, `+KILL`, "soft" control-lines; all others are "hard" control-lines, acting as terminators for material associated to a `+KEEP` (and various other control lines, like `+ADD`, `+SELF`, `+IF`).

The `T=type` parameter may be used:

`T=Append` allows the sequence material to be appended to an existing sequence definition, see below for examples; any sequence may be appended to only once.

`T=Nolist` prevents the contents of this sequence from being listed when called in a deck with `LIST` mode selected.

`T=Dummy` causes this definition to be ignored; this exists only for back-compatibility.

.Calling

Sequences may be called or quoted with

```
+SEQ, (Z=)sname, s2, ..., T=PASS, IF=...
+CDE, (Z=)sname, s2, ..., T=PASS, IF=...
```

The form `+CDE` is exactly synonymous with `+SEQ`, it is usually preferred for `CDE` declaratives. If several sequences are called, substitution occurs in the order indicated on the control line. If a sequence is called which has not been defined upstream, a "missing sequence" diagnostic is given unless the option `T=Pass` is specified. A well constructed `PAM` file does not cause this diagnostic without good reason.

Sequences may also be called with

```
+SELF, (Z=)sname, s2, s3,..., T=PASS, IF=...
 <default self material>
```

Depending on whether or not the *first* sequence "sname" is defined, either the sequences are called and the default self material is ignored, or the default self material is accepted and the sequence names are ignored. For more details see para. 2.6.

Dummy sequence calls of the form

```
+SEQ, sname, s2, ..., T=DUMMY, IF=...
```

are accepted for back-compatibility; they have no effect, except that processing modes attached to the patches quoted in the IF= parameter are inherited.

.Overruling

Multiple definitions of the same sequence are resolved thus:

a) the global sequence takes precedence over the patch-directed sequence, and this over the deck-directed sequence;

b) if both sequences are on the same level, the upstream definition (i.e. coming earlier in the input stream) takes precedence over the downstream definition.

In fact, a sequence definition is skipped during input if it is already overruled at that moment.

.Conditional definitions

Conditional definitions may be given either by placing the corresponding +KEEP definitions into separate patches which are, or are not, USE selected, or else by giving IF parameters on the +KEEP lines, as in this example:

```
+KEEP, RUST, IF=ROSA.
 < sequence material in case P=ROSA is selected >
+KEEP, RUST, IF=LILIUM.
 < sequence material in case P=LILIUM is selected >
+KEEP, RUST.
 < default sequence material >
```

In this example, if patch ROSA is USE selected the first definition of RUST is accepted, and the other two are over-ruled. If neither ROSA nor LILIUM are USE selected the first two definitions are skipped, and the third one holds.

If only part of the material is conditional one can use the control-lines +_IF etc, for example:

```
+KEEP, RUST.
   < line group a >
+_IF, ROSA.
   < line group b1 >
+_ELSE, LILIUM.
   < line group b2 >
+_ELSE.
   < line group b3 >
+_ENDIF.
   < line group c >
```

Such a set of conditions must be complete within the sequence definition. The conditions are evaluated when the sequence is called up for the first time.

.Composites

Quoting a sequence in the definition of an other sequence is possible, as already said, for example:

```
 +KEEP, AIR.
< sequence material >
 +SEQ,  WATER, NOBLE, IF=EXACT.
< more sequence material >
```

The definition of Z=AIR quotes the sequences WATER and NOBLE, which must be defined by the time Z=AIR is called for the first time; it is at this moment that the IF=EXACT is evaluated. Up to 36 levels of quotation are allowed.

.Appending

Appending to a sequence: the situation does occasionally arise that one would like to augment an already defined sequence by more material, and one might wrongly be tempted to give for example:

```
+KEEP, Q.       Wrong !!
+CDE,  Q.
<more sequence material>
```

This definition would simply be ignored, because `Z=Q` is already defined. The desired effect can be achieved with

```
+KEEP, Q, T=APPEND.
 <more sequence material>
```

If at all, this feature has to be used with care, because the effect of `+CDE,Q.` is different before and after the re-definition, and because the original definition does not remain available.

A better way of solving this problem is to define, for example:

```
+KEEP, QQ.
+CDE,  Q.
 <more sequence material>
```

and to call `Z=QQ` everywhere instead of `Z=Q`.

.Immediate substitution

To permit the construction of frequently called composite sequences, which will appear in listings as simple sequences, Nypatchy can do a direct substitution, thus reducing the composite sequence to a simple sequence. The condition for this to happen is the following:

```
+KEEP, shigh.      "shigh" contains only 1 line, it is global
<1 line only>      and it is upstream to "slow".
 .........

+KEEP, slow.       "slow" quotes "shigh" with a  +SEQ line
 ...               which quotes no other sequence.
+CDE, shigh.
 ...
```

Normally, the different pieces of composite sequences are kept separately in memory, and are assembled every time the sequence is called. But in the above case Nypatchy will replace in memory the line `+CDE,shigh.` by the line belonging to "shigh" whilst reading the definition of "slow", thus joining the three pieces of "slow" into one. On any listed output, this line from "shigh" will carry the line-number of the line `+CDE,shigh.`

.Built-in sequences

A number of sequence names are reserved; these so-called built-in sequences are used to make Nypatchy do certain things or deliver context information. They are decribed in the next paragraph; the reserved names are:

```
DATEQQ    QENVIR    QFTITLE   QFVSNUM
QCARD1    QFHEAD    QFVERS    QTERMHD
QDATE     QFNAME    QFVPRIM   QTIME
QEJECT    QFTITLCH  QFVSEC    TIMEQQ
```

2.4 Built-in sequences

Unlike normal sequences which have to be defined by the user with a control-line +KEEP, the built-in sequences have a pre-defined meaning for Patchy. Most of them deliver text like normal sequences, but some are pseudo-calls calling for an action of Patchy:

```
+SEQ, QCARD1, R=rname.
```

This pseudo-call can be used either to change the routine name, or to mark the beginning of a new routine in a multi-routine deck, specifying the routine name to be used with a routine-header line or for a file name in split-mode output, cf. para. 4.8. It has no effect if the parameter R=rname is absent.

```
+SEQ, QEJECT, N=n
```

This may be used for listings with Nylist to cause a page eject if there are less than n lines left on the current page. If the N=n parameter is absent the page eject is unconditional.

```
+SEQ, QDATE, S=?  .txa?txe
+SEQ, QTIME, S=?  .txa?txe
```

These calls can be used to obtain the current date or the time of the Nypatchy run. They will deliver one line, containing the text given in the comment field with the escape character replaced by the date or the time of the run. The escape symbol is "?" by default, it can be changed with the S= parameter.

```
  Examples: +SEQ, QDATE   .      PARAMETER (IPDATE=?)
might give:       PARAMETER (IPDATE=19920229)

            +SEQ, QTIME   .#define now ?
might give: #define now 2204
```

```
+SEQ, QENVIR  .statement
```

This call will deliver one line, containing the statement given in the comment field with environment variable substitution; the name of the variable is specified in the Unix way with the curly brackets required, and case-sensitive.

```
   Example: +SEQ, QENVIR  .      PARAMETER (DIR='${XDIR}/')
might give:       PARAMETER (DIR='/cern/dev/')
```

For back-compatibility only we have the following built-in sequences to obtain date and time of the Nypatchy run and the PAM title line of the current PAM file for a Fortran context only:

```
+SEQ, DATEQQ.  is replaced by   IDATQQ=yymmdd

+SEQ, TIMEQQ.  is replaced by   ITIMQQ=hhmm

+SEQ, QFTITLE, N=n.    is replaced by the FORTRAN continuation line
    + nnH< first n characters of last PAM title >

+SEQ, QFTITLCH, N=n.    is replaced similarly by
    + '< first n characters of last PAM title >'
```

The first n<63 significant characters of the title are transmitted; if the parameter N=n is absent, n=8 is used.

```
+SEQ, QFxxxx, S=?, L=lim, N=n  .txa?txe
```

This set of sequences gets components of the PAM file header line. Each sequence call will expand into one line of text starting with the text "txa" given in the comment field, with leading blanks significant, followed by the "value" substituted for the escape symbol (here and by default "?"), followed by the text "txe", if any.

The optional parameters are:

- **S=sym** can be used to specify the escape symbol, default "?".
- **L=lim** may be used to limit the length of the value string, default is the 'natural length' of the item, which for QFHEAD includes the file ID, the version, date and time of the file.
- **N=n** may give the ordinal number of the PAM file whose header line is to be used,
  if n > 0: take the n'th PAM file
  if n < 0: take the n'th PAM file before the current
  default: the current PAM file.

Get the PAM file header line as is: **QFHEAD**

```
    Example: +SEQ, QFHEAD  .      DATA VIDQQ /'@(#)?>'/
 might give:       DATA VIDQQ /'@(#)WYLBUR  1.08 /7  1992/02/29 12.00>'/
```

Get the Pam file identifier: **QFNAME**

```
    Example: +SEQ, QFNAME  .      CHID = '?'
 might give:       CHID = 'WYLBUR'
```

Get the version as a text string: **QFVERS**

```
    Example: +SEQ, QFVERS  .      CHVERS = '?'
 might give:       CHVERS = '1.08 /7'
```

Get the version as a number: **QFVSNUM**

```
    Example: +SEQ, QFVSNUM  .      int pamvers = ?;
 might give:       int pamvers = 10807;
```

Get the primary version number: **QFVPRIM**

```
    Example: +SEQ, QFVPRIM, S=@  .      NUMVERS = @ * 100.01
 might give:       NUMVERS = 1.08 * 100.01
```

Get the secondary version number: **QFVSEC**

```
    Example: +SEQ, QFVSEC, S=@  .      NUMALL = 1000*NUMVERS + @
 might give:       NUMALL = 1000*NUMVERS + 7
```

2.5 Action lines: +ADB, +ADD, +REPL, +DEL

Action lines modify the assembly process of decks downstream. They can be used to change the result from a deck without actually changing the deck on the PAM file, but they are now not normally created by the user, they are used in correction cradles generated by the Auxiliary program Nydiff.

The lines

```
+ADBEFORE, (P=)pname, D=dname, L=cnum, IF=...
 <action material>

+ADD, (P=)pname, D=dname, L=cnum, IF=...
 <action material>
```

cause the action material to be added just before or just after the specified line, unless the IF result is "false". The version 4 parameter C= is recognised to mean L=.

The line

```
+REPLACE, (P=)pname, D=dname, L=c1-c2, IF=...
 <action material>
```

causes the lines c1 to c2 inclusive to be deleted and replaced by the action material, unless the IF result is "false". For single-line replacement the parameter is written as "L=cnum".

The line

```
+DELETE, (P=)pname, D=dname, L=c1-c2, c3-c4,.., IF=...
```

causes deletion of lines c1 to c2, c3 to c4,... (inclusive), unless the IF result is "false".

As for sequences, the action material is terminated by the next "hard" control-line or the end-of-deck, it may contain "soft" control-lines, that is any of +SEQ, +IF etc, +INCL, +NIL, +KILL. Thus the action material may "quote" sequences, even if they are not yet defined, and carry other soft control-lines to the target.

The course taken for conflicting actions is described in para. 4.11, Clash handling.

.Delayed control lines

To send a hard control-line to some downstream deck one has to replace the "+" in col.1 by a "-", for example:

```
+ADD, *ROSACEAE, L=12.
-USE, CYDONIA.
```

which will add the line +USE,CYDONIA. after line 12 in the blank deck of P=*ROSACEAE. Nypatchy will turn the "-" in column 1 into a "+", provided the key in cols. 2-4 is that of a Patchy control line. In a normal run the control line will operate at the place where it is put; in an UPDATE run it will appear in the updated PAM file at the right place.

.Implied patch/deck name

If, in the same deck, a series of consecutive action lines all address the same patch or deck, the corresponding designators need not be repeated; for example:

```
+ADD, P=TULIPA, D=BOEOTICA, L=12.
 <action material>
+DEL, L=25, 39-51, 92, IF=FORMICA.
+REPL, D=SILVESTRIS, L=36-37.
 <action material>
+DEL, L=25.
+ADD, P=SCILLA, L=64.
 <action material>
+DEL, D=BIFOLIA, L=12.
```

.Omitted parameter keys

As a special facility for action lines +ADB, +ADD, +REPL, +DEL, Nypatchy accepts omitted parameter keys as shown by the following transcription of the preceding example:

```
+ADD, TULIPA, BOEOTICA, 12.
 <action material>
+DEL,,, 25, 39-51, 92, IF=FORMICA.
+REPL,, SILVESTRIS, 36-37.
 <action material>
+DEL,,, 25.
+ADD, SCILLA,, 64.
 <action material>
+DEL,, BIFOLIA, 12.
```

Rules: a) on a given action line, parameter keys P=, D=, L= (only) are implicitly substituted for "omitted keys" as long as no key has been given previously. b) "Omitted designators" must be marked by ",".

Valid examples:

```
+ADD, P=TULIPA, D=BOEOTICA, L=12, IF=FORMICA.
+ADD,   TULIPA, D=BOEOTICA, L=12, IF=FORMICA.
+ADD,   TULIPA,   BOEOTICA, L=12, IF=FORMICA.
+ADD,   TULIPA,   BOEOTICA,   12, IF=FORMICA.
```

Invalid forms:

```
+ADD, P=TULIPA, BOEOTICA, 12, IF=FORMICA.  faulty!
(multiple parameter values to P=)

+ADD, TULIPA, BOEOTICA, 12, FORMICA.  faulty!
(key other than P,D,C).
```

.Notes

To delete a sequence definition or an action with Patchy 5 one has to delete the +KEEP or the action line itself and the associated action material exactly; this was required with Patchy version 4 only in UPDATE mode, but not in normal mode.

For backward compatibility, the line

```
+REPL,..., Z=sname, s2, ...
 <action material>
```

is accepted, interpreted as

```
+REPL,...
+SEQ, Z=sname, s2, ...
 <action material>
```

Similar forms are allowed for +ADB and +ADD.

.Actions on foreign material

Level 1 actions operate on the self material or control material of the deck addressed. Level 2 actions are operations on the foreign material of level 1 actions or sequences contained in the deck addressed; Nypatchy handles actions up to level 80.

The meaning of higher level actions is straightforward, but it may be useful to visualize the following examples:

Suppose we have this patch **P=HYBRID**:

```
 0 - +PATCH, HYBRID.
 0 - +DECK, TETRAHIT.
 1 - +KEEP, FOLIUM.
 2 - line 1
 3 - line 2

 4 - +ADD, P=GALEOPSIS, D=SEGETUM, L=36.
 5 - line 1
 6 - line 2

 7 - +DELETE, L=41-44.

 8 - +REPLACE, L=75-124.
 9 - line 1
10 - line 2
11 - line 3

12 - +ADBEF, L=148
13 - line 1

 0 - +DECK, SULPHUREA.
```

and the following actions:

```
+ADD, P=HYBRID, D=TETRAHIT, L=2.
<action material>
```

this causes the action material to become part of the sequence **Z=FOLIUM**.

```
+REPL, P=HYBRID, D=TETRAHIT, L=3-13.
<action material>
```

this causes the action material to become part of the sequence **Z=FOLIUM** and deletes the rest of **D=TETRAHIT**.

```
+REPL, P=HYBRID, D=TETRAHIT, L=7-11.
<action material>
```

this causes the action material to become part of the **+ADD** action and it deletes the **+DEL** and **+REPL** actions.

```
+DEL, P=HYBRID, D=TETRAHIT, L=7-11.
```

this causes the **+DEL** and **+REPL** action to be deleted.

2.6 Sections of self material: +SELF, +SKIP

The control-line +SELF is used to structure the contents of a deck. It terminates any material which preceds it, whether self material or foreign material (i.e. sequence material headed by +KEEP or action material headed by +ADD etc.) It opens a new section of self material, possibly conditional or optional.

A section of self material is terminated by the next "hard" control-line or the end-of-deck, it may contain "soft" control-lines, that is any of +SEQ, +IF etc, +INCL, +NIL, +KILL. Sequence calls within self-material are part of it.

```
+SELF, IF=...
 <conditional self material>
```

starts a section of conditional self material, which is skipped if the IF result is "false", or is accepted if "true". In the absence of IF parameters the self material is unconditional, i.e. always accepted.

```
+SKIP, IF=...
 <conditional self material>
```

is much like +SELF,IF=..., it also starts a section of conditional self material, but this is skipped if the IF result is "true".

Example:

```
+KEEP, AQUA.
 < sequence material >  foreign material goes to memory
+SELF.
 < self material >      self material goes to the ASM file
```

However, mixing of foreign and self material in the same deck is not recommended.

```
+SELF, (Z=)sname, s2, ..., T=PASS, IF=...
 <default self material>
```

starts a section of optional default self material: provided the IF result is "true", the assembly result depends on whether or not the *first* sequence "sname" is defined:

yes: the sequences "sname", "s2",... are called,
missing sequences "s2",... cause a diagnostic unless T=PASS is specified,
the default self material is skipped.

no: the sequence calls are ignored, the default self material is accepted.

If the IF result is false, both the sequence calls and the default self material are skipped.

For several successive pieces of conditional code in self material using +SELF rather than +IF can be more readable, for example:

```
+SELF, IF=LIGHT.   instead of:  +IF, LIGHT.
  < material a >                  < material a >
+SELF, IF=WATER.                +ENDIF.
  < material b >                +IF, WATER.
+SELF, IF=HUMUS.                  < material b >
  < material c >                +ENDIF.
+SELF.                          +IF, HUMUS.
                                  < material c >
                                +ENDIF.
```

2.7 Conditional material: +IF, +ELSE, +ENDIF

These control lines can be used to make ordinary text lines, interspersed with "soft" Patchy control lines, conditional. They may be part not only of self material, but also of foreign material, *i.e.* sequence or action material.

```
+_IF, (IF=)...     open an IF section of material,
  <material>       accept the material if "true"

+_ELSE, (IF=)...   accept if the initial +IF and
  <material>       all previous +ELSE have failed,
                   and if the IF condition is true.

+_ENDIF            terminate an IF section.
```

For convenience Patchy accepts also:

```
+_IFNOT, (IF=)...  is the opposite of "+_IF, ...",
  <material>       i.e. accept the material if "false"
```

A section of conditional code must be opened with `+IF` (or `+IFNOT`), and it must be terminated by `+ENDIF`; inside it may contain any number of `+ELSE` (left example).

`IF` sections may be nested within other `IF` sections, in which case the number of underscores should indicate the nesting level. Level 1 may be indicated either by zero or by one underscore, level 2 should have one more, etc (right example).

.Examples:

```
+KEEP, RUST.
   < line group a >
+_IF, ROSA.
   < conditional group b1 >
+_ELSE, LILIUM.
   < conditional group b2 >
+_ELSE, IRIS.
   < conditional group b3 >
+_ELSE.
   < conditional group b4 >
+_ENDIF.
   < line group c >
```

```
+_IF, LILIACEAE.
+__IF, LILIUM.
+___IF, CANDIDUM.
    < material for Lilium candidum>
+___ELSE, MARTAGON.
    <material for Lilium martagon>
+___ELSE, BULBIFERUM.
    <material for Lilium bulbiferum>
+___ELSE.
    <material for Lilium sp.>
+___ENDIF.
+__ELSE, PARADISEA.
+___IF, LILIASTRUM.
    <material for Paradisea liliastrum>
+___ELSE, LUSITANICA.
    <material for Paradisea lusitanica>
+___ELSE.
    <material for Paradisea sp.>
+___ENDIF
+__ELSE, ANTHERICUM.
    <material for species of Anthericum>
+__ELSE.
    <material for other genera of Liliaceae>
+__ENDIF.
+_ELSE, ROSACEAE.
    <material for the family of Rosaceae>
+_ELSE, LABIATAE.
    <material for the family of Labiates>
+_ENDIF.
```

2.8 Usage of "include" files: +INCLUDE

The purpose of this control line is to make Patchy aware of the dependency of decks of data type CC on include files, to trigger recompilation of decks which depend on include files which have changed.

The line

```
+INCLUDE, (Z=)iname, IF=...
```

is replaced by Nypatchy with

```
#include "iname.h"
```

Like +SEQ, this is a "soft" control line, *i.e.* it can be part of foreign material or of conditional material under control of a +IF.

An include file present on the user's Pam file is defined in a deck of its own with data type INCL, for example:

```
+DECK, MZINC, T=INCL.
 < text of the include file >
```

and its use is signalled with the control line

```
+INCLUDE, MZINC.
```

which line is replaced by Nypatchy with

```
#include "mzinc.h"
```

If the ASM output is generated in MODIFY mode, Nypatchy itself knows whether an include file has changed and marks itself dependent decks for re-compilation. At the next step, Nyshell will check on the dates of the object files and dependent include files of all decks which Nypatchy has signalled as unchanged to make sure that re-compilation is really not needed.

2.9 Patchy comment lines: +NIL

A control line whose key is +NIL is ignored by Nypatchy when seen as part of self material.

Many-line comments are more easily done by heading them with +SKIP. and terminating with +SELF, for example:

```
+NIL.  To extract the installation job for a given machine:
+NIL.    #!/bin/csh -f -v
+NIL.    nypatchy patchy.car job.sh .go  <</
+NIL.    +EXE.
+NIL.    +USE, *INSTAL.
+NIL.    +USE, APOLLO.     (for example)
+NIL.    +PAM.
+NIL.    /
```

is better written as:

```
+SKIP.
To extract the installation job for a given machine:
  #!/bin/csh -f -v
  nypatchy patchy.car job.sh .go  <</
  +EXE.
  +USE, *INSTAL.
  +USE, APOLLO.     (for example)
  +PAM.
  /
+SELF.
```

Chapter 3: Control lines to define program versions

3.1 Adding to a program version: +USE

The program version wanted with a particular run of Nypatchy is selected with +USE lines given in the cradle, for example:

```
+USE, *PATCHY, APOLLO.
```

which selects the complete source code of Patchy to be used on Apollo. Just what are the constituents of this complete program version is programmed on the PAM file itself, here in the "pilot patch" P=*PATCHY. This contains essentially a series of +USE lines, some of them conditional on the USE status of P=APOLLO. (By convention, patch names of pilot patches start with the character "*".)

The line

```
+USE, (P=)pname, p2, ..., T=type, IF=...
```

causes the patches "pname", "p2", ... to be USE selected, if it occurs in a patch or deck which is itself USE selected (unless T=INHIBIT or unless the IF result is "false").

The line

```
+USE, (P=)pname, D=dname, d2, ..., T=type, IF=...
```

causes the decks "dname", "d2", ... of patch "pname" to be USE selected, without affecting the USE status of patch "pname" as a whole.

The line

```
+USE.     without P/D - parameters
```

given in the cradle is a global selection of everything. This should be used only in UPDATE mode.

The type parameter T=Inhibit turns any +USE line into its opposite: it definitely excludes the specified patches or decks from the wanted program version, irrespective of any past or future +USE; thus USE inhibition is stronger than USE selection.

The type parameter T=Repeat indicates that there are several patches of the same name. Unless duplication of a patch name "pname" is specified by +PATCH,pname,T=REPEAT., Patchy assumes that any patch-name occurs only once and discards any material for a patch whose processing is terminated. If this has been forgotten on the PAM file it can be corrected from the cradle with +USE,pname,T=REPEAT.

The type parameters T=List, Exe, Divert, Xdiv may be used to select the processing modes straight away with the +USE line, but this is rarely useful and, if at all, only in the cradle, see para. 5.4.

The cradle patch P=CRA* is preset to be USE selected, so is the option patch PY_VS5 which signals that Patchy version 5 is running. Other patches which the user cares to present in the cradle become part of the wanted program version only if explicitly USE selected.

.Examples:

```
+USE, CONVALLARIA, POLYGONATUM, MAIANTHEMUM.
```

USE selection of 3 patches.

```
+USE, CENTAUREA, D=CYANUS, MONTANA.
```

USE selection of 2 decks in P=CENTAUREA. Normally USE selection is issued for entire patches only, but occasionally it is convenient to group a number of utility routines together into a single patch and to select them individually as decks.

```
+USE, TCGEN, D=UCOPY, T=INHIBIT.
```

USE inhibit of a particular deck. This can be used to overrule some default from a chosen preference.

```
+USE, RIBES, T=INHIBIT.
```

USE inhibit of a particular patch.

.Notes:

Any patch (or deck) which is not USE selected by the time it comes along in the cradle / PAM input stream is simply skipped over.

USE selection only indicates which material is to be taken into consideration, it does not in itself imply what should happen to it. This is done in the cradle by assigning "processing modes" either to the complete wanted program version or only to parts of it, as described in para. 4.5.

Be sure to specify always the complete wanted program version, even if the whole program exists compiled on a library and you only want to re-compile one routine.

```
 ---------------------------------------------------------
|                                                         |
|   You should never try to get selective compilation by  |
|   not including material into the USE selected program. |
|                                                         |
|   Instead you should define the complete program with   |
|   USE lines, and select just the parts to be compiled   |
|   with EXE lines.                                       |
|_________________________________________________________|
```

If you do not heed this advice all sorts of traps will be waiting. For example, some foreign material from somewhere which you forgot about will not arrive in just the particular subroutine which you want to compile.

3.2 Imitate USE status: +IMITATE

The line

```
+IMITATE, (P=)pname, p2, ...
```

causes the patches "pname", "p2",... to appear USE selected for any evaluation of IF parameters, but inhibits the actual processing of those patches.

Example:

```
+PATCH, NEWDEBUG.
+IMITATE, DEBUG.
...
```

P=NEWDEBUG imitates P=DEBUG, that is P=DEBUG is actually excluded from the wanted program version, yet the parameter IF=DEBUG has still the value "true".

Past or future +USE,DEBUG. change nothing; but giving a line +USE,DEBUG,T=INHIBIT. overrules, such that IF=DEBUG will have the value "false".

3.3 Kill the run for bad selection: +KILL

The line

```
+KILL, IF= ...
```

if reached and if the IF condition is true, will stop the Nypatchy run immediatly, causing an error exit.

This (soft) control-line can be used to check against incompatible version selection, for example:

```
+KILL, IF=SUN, IF=VAX.
```

will stop the run with errors if both SUN and VAX are selected at the same time.

Chapter 4: Control lines to steer Nypatchy

4.1 Running Nypatchy

.Program call

At CERN the ready-made module of Nypatchy normally resides on /cern/pro/bin. To run it one calls it into operation with the program-call statement and gives it a cradle, containing the instructions on what to do. The program call is:

```
nypatchy  pam fort  cradle print  cc as data  fort:2 cc:2 as:2 data:2
```

The parameters, which may be given on the program-call statement, or interactively, are:

pam	the name of the PAM file may be given here, if any;
fort	the name of the primary ASM output file may be given here, if any;
cradle	the file name of the cradle, default: standard input;
print	the name of the file to receive the printed output, default: standard output;
cc ...	the names of the ASM files for data types CC, AS, DATA may be given here;
fort:2	the names of the ASM output files to receive the diverted material for data types FORT, CC, AS, DATA may be given here.

Significant leading void parameters are marked by giving a "-", trailing void parameters are cut short by giving ".", to start execution right away one gives ".go" instead.

To get details on the way the parameters are to be specified one can type on the computer:

```
nypatchy help .no
```

In particular, note that on the VAX or Alpha with system VMS a file name may be given in Unix style; the correspondence is for example:

```
/cern/new/src/car/patchy.car;4
cern:[new.src.car]patchy.car;4
```

these forms also work:

```
~/dir/name   dir/name   ../name  etc.
```

On the IBM with VM/CMS file names must be given in one of two possible ways, for example:

```
          patchy.car.a   or   a/patchy.car
to mean   PATCHY CAR A
```

.Error codes

Status return codes from Nypatchy are:

```
               UNIX            VAX        IBM
$? or $status = 0     $STATUS = 1     rc = 0   normal
                1               9          1   warning (not used)
                2               4          8   error
```

Note that it is necessary to test the completion status of Nypatchy, because it will stop delivering ASM output as soon as it has detected the first error, although it will continue to run, to report all errors if possible. This test might look like this in a shell script:

```
C shell:  set rc = $status        Bourne shell:  rc=$?
          if ($rc != 0)  exit 7                  if [ $rc != 0 ]
                                                    then
                                                      exit 7
                                                 fi
```

Note that the status-code has to be copied to a shell-variable if more than one test is to be done on it.

.Examples:

```
nypatchy -- xyz  .go
if ($status != 0)  exit
```

In this example the cradle is read from file xyx.cra, all other files to be used are specified in the cradle.

```
nypatchy  .go  <<\\
 <cradle>
\\
```

Here the cradle is given as a here-document in the shell script.

```
nypatchy  zebra.car zebra.fca  zebra.cra zebra.lis .go
```

reads the PAM file zebra.car, delivering the ASM output to zebra.fca, taking the cradle from zebra.cra, and printing is done to file zebra.lis.

.About the cradle

The cradle can, but need not, be structured into patches and decks like a PAM file. The beginning of the cradle, up to the first line +PATCH, if any, is assumed to belong to the default cradle patch CRA*.

The very beginning of the cradle, up to the first line +DECK, or +PATCH, or +PAM, is assumed to belong to the "blank deck" of the patch CRA*. A certain number of run-initializing control lines, *viz.* +UPDATE, +ASM, +NAMES, +GAP, +MORE, can only be given in this blank deck of P=CRA*, before normal processing starts.

Material after every line +PAM, and up to the next line +PATCH, if any, is again assumed to belong to P=CRA*. Its beginning, up to a line +DECK, if any, is assumed to belong to a deck D=CRA* of P=CRA*. As for the blank deck of P=CRA*, self material in such a deck is ignored.

In the cradle one does the following things:

1) One can limit the number of significant characters in Patchy names to less than 32 characters with control line +NAMES. Very exceptionally one tunes the memory utilisation parameters with lines +GAP or +NAMES.
2) If the purpose of the run is to update the PAM file one has to give the control line +UPDATE, see para. 4.10.
3) Except in the simplest cases, one sets up the ASM output streams with +ASM lines.
4) With lines +USE one has to select the wanted program version.
5) Processing modes, selected with lines +EXE, +LIST, +DIVERT, +XDIV,
   have to tell Nypatchy what it should do with all or parts of the program version.
6) Certain minor options can be taken with lines +OPTION and +PARAM.
7) Corrections, headed by +ADD, +REPL, etc, and over-ruling sequence definitions, headed by +KEEP, may be given.
8) Lines +PAM direct Nypatchy to pause taking input from the cradle and to continue by reading a PAM file.
9) New routines may be given by adding them in the cradle; new decks not headed by a line +PATCH will belong to P=CRA* and are thereby automatically USE selected.
10) The cradle is terminated by a control line +QUIT, or by an End-of-File.

4.2 Continue the cradle from a file: +MORE

With this control line it is possible to use a ready-made cradle residing on a file, but augmented for the purpose of the current run.

```
+MORE    .file-name
```

This line, given in the blank deck of the cradle, adds the content of the specified file to the blank deck read so far. This file in turn could be terminated by an other **+MORE** line, as long as none of **+DECK**, **+PATCH**, or **+PAM** have been present.

Example:

```
nypatchy patchy.car .go <</
+DIVERT, ARRIVE, D=INTERM.
+USE, QBEBUG.
+MORE.        patchy.cra
/
```

4.3 Memory tuning: +NAMES, +GAP

These control lines, which are not normally needed, can only be given at the very beginning of the cradle.

Patchy keeps the names of patches, decks, sequences, occuring on all control lines seen, on the "name stack", to permit representing a name by a small integer in the internal data structures. Enough memory is reserved for this stack at the top end of the dynamic text store even for the needs of very many PAM files. Should this ever overflow one can increase this memory with the control line

```
+NAMES, (N=)len, nsl, nby
```

with the parameters:

- **len** if >0: reset the number of significant characters in P/D/Z names to be *len* characters, not less than 8, and not more than 32, default: 32.
- **nsl** if present and >0: allocate *nsl* name slots, not less than 200, default: 6000.
- **nby** if present and >0: allocate *nby* bytes of text store for all names together, not less than 1200, default: 48000.

The status of the name stack is printed as part of the full summary at the end of the run (if this summary is printed at all, see +OPTION,VERBOSE.)

Nypatchy processes one deck at a time, it requires the deck to be complete in memory, and that there is enough free space to receive the result of the processing. Garbage collection can only be handled between decks, and therefore some parameters are needed to forcast the expected requirements, the "gap" parameters. Their default values can be overruled with

```
+GAP, (N=)gwd, gsl, maxl, chpl
```

with the parameters:

- **gwd** if >0: the number of free words in the control store, at least 1000, at most 10 per-cent of the control store, default: 4000.
- **gsl** if present and >0: the number of free line slots for lines which have to be constructed, at least 100, at most 400, default: 200.
- **maxl** if present and >0: the expected maximum number of lines in any deck, at least 2000, at most 10 per-cent of the text store, default: 5000.
- **chpl** if present and >0: the average number of characters per line, at least 20, at most 80, default: 40.

If you give this control line at all, be careful not to exaggerate: the total amount of memory is fixed, all one can do is to vary the occupation level at which garbage collection occurs. But if there is no garbage to be abandoned, and if you reserve too much space for processing, you may find that there is no space left to read the deck into.

To inspect the state of the memory after processing of a PAM file is complete, one may give the control line +SHOW,MEMORY. just after the corresponding line +PAM.

4.4 Operating options: +OPTION, +PARAMETER

.On/Off operating options

These can be set or reset by the user with:

```
+OPTION, (T=)opt1, opt2, ...        select options;

+OPTION, (T=)OFF, opt1, opt2, ...   de-select options;
```

where "opt" is a word indicating the option. At least 3 letters of the option word have to be given, further letters, if any, have to be correct.

Only the options mentioned on the control line change status, others remain untouched, i.e. the effect of several lines `+OPTION,...` is cumulative.

Lines `+OPTION` and `+PARAM` are only allowed in the cradle; if found on a `PAM` file they cause an error.

These are the available options:

```
ALL       final summary to contain all patch names seen;
```

when starting a new patch, Nypatchy will normaly forget its name if it has not been addressed or activated in any way by upstream material, and hence it will not appear in the final summary, unless this option is given.

```
BACkcom   back-compatible operation;
```

this relaxes the new syntax checks of Patchy version 5 somewhat, it also allows some version 4 constructs. Set "on" by default for the time being.

```
COMpact   listing is compact;
```

this suppresses page ejects at start of deck in Nypatchy listing. Set "on" by default.

```
EJEct     user control of page ejects in Nypatchy listings;
```

if this option is on Nypatchy will cause page ejects for control lines `+SEQ,QEJECT,N=n`.

```
FULl      full listing;
```

the Nypatchy listing is to include control lines and foreign material. If this option is `OFF` the listing only shows the lines which would go to the `ASM` file. Set "on" by default.

```
MAPasm    monitor the decks written to the ASM files;
```

this causes Nypatchy to print one line for every deck which has been written to the `ASM` file, showing its size and where it has been written to.

```
VERbose   print full summary at the end;
```

to avoid too much output to the screen when running Nypatchy interactively, the final full summary is not normally printed if the terminal receives the printed output. With this option one can select to have the summary anyway.

The version 4 options `BIGUSE`, `KEEPORD`, `LOST`, `SEQORD`, `TERM`, `UREF`, `WARN`, `YESIF`, `DETACH`, `EOF`, `HOLD`, `RESUME`, are no longer available.

.Parameteric options

Some operating options take an integer value, they can be set with:

```
+PARAMETER, (T=)opt, N=value.
```

where "opt" selects the option:

```
CLAsh, N=n    report clashes at or above level n= 1 or 2.
```

Default is `N=2` for normal mode, and `N=1` for `UPDATE` mode.
For clash handling see para. 4.11.

```
LINes, N=n    set page-size for the Nypatchy listed output.
```

The default is preset to 110 lines per page.

```
COLumns, N=n  set the page-width for the Nypatchy printed output.
```

The default is preset to 120 columns across the page; it can be increased, or decreased, but not below 90.

.Examples:

```
+OPTION, ALL, MAPASM.
+OPTION, COMPACT, OFF.
+PARAM,  CLASH, N=1.
```

4.5 Select processing modes: +EXE, +DIVERT, +XDIV, +LIST

Processing of any deck of the selected program version is done in zero, one, or several of the following modes:

- **EXE** write the assembly result of the deck onto an **ASM** output stream.
- **DIVERT** divert the output, if any, to the diverted **ASM** output stream.
- **XDIV** divert the output, if any, to the "extra diverted" streams, see para. 4.8.
- **LIST** list the composition of the deck on the print file.

Processing mode selections (except **+DIVERT**) should only ever be done in the cradle; it is only for back-compatibility to Patchy version 3 that they are allowed also on **PAM** files.

Activations of any of the processing modes, **EXE**, **DIV**, **XDIV**, **LIST**, are handled exactly in the same manner. In order to simplify the reading of the present paragraph, the description is given in terms of the **EXE** mode, but everything stated is also valid, *mutatis mutandis*, for the other modes **DIV**, **XDIV**, and **LIST**, with one exception noted (filtering).

```
+EXE, T=type.   without P/D parameters
```

causes global **EXE** selection for all **USE** selected material that follows. Global selection is allowed only in the cradle; if given right at the beginning, it causes **EXE** selection for the complete "wanted program version".

```
+EXE, (P=)pname, p2, ..., T=type, IF=...
```

causes the patches "pname", "p2",... to be **EXE** selected, normally with "**EXE** for self and foreign", unless specified otherwise by the type parameter.

```
+EXE, (P=)pname, D=dname, d2, ..., T=type, IF=...
```

causes the decks "dname", "d2", ... of patch "pname" to be **EXE** selected, normally with "**EXE** for self and foreign", unless specified otherwise by the type parameter.

The type-parameter may be used to qualify **EXE** selection (see below):

```
T=ONLy       restrict to "EXE for self material" only;
T=Transmit   extend to "EXE via USE lines" also;
T=Inhibit    inhibit past and future EXE selection.
```

The type parameter may be used to select other processing modes at the same time:

```
T=Exe, List, Divert, Xdivert
```

for example the following two lines are equivalent:

```
+XDIV, CONVALLARIA, T=DIV.
+DIV,  CONVALLARIA, T=XDIV.
```

Here are examples of the different kinds of EXE selection:

```
+EXE, X.              normal EXE, self and foreign material of patch X;
+EXE, X, T=ONLY.      self material of X only;
+EXE, X, T=TRANS.     transmit EXE also via USE lines;
+EXE, X, T=INHIBIT.   EXE of X totally inhibited.
```

Normal EXE selection of patch X attaches EXE mode to all the decks of patch X, and also to the self material of all decks elsewhere which depend on X, either because they receive foreign material from X, or because they refer to X with the IF parameters of some of their control lines.

By giving the T=ONLY parameter one can *limit* the EXE mode to the self material of patch X only, without propagation to decks outside X; for example:

```
+LIST, QCDE, T=ONLY.
```

will cause listing of the patch QCDE (supposed to contain a set of sequence definitions), without triggering LIST mode of all the decks which call these sequences.

By giving the T=TRANSMIT parameter one can *extend* the EXE mode propagation such that everything USE selected from X, directly or indirectly through all levels, is also selected for EXE. This is of interest only in very special situations, namely when the part selected by some sub-pilot should be handled separately from the main stream; an example using this can be found in para. 5.3.

.Mode filter

A special, non-obvious feature is filtering of processing-mode activation: EXE selection (and EXE inhibition) by a line +EXE,... is effective only if this line is

a) either in the cradle,

b) or in a deck on the PAM file which is itself EXE selected for "self and foreign".

Here is the exception: this filter does not operate for control lines +DIVERT given on the PAM file, that is: a control line +DIV on a PAM file is always honoured (if it occurs in a USE selected deck, of course). But note that lines +EXE, +XDIV, +LIST should anyway not be present on PAM files.

.About the LIST mode

Processing a deck in LIST mode will show how the deck has been assembled, by listing its self material plus any foreign material coming into it, along with identification tags showing its origin.

Material which has been deleted under a +REPLACE or +DELETE action is not displayed, unless the T=DISPLAY option is taken on the current +PAM control line, see para. 4.7 (this is new with version 5.03/8).

If listing mode is FULL the control-line content and foreign material going out from this deck is also shown. Whether listing is full or short is a general operating option, by default +OPTION,FULL. is assumed.

.Modes for the cradle

The very beginning of the cradle, the blank deck of `P=CRA*`, is initialized to ignore self material, in order not to write on the `ASM` files before they are established. No processing modes are pre-selected for the blank deck, except straight listing corresponding to

```
+OPTION, FULL, COMPACT.
+LIST, P=CRA*, D=, T=ONLY.
+USE, P=CRA*, T=REPEAT.
```

This deck is the only deck which can change its own processing mode, as shown by the following example:

```
+USE, P=*ROSACEAE.                   select program version
+ADD, P=RUBUS, D=IDAEUS, C=96.       without EXE activation
 ...
+EXE, P=CRA*, D=.                    "EXE on foreign"
+REPLACE, P=ROSA, D=ALPINA, C=24-32. with EXE activation
 ...
+EXE, P=CRA*, D=, T=INHIB.           "EXE inhibit"
+DEL, P=CYDONIA, D=OBLONGA, C=44.    without EXE activation
 ...
```

The cradle patch `P=CRA*` has no processing modes pre-selected. To do so, one has to specify in the blank deck what is wanted, like:

```
+LIST.                  list selection for everything.
+LIST, CRA*, T=ONLY.    simple list selection without propagation.
+LIST, CRA*.            list P=CRA* and everything modified from it.
+EXE,  CRA*, T=DIV.     select for EXE and DIVERT all decks of
                        P=CRA* and any deck modified from it.
```

The decks `D=CRA*` of `P=CRA*`, that is the lines after any `+PAM,...`, are pre-set like the blank deck: self material ignored, simple listing pre-selected *plus* whatever has been selected for `P=CRA*` as a whole. The contents are not addressable, as there may be many such decks, Nypatchy simply ignores a line like `+REPLACE,CRA*,CRA*,27.`

.Notes:

Note that `+DIVERT` and `+XDIV` only select diversion; for any deck to actually appear on the `ASM` files it has to have been given `EXE` mode also.

Control lines to select processing modes should not be present on a `PAM` file, except for lines `+DIVERT` which are used to indicate which routines have to be compiled with down-graded optimization under which circumstances.

The processing mode `MIX` of Patchy version 4 is no longer available.

Self material is always ignored in the blank deck of `P=CRA*`, and also in all decks `D=CRA*` of `P=CRA*`, which are any lines just after a line `+PAM` until either `+PATCH` or `+DECK`. One can use this fact for easy comments.

4.6 Forcing processing modes: +FORCE, +SUSPEND

With the lines **+EXE**, **+DIV**, **+LIST**, described in the previous paragraph, processing modes may be activated for individual patches and individual decks, with and without transmission to other decks or patches affected by foreign material or by **USE** lines. Processing modes can be inhibited, again for individual patches and decks, with the type parameter **T=INHIBIT**. Inhibition is final and cannot be overruled again.

Sometimes it is convenient to be able to switch on a processing mode, not for individual patches but for a whole section of the cradle / **PAM** input stream. This can be done, for example for the **EXE** mode, with

```
              +FORCE, (T=)EXE.
```

This forces "**EXE for self**" for every deck that comes along, provided the deck is part of the wanted program version or unless the deck is **EXE** inhibited, until forcing is taken off again with

```
              +FORCE, (T=)EXE,OFF.
```

Forcing can be switched on and off any number of times.

Similarly, it is sometimes convenient to be able to cancel a processing mode for a whole section of the cradle / **PAM** input stream. This can be done, for example for the **EXE** mode, with

```
              +SUSPEND, (T=)EXE.
```

This cancels "**EXE for self**" for every deck that comes along until suspending is taken off again with

```
              +SUSPEND, (T=)EXE,OFF.
```

Suspending can be switched on and off any number of times.

Lines **+FORCE** and **+SUSPEND** are allowed only in the cradle; if found on a **PAM** file they are signalled as faulty control-lines.

Examples:

```
+LIST.
+USE, ...                              |
 ...                                   | +USE, ...
+SUSPEND,LIST.                         |  ...
+PAM, T=ATTACH              .hycde     | +PAM, T=ATTACH              .hycde
+PAM, T=ATT,HOLD, R=GEOCDE  .hygeom    | +PAM, T=ATT,HOLD, R=GEOCDE  .hygeom
+SUSPEND, LIST, OFF.                   | +FORCE, LIST.
+PAM, T=RESUME.                        | +PAM, T=RESUME.
+QUIT.                                 | +QUIT.
```

Both examples give a listing of the complete wanted program version, except that material coming from the **PAM** file **HYCDE** and from the beginning of the **PAM** file **HYGEOM** is not listed. Another example is found at the end of para. 6.3.

4.7 Invoking Pam file input: +PAM

A line `+PAM` read by Nypatchy on the cradle terminates the current deck and patch, and directs Nypatchy to continue input from the indicated `PAM` file, provided the `IF` result is "true". Lines `+PAM` are allowed only in the cradle, not on a `PAM` file.

The name of the `PAM` file to be read can be given as the first parameter on the program-call statement "`nypatchy pam asm ...`", and then reading can be initiated by the control line

```
+PAM.          (without parameters)
```

If one has more than just one file to read, one gives several lines `+PAM`, with the name of each file on the comment field, having selected the `ATTACH` option, for example:

```
+PAM, T=ATTACH   ./cern/pro/src/car/wylbur
```

where the default file-name extension ".car" is assumed, if none given.

One can also use the `LABEL` parameter to select a file for re-read, as explained further down.

Options for processing a `PAM` file may be taken with:

```
+PAM, (LABEL=)n, N=xx, T=opt, RETURN=pname, IF=...  .file-name
```

with these parameters:

`Label=n`	a small integer to identify the file, this is only needed for stop-go reading, see below.
`N=ndo`	number of `PAM` files to be processed on this file; default: ndo=999
`N=nsk,ndo`	ignore nsk `PAM` files and then process ndo `PAM` files on this file.
`T=Attach`	open the file whose name is given in the comment field and process it.
`T=Cards`	is accepted and ignored for back-compatibility.
`T=Hold`	no rewind of the file after processing, default: rewind after.
`T=Resume`	no rewind of the file before processing, default: rewind before.
`T=Update`	process the file in `UPDATE` mode, see para. 4.10, 6.3.
`T=MERge`	accept all actions, even if clashing, see para. 4.11; normally used only in `UPDATE` mode; new with version 5.03/8 !
`T=DISplay`	display deleted material, see para. 4.5; new with version 5.03/8 !
`R=pname`	stop processing the file when the processing of patch "pname" is complete and revert to the cradle.
`IF=...`	if the `IF` result is "false" the file is not actually read, input continues from the cradle.

The 'compact binary' format for `PAM` files of Patchy version 4 is no longer available.

After processing of the file is complete, Nypatchy reverts to taking lines from the cradle. These are initially assumed to belong to `P=CRA*`, `D=CRA*` until the next `+PATCH` or `+DECK` arrives, if any. As for the blank deck of `P=CRA*`, self material in `D=CRA*` is ignored, lines in this deck cannot be addressed.

.Re-reading files

Suppose one has the following problem: from a set of 3 PAM files, whose names are given in the 3 environment variables PAMA, PAMB, PAMC, the beginnings have to be read first to collect the declaratives, before reading the bulk of the code. Using the LABEL parameter we can write:

```
+PAM, 11, N=1, T=HOLD, ATTACH   .${PAMA}
+PAM, 12, N=1, T=HOLD, ATTACH   .${PAMB}
+PAM, 13, N=1, T=HOLD, ATTACH   .${PAMC}
+PAM, 11, T=RESUME.
+PAM, 12, T=RESUME.
+PAM, 13, T=RESUME.
```

For each file, the first +PAM attaches the file, assigns a label to it, reads the first PAM from the file, and leaves the file positioned just before the second. The second control line +PAM for the same file requests to go back to the file designated by its label, to continue reading it. The parameters T=HOLD and T=RESUME are needed to keep the file positioned. This example is fine if each file is subdivided into at least two PAM files, with the first containing the declaratives. If this is not the case one has to resort to specifying the patch name up to which the first read should go, this can be done with the RETURN=pname parameter.

4.8 Defining physical output streams, concepts

The assembly result of any deck processed in EXE mode is written onto an ASM file as line images ready to be used as input to the compiler, or to the assembler, or any other purpose. Nypatchy tries to give the user full control to steer the output onto maybe many different files, in the simplest possible way. This can be done with +ASM lines given at the beginning of the cradle.

.Logical streams

The data type of the deck (cf. para. 2.2) combined with the processing modes DIVERT and XDIV determine the "logical stream" through which the result is written. The name of the data type and a numeric suffix identify the logical streams, for example:

```
LATEX:1   normal/normal
LATEX:2   normal/diverted
LATEX:3   extra/normal
LATEX:4   extra/diverted
```

are the 4 possible logical streams for the user data type LATEX. The generic logical stream LATEX (or LATEX:0) designates all logical streams of this data types.

.Physical streams for output

A logical stream can be made physical by giving output instructions for it (using stream LATEX:0 as an example):

```
+ASM, LATEX, T=ATTACH   .file name
+ASM, LATEX, T=SPLIT
+ASM, LATEX, T=MODIFY
+ASM, LATEX, T=BYPASS.
```

These are the 4 possible ways of establishing the physical stream LATEX. In the first case all material coming through this stream shall be written to the file whose name is given. In the second and third case each deck is to be written to a separate file whose name is to be constructed from the deck name plus the file-name extension suffix proper to the data type of the deck (and possibly a directory prefix). In the last case all material coming to this stream is flushed.

For some of the standard data types some of the streams can be made physical merely by giving file names on the nypatchy program-call statement (cf. para. 4.1). This is true for the generic logical streams **FORT**, **CC**, **AS**, **DATA**, and for the diverted streams **FORT:2**, **CC:2**, **AS:2**, **DATA:2**. Unless over-ruled with an **ASM** line, **T=ATTACH** is assumed for a stream made physical in this way.

.Binding of logical streams

"Binding" of a given logical stream to some other logical stream means that the output of this stream goes to the same file as the output from the stream it is bound to. Initially all logical streams for the standard data types are set up bound directly or indirectly to the generic stream **FORT:0**. User data types can be declared with, for example: **+ASM,LATEX,T=USED.**, which causes the creation of the logical streams for **LATEX** bound into the streams for the standard data types. If a logical stream is made physical it retains the bonds of other logical streams bound to it, but it itself is no longer bound to some other stream.

Default binding is initialized such that the wanted set-up can be specified with minimal instructions, this default is:

a) the generic logical streams **CC**, **AS**, **SHELL**, **CRAD**, **DATA** are bound to stream **FORT**;

b) the generic logical stream **INCL** is bound to the generic stream **CC**;

c) the generic streams of all user data types are bound to the generic stream **DATA**;

d) all logical streams **X:i** of data type **X** are bound to the generic stream **X:0**;
if this is itself bound to the generic stream **Y:0**, then the logical streams **X:i** are bound to the corresponding streams **Y:i**;

e) if the logical stream **X:1** is made physical all logical streams **X:i** are bound to it, provided they have not been bound explicitly to somewhere else; similarly if the logical stream **X:3** is made physical, the logical stream **X:4** is bound to **X:3**.

If no **+ASM** lines are given in the cradle the result depends entirely on the file-names given on the Nypatchy program-call:

```
nypatchy  pam fort  cradle print  cc as data  fort:2 cc:2 as:2 data:2
```

1) if a file name is given *only* for the 2nd parameter "fort" all the output will go to this file;

2) if a file name is given *also* for the 8th parameter "fort:2" all the diverted material of all data types will go to this file, whilst all the normal material will go to the first file;

3) file names given for other parameters will capture the corresponding material.

Alternatively one can have the same effect by giving the following **+ASM** lines instead of the file names on the program-call:

```
+ASM, FORT,   T=ATTACH  .file-name_1   case 1)
+ASM, FORT:2, T=ATTACH  .file-name_2   case 2)
+ASM, CC,     T=ATTACH  .file-name_3   case 3)
```

Explicite binding can be done with, for example:

```
     +ASM, AS, MACRO, T=BIND.
or   +ASM, AS, MACRO, T=ATTACH  .file-name
```

In both cases the user data type **MACRO** is declared and bound to the generic logical stream **AS**, which in the second case is also established as physical.

The summary printed at the end of each Nypatchy run will show the data flow through all active logical streams, and the file-names associated with the active physical streams. If this is not enough, and if there is something one does not understand, one can give the control line **+SHOW,ASM.** at the end of the cradle. This produces rather a lot of printed output.

.Split mode

One may request that Nypatchy writes each deck coming to a given physical stream to a separate file, whose name is derived from the deck-name with an extension added. For example:

```
    +ASM, FORT, T=SPLIT.
or  +ASM, FORT, T=MODIFY.
```

requests splitting of all decks arriving on the physical stream `FORT`, that is the decks of all logical streams bound to this physical stream.

Giving `T=MODIFY` instead of `T=SPLIT` will cause Nypatchy to compare the text of each deck against the corresponding file, if it pre-exists, and to suppress the output if the two are identical. Thus the date of the existing and unchanged file is not disturbed.

Nypatchy knows the extensions needed for the standard data types `FORT`, `CC`, `INCL`, and `AS`, on the machine one is running on, on most machines they are .f, .c, .h, and .s. For data type `CRAD` the extension used is .cra; for `SHELL` .sh is used on `UNIX`, .com on `VAX`. For data type `DATA` .dat is used on the `VAX`, and nothing on `UNIX`.

One may want to change the extensions used, or for the user data types one may want to set the extension; this can be done with, for example:

```
+ASM, LATEX, T=EXT   ..tex
```

The directory into which the files are written is the current working directory by default; this can be changed by adding a common prefix to all file names, with for example:

```
+ASM, FORT, T=SPLIT, PREFIX   .dir/
```

One may request that a record of the files created be written to a file to be digested by the Patchy Auxiliary Nyshell, with for example:

```
+ASM, FORT, T=SPLIT, LOG   .file-name
```

The last two examples can be combined if the prefix is a pure directory, for example:

```
+ASM, FORT, T=SPLIT, PREFIX, LOG   .dir/file-name
```

The resulting log file will contain one line for each deck written, giving its logical stream name and its file name, as for example:

```
fort:1    chdirf.f
cc:1      chdiri.c
fort:1    getenvf.f  same
cc:1      geteni.c   same
cc:1      getpidf.c
fort:2    getwdf.f
cc:2      getwdi.c   same
```

If the "same" flag is present the file has not actually been written because it is unchanged. This can only happen in `MODIFY` mode.

4.9 Defining physical output streams: +ASM

Context information about the ASM output streaming has been given in the previous paragraph. Here comes a more formal description of the forms which +ASM control lines can take. As always, a control line +ASM can be made conditional by adding IF= parameters; for simplicity they are not shown here.

.Non-split mode

To establish a physical stream:

```
      +ASM, phys, log1, ..., T=Attach  .file-name
      +ASM, log1, log2, ..., T=BYpass.
```

In the first case logical streams *log1,...*, if any, are bound to stream *phys*, and all output goes to the file whose name is given. In the second case output for all streams *log1,...* is suppressed.

.Split mode

To establish a physical stream:

```
      +ASM, phys, log1, ..., T=SPLit  [,PREfix] [,LOG]
   or +ASM, phys, log1, ..., T=MODify [,PREfix] [,LOG]
```

Streams *log1,...*, if any, are bound to stream *phys*, and all output coming here is processed in SPLIT or MODIFY mode. The prefix and/or the name of the log file can be given right away, or separately:

```
      +ASM, phys, T=PREfix       .prefix
      +ASM, phys, T=LOG          .file-name
      +ASM, phys, T=PREfix, LOG  .dir/file-name
```

The file-name extension to be used can be set with

```
      +ASM, log1, ..., T=EXT.  extension
```

Be careful to give the "." which starts the comment field as well as the "." which normally belongs to the extension.

.Binding only, Aliasing

```
      +ASM, here, log1, ..., T=BINd.
      +ASM, here, typ1, ..., T=ALIas.
```

The first binds logical streams *log1,...* to stream *here*.

The second declares that *typ1,...* are just other names for *here*. Neither *here* nor *typi* may contain a ":" in this case.

In the first case the logical stream keeps its identity; in the second case it looses it: in SPLIT mode it will appear under the *here* name on the log-file.

.User type confirmation

All user data types whose material on the PAM files is selected with +USE should have been declared on some +ASM line in the blank deck of the cradle. If no +ASM line for such a user data type is needed for any purpose, it should be declared with:

```
+ASM, typ1, ..., T=USEd.
```

If Nypatchy finds a non-declared data type when processing USE-selected material on the Pam file, it prints a warning and opens a generic stream for this data type, which it then binds to the stream DATA.

.Control-character substitution

This can be requested with:

```
+ASM, log1, ..., T=CCHsubs   .mask
```

The data types SHELL and CRAD are initialized as with:

```
+ASM, CRAD, T=CCHSUB .&+
```

to make cradles contained on a PAM file ready for use with a second run of Nypatchy. This could be changed, for example with:

```
+ASM, CRAD, T=CCHSUB .&+!+
```

This example causes changing the control-character "&" or "!" in column 1 on any line of the output written to the ASM file for all logical streams of type CRAD into a "+" if the key of the line is that of a Patchy control line.

This could be switched off by giving an empty mask.

.Routine-header lines

Arbitrary routine header lines can be defined for a given data type either for the generic or for a particular stream with:

```
+ASM, log, T=RHead  .mask
```

In the comment field of the control line one gives the mask for generating the routine header line; the very first character is the escape character used to indicate the place where the routine name has to be inserted.

On the machines where this useful, routine header lines are pre-initialized as needed by FCAsplit on the machine where one is running on to something like:

```
+ASM, FORT, T=RH   .!CDECK  ID>, ! .
+ASM, CC,   T=RH   .!/*DECK ID>, ! . */
+ASM, AS,   T=RH   .!|DECK  ID>, ! .
```

This can be switched off, for example with:

```
+ASM, AS, T=RH.
```

with an empty comment field.

Routine header lines are not generated in SPLIT mode.

4.10 Updating a PAM file: +UPDATE

The purpose of a run of Nypatchy in Update mode is to deliver again a **PAM** file with actions like **+ADD** applied (and not to deliver compilable material in the normal fashion).

Although one could make corrections with **+ADD**, **+REPLACE**, etc. by hand, this is now rarely used. Instead one changes the **PAM** file directly with a text editor, and if needed one gets the corrections by taking differences between the changed and the original file using the auxiliary Nydiff, as explained in para. 6.3.

To use this one has to do two things: one has to signal Update mode by giving the line

```
+UPDATE.
```

right at the beginning of the cradle, *and* one has to give the parameter **T=UPDATE** on the **+PAM** call for the file which is to be updated.

.Example:

Look at this example taken from para. 6.3:

```
nypatchy - mumerge.car - mumerge.lis  .go  <</
+UPDATE.
+USE, T=EXE.
+LIST, MUOSLO, MULUND.
+PAM, T=ATT           .muoslo.ucra
+PAM, T=ATT           .mulund.ucra
+PAM, T=ATT, UPDATE   .muorg.car
+QUIT.
/
```

The **PAM** file "**muorg.car**" is updated to give file "**mumerge.car**";
the corrections are supposed to be contained in the two patches **P=MUOSLO** and **P=MULUND**, on the 2 files "**muoslo.ucra**" and "**mulund.ucra**".

(If one is using Patchy version 5.03/8 or higher one can give

```
+PAM, T=ATT, UPDATE, MERGE, DISPLAY   .muorg.car
```

see para. 6.3 for details.)

4.11 Handling of clashing actions

With the original philosophy of using Patchy, one would never for a long time change the content of patches on a PAM file directly, but rather do the necessary changes by sets of corrections with actions +ADD, etc. Once a given development was finished, one would add the corrections as a correction patch right at the beginning of the PAM file, which itself would then no longer change. This cumulation of correction patches required a defined behaviour for apparently conflicting actions. For example: an action

```
+REPL, MQ, MQLIFT, 72-79.
```

in some (younger) correction patch, followed by an action

```
+ADD, MQ, MQLIFT, 76.
```

in an (older) correction patch somewhere further down on the PAM file, was, and is, a correct operation, replacing a piece of code including the (older) correction.

These two actions are said to be "clashing at level 1", and do not cause a warning printed, unless asked for with

```
+PARAM, CLASH, N=1.
```

If however the order of these two actions on the PAM file were reversed, this would most likely be a mistake, trying to add to material already deleted; this would be signalled as a "clash of level 2".

Two actions +ADD (or +ADB) addressing the same target line are considered to be clashing at level 1, and both actions are accepted, the upstream material being taken first.

Of two actions +REPL or +DEL with overlapping target ranges only the action with the lower target edge is taken, the other is considered over-ruled, except that the delete ranges are merged.

Note that the following are not at all clashing with the above +REPL action:

```
     +ADB, MQ, MQLIFT, 72.
or   +ADD, MQ, MQLIFT, 79.
```

because they add just before or just after the delete range 72-79.

.Version 5

To be back-compatible, the "normal" clash handling of Patchy version 5 is the same as that of version 4, and this is what has been described above.

In the context of parallel developments starting from the same PAM file, managed by the new auxiliary Nydiff and described in para. 6.3, we need a different clash handling:

When merging two correction cradles operating on the original PAM file, it would not be correct to skip one of two actions with overlapping line ranges. Thus Nypatchy has to accept all actions on the same footing, leaving it to some responsible person to sort out the problem. All it can do is to provide clash warnings, and context listing. The concepts "older" and "younger" do not apply, hence all actions with colliding line ranges are signalled as of level 2.

This (new) behaviour is selected by giving the T=MERGE parameter on the control line +PAM calling for the the PAM file to be processed in this mode, for example (from para. 6.3):

```
+PAM, T=ATTACH, UPDATE, MERGE, DISPLAY   .muorg.car
```

(The DISPLAY parameter demands that deleted material should also be displayed in the listings from Nypatchy, see para. 4.5.)

Please note that the parameters MERGE and DISPLAY are new with version 5.03/8, which is presently in the NEW area.

Chapter 5: Example jobs using Nypatchy

In this chapter we will give a number of examples showing the use of Nypatchy, and for the first few, its collaboration with Nyshell.

5.1 Ex1: Create the Patchy modules from the PAM file

As the first example we take the jobs used to make a new version of Patchy from its `PAM` file, as used on Unix, `DEC VMS`, and `IBM/VM`, (taken from `P=INSTAL` of the Patchy `PAM` file).

This is the classical example of how to handle most conveniently complete programs or packages. One can edit the `PAM` file to bring it into the shape wanted, and then one runs a job, like the one shown here, to make the library for the package, and maybe one or even several executable modules. Nyshell will look after re-compiling just the routines which have been changed. This is how the job works:

A directory, called `wk_patchy` in this example, is reserved to contain the source and object files of this package only. Nypatchy is made to operate in `MODIFY` mode, to deliver all routines one-by-one onto separate source files, but without actually writing a particular source file if it exists already and if its content is what it should be, thereby keeping the time-stamp intact. Apart from the source files themselves, Nypatchy delivers their names and dependencies on the log file, here its name is source.log, to be used by Nyshell.

To make sure that all and only those routines which need to be, are re-compiled, Nyshell will compare for each routine the date of its .o file, of its source file, and of any include file used, and whether the routine is to be compiled in the same way as last time. It will also check all .o files present in the directory, and delete any trailing .o file if its name no longer appears on the log file. The output from Nyshell is a shell-script, called source.shfca in this example, to compile the routines which need to be compiled. A more complete description of Nyshell is found in Chapter 6.

The compilations are then performed by running this resulting shell-script, the libary is built by collecting all .o files.

In our example the executable modules are linked using the new library.

Note that it is very important to test the return status of Nypatchy, because it stops delivering `ASM` material as soon as it detects an error, thereby cutting the log file short, although it continues to idle through the entire run to see whether there are more errors.

Status return codes from Nypatchy and the Auxiliary programs are:

```
               UNIX            VAX       IBM
 $? or $status = 0    $STATUS = 1    rc = 0   normal
                 1              9         1   no-op from Nyshell
                 2              4         8   error
```

Note however that Nyshell is not available on `IBM/VM`.

This is the Unix job to run on SUN:

```
#!/bin/csh -f -v
# Create Patchy 5 modules

  set  PRO     = /cern/pro/bin/
  set  PAM     = /cern/new/src/car/patchy.car
  set  KERNLIB = /cern/new/lib/libkernlib.a

  set  NEW     = ~/uty/new
  set  EXTRA   =  ' '

  setenv  MACHINE   SUN
     set  LOAD   = 'f77 -Bstatic'

 cd  $NEW                                 | operate in the
 if (-d wk_patchy == 0)  mkdir wk_patchy    | directory $NEW/wk_patchy
 cd     wk_patchy                         |

 ${PRO}nypatchy  $PAM  .go  <</         | run Nypatchy in MODIFY mode
+ASM, FORT, T=MODIFY, LOG  .source.log  | deliver all routines
+EXE.                                   | for machine MACHINE
+USE, *PATCHY, ${MACHINE}.              | create file source.log to
+OPT, MAP.                              | be read by Nyshell
+PAM.                                   |
/                                       |
 set rc = $status                       | stop on Nypatchy errors
 if ($rc != 0)  exit                    |

 ${PRO}nyshell source  u  .go <</   | run Nyshell
/                                   | input:  source.log
 set rc = $status                   | output: source.shfca
 if ($rc != 0)  exit                |

 csh -f -v  source.shfca        | compile

 if (-f p5lib.a)  rm p5lib.a     | put all .o files onto a library
 ar rc  p5lib.a   *.o           |
 ranlib p5lib.a

 $LOAD -o $NEW/nypatchy  npatch.o   p5lib.a $KERNLIB $EXTRA
 $LOAD -o $NEW/nyindex   nindex.o   p5lib.a $KERNLIB $EXTRA
 $LOAD -o $NEW/nylist    nlist.o    p5lib.a $KERNLIB $EXTRA
 $LOAD -o $NEW/nydiff    ndiff.o    p5lib.a $KERNLIB $EXTRA
 $LOAD -o $NEW/nyshell   nshell.o   p5lib.a $KERNLIB $EXTRA
 $LOAD -o $NEW/nycheck   ncheck.o   p5lib.a $KERNLIB $EXTRA
 $LOAD -o $NEW/nytidy    ntidy.o    p5lib.a $KERNLIB $EXTRA
 chmod  755  $NEW/ny*
```

The corresponding job to run on **DEC VMS**:

```
$ set verify = procedure
$ set verify = noimage
$ on error     then $ goto exit
$ on control_y then $ goto exit
$!
$!   COM-file to create the Patchy 5 modules
$!
$ assign  [zoll.p5.dev]   OPDIR
$ assign  [zoll.p5.wyl]   SRCDIR
$ assign  [zoll.vaxlib]   LIB
$ assign  [zoll.vaxnew]   NEW
$ assign  lib:kernlib     KERNLIB
$ assign  lib:p5lib       P5LIB
$ uselib :== KERNLIB/LIB,SYS$LIBRARY:VAXCRTL/LIB
$!
$ set default OPDIR
$ set default [.wk_patchy]
$ nypatchy  SRCDIR:patchy.car  .go
+ASM, FORT, T=MODIFY, LOG   .source.log
+EXE.
+USE, *PATCHY, VAX.
+OPTION, MAPASM.
+PAM.
+QUIT.
$ if $status .ne. 1 then goto exit
$ nyshell source.log u eof  .go
$ if $status .ne. 1 then goto exit
$ @source.shfca
$ purge/nolog/noconfirm
$ lib/create P5LIB *.obj
$ link/nomap/exe=NEW:nypatchy  P5LIB/INC=npatch/LIB,'uselib
$ link/nomap/exe=NEW:nyindex   P5LIB/INC=nindex/LIB,'uselib
$ link/nomap/exe=NEW:nylist    P5LIB/INC=nlist/LIB,'uselib
$ link/nomap/exe=NEW:nydiff    P5LIB/INC=ndiff/LIB,'uselib
$ link/nomap/exe=NEW:nyshell   P5LIB/INC=nshell/LIB,'uselib
$ link/nomap/exe=NEW:nycheck   P5LIB/INC=ncheck/LIB,'uselib
$ link/nomap/exe=NEW:nytidy    P5LIB/INC=ntidy/LIB,'uselib
$exit:
$ set default OPDIR
$ set noverify
$ exit
```

Although Nyshell is not available on IBM VM/CMS, for completeness we still show the installation of Patchy on this machine:

```
/*BATCH TIME  9:00 */
/*BATCH PUNCH 100K */
/*BATCH PRINT 100K */
/*BATCH STORAGE   48M       */
/*BATCH NORETURN "LOAD MAP"        */
Trace C
fortlev  =  4                           /* accepted error level*/
FORTMOD  = 'FORTVS2'                    /* Fortran Compiler  */
FORTLIBS = 'CMSLIB VSF2LINK VSF2FORT'   /* Fortran Libraries */
MACLIBS  = 'CMSLIB DMSSP'               /* Macro Libraries   */

/**   run NYPATCHY to make the compilable file   **/
'nypatchy  a/patchy  a/p5asm  a/p5cra  .go'
if RC /= 0  then Exit 9

/**   compile to make P5ASM TEXT           **/
'EXEC VFORT P5ASM (NOLIST'

/**   run EDITLIB + TXTLIB to make P5LIB TEXTLIB   **/
'FILEDEF FT06F001 DISK P5WK LISTING A'
'EDITLIB P5ASM'
'TXTLIB GEN P5LIB P5ASM'

/*    create a dummy routine ENDMODU to be used to mark the end  */
line.1="      BLOCK DATA  ENDMODU"
line.2="      END"
line.3=""
'EXECIO * DISKW ENDMODU FORTRAN A ( STEM LINE. FINIS'
fortmod 'ENDMODU (NOPRINT'

/**   create the modules of Patchy 5          **/
'GLOBAL TXTLIB P5LIB KERNLIB 'FORTLIBS

'LOAD NPATCH ( CLEAR NOAUTO' ; 'INCLUDE ENDMODU'
'GENMOD NYPATCHY MODULE A ( FROM NPATCH TO ENDMODU'
say 'GENMOD creation of NYPATCHY returned RC='RC

'LOAD NINDEX ( CLEAR NOAUTO' ; 'INCLUDE ENDMODU'
'GENMOD NYINDEX MODULE A ( FROM NINDEX TO ENDMODU'
say 'GENMOD creation of NYINDEX returned RC='RC

'LOAD NLIST ( CLEAR NOAUTO' ; 'INCLUDE ENDMODU'
'GENMOD NYLIST MODULE A ( FROM NLIST TO ENDMODU'
say 'GENMOD creation of NYLIST returned RC='RC

'LOAD NDIFF ( CLEAR NOAUTO' ; 'INCLUDE ENDMODU'
'GENMOD NYDIFF MODULE A ( FROM NDIFF TO ENDMODU'
say 'GENMOD creation of NYDIFF returned RC='RC

'LOAD NCHECK ( CLEAR NOAUTO' ; 'INCLUDE ENDMODU'
'GENMOD NYCHECK MODULE A ( FROM NCHECK TO ENDMODU'
say 'GENMOD creation of NYCHECK returned RC='RC

'LOAD NTIDY ( CLEAR NOAUTO' ; 'INCLUDE ENDMODU'
'GENMOD NYTIDY MODULE A ( FROM NTIDY TO ENDMODU'
say 'GENMOD creation of NYTIDY returned RC='RC

/*BEGIN P5CRA CRADLE RECFM F LRECL 80         ---------------*/
+EXE.
+USE, *PATCHY, IBMVM.
+PAM.
+QUIT.
```

5.2 Ex2: Make test versions of Kernlib

This example makes test versions of the KERNLIB library for the Apollo. The approach is the same as in example 1, but note the differences: We are reading 2 PAM files, kernapo and kernfor. Their names are given as environment variables, because only those Nypatchy can substitute, it does not recognize shell variables (although as written here the substitution is actually done by the shell).

Furthermore we are using the same source files to make 2 libraries, a normal one and a library to be used with the debugger. The debug options are selected as parameters to Nyshell: "-dbs" for Fortran and "-g" for CC, which are to be added to the normal compiler options.

```
#!/bin/csh -f -v
# Shell script to create short KERN libraries on Apollo

 setenv KERNMACH    /cern/new/src/car/kernapo.car
 setenv KERNFOR     /cern/new/src/car/kernfor.car
    set KERNLIB   = ~/kern/inst/libkernsh.a
    set KERNLIBDB = ~/kern/inst/libkerndbg.a

 cd   ~/kern/inst
 if (-d wk_kern == 0)  mkdir wk_kern
 cd     wk_kern

 nypatchy  .go  <</
+ASM, FORT, T=MODIFY, LOG  .source.log
+EXE.
+USE, *KAPO.
+PAM, T=ATTACH   .${KERNMACH}
+PAM, T=ATTACH   .${KERNFOR}
+QUIT.
/
 if ($status != 0)  exit

#  ----------------------------------------  compile debug library
 if (-d wk_dbg == 0)  mkdir wk_dbg      | cd is ~/kern/inst/wk_kern/wk_dbg
 cd     wk_dbg

 nyshell ../source  u  .go <</          | take the source from ../
 +fopt    -dbs                          | and re-compile the changes
 +copt    -g                            | into ./
/                                       |
 if ($status != 0)  exit

 csh -f -v  source.shfca

 if (-f $KERNLIBDB)  rm $KERNLIBDB
 ar rc  $KERNLIBDB  *.o
 cd ../

#  ----------------------------------------  compile normal library
 nyshell source  uq  eof  .go           | cd is ~/kern/inst/wk_kern
 if ($status != 0)  exit                | take the source from here
                                        | and re-compile the changes
 csh -f -v  source.shfca                | into here
                                        |
 if (-f $KERNLIB)  rm $KERNLIB
 ar rc  $KERNLIB  *.o
```

5.3 Ex3: Make a test version of the Zebra library

This example makes a test version of the Zebra library, and at the same time a library containing some routines needed only for the standard verification runs, which check the correct operation of Zebra. On the PAM file, the pilot patch *ZEBRA selects all the normal routines, the others are USE selected from the sub-pilot P=QTESTLIB.

To separate the 2 categories we drive the normal routines into the directory wk_lib with the first +ASM line, and with the second +ASM line, for stream FORT:3, we send the verification routines into the directory wk_test. To identify them we attach the processing mode XDIV to all the material which is USE selected starting from QTESTLIB through all levels: T=TRANS.

```
#!/bin/csh -f -v
# Shell script to create the ZEBRA libraries

 set ZPAM   = ~/zebra/pro/zebraq.car
 set ZLIB   = ~/zebra/pro/zebra.a
 set ZLIBTE = ~/zebra/pro/zebrate.a

 cd   ~/zebra/pro
 if (-d wk_lib  == 0)  mkdir wk_lib
 if (-d wk_test == 0)  mkdir wk_test

 nypatchy  $ZPAM   .go  <<\\
+ASM, FORT,   T=MODIFY, PREFIX, LOG  .wk_lib/source.log
+ASM, FORT:3, T=MODIFY, PREFIX, LOG  .wk_test/source.log
+EXE.
+USE, QTESTLIB, T=XDIV, TRANS.
+USE, *ZEBRA, APOLLO.
+OPTION, ALL, MAP.
+PAM.
+QUIT.
\\
 if ($status != 0)   exit

#   ------------------------------     Compile for library zebra.a
 cd    wk_lib
 if (-f $ZLIB)  ar xo $ZLIB         | restore the .o files

 nyshell source  u  eof   .go
 set rc = $status                   |
 if ($rc == 1)   goto libok         | status 1: no new compilations
 if ($rc != 0)   exit               |        2: error

 csh -f -v  source.shfca
 if (-f $ZLIB)  rm $ZLIB
 ar rc  $ZLIB  *.o
libok:
 rm  *.o                            | delete the .o files

#   ------------------------------     Compile for library zebrate.a
 cd ../wk_test

 nyshell source  u  eof   .go
 if ($status != 0)   exit

 csh -f -v  source.shfca
 if (-f $ZLIBTE)  rm $ZLIBTE
 ar rc  $ZLIBTE  *.o
```

It is somehow a pity to keep all the .o files twice, once as individual files and again their copies on the library. In this example we remove the .o files, and regenerate them from the library on the next run, so that Nyshell can look at their dates, this is done with the "xo" parameter to ar on some systems, on others the "x" alone is enough to preserve the date. This procedure saves disk space but costs time.

5.4 Ex4: Develop a new part of Zebra

Suppose one has the PAM file and the library of the previous example, and that one wants to develop a new part to be added to the Zebra package, without modification to the existing material. Suppose also that there are only a few routines, the time to recompile them for each shot is negligeable. Suppose further that patch P=NEW on file zenew.car contains all the new material, all the new routines and also any code to test them, including the Main program.

```
#!/bin/csh -f -v
# Shell script to develop a new part of ZEBRA

 set ZPAM    = ~/zebra/pro/zebraq.car
 set ZLIB    = ~/zebra/pro/zebra.a
 set ZLIBTE  = ~/zebra/pro/zebrate.a
 set KERNLIB = /cern/pro/lib/libkernlib.a

 nypatchy  $ZPAM  zetestxf.f -- zetestxc.c  .go  <<\\
+ASM, INCL, T=SPLIT.  (only if C include files in the new stuff)
+USE, NEW, T=EXE.
+USE, *ZEBRA, SUN.
+OPTION, MAP.
+PAM.
+PAM, T=ATTACH   .zenew.car
+QUIT.
\\
 if ($status != 0)   exit

 cc  -g -c zetestxc.c
 f77 -g -o zetest zetestxf.f zetestxc.o $ZLIBTE $ZLIB $KERNLIB
 zetest
```

5.5 Ex5: Using Nypatchy without a PAM file

This is a shell-script to create an executable module from the source code given as a here-document to Nypatchy in the script itself. This gives easy propagation of the COMMON declaration, and easy editing with Wylbur because the source is structured into decks.

```
#!/bin/csh -f -v
#     totex.sh: script to make the module totex

 set  KERNLIB = /cern/pro/lib/libkernlib.a

 nypatchy - totexxf  .go   <<'EOF'
+EXE.
+USE, TOTEX.
+PATCH, TOTEX.
+KEEP, ME.
      CHARACTER      CHIN*256, CHEX*256, COL(132)*1, COLE(256)*1
      COMMON /ME/    NBLK, NBLN, NCHIN, NCHTK, NCHEX, CHIN, CHEX
                     EQUIVALENCE (COL,CHIN), (COLE,CHEX)
      COMMON /SLATE/ NDSLAT, NESLAT, NFSLAT, NGSLAT, NNSLAT(36)
+DECK, MAIN, T=JOIN.
      PROGRAM TOTEX

C-    Translate flat text to Latex

+SEQ, ME.
      ...

      END
+DECK, ARRIVE.
      SUBROUTINE ARRIVE

C-    Read next line, store it into CHIN, length NCHIN
C-    skip and count blank lines in NBLN

+SEQ, ME.
      ...

      END
+DECK, DEPART.
      SUBROUTINE DEPART (JLEV)
      ...

+DECK, VERBATIM.
      SUBROUTINE VERBATIM
      ...

+DECK, NOSUB.
      SUBROUTINE NOSUB
      ...

+DECK, BOLD.
      SUBROUTINE BOLD
      ...

+DECK, DOLINE.
      SUBROUTINE DOLINE
      ...

+DECK, SCAN.
      SUBROUTINE SCAN (JLOOK, JFIND, LENWD)
      ...

+DECK, COPST.
      SUBROUTINE COPST (JF,N)
      ...

+QUIT.
'EOF'
 if ( $status != 0 ) exit
 f77   -o totex  totexxf.f    $KERNLIB
 rm totexx*
```

Chapter 6: Auxiliary programs

To get help for the auxiliaries one can type on the computer for example:

```
      nylist help .no
   or nylist - h
```

The first will give details on how the parameters, file names and options, are to be specified; with the second the H option is given which will cause the particular program to print a short description of itself.

For the "option" parameter one gives one letter for each option selected (case insensitive), for example:

```
nyindex mypam z   mylist .go
nylist  mypam es +mylist .go
```

selects the Z option for Nyindex, and the E and S options for Nylist.

On the **VAX** or Alpha with system **VMS**, a file name may be given directly in Unix style; if given in **VAX** style Patchy will convert it for internal handling and analysis to Unix style, the correspondence is for example:

```
/cern/new/src/car/patchy.car;4
cern:[new.src.car]patchy.car;4
```

Here are some more **VAX** equivalents:

```
//node/log/a/b/f.e;17            ~/a/b/f.e          ~/f.e
node::log:[a.b]f.e;17   disk:[name.a.b]f.e   disk:[name]f.e

      a/b/f.e    ../a/b/f.e    ../f.e    /(a.b/f.e
    [.a.b]f.e    [-.a.b]f.e    [-]f.e     [a.b]f.e
```

On the IBM with VM/CMS, file names have to be given in one of two possible ways, for example:

```
         patchy.car.a   or   a/patchy.car
to mean  PATCHY CAR A
```

The status returned is:

```
           UNIX            VAX       IBM
$? or $status = 0    $STATUS = 1    rc = 0   normal
                1              9         1   warning | void operation
                2              4         8   error
```

The meaning of the "warning" status depends a bit on the program, for Nyshell it means that nothing needs to be re-compiled.

6.1 Nyindex and Nylist – to print `PAM` file listings

Nyindex will print a table of contents of the patch and deck names encountered, as well as a sorted index of patch and deck names, and of sequences defined on the `PAM` file:

```
nyindex  pam.car  options  print
```

with:

pam the name of the `PAM` file, .car is the default extension;

options B – bare, table-of-contents without comments,
H – print the help information only,
P – patch names only in the table-of-contents,
Q – quick, suppress the table-of-contents,
X – suppress the index of patch, deck, sequence names,
Y – suppress the index of deck and sequence names,
Z – suppress the index of sequence names,
n – page size control, as for Nylist, see below,
+ – inhibit the initial page-eject.

print printed output file, default: standard output.

Nyindex will check if there are two decks of the same name in the same patch, and if so, it will print a "duplicate" warning and take the "warning" exit (status =1 on Unix). (With version 5.03/0 there was a bug: if the X option has been selected the status will be 0 always.)

When printing the table-of-contents Nyindex will normally include the comment fields found on the `+PATCH` and `+DECK` lines; if this is not wanted one can give the B option.

Nylist will list a Pam file, with line numbering, both local to the deck and global in the file. If the S option is given each deck will start a new page, unless the page is almost empty, or if its `+DECK` line carries a `T=JOIN`. If the E option is given, page breaks within a deck can be controlled with lines `+SEQ,QEJECT,N=n`.

```
nylist  pam.car  options  print
```

with:

pam the name of the `PAM` file, .car is the default extension;

options H – print the help information only,
S – start a new deck on a new page,
E – page ejects with `+SEQ,QEJECT` honoured (only with S),
n – page size control, see below,
+ – inhibit the initial page-eject.

print printed output file, default: standard output.

The page size of the print medium is assumed to allow for 110 lines per page, this can be overruled by giving options 0, 1, 2, 3, or 4, to set 58, 62, 74, 84, or 98 lines per page.

Starting with version 5.03/8, one can alternatively give the number of lines in clear as an integer within the option parameter, for example:

```
xbanner 1=mylist.lis  |List||mypam|Vs|1.24|
nyindex mypam z104  +mylist .go
nylist  mypam es104 +mylist .go
nycheck mypam +u999 +mylist .go
```

6.2 Nysynopt – to print synoptic PAM file listings

Nysynopt will print a line-by-line listing of PAM files in the style of Nylist, but with in-line expansion of sequence calls, and with markers representing actions +ADD etc. at the point where the action would operate. Thus, if one is working on a listing of a particular routine, one has the declaractions of the COMMON variables used by this routine also displayed on one's listing. For optional code controlled by multiple sequence definitions one obtains thus a synopsis of all the possibilities. (Nysynopt is available starting with version 5.03/28).

The operation of Nysynopt can be controlled in detail with control lines given in the cradle (which is not possible with Nylist). These control lines are a sub-set of the control lines for Nypatchy, as explained later.

.Program call statement

```
nysynopt  pam.car  options  cradle  print
```

with:

- `pam` the name of the PAM file, .car is the default extension;
  PAM files may also be attached with lines +PAM given in the cradle, as for Nypatchy.
- `options` H – print the help information only,
  S – start a new deck on a new page,
  E – page ejects with +SEQ,QEJECT honoured (only with S),
  M – missing sequences to be signalled,
  I – line numbers for individual PAM files each start at zero,
  X – do not signal actions,
  Y – do not expand sequences called by +CDE,
  Z – do not expand sequences called by +SEQ,
  n – page size control as for Nylist, see above,
  + – inhibit the initial page-eject.
- `cradle` the file name of the cradle, default: standard input;
  one may give the 3 characters EOF in this position to indicate that there is no cradle, in which case Nysynopt will simply list the complete PAM file given.
- `print` printed output file, default: standard output.

.Control lines in the cradle

Nysynopt will act on a certain set of control lines given in the cradle, but, unlike Nypatchy, not on control lines occurring on the PAM files to be listed, with the exception of sequence definitions and calls, and of the actions +ADD etc.

These *cradle* control lines are:

```
+LIST.
+LIST, (P=)pname,...
+LIST, (P=)pname, D=dname,...
```

The material to be listed has to be indicated, either everything or selected pieces. Control lines +FORCE and +SUSPEND are also available. Alternatively one may give the parameter T=LIST on the control line +PAM to list everything on the particular PAM file.

```
+USE, (P=)pname, [D=dname,] ..., T=INHIBIT.
```

Nysynopt is initially set up to accept everything, corresponding to `+USE.` without parameters. If one wanted to exclude a patch or a deck, so as to prevent the sequences or actions defined in there to appear in the listing of the other material, one can inhibit its use.

```
+PAM, (LABEL=)n, N=xx, T=opt, RETURN=pname  .file-name
```

is used to call up a `PAM` file for processing. The syntax is the same as for Nypatchy, except that the parameters `T=UPDATE`, `T=MERGE`, `T=DISPLAY`, and the `IF=` parameters are meaningless. Special to Nysynopt are the parameters `T=LIST` and `T=INDIVIDUAL`, here is the summary:

`Label=n`	a small integer to identify the file, this is only needed for stop-go reading, as for Nypatchy.
`N=ndo`	number of `PAM` files to be processed on this file; default: ndo=999
`N=nsk,ndo`	ignore nsk `PAM` files and then process ndo `PAM` files on this file.
`T=Attach`	open the file whose name is given in the comment field and process it.
`T=Hold`	no rewind of the file after processing, default: rewind after.
`T=Resume`	no rewind of the file before processing, default: rewind before.
`T=LISt`	do list all the material on this file.
`T=INDiv`	if there are several `PAM` files on this file, start line numbers from zero for each individual `PAM` file.
`R=pname`	stop processing the file when the processing of patch "pname" is complete and revert to the cradle.

```
+KEEP, sname.
+KEEP, sname, T=SINGLE.
```

Many programs use sequence definitions for simple functions which are slightly different on different machines, and it might be a nuisance to see all of them expanded in every routine. By using the first form one may completely suppress expansions of the particular sequence; using the second form will suppress showing further definitions beyond the first. Note well that this is for `+KEEP` given in the *cradle*.

Other Nypatchy control lines available in the cradle for Nysynopt are:

```
+OPTION, COMPACT, EJECT (,OFF)
+PARAM,  LINes, N=n.
+PARAM,  COLumns, N=n
```

.Usage of Nysynopt, example 1

Straighforward listing of the PAM file micky.car:

```
#!/bin/csh -f
#  script to list PAM file micky
 set verbose

 set TIME  = "july 95"
 set NAME  = micky
 set PAM   = ~/kern/wyl/$NAME
 set LIST  = $NAME.lis

 if (-f $LIST)   rm $LIST
 xbanner  1=$LIST  KERN SYNOPT "$TIME" // $NAME .car

 nyindex   $PAM  -      +$LIST  .go
 nysynopt  $PAM  - EOF  +$LIST  .go
```

In this example everything is listed because there is no cradle; listing is compact because the S option is not given.

.Usage of Nysynopt, example 2

This shell script might be used to list the PAM file patchy.car:

```
#!/bin/csh -f
 set verbose

 set VS   = "5.03/28"
 set PAM  = /cern/new/src/car/patchy.car
 set LIST = synoptk.lis

 if (-f $LIST)   rm $LIST
 xbanner   1=$LIST  "|PATCHY 5|LIST|patchy||$VS||SYNOPT|"

 nyindex   $PAM  -     +$LIST  .go
 nysynopt  $PAM  ES -  +$LIST  .go  <</
+LIST.             list everything
+KEEP, Q_AND.      do not expand these sequence calls
+KEEP, Q_OR.
+KEEP, Q_SHIFTL.
+KEEP, Q_SHIFTR.
+KEEP, Q_JBIT.
+KEEP, Q_JBYT.
+KEEP, QCARDL.
+KEEP, NEWLINE, T=SINGLE.  show only the first definition
+PAM.                      go
/

 xprint -d 4050  -f P1RM -cc -r X8  $LIST
 rm  $LIST
```

In this example everything is listed because of the line +LIST.; there is only one PAM file, its name is given in the program-call, and it is called for processing with the line +PAM.

.Usage of Nysynopt, example 3

The master source of Zebra is held on a number of separate files for ease of handling, listing in particular. The COMMON declarations for all packages are on file zecde.car. The various packages are on files zemq.car, zefq.car, zejz91.car, etc. (These files are concatenated for release delivery to zebra.car).

To list one of these file, the file-I/O package for example, one calls a shell script with

```
synzeb zefq
```

where the script synzeb is:

```
#!/bin/csh -f -e
 unset verbose
 set NAME  = "$1"
 if ( $NAME == '' ) then
   echo " Usage:  synzeb name  :> print zebra/wyl/name.car"
   exit
  endif

 set verbose
 set VERS  = "3.76"
 set DATE  = "Oct 95"
 set LIST  = zebk.lis

 setenv CDE  ~/zebra/wyl/zecde.car
 setenv PAM  ~/zebra/wyl/$NAME.car

 if (-f $LIST)   rm $LIST
 xbanner   1=$LIST  "|ZEBRA|LIST||$VERS|$DATE||$NAME|"

 nyindex   $PAM -     +$LIST
 nysynopt  -    ES -  +$LIST  <<'stop'
+KEEP, QCARDL.
+KEEP, Q$ANDOR.
+KEEP, Q$SHIFT.
+KEEP, Q$JBIT.
+KEEP, Q$SBIT.
+KEEP, Q$SBYT.
+KEEP, Q$CBYT.
+KEEP, Q$JBYTET.
+KEEP, MZBITS,   T=SINGLE.
+KEEP, MZCA,     T=SINGLE.
+KEEP, ZMACHFIX, T=SINGLE.
+KEEP, ZCETA,    T=SINGLE.
+KEEP, QTRACE,   T=SINGLE.
+KEEP, QTRACEQ,  T=SINGLE.
+KEEP, QTRACE99, T=SINGLE.
+KEEP, QTOFATAL, T=SINGLE.
+PAM, T=ATTACH     .${CDE}
+PAM, T=ATT, LIST  .${PAM}
'stop'

 xprint -d 4050  -f P1RM -cc -r X8  $LIST
 rm  $LIST
```

In this example there are two PAM files, the first is not listed, it only supplies sequence definitions; the second is listed because of the T=LIST parameter.

6.3 Nycheck and Nytidy – to clean PAM files

Nycheck will check all the control lines on a Pam file, reporting syntax errors. This is more thorough than Nypatchy itself, which tolerates syntax errors on `IF` de-selected control lines, and which does not look at the contents of patches or decks which have not been `USE` selected.

```
nycheck  pam.car  options  print
```

with:

`pam`	the name of the `PAM` file, .car is the default extension;
`options`	H – print the help information only, U – presence of user data types to cause a warning, n – page size control, as for Nylist, see above, + – inhibit the initial page-eject.
`print`	printed output file, default: standard output.

The error exit is taken if errors have occurred, similarly for warnings.

Nytidy will make a cleaned-up copy of a Pam file:

```
nytidy  org.car  new.car  options  print
```

with:

`org`	the name of the original `PAM` file, .car is the default extension;
`new`	the name of the new `PAM` file, if this is not given the result will replace the original;
`options`	C – the original is a `CMZ` output file, H – print the help information only, V – verbose, print each change, + – inhibit the initial page-eject.
`print`	printed output file, default: standard output.

Nytidy will remove trailing blanks on all lines. In decks of type `FORT` trailing comment lines will be removed, starting with version 5.03/8 leading comment lines are also removed. It will replace old forms of some built-in sequences by their new forms; it will remove lines `+DECK,BLANKDEK.` coming from `CMZ`.

If "new" is a pure directory, the file-name part will be inherited from "org" (starting with version 5.03/12).

6.4 Nydiff – to compare two PAM files for differences

Nydiff compares two different evolutionary stages of the same PAM file and delivers the differences found as a correction patch, which when applied with Nypatchy to the file of the earlier stage will upgrade it to the later stage. Re-ordering of patches or decks cannot conveniently be done with Nypatchy, therefore such operations are to be done with Wylbur, for which Nydiff delivers an exec file. The program call is:

```
nydiff  org.car  new.car  name  options  print
```

with the parameters:

org the name of the original, the reference PAM file;

new the name of the derived PAM file, .car are the default extensions.

name the name to be used for the correction files to be created:
name.ucra: the correction patch for Nypatchy,
name.uexe: the re-ordering instructions for Wylbur.

options A – anyway, by-pass the first test on sufficient matching,
F – force, by-pass the second test on sufficient matching,
H – print the help information only.

print printed output file, default: standard output.

When deriving the corrections, it is recommended to check back immediately in the same job that everything is well, as for example with:

```
#!/bin/csh -f -v
#     Make the Muon correction cradles
 set  ORGF  = muorg
 set  NEWF  = munew
 set  CRAD  = muoslo

# get the correction files             status returned by Nydiff:
 nydiff  $ORGF  $NEWF  $CRAD   .go            = 0  fine
 if ($status > 1)  exit                          1  warning
                                                 2  error

# apply .ucra
 nypatchy  -  temp.car   .go  <</
+UPDATE.
+USE, T=EXE.
+PAM, T=ATT          .${CRAD}.ucra
+PAM, T=ATT, UPDATE  .${ORGF}
/
 if ($status != 0)  exit

# apply .uexe
 use temp.car <</
@$CRAD.uexe
save -unn -rep
quit
/

# check for identity
 diff  temp.car  $NEWF.car
 if ( $status != 0 ) exit
 echo " Success."
 rm temp.car
```

Nydiff is provided to help in the coordination of parallel developments: two people working independently on the same piece of code, starting from the same PAM file, each one will have taken his private copy and change it with a text editor until his PAM file is in the wanted shape.

To merge the resulting two PAM files, one uses Nydiff to get the two correction patches, which are then applied together to the original file in one run of Nypatchy in update mode.

The real problem, however, is not so much the mechanics of merging the code of the parallel developments, but to verify their compatibility, a task which can only be done by a person who understands the code. All that Patchy can do to help here is to give "clash warnings" if two corrections address the same lines, and, very important, to provide an in-context listing of all the corrections done.

To be useful, the correction patch produced by Nydiff has to be *minimal*, for example: if a deck has changed its position in its patch, it would be useless to merely delete the old deck and add the new one in the new position, because changes from different people to this same deck could not then be collated.

For this reason, Nydiff makes a big effort to first find out who is who, before actually making the differencing between a deck on "new" and its corresponding deck on "org". It might be helpful to visualize this matching process:

1) matching by name assumes that patches of the same name on the two files are related, and also decks of the same name in related patches. Thus if one must change patch or deck names, never exchange them, but use new names. At the end of this process there is a test (which can be bypassed with the A option) to check that at least half of all decks or patches have been matched.
2) matching of patches by content compares all as-yet unmatched patches on "org" against all unmatched patches on "new": if either 3/4 of the deck names or 80 per-cent of the content are the same, the two patches are assumed to be related. Thus if one must change the name of a patch, try not also to change its deck names.
3) matching of decks by content compares all as-yet unmatched decks of related patches: if the content of a pair of decks is the same to the 80 per-cent level, they are assumed to be related.
4) a check is made to see whether an old patch has been split by inserting a new line +PATCH. After this there is a test (which can be bypassed with the F option) to check that at least 3/4 of all decks have been matched.
5) To find "lost" decks, decks which may have been transferred from one patch to another one, Nydiff takes each unmatched deck on "org" (of any patch which survives) and scans all patches on "new" to find an as-yet unmatched deck of the same name and with the same content to the 80 per-cent level.

On the result of the matching process the correction patch is built: unmatched patches or decks on "org" are deleted with +USE,T=INHIBIT. For matched decks the differences of content are recorded headed with +ADD, +REPLACE, +DELETE lines. Unmatched patches or decks on "new" are added, headed with +ADD. Instructions to re-order the PAM file resulting from the corrections are delivered as an Exec file for the Wylbur editor.

Obviously, as long as parallel developments of a PAM file are under way one should avoid cosmetic changes completely; rationalizing variable names, routine names, or statement numbers, tidying up the lay-out, re-ordering patches or decks, should be left to the person in charge of the PAM file, to be done after all developments have been merged. Although painful, this gives one a chance not to loose control over the evolution of a piece of software.

.Errors and warnings

Nydiff will give a "duplicate" informative message if it finds two decks with the same name in the same patch. A warning is given if a correction needs to be constructed for the first of a set of duplicate decks. A correction for any deck of a set of duplicates other than the first is an error, because Nypatchy would apply it to the first.

The error exit is taken if errors occured, similarly for warnings.

.Example:

Suppose two people in Oslo and Lund have worked starting from the same file muorg.car. When finished they deliver the corrections muoslo.ucra and mulund.ucra, obtained as shown in the first example of this paragraph. To merge the developments and to get a listing collating the changes one has to do this:

1) change the patch names to identify the origin of each correction. One can do this by hand of course, using Wylbur one would do this:

```
use  muoslo.ucra  <</
change 5/l to ', muoslo.' in '+PAT'1 xqt 1
save -unn -rep
use  mulund.ucra
change 5/l to ', mulund.' in '+PAT'1 xqt 1
save -unn -rep
quit
/
```

(This is needed at the moment; starting with version 5.03/8 Nydiff uses the file name also as name for the patch.)

2) run Nypatchy in Update mode:

```
nypatchy - mumerge.car - mumerge.lis  .go  <</
+UPDATE.
+USE, T=EXE.
+LIST, MUOSLO, MULUND.
+PARAM, CLASH, LEV=1.
+suspend, list.
+PAM, T=ATT           .muoslo.ucra
+PAM, T=ATT           .mulund.ucra
+suspend, off, list.
+PAM, T=ATT, UPDATE   .muorg.car
+QUIT.
/
```

Apart from delivering the updated `PAM` file on mumerge.car, it will list on file mumerge.lis all routines which have been changed, each correction being labelled with its origin. If one does not want to see the initial listing of the patches `MUOSLO` and `MULUND` one could suppress it with the control lines `+SUSPEND` as indicated.

Starting with version 5.03/8 one will probably want to give instead

```
+PAM, T=ATT, UPDATE, MERGE, DISPLAY   .muorg.car
```

The `MERGE` option tells Nypatchy to take all corrections, even if they have overlapping line ranges (cf. para. 4.11), and the `DISPLAY` option asks for deleted material also to be listed.

6.5 Nymerge – to ready PAM files for release

Nymerge is intended to help in the preparation of a new release of some piece of software. It copies the "old" PAM file to the "new" file, with:

a) optionally updating the version number and the date/time field on the PAM file title(s),

b) replacing decks for which there is a new version presented on the "merge" file, if any. The order of the decks on the "merge" file has to be the same as that of the corresponding decks on the "old" file.

The program call statement is:

```
nymerge  merge.car  old.car  new.car  options  print
```

with:

`merge` the name of the file containing the replacement decks, give "-" if none;

`old` the name of the PAM file to be updated;

`new` the name of the resulting PAM file, .car are the default extensions.

`options` H – print the help information only,
U – update version/date/time on all PAM titles found on the file,
F – update only the first PAM file title,
P – ponly: log only the patch names, not the decks,
Q – quick: no log printing of patch/deck names,
+ – inhibit the initial page-eject.

`print` printed output file, default: standard output.

.Example

The shell script on the next page is used to prepare the release of the file kernfor.car:

The current release is PUBPAM; the master source is ORGPAM, the file on which all the changes have been done with a text-editor. The job steps to create the file NEWPAM for release are:

1) print the banner page to LIST identfying the output.

2) Nypatchy in UPDATE mode is used to extract all the routines which have changed. Note that the blank decks of the patches concerned are also pulled out, to safeguard against duplicate deck names in different patches.

3+4) Wylbur is used to place the version stamp into all the new routines, which are then listed with Nylist.

5) Nymerge is used to merge the new decks into ORGPAM, also updating the PAM file title.

6) Wylbur is used to compare the new file against the current release, showing all differences and where they occur. In an extra step the output is copied to the print file LIST, left-shifted by inserting one blank at column 1.

7) Send the output LIST to print.

```
     #!/bin/csh -f -v -e
     # script to prepare kernfor.car for the next release
      set verbose

      set OLD     = "441"
      set TIME    = "sept 95"
      setenv STAMP '.VERSION KERNFOR  4.42  951011'

      set PUBPAM  = /cern/pro/src/car/kernfor.car  # the current release file
      set ORGPAM  = ~/kern/wyl/kernfor.car         # the master source file
      set NEWPAM  = ~/kern/new/kernfor.car         # the file for the next release
      set LIST    =  upfor.lis

      set xwork   = ~/work/upfor         # temporary working files:
      set newmat  = ${xwork}x2k          #   the new or changed routines
      set wylog   = ${xwork}.wyllog      #   the Wylbur diff result

1)    xbanner  1=$LIST   KERN UPDATE "$TIME" /// "old $OLD" kernfor

     # ----------------------------------------  extract the new material
2)    nypatchy  $ORGPAM  $newmat.car - +$LIST  .go  <<\\
     +UPDATE.
     +USE.
     +EXE, KERNFOR, D=, UPDATE.
     +EXE, UTYGEN, D=, FCASPLIT.
     +EXE, CCGEN, D=, VXINVB, VXINVC.
     +EXE, CCGEN, D=SIGUNBL, SIGPRNT.
     +EXE, TCGEN, D=, CCOSUB, CENVIR.
     +OPT, MAPASM.
     +PAM, T=UPDATE.
     \\

     #  ---------------------------------------  Wylbur, version stamps
3)    use -u  $newmat.car   <</
     change 38/l to '    :env:STAMP' in 'LIB#' & '.VERSION KERN'38> sh \pd
     sur
     quit
     /

     #  ---------------------------------------  list the new material
4)    nylist  $newmat  -  +$LIST  .go

     #  ---------------------------------------  make the new PAM file
5)    nymerge  $newmat  $ORGPAM  $NEWPAM  u  +$LIST  .go
      rm  $newmat.car

     #  ---------------------------------------  compare to last release
6)    use -u  $NEWPAM     >$wylog <</
     load -u  $PUBPAM
     diff e1 show \pd
     quit
     /
                              # copy result from previous step to the printed
6b)   use -u $LIST <</        # output shifted with blank inserted at col. 1
     end 1   ---------------  compare to last release
     end  .
     set val w11 *
     collect from $wylog  -nol
     change 1 to ' ' in :w11+/l -nol
     sur
     /
      rm  $wylog
7)    xprint -d 4050  -f P1RM -cc -r X8  $LIST
```

6.6 Nyshell – to construct the commands to compile

The purpose of Nyshell is to help in the making of load libraries containing the routines of complete software packages, in the following context:

The facilities of Unix being what they are, one has to have on a separate file the *source* of each routine which is to appear on the library as an individual object. Using the compilers, Fortran, C, or Assembler, depending on the nature of each source file, one compiles it to create the corresponding .o object file. Finally the wanted object files are to be collected together onto the library file, using the "ar" utility.

Thus the single concept "package X", whose source is one file "`PAM` file X", and whose object is again one file "library for X", has to be represented during the transition from the source to the object by a multitude of maybe very many .f, .c, .s, .o files. Fortunately the creation and deletion of directories is really easy with Unix. This allows a trick to represent all the intermediate files as one single entity: the "directory for X" containing this multitude of files, and *only* those files. Thus *all* the .o files in this directory represent the wanted library.

In practice, then, we proceed as can be seen from the first 3 examples in Chapter 5:

1) create the directory for package X if it does not yet exist;
2) run Nypatchy in `MODIFY` mode, sending the source of each routine as a separate file into this directory, and recording on the log file the nature and the name of each routine;
3) run Nyshell to read the log file, to decide which routines need to be (re)compiled, and to deliver a shell-script containing the commands to compile;
4) run this script to do the compilations;
5) run the ar utility to make the library.

To do what it is supposed to do, Nyshell needs several pieces of information:

a) the names and properties of all the routines, this is provided by the log file coming from Nypatchy, for example (Apollo):

```
fort:1     wylmain.ftn    same
fort:1     wyldo.ftn      same
       ...
fort:1     wwr_save.ftn   same
fort:1     wws_apo.ftn
fort:1     wws_stat.ftn
fort:2     wws_del.ftn
fort:1     wws_rolc.ftn   same
fort:1     wws_rolx.ftn   same
fort:1     wws_save.ftn   same
fort:1     wws_pseu.ftn
fort:1     wwx_stat.ftn   same
fort:1     wwx_del.ftn    same
fort:1     wwx_rolc.ftn   same
cc:1       wwx_fork.c          ww_incl.h
cc:1       wwx_poll.c     same ww_incl.h
cc:1       wwx_chk.c      same ww_incl.h
cc:1       wwx_ewait.c         ww_incl.h
fort:1     wwx_init.ftn   same
fort:1     wwx_fisc.ftn   same
cc:1       wwx_ficc.c          ww_incl.h
```

The first column gives the property and the second column gives the name of the source file; the *same* flag signals that Nypatchy did not actually write the source file because its content was unchanged; names of include files used, if any, are given at the end.

Note that the "property" is described by the name of the logical stream through which the routine was sent (cf. para. 4.8), thus it gives the data type *and* the diversion mode. In the example there is one routine which came through **`fort:2`**, to be compiled without optimization. For each data type there are 4 streams available, corresponding to the 4 diversion states.

b) are also needed the compiler options and the shape of the command to call the compiler for each logical stream, as well as the name of the compiler for each data type. Nyshell has built-in defaults for the machine it is running on; this can be replaced or modified with data lines given in the cradle to Nyshell.

c) finally, the state of affairs at the previous run of Nyshell is needed, to detect a change in the compiler options used, or a change in the diversion status of a particular routine. Each run of Nyshell records this on the ".xqtlog" file to be used with the next run, for example (Apollo):

```
>.xqtlog
>fort:1  -bounds_violation -info 1 -indexl -cpu mathlib_sr10 -dbs -opt 3
>fort:2  -bounds_violation -info 1 -indexl -cpu mathlib_sr10 -dbs -opt 0
>cc:1    -c -g -O
 fort:1  wylmain
 fort:1  wyldo
 fort:1  initapo
        ...
 cc:1    wwx_chk
 cc:1    wwx_ewait
 fort:1  wwx_init
 fort:1  wwx_fisc
 cc:1    wwx_ficc
```

This information is the basis for the decisions taken by Nyshell, when it is invoked with the U (up-to-date) option. A particular routine does *not* need to be recompiled if all of the following conditions are satisfied:

1) Nypatchy sends the *same* flag;
2) the compiler options for this routine are unchanged;
3) the "last modified" time stamps indicate that the .o file exists and is more recent than the source file, or any .h file used.

Moreover Nyshell has to check all the .o files present in the directory, to remove any file for which there is no source file indicated in the log. This is necessary to safeguard against name changes or deletions in the source.

This whole procedure is perfectly safe, one can kill a job whilst Nypatchy or Nyshell are running, and restart. One will never loose a needed compilation.

The shell-script delivered by Nyshell might be for example:

```
#!/bin/csh -f
# Script from nyshell for file  source.log
set FILE =  source
set s    = " "
set FO_1 = "-bounds_violation -info 1 -indexl -cpu mathlib_sr10 -dbs -opt 3"
set FO_2 = "-bounds_violation -info 1 -indexl -cpu mathlib_sr10 -dbs -opt 0"
set FC   = "/com/ftn"
set CO_1 = "-c -g -O"
set CC   = "cc"
${FC} ${s}wws_apo.ftn  ${FO_1};  mv wws_apo.bin  wws_apo.o
${FC} ${s}wws_stat.ftn ${FO_1};  mv wws_stat.bin wws_stat.o
${FC} ${s}wws_del.ftn  ${FO_2};  mv wws_del.bin  wws_del.o
${FC} ${s}wws_pseu.ftn ${FO_1};  mv wws_pseu.bin wws_pseu.o
${CC} ${CO_1} ${s}wwx_fork.c
${CC} ${CO_1} ${s}wwx_ewait.c
${CC} ${CO_1} ${s}wwx_ficc.c
# End of the shell script
```

With Unix one relies on the shell to do the parameter substitution; on `DEC` machines with `VMS` the substitution is actually done by Nyshell itself.

The program call to Nyshell (not available on `IBM VM`) is:

```
nyshell  name.log  options  cradle  print
```

with the parameters:

name the name of the log file coming from Nypatchy, .log is the default extension; the output files delivered will be:
name.shfca: the shell-script to be run for compilation,
name.xqtlog: the file remembering the present state.

options A – force recompilation of all routines,
B – bypass the tests based on the .xqtlog file,
E – bypass the tests based on the .xqtlog file if the file is empty,
H – print the help information only,
Q – quick, do not print the set-up,
S – print the setup only, only together with H,
U – up-to-date, check that all .o files are ready to go to the library,
V – verbose, print the complete set-up.

cradle name of the file with the user set-up commands, if the file does not exist this is not an error but a signal to use the set-up as is,
one may also give `EOF` to use the default set-up as is,
default: standard input.

print printed output file, default: standard output.

Nyshell and the resulting script *name*.shfca have to be run in the directory where the .o files are, or are to be. The source files, however, can be together with the log file in some other directory, example 2 of Chapter 5 uses this.

In the cradle one may give on data lines instructions on how the compilation commands are to be constructed. On each machine, Nyshell has a default set-up, which may be looked at as indicated on the next page. This response from Nyshell has been formatted to print valid lines as they would be given in the cradle (apart the annotations added by hand in the left margin).

The first word on each line is a tag identifying the information given on the right. Thus "fc", "cc", and "as", specify the command verbs to be used to call the compiler for Fortran, C, and assembler, respectively.

"fopt" gives the compiler options to be used for all logical streams of Fortran, and "fo:i" gives the extra options to be added to the common options for stream `FORT:i`.

"fort:i" specifies the complete command to be given to the shell to compile one routine, the escape `$*` indicating the place(s) where the routine name has to be inserted.

The default set-up assumes that routines coming to streams :1 and :3 are to be compiled normally, and that routines coming to streams :2 and :4 have problems and should be compiled without optimisation.

One could over-rule the default with a line like:

```
fopt     -bounds_violation -info 1 -indexl -cpu mathlib_sr10 -dbs
```

or one may add to the default with a line like:

```
+fopt     -dbs
```

From the set-up Nyshell constructs shell parameter definitions followed by the shell commands to compile, as can be seen from the example shown earlier which is the result of the following call (adding the debug options for Fortran and C on Apollo):

```
nyshell source.log  u  .go <</
  +fopt  -dbs
  +copt  -g
/
```

The default set-up on any machine can be seen by typing on that machine the command `nyshell` asking for help like (on Apollo):

```
                    Apo.1: nyshell ../xyz hs

                     Nyshell 5.02 /7 executing
                     Options  :     All, By, Empty, Help, Quick, HSetup, Uptodate, Verbose
                     Default ext. : .log        .cra  .lis
                     Stream names : LOG   opt   read  print
                       1  LOG       ../xyz.log
                          opt      HS
                      ...  ...  ...

                       Input file:  ../xyz.log
                     Shell script:  xyz.shfca
                         Log file:  xyz.xqtlog

                     Actual set-up used:

Prelude              start:   # Script from nyshell for file  ../xyz.log
Compiler options
        specific     fo:1     -opt 3
                     fo:2     -opt 0
                     fo:3     =:1
                     fo:4     =:2
          common     fopt     -bounds_violation -info 1 -indexl -cpu mathlib_sr10
Compiler name        fc       /com/ftn
Commands             fort:1   ${FC} ${s}$*.ftn ${FO_1}; mv $*.bin $*.o
                     fort:2   ${FC} ${s}$*.ftn ${FO_2}; mv $*.bin $*.o
                     fort:3   =:1
                     fort:4   =:2
Compiler options
        specific     co:1     -O
                     co:2
                     co:3     =:1
                     co:4     =:2
          common     copt     -c
Compiler name        cc       cc
Commands             cc:1     ${CC} ${CO_1} ${s}$*.c
                     cc:2     ${CC} ${CO_2} ${s}$*.c
                     cc:3     =:1
                     cc:4     =:2

                     ao:1
                     ao:2     =:1
                     ao:3     =:1
                     ao:4     =:2
                     aopt
Assembler name       as       as
                     as:1     ${AS} ${AO_1} ${s}$*.s
                     as:2     =:1
                     as:3     =:1
                     as:4     =:2

Apreslude            end:     # End of the shell script
Source
directory            sdir     ../
```

The special operator `=:` means "same as", thus the line `fo:3 =:1` means: use for stream `fort:3` the same specific compiler options as for stream `fort:1`.

The source files are assumed to be in the same directory as the log file, therefore the parameter "sdir" is pre-set to the prefix given for the log file. This will be passed through the shell variable `${s}`.

The names of the shell variables used, like `FC`, `s`, `FO_1`, are fixed by convention and cannot be changed. The special escape `$*` indicates the places where the name of the routine has to be substituted.

For more ample illustration we give here the result of this call for help also on a SUN and, on the next page, for a VAX:

```
sun.3: nyshell ../abc hs
       ...
 Nyshell executing with files / options

   1  LOG       ../abc.log
      opt      HS
   3  read     -
   4  print    tty

 Version: PATCHY  5.02 /7  1995/02/23 17.40

   Input file:  ../abc.log
 Shell script:  abc.shfca
     Log file:  abc.xqtlog

 Actual set-up used:

 start:   # Script from nyshell for file  ../abc.log

 fo:1     -O
 fo:2
 fo:3     =:1
 fo:4     =:2
 fopt     -c -w66
 fc       f77
 fort:1   ${FC} ${FO_1} ${s}$*.f
 fort:2   ${FC} ${FO_2} ${s}$*.f
 fort:3   =:1
 fort:4   =:2

 co:1     -O
 co:2
 co:3     =:1
 co:4     =:2
 copt     -c
 cc       cc
 cc:1     ${CC} ${CO_1} ${s}$*.c
 cc:2     ${CC} ${CO_2} ${s}$*.c
 cc:3     =:1
 cc:4     =:2

 ao:1
 ao:2     =:1
 ao:3     =:1
 ao:4     =:2
 aopt
 as       as
 as:1     ${AS} -o $*.o ${s}$*.s
 as:2     =:1
 as:3     =:1
 as:4     =:2

 end:     # End of the shell script
 sdir     ../
```

And for the VAX:

```
AxCrnC$ nyshell ../rst hs
       ...
Nyshell executing with files / options

  1  LOG       ../RST.log
     opt      HS
  3  read     -
  4  print    tty

Version: PATCHY  5.02 /7  1995/02/23 17.40

  Input file:  ../RST.log
Shell script:  RST.shfca
    Log file:  RST.xqtlog

Actual set-up used:

start:   $! shfca:
         $ proc_ver = f$environment("verify_procedure")
         $ imag_ver = f$environment("verify_image")
         $ set verify=(procedure,noimage)

fo:1     /opt
fo:2
fo:3     =:1
fo:4     =:2
fopt     /nolist/check=noover
fc       fortran
fort:1   $ ${FC} ${FO_1} ${s}$*.for
fort:2   $ ${FC} ${FO_2} ${s}$*.for
fort:3   =:1
fort:4   =:2

co:1
co:2
co:3     =:1
co:4     =:2
copt     /warn=noinfo
cc       cc
cc:1     $ ${CC} ${CO_1} ${s}$*.c
cc:2     $ ${CC} ${CO_2} ${s}$*.c
cc:3     =:1
cc:4     =:2

ao:1
ao:2     =:1
ao:3     =:1
ao:4     =:2
aopt     /nolist
as       macro
as:1     $ ${AS} ${AO_1} ${s}$*.mar
as:2     =:1
as:3     =:1
as:4     =:2

end:     $ temp=f$verify(proc_ver,imag_ver)
sdir     ../
```

Note that Patchy operates internally with Unix-style file names even under `VMS`, they are converted to `VMS` style just before being used.

Chapter 7: Index