Virtual Master 3.4 script writing manual

By Sven B.

This manual assumes you are familiar with Virtual Master. If you are not familiar with Virtual Master, you should read the document **User guide.pdf**. That is a short document explaining the concept of Virtual master, providing an introductory guide through the program.

You can find a chapter about new features in this release at the bottom of this document.

Table of contents

Virtual Master 3.4 script writing manual 1
Create your own rules 9
Script syntax 10
Basic elements 11
General section: must always be present 12
What must be there 12
Optional keywords 12
What happens the first time the sub opens the script? 12
Status 13
What must be there 13
Use status 13
Changing status 14
Substatus 14
NewSubStatus 14
Define the status as a substatus 14
Return to the original status 15
Automatic substatus: 15
Status groups 15
How much has the sub been home? 16
Reports 17
What must be there 17
Making relevant reports appear at the top of the report list 17
Confessions 18
Automatic confessions 18
Permissions 20
What must be there 20
What about the next time the sub asks? 20
Telling the sub when waiting time is over 21
Punishments 22
Punishing the sub 22
Specifying the punishments the sub can receive 22
What must be there 22
Repetitive punishments 22
Punishments measured in time 23
Punishments without repetitions or time 23
Control the severity of the punishment 23
Don't use a heavy punishment for minor offences 24
Use some punishments more often than others 24
Make the "punishment fit the crime" - Punishment groups 24

"Long-running" punishments 25
Other punishment keywords 25
Punishments forbidding the sub to do something 26
Can I start a punishment immediately? 26
What if the punishment is not done on time? 26
Controlling punishments 27
Jobs 28
What must be there 28
Giving further instructions 28
Repeating the job with an interval 28
Jobs on specific days 28
When to do the job 28
What if the job is not done on time? 29
Other job keywords 30
More job control 30
Jobs to run only one time 30
Add a job to the assignment list 30
Assignment keywords (both jobs and punishments) 31
Mark an assignment done, abort it or delete it 31
Preventing an assignment from being started or marked done 31
More assignment keywords 32
Control when assignments may popup with messages 33
Merits 34
Setting the limits 34
Defining the start score 34
Adjusting the merit score 34
Giving and taking merit points 34
Giving a daily “allowance” 35
Using the merit score 35
Hiding the merit score 35
Advice on how to use merits 35
Prevent Cheating 36
If you're writing a script for another person: 36
If you're writing a script for yourself: 36
Autoencrypt the script 36
Not knowing what is going to happen 37
Making extreme values possible 37
Instructions and clothing 38
Definitions 38
Instructions 39
How to use instructions 39
Defining the instructions - What must be there 39
When can the sub ask for instructions? 39
Defining a choice 39

What if nothing is chosen? 40
When does the program show a new selection? 40
Using sets to control the choices 40
More control of what is chosen 41
Sets 42
If sets are not always to be used 42
An old way to define choices 42
Cloth reports 44
Defining the cloth types 44
Ordering the sub to make a cloth report 44
Importing and exporting clothes 44
When the sub adds a cloth type 45
Advanced script writing 46
Changing the name of elements 47
Procedures 48
What must be there 48
What can you do in a procedure? 48
How to use a procedure? 48
Conditional actions 49
Weighting procedures with Select=Random 50
Popups 51
How to define a popup 51
How to activate popups 51
How to enable and disable popups by use of flags 51
Use some popups more often than others 51
How to change the sound when a popup occurs 52
How to control response times for popup 52
How to use popup groups 52
Defining popup groups 53
Using popup groups 54
Timers 55
What must be there 55
Using status with timers 55
Using timers with flags 55
Timers and weekdays 56
Writing messages to the sub 57
Asking questions to the sub 59
Input, the really simple way 59
Input to variables: a bit more complex 59
Advanced questions: how to define them 59
Advanced questions: how to ask them 60
Duration keywords, being too quick or too slow 61
Using MinTime and MinTimeProcedure with jobs and punishments 62
Response time for popups and timers 62

Advanced permission use 64
Permissions based on time 64
Permissions based on flags 64
Permissions based on the merit score 64
More variation in permission limits 64
Variable permissions based on the merit score 65
Use a procedure to decide whether permissions should be granted 66
Take action when a permission is denied 66
Sign in 68
Punishing the sub for late sign in 68
Time keywords, being late or early 69
Specifying Latest and Earliest 69
Controlling menus based on time 70
Prevent simultaneous assignments 71
Events 72
Events about program start, stop etc. 72
Events about the report file 72
Events about reports and confessions 72
Events about permissions 73
Events about punishments 73
Events about jobs 73
Events about assignments 73
Events about clothes 73
Other events 74
Starting assignments - Keeping the sub busy 75
How to do it 75
Estimating jobs and punishments 75
Defining which status to use 75
Defining extra jobs 75
Starting automatic assignments 76
Knowing what happens 76
Examples 76
Advanced cloth issues 78
Check the sub's cloth report 78
The concept 78
How to define the cloth report. 78
How to define the cloth type 79
How to activate / deactivate the requirements 80
How do define your own actions 81
Setting flags in cloth reports 82
Setting flags in instructions and clothing instructions 82
Send email from the program 84
What must be there 84
Optional 84

How to send a mail to the Master 85
How to send a mail to the sub 85
How to automatically send a created report 85
Line writing as jobs or punishments 86
What must be there 86
The same or different lines, or a whole text 86
What should be in the Value parameter 87
Examples: 87
Line writing job examples: 87
Changing the page size 88
Detention as punishment 89
What must be there 89
Adding additional text 89
Examples: 89
Use FTP 90
What must be there 90
Using a subdirectory 90
Automatically update the script and/or the program 91
Uploading reports 91
Other FTP options 91
An FTP example: 92
Using a web cam 93
Monitoring a status 93
Having the sub pose for the camera 93
How to send pictures 94
Where to save the pictures 95
Other options 96
Playing sounds 96
Showing pictures 96
Show a picture when in a status 96
Display a picture in a separate window 97
Starting files and other programs 97
Writing to the report file 97
Make a new report file 97
The rules menu 98
Omitting permissions, reports and instructions from the Rules menu .. 98
Creating your own rules 98
Omitting the Rules menu completely 98
Disabling the Rules menu in a specific status 99
Minimizing or closing the program 99
Setting the program on standby 99
Quick reports 99
The global quick report 100
The assignment quick report 100

The local quick reports 100
Changing font size 101
Show buttons that are outside the window 101
Hiding the program 101
Setting a password on the program 101
Minimize to the system tray 102
How to handle popups when minimized 102
Variables 103
Flags 104
Set and remove flags 104
Use flags 104
Define flag attributes 105
Using flags to make something happen after a period of time 105
Predefined flags 106
String variables 107
Setting a string variable 107
Using a string variable 107
Removing a string variable 107
Examples of string variables 107
Predefined string variables 107
Counters 109
Setting a counter 109
Calculating on counters 109
Generating a random value 109
Removing a counter 110
Using a counter 110
Examples of counters 110
Predefined counters 110
Time variables 112
Time constants 112
Setting a time variable 112
Calculating on time variables 113
Converting a time variable to a counter 113
Converting a counter to a time variable 113
Rounding a time variable 113
Generating a random value 114
Removing a time variable 114
Using a time variable 114
Predefined time variables 114
Using variables in If and NotIf 116
Divide your script in parts 118
Conditional execution (case) 119
Foreign language translation 123
Advice for self scripters 124

Compability with older versions 125
Testing your script 126
The test menu 126
The test menu 126
Avoiding Internet connection when testing 126
Avoiding entering the open password when testing 127
Making reports only for test usage 127
Debug help 127
Trouble shooting 128
I cannot see the buttons on the Cloth Report form 128
What's new 129
What's new in 3.1 129
What's new in 3.2 129
About Foreign language translation: 131
What's new in 3.3 131
What's new in 3.4 133

Create your own rules

The program is shipped with two sample demo scripts, one for male subs and one for female subs. Make a copy of the demo file of your choise and edit it to create your own rules. You can call the new file anything you want, but it must end with .ini. The file can be edited by a simple text editor like Notepad. Usually you can double click on the file to start Notepad. **Warning:** If you use a text processing system like Word to edit the script, make sure that you save it as a text file. If you save it as a Word file, you will not be able to use it in the program.

For practical reasons the sub is most of the time called "he" in this manual. Most of my examples are for a female slave, so sometimes I write “she”. In reality, the sub can be a man or a woman. The program doesn't care.

While you read this manual, I suggest that you look at the demo script at the same time, to get some examples of how to use the keywords described in this manual.

In the end of the manual, you will find a chapter with advices if you are scripting for yourself, and a chapter about testing your script.

Script syntax

The script is divided into sections. Most of the sections define elements the sub will see, like a permission, a report or a punishment.

A few of the sections describe options common to the whole script. These are the [General], [Init] and [Events] sections.

Each section starts with a name in [ ]. Each section has its own keywords.

Each keyword must be on a separate line. You can start the line with as many blanks as you wish before the keyword.

You can make comments, headings, explanations etc. in the script by starting the line with a semi-colon (;). The program will not "see" and try to read any line beginning with a semi-colon. You can also use this to deactivate a line of code without deleting it entirely.

If you want to deactivate a whole section, add the semicolon (;) after the opening bracket ([). If you add the semicolon at the start of the line, the program will not "see" the header and it will believe that the keywords belongs to the previous header.
Example: [;Permission-Go out].

Here is a description of the syntax of script.

Basic elements

This section describes the basic elements: the status, the reports, the permissions, the jobs, the punishments etc.

It also describes the general stuff that must always be present in a script, and what happens the first time the sub starts the script.

General section: must always be present

The General section defines general preferences. **It must be present in the script**. These are the most important keywords of the [general] section. There are other keywords, that will be described later.

What must be there

[General]	This line starts the general section.
Master=*Master's name*	Give your virtual Master a name. The name shows on the title bar.
SubName=*sub's name*	Name the sub. If present, the *sub's name* is used in some of the messages. You can have more than one SubName line. If you have, the program will choose a random name.
MinVersion=3.3	Tells which version of the program the script is made for. If you try to run the script on an older version of the program, you will get an error message. Write the version number as you can read it in the About box under the help menu. MinVersion must be present in the script.

Optional keywords

Version=*xxx*	Use this to write in the report which script version is used. Replace *xxx* by any identification of the script you want. It can be a date, but it can also be anything else. If you use an encrypted script, the program will use the encryption date and time as the script version and ignore this keyword.

What happens the first time the sub opens the script?

The Init section defines what happens first time a new script is used. The Events section has the keyword FirstRun, where you can specify a procedure to run the first time a script is used. The FirstRun procedure is called after the Init section is executed and is intended to supplement the Init section. Read more about procedures later.

[init]	This line starts the init section.
NewStatus=*statusname*	This defines the first status to enter
Merits=*startmerits*	The sub will start with *startmerits* merit points. If you do not specify this, the sub will start with half of MaxMerits.
[events]	This line starts the events section.
FirstRun=*procedure*	Activates *procedure* the very first time a new script is used. See more about procedures later.

Read more about the Events section in the chapter Events.

Status

Status is an important concept of the program. The status should reflect what the sub is currently doing, so VM can react accordingly.

What must be there

[status-*name*]	This line starts a status definition. Replace *name* with the name of the status.

Use status

You can use status to decide when something is possible. You do this by adding the keyword PreStatus to a permission, report, confession, procedure, timer, popup or set.

Prestatus=*status*	Means that sub must be in status *status* for the permission, report etc. to be possible. You can have more than one Prestatus line. Or, you can specify several possible statuses on the same line by separating them with commas. You can use the name of a status group instead of the name of a single status.

When prestatus is used in a report, the report will only be visible in the communication menu when the sub is in the specified status.
When prestatus is used in a confession, the confession will only be visible in the communication menu when the sub is in the specified status.
When prestatus is used in a permission, the permission will only be visible in the communication menu when the sub is in the specified status.
When prestatus is used in a procedure, the procedure will only be executed when the sub is in the specified status.
When prestatus is used in a timer, the timer will only be executed when the sub is in the specified status.
When prestatus is used in a popup, the popup will only pop when the sub is in the specified status.
When prestatus is used in a set, the set will only be used when the sub is in the specified status.

ReportsOnly=1	If you add this keyword to a status, all the sub can do is make a report. It will not be possible to ask permission, confess, ask for instructions, report clothing or look at assignments. This is useful for statuses where the sub is only supposed to do one thing, like going to the toilet or standing in the corner.

Assignments=0	If you add this keyword to a status, the sub will not be able to access the Assignments menu.
Permissions=0	If you add this keyword to a status, the sub will not be able to ask permissions.
Confessions=0	If you add this keyword to a status, the sub will not be able to make confessions.
Reports=0	If you add this keyword to a status, the sub will not be able to make reports.
Rules=0	If you add this keyword to a status, the sub will not be able to look at the rules.

Changing status

You can change status in permissions, reports, confessions, procedures, timers or popups.

NewStatus=*newstatus*	Status is set to *newstatus*.

If you use NewStatus in jobs and punishments, the program will change to *newstatus* when the job or punishment starts and will automatically change back when it is marked done. You cannot use NewSubStatus in jobs and punishments.

Substatus

Sometimes you want to switch to another status for a short period and then return to the original status. One example is when the sub is allowed to go to the toilet. You then want the status to reflect that the sub is using the toilet, and afterwards you want to return to the original status. You can do this in two ways. Either use NewSubStatus instead of NewStatus when switching status, or define the status you temporarily switch to as a substatus. A substatus is a status from which you can return to the previous status. A primary status is a status that is not a substatus.

If you go from a substatus to a primary status using NewStatus=*statusname*, you lose the substatus and will no longer be able to use NewStatus=&LastStatus.

NewSubStatus

You can use NewSubStatus anywhere you can use NewStatus, except in the [init] section or in jobs or punishments, as mentioned above..

NewSubStatus=*newstatus*	Status is set to *newstatus*.

Define the status as a substatus

In the status definition add:

SubStatus=1	Means that this status is a substatus, and that you can use &laststatus to return.

Return to the original status

When you want to return to the original status, add this to a permission, report, confession, procedure, timer or popup:

NewStatus=&LastStatus	Returns to the previous status if you are in a substatus.

If you use NewStatus=&LastStatus when there is no saved LastStatus, then the program cannot return to a previous status. There are two ways of handling this. If you have set a default status (see below), the program will go to that status. If not, the program will stay in the current status. In both cases an error line is placed in the report. The sub will not receive an error.

DefaultStatus=*statusname*	Defines the status that the program should return to when no last status exists. This keyword must be put in the [General] section.

Automatic substatus:

Another and more simple way to define a substatus is to use the EndReport keyword. EndReport is useful in simple situations like going to the toilet.

EndReport=*reporttext*	If you use EndReport in combination with NewStatus, an automatic report is created with the name *reporttext* and the new status is automatically used as a substatus. When you use EndReport, you don't even have to define the status, unless you want to use other status-related keywords. Furthermore, when you use EndReport, the sub will have no other choices than to use this report to end the status.

The generated report will automatically be used as a quick report. This means that the sub can report it by clicking a button in the main window or by pressing the F9 key.

EndText is an obsolete name for EndReport and works the same way.

Status groups

You will probably have several reports, permissions, etc. that are possible when in a number of statuses. As an example, you may ask permission to use the toilet most of the time when home. Instead of using a PreStatus line for every possible status, you can place all the relevant statuses in a status group called "Home". You can use the status group name in the PreStatus keyword.

In a status definition use

Group=*name*	Means that this status belongs to the status group *name*.

Example:

```
[status-Normal]
  Group=home
[status-Eating]
  Group=home
[status-Watching TV]
  Group=home
[permission-Use the toilet]
  PreStatus=Home
```

How much has the sub been home?

Optionally you can choose to mark all status where the sub is not home with the keyword Away=1. When a day ends, the program lists how much time the sub have spent at home and how much time away from home. The program uses the Away keyword to determine when the sub is home.

In a status definition use

Away=1	Means that when in this status, the sub is not home. Away=0 or not coding the Away keyword means that the sub is home.

Reports

Reports are the way the sub tells the program that something has happened, that he is going to do something that he doesn't need to ask permission for, or that he has finished doing something. Reports are often, but not always, used to change to a new status.

What must be there

[report-*name*]	This line starts a report definition. Replace *name* with the name of the report.

Making relevant reports appear at the top of the report list

Usually reports are shown in the order they are defined. But if the sub is doing something that he must report finished, you usually want that report to be on the top of the report list.

OnTop=1	If you add OnTop=1 to a report definition, then the report will be on top of the list when the sub wants to make a report.

When the sub is doing something else, you don't want the ”finished” report to show up at all. By adding a Prestatus keyword (as described above) to the report definition, you ensure that the report will only be available when that status is active.

Confessions

This element specifies the confessions the sub has to make. Confessions are a lot like reports. The only difference is that they have a separate entry in the Communications menu.

[confession-*name*]	This line starts a confession definition. Replace *name* with the name of the confession.

You can use the same keywords for confessions as for reports.

Automatic confessions

The program generates two special confessions: "Forgot to ask permission" and "Did something though permission was denied". When confessing one of these, the sub is asked what he has to say for himself. Then he may be punished. You can specify penalties for these confessions with the keywords ForgetPenalty and IgnorePenalty. These keywords are used in a permission definition to specify a special penalty for forgetting/ignoring this permission or in the [General] section to specify a general penalty for all permissions, except those which have an individual specification.

ForgetPenalty=*points*	If the sub confesses that he has forgotten to ask permission, he will be punished with a severity of *points*. If you specify two values separated by comma, the program will use a random value in that range. You can use a counter instead of a value.
IgnorePenalty=*points*	If the sub confesses that he has done something, although he was denied permission, he will be punished with a severity of *points*. If you specify two values separated by comma, the program will use a random value in that range. You can use a counter instead of a value.

If for some reason you don't want automatic confessions, you can disable them by using these keywords in the [General] section.

ForgetConfession=0	There will be no automatic confessions for forgetting to ask permission.
IgnoreConfession=0	There will be no automatic confessions for doing something that was denied.

If you want to control what happens when an automatic confession is made, you can use the events ForgetConfession and IgnoreConfession. See Events.

If you want to control what happens when an automatic confession is made for a specific permission, you can use these keywords in the permission

ForgetProcedure= *procedure* If the sub confess that he has forgotten to ask permission, *procedure* is called.
IgnoreProcedure= *procedure* If the sub confess that he has done something, although he was denied permission, *procedure* is called.
If you use both ForgetProcedure / IgnoreProcedure and events, the ForgetProcedure / IgnoreProcedure is called first and then the event.

Permissions

Permissions define what the sub has to ask permission for and sets rules for when permission is given.

What must be there

[permission-*name*]	This line starts a permission definition. Replace *name* with the name of the permission.
Pct=*nn*	Gives the sub *nn* % of chance for an immediate permission.

What about the next time the sub asks?

If the program refuses permission, you don't want the sub to be able to just ask again, and then maybe get permission. If permission is granted, you may not want the program to grant permission once again shortly after. If the sub is refused permission to use the toilet, you want to make sure he isn't going to wait forever. This chapter describes how to control these intervals.

MinInterval=*hh:mm,hh:mm*	Mininterval defines the minimum interval that must pass after the permission is granted before it can be granted again. Thus mininterval=24:00 insures that this permission is never granted twice within 24 hours. If you specify two values separated by comma, the program will use a random value in that range.
Delay=*hh:mm,hh:mm*	If permission is not granted, delay defines the minimum amount of time to pass before permission can be granted. If you specify two values separated by comma, the program will use a random value in that range. You can use a time variable instead of *hh:mm*. Note: If the sub asks permission again before the calculated time arrives, the program calculates a new time, and if it is later than the old time the new time will be used.
MaxWait=*hh:mm,hh:mm*	Specifies the maximum time the sub can wait. If the sub keeps asking, the delay will continue to extend the time the sub has to wait, but never past the time in the MaxWait parameter (calculated from the first time the sub is denied the permission). This is useful for things that can't wait forever, such as going to the toilet. If you specify two values separated by comma, the program will use a random value in that range. You can use a time variable instead of *hh:mm*.

Telling the sub when waiting time is over

Usually when a permission is denied, you can simply leave it to the sub to guess when permission can be asked again. But in some cases, you may want to tell the sub when waiting time (defined by the Delay parameter) is over.

Notify=2	The program will inform the sub when the delay time has passed. The sub is told that he will be informed, but not how long he has to wait.
Notify=1	The program will inform the sub when the delay time has passed. The sub is told that he will be informed, and how long he has to wait.
Notify=0	The program will not inform the sub when the delay time has passed. The sub has to guess when to ask again.

Not specifying Notify is the same as Notify=0.

Punishments

Punishing the sub

To punish the sub, you can add the following keywords to a report, confession, timer, popup or procedure. You can also use them in a permission, in which case the sub is punished when the permission is given.

Punish=*points*	The sub will be punished with a severity of *points*. *Points* can be a value or the name of a counter. One severity point is equal to one merit point. If you give two values separated by a comma, then the program will chose a random value between the two.
PunishMessage=*text*	Use PunishMessage to specify the text that is presented to the sub. This keyword is ignored if Punish is not used. If you do not code PunishMessage, the system will generate one.
PunishmentGroup=*group,group,*	If a punishment is given, it will be chosen from one of the groups specified. If you do not code PunishmentGroup, the system will pick a random punishment. Note that in combination with the RemindPenalty keyword, you should use RemindPenaltyGroup instead of PunishmentGroup.

Specifying the punishments the sub can receive

The [punishment-] element defines the possible punishments the program can give.

What must be there

[punishment-*name*]	This line starts a punishment definition. Replace *name* with the description of the punishment.

Repetitive punishments

Often you want a punishment to consist of a number of repetitions, like "6 beats with the cane". Use # in the name where you want the program to put in the number of repetitions. Example: "[punishment-# beats with the cane]".

Punishments measured in time

Sometimes you want to measure a punishment in time. You can do this in minutes, hours or days. Examples: "Stand in the corner for 15 minutes", "Wear a gag for for 3 hours", "No TV for 3 days". The program will ensure that the sub doesn't finish too fast. Use # in the name where you want the program to put in the time (Example: "[punishment-Stand in the corner # minutes]"), and use the ValueUnit keyword as described below.

ValueUnit=*unit*	*unit* must be 'minute', 'hour' or 'day'.

Punishments without repetitions or time

Some punishments are not time-based and are not repeated. The sub is given an order, and that is that.

ValueUnit=Once	Tells that there are no repetitions.

Control the severity of the punishment

value=*number*	*number* tells the severity of one repetition/minute/hour/day of the punishment measured in merit points. The severity of the punishment (given with the Punish keyword) is divided by *number* to calculate the actual punishment. If you do not code value, 1 is used, unless ValueUnit=Once. You may use decimals, e.g. value=0.2.
max=*number*	*number* gives the maximum number of repetitions/minutes/hours/days allowed for this punishment. If the sub needs more punishment than that, more punishment sessions are scheduled. If you do not code max, 20 is used. Max is ignored if ValueUnit=Once
min=*number*	*number* gives the minimum number of repetitions/minutes/hours/days given in this punishment. If you do not code min, 1 is used.

Example:

```
[punishment-Stand # minutes in the corner]
   value=2
   max=20
   min=5
[confession-Late home]
   punish=20
[confession-Lied to my Master]
   punish=50
[confession-Overslept]
   punish=8
```

If the sub confesses late home, he will be punished with a severity of 20. Corner time has a value of 2, which gives 20 divided by 2 = 10 minutes in the corner.
If the sub confesses to have lied, he will be punished with a severity of 50. Corner time has a value of 2, which gives 50 divided by 2 = 25 minutes in the corner. This is above maximum, and the sub will be punished with 20 minutes in the corner (maximum) and given a supplemental punishment.
If the sub confesses to have overslept, he will be punished with a severity of 8. Corner time has a value of 2, which gives 8 divided by 2 = 4 minutes in the corner. This is below minimum, so the sub is punished with 5 minutes in the corner (minimum).

Don't use a heavy punishment for minor offences

You may have some punishments that are always severe, and which you don't want to use with minor offences. Or you may have mild punishments you don't want to use for severe offences.

MinSeverity=*points*	If the severity asked for is less than *points*, this punishment will not be used and another punishment will be chosen.
MaxSeverity=*points*	If the severity asked for is higher than *points*, this punishment will not be used and another punishment will be chosen.

Use some punishments more often than others

When the sub must be punished, the program will search through all possible punishments and select one (or sometimes more). It is possible to influence this selection and have some punishments turn up more often than others. You do this by weighting the punishments. The higher the weight, the more likely the punishment is to be chosen. A punishment with a weight of 3 is chosen 3 times more often than one with a weight of 1.

Weight=*weight*	Tells the weight of the punishment. *Weight* can be a positive number or a counter. Or it can be two values separated by comma, and the program will chose a random number between the two values. If you don't code weight, the punishment will have a weight of 1.

Make the "punishment fit the crime" - Punishment groups

A punishment can belong to one or more groups, allowing the Master to control which punishments can be given for each offence, to "make the punishment fit the crime". If the program gives a punishment on the basis of a section where punishmentgroup(s) are

defined, it will choose a punishment from one of the listed groups. If the sub declines the punishment, he/she will be given a new punishment from the groups.

Group=*group,group,..*	List one or more punishment groups that the punishment belongs to. You can add a number of groups separated by commas, or you can add more than one Group keyword.
GroupOnly=1	If you use GroupOnly=1, then this punishment will only be used if it is selected from a punishment group. In other words, it will only be used if there is a PunishmentGroup keyword where the punishment is generated.
PunishmentGroup=*group,group,*	If this punishment results in a new punishment, e.g. for being late, the new punishment will be chosen from one of the groups specified.

Use the keyword PunishmentGroup in the same section where you have the keyword Punish to tell the program which group(s) to use. See "Punishing the sub" above.

"Long-running" punishments

A "long-running" punishment is a punishment that can be done while the sub goes on with other tasks, even other punishments. It need not actually run for a long time, though it is usually connected with time. Examples: "No TV for 3 days", "Wear a butt plug for 3 hours", "Wear a gag for 17 minutes".

ValueUnit=day	If you use ValueUnit=day the punishment will always be considered long-running.
LongRunning=1	Tells that this is a "long-running" assignment, even if ValueUnit is not day.

Other punishment keywords

MustStart=1	Means that the sub must report when starting the punishment, that it is not possible to report a punishment done, if it is not started. This keyword is ignored if ValueUnit=day, hour or minute or if LongRunning=1 is specified, as the sub must always start those.
Accumulative=1	Means that if a punishment of the same type is already in the assignments list when it is given, a new punishment will not be created, but the existing punishment will be increased. However, if the increased punishment will

	exceed the Max keyword, a new assignment will be created instead.
Respite=*hh:mm*	Specifies how much time (in hours and minutes) before the sub must have finished the punishment. For some punishments the program will make its own calculations and add it to your specification. You can use a time variable instead of *hh:mm*. Can be abbreviated Respit.
Estimate=*hh:mm*	The punishment is estimated to take the sub *hh:mm* hours and minutes to complete. Maximum is 23:59. Estimate will be ignored if the punishment is measured in time. Estimate can be useful in combination with auto assignments.

See also Assignment keywords (jobs and punishments)

Punishments forbidding the sub to do something

You can have a punishment forbid something the sub has to ask permission for. If the sub asks permission while the punishment is started, it will always be denied.

Forbid=*permission*	While this punishment is active, the sub will not be allowed *permission*. *Permission* must be defined as a normal permission. Example: Forbid=Watch TV

Can I start a punishment immediately?

I am often asked if it is possible to make a punishment start immediately. It is not. The reason is this: Punishments can be given at any time. And in real life it is not always possible to do a punishment right away. The sub can receive a punishment just before going to work, or just before someone comes to visit. Or the sub can have some other activity that makes it impossible to do the punishment at the instant it is given.

What if the punishment is not done on time?

Usually, if the sub doesn't finish on time, you simply want to remind him and maybe punish him for being late.

RemindInterval=*hh:mm*	If the sub is late doing the punishment, he is reminded and punished every *hh:mm* hours and minutes. You can specify RemindInterval on jobs and punishments and in the [General] section. The interval specified in the [General] section will be used for any job or punishment that does not have RemindInterval specified. If you do not specify the interval, it is set to 24 hours. You can code two values

separated by comma. The program will then choose a random value between the two values.

RemindPenalty=*severity* Every time the sub is reminded about being late, he is punished with a severity of *severity*. You can specify a single number, or two numbers separated by comma. If you use two numbers, the program will pick a random number between them. You can specify RemindPenalty on the job or in the [general] section.

RemindPenaltyGroup=*punishmentgroup* If you use RemindPenalty, you can use RemindPenaltyGroup to tell which group the punishment should be taken from. See Punishing the sub.

LateMerits=*points* If the job is not done on time, the sub's merit points are reduced by *points*. When the job is done, the points are returned. You can use this if you do not wish to punish the sub for being late, but still want to show that you are not satisfied. If you specify two values separated by comma, the program will use a random value in that range. You can use a counter instead of a value.

Controlling punishments

In the [general] section you can add some keywords to generally control the behaviour of punishments.

MaxDecline=*number* The sub is only allowed to decline a punishment *number* times. If you do not specify this keyword, it is set to 3. Every time the sub declines, the punishment severity is increased by 20%.

MinPunishment=*number* When the sub is punished, it will be with a minimum severity of *number* points, even for minor offences. If you do not specify this, a value of 1 is used.

MaxPunishment=*number* A punishment can be of maximum severity *number*. If you do not specify *number* or if you specify MaxPunishment=0, there is no limit.

AskPunishment=*min,max* When the sub asks for a punishment, *min* and *max* defines the minimum and maximum severity given, measured in merit points. If you don't code AskPunishment, the sub cannot ask for punishments.

AskPunishmentGroup=*group,group,…* When the sub asks for a punishment, it will be chosen from one of the groups specified.

Jobs

Jobs are something you want the sub to do regularly. It can be simple housework, training activities or anything you want.

What must be there

[job-*name*]	This line starts a job definition. Replace *name* with the name of the job.

Giving further instructions

Text=*instructions*	The instructions for the job.

Repeating the job with an interval

Interval=*min,max*	The job must be done regularly with intervals of minimum *min* and maximum *max* days. Interval=0 or not specifying Interval means that the job will not be scheduled automatically. *Min* and *max* can be an interval or a counter.

When you start a new script, or when you introduce a new job into an existing script, you may not want to wait for the first occurrence of the job to be scheduled according to the interval specifie. Then you can use the FirstInterval keyword.

FirstInterval=*min,max*	Tells how soon the first occurrence of the job must be finished.

Jobs on specific days

You can use the Run keyword to have the job run daily or on specific weekdays. If you have more than one run keyword, the job must be done on all appropriate days.

Run=daily	The job must be done every day.
Run=*weekday*	The job must be done on every *weekday*, where weekday is Monday, Tuesday, Wednesday, Thursday, Friday, Saturday or Sunday.
Norun=*weekday*	The job will not be done on *weekday*. Use this to make exceptions from a Run=Daily.

There's one disadvantage of using the run keyword. The sub cannot be notified earlier than the same day as the deadline. Regardless of what you write in Respite, the sub will not be notified earlier than the last midnight before the job must be done.

When to do the job

EndTime=*hh:mm*	Specifies the time of day where the sub must have completed the job. Use the 24 hour clock, e.g.

EndTime=14:00 means 2 PM.
If you specify two values separated by comma, the program will use a random value in that range. You can use a time variable instead of *hh:mm*.

What if the job is not done on time?

Usually, if the sub doesn't finish on time, you simply want to remind him and maybe punish him for being late.

RemindInterval=*hh:mm* If the sub is late doing the job, he is reminded and punished every *hh:mm* hours and minutes. You can specify RemindInterval on jobs and punishments and in the [General] section. The interval specified in the [General] section will be used for any job or punishment that does not have RemindInterval specified. If you do not specify the interval, it is set to 24 hours. You can code two values separated by comma. The program will then choose a random value between the two values.

RemindPenalty=*severity* Every time the sub is reminded about being late, he is punished with a severity of *severity*. You can specify a single number, or two numbers separated by comma. If you use two numbers, the program will pick a random number between them. You can specify RemindPenalty on the job or in the [general] section.

RemindPenaltyGroup=*punishmentgroup* If you use RemindPenalty, you can use RemindPenaltyGroup to tell which group the punishment should be taken from. See Punishing the sub.

LateMerits=*points* If the job is not done on time, the sub's merit points are reduced by *points*. When the job is done, the points are returned. You can use this if you do not wish to punish the sub for being late, but still want to show that you are not satisfied. If you specify two values separated by comma, the program will use a random value in that range. You can use a counter instead of a value.

Sometimes, if a job is not done on time, you don't want it done at all.

ExpireAfter=*hh:mm* If the job has not been done after *hh:mm* hours and minutes after it should be finished, it expires and is removed from the assignment list. You can code two values, ExpireAfter=*hh:mm,hh:mm*, and the program will pick a random time between the two values.
You can use a time variable instead of *hh:mm*.

ExpirePenalty=*n* If the job expires, the sub is punished with a severity of *n*. You can code two values, ExpirePenalty=*n1,n2*, and the program will pick a random number between *n1* and *n2*.

ExpireProcedure=*procedurename* When the job expires, the procedure *procedurename* is called.

Other job keywords

Respite=*hh:mm* Specifies the time (in hours and minutes) in advance of the deadline that the sub will be informed about the job he/she has to do. That is, the time the sub has to find a suitable time to do the job.
You can use a time variable instead of *hh:mm*.
If you do not code Respite, it will be 48 hours.
Can be abbreviated Respit.

Estimate=*hh:mm* The job is estimated to take *hh:mm* hours and minutes to complete. Estimate can be useful in combination with auto assignments.
You can use a time variable instead of *hh:mm*.

See also Assignment keywords (jobs and punishments)

More job control

Jobs to run only one time

Sometimes you may want a job to run only one time. You can do that by adding the OneTime keyword to the job definition.

OneTime=1 Means that the sub will only be asked to run this job one time.

Add a job to the assignment list

You may want to add a job to the assignment list from the script, instead of letting the program control it.

You do it with the keyword Job, which can be used in permissions, reports, confessions, procedures, timers or popups.

Job=*jobname* Announces the job *jobname* to the sub and adds it to the assignment list. *Jobname* must not include the “job-” prefix.

Assignment keywords (both jobs and punishments)

Mark an assignment done, abort it or delete it

Sometimes you want a report or another action to mark an assignment as done, without requiring the sub to do it himself. For example, you may have a job to go jogging. When the sub reports back from the running, you want to automatically mark the job as done.

In other cases you may want to abort an assignment. For example, the sub is standing in the corner as a punishment when someone comes to the door. The sub reports that guests have arrived, and you want to make sure the punishment is aborted so that the sub must start over another time.

You may also want to delete an assignment from the assignments list.

You do this with the keywords MarkDone, Abort and Delete, which can be used in permissions, reports, confessions, procedures, timers or popups.

MarkDone=*name*	Marks the assignment *name* done. *Name* must include the “job-” or "punishment-" prefix. The job will be removed from the assignment list.
Abort=*name*	Aborts the assignment *name*, if it is active. *Name* must include the “job-” or "punishment-" prefix. The assignment will be returned to the assignment list as not started.
Delete=*name*	Deletes the assignment *name* from the assignment list. *Name* must include the “job-” or "punishment-" prefix.

NOTE: MarkDone and Delete cannot be used in the event JobAnnounced or in an AnnounceProcedure, RemindProcedure, BeforeProcedure, StartProcedure, AbortProcedure, BeforeDoneProcedure, DoneProcedure or DeleteProcedure.

Preventing an assignment from being started or marked done

You can control when an assignment can be started or marked done.

Use the keyword BeforeProcedure (see below) to call a procedure before an assignment is started. If the procedure sets the flag zzDeny, the assignment will not be started.

Use the keyword BeforeDoneProcedure (see below) to call a procedure before an assignment is marked done. If the procedure sets the flag zzDeny, the assignment will not be marked as done. See Flags for information about flags.

In both cases you must do something to tell the sub that the assignment have not been started or completed. The program will not do so.

More assignment keywords

Assignment keywords are keywords common to both jobs and punishments.

AddMerit=*points* When the job is done, *points* points are added to the sub's merit points. *Points* can be a value or the name of a counter.

LongRunning=1 Tells that this is a "long-running" assignment. A long-running assignment can be going on during other activities.

MustStart=1 Means that the sub must report when starting the assignment. It is not possible to report the assignment done if it is not started.

StartFlag=*flagname* Sets the flag *flagname* when the assignment is started and removes it again when assignment is done or aborted.

NewStatus=*newstatus* Status is set to *newstatus* when the assignment is started and set back when the assignment is done.

AnnounceProcedure=*procedurename* When the job is announced or the punishment given, the procedure *procedurename* is called.

RemindProcedure=*procedurename* When the sub is reminded about being late, the procedure *procedurename* is called.

BeforeProcedure=*procedurename* When the assignment is starting, the procedure *procedurename* is called before the assignment is started. If the procedure sets the flag zzDeny, the assignment will not be started.

StartProcedure=*procedurename* When the assignment is started, the procedure *procedurename* is called after the assignment is started.

AbortProcedure=*procedurename* When a started assignment is being aborted, the procedure *procedurename* is called.

BeforeDoneProcedure=*procedurename* Before the assignment is marked done, the procedure *procedurename* is called. If the procedure sets the flag zzDeny, the assignment will not be marked done.

DoneProcedure=*procedurename* After the assignment is marked done, the procedure *procedurename* is called.

DeleteAllowed=1 Allows the sub to delete the assignment from the assignment list. If you want to be able to deny the deletion, use a BeforeDeleteProcedure.

DeleteProcedure=*procedurename* When the sub deletes the assignment using the Delete button, or when the Delete keyword is used in the script, the procedure *procedurename* is called.

BeforeDeleteProcedure=*procedure* Activates *procedure* when the sub tries to delete the assignment. If the *procedure* sets the flag zzDeny, the assignment will not be deleted.

Control when assignments may popup with messages

When it's time for a job to announce itself to the sub, it pops up with a message box and sounds the alarm. When the deadline for an assignment (job or punishment) has been passed, the sub is reminded and possibly punished. Again a dialogue box pops up with an alarm. These dialogue boxes may not be practical when the sub is sleeping, is away or has guests.

To prevent these messages from popping up at undesired times, use the InterruptStatus keyword in the [general] section.

InterruptStatus=*statuslist*	Assignment popups will only occur in the status(es) listed in the *statuslist*. *Statuslist* can be one status name or a list separated by commas.

Merits

The merits show how well the sub is doing. A high merit score means that the sub is doing well, and should be allowed privileges. A low merit score means that the sub is not behaving well, and should have fewer privileges.

Setting the limits

In the [General] section you define the limits of the merits and how they are displayed.

MinMerits=*points*	The lowest merit score displayed will be *points*. A score below this will not be shown on the slider bar, but the bar will turn black. The score will still be shown as a number.
MaxMerits=*points*	The highest merit score possible will be *points*. If the merit score gets above this level, it is truncated to *points*. See note below. If you do not code MaxMerits, the value is 1000.
Yellow=*points*	If the merit score gets below *points*, the slider bar will become yellow. When the merits are above *points*, the slider bar will be green.
Red=*points*	If the merit score gets below *points*, the slider bar will become red.

Note: If the sub has unfinished punishments, the maximum possible number of merit points will be calculated as MaxMerits reduced by the value of the unfinished punishments. It is not possible for the sub to reach maximum with unfinished punishments.

Defining the start score

In the [Init] section you can specify what the merit score should be when the sub opens the script for the first time.

Merits=*points*	The sub will start with *points* merit points.

Adjusting the merit score

The merit points are automatically reduced when the sub is punished and restored when the punishment is done. You can influence the merit score in other ways. You can add merit points when the sub has finished his jobs. You can deduct merit points when the sub is confessing bad behaviour or asking for privileges. You can give the sub an "allowance" of a certain number of points per day.

Giving and taking merit points

You can use the AddMerit keyword in permissions, reports, confessions, procedures, timers or popups. You can also use AddMerit in jobs, meaning that when the job is done, merit points are added.

AddMerit=*points*	Add *points* points to the subs merit points. *Points* can be a value or the name of a counter. *Points* can be negative.
AddMerits=*points*	Same as AddMerit.
SetMerit=*points*	Set the merit score to *points*, no matter what it was before. Use this with care. This is only intended for very special scripts.
SetMerits=*points*	Same as SetMerit.

Giving a daily "allowance"

You can give the sub an "allowance" of a certain number of points per day. Use DayMerits in the [General] section.

DayMerits=*points*	Add *points* points to the sub's merit points each day. *Points* can be negative. Points are only given on days the program has been opened.

Using the merit score

You can use the merit score to decide whether the sub gets a permission. See Permissions based on the merit score.

Hiding the merit score

In the [General] section you can add the keyword HideMerits to hide the merit score from the sub, if you wish to do so.

HideMerits=1	Hide the merit score and slider bar.

Advice on how to use merits

Balancing the merits is one of the most difficult tasks in writing a script.. It is very easy to allow the score to get too high too quickly, or conversely, to make it too difficult for the sub to achieve and maintain an acceptable score. It's my experience that it is best to use the maximum score as the standard; whenever the score is below the maximum, the sub must work to get it up. If the sub gains too many merits when already at maximum, they are simply discarded.

Prevent Cheating

If you modify the demo script to make your own script, it will be fairly easy to cheat by reading the report and by modifying the status file. There are two ways to prevent this, depending on your situation.

If you're writing a script for another person:

1. Add the line
   ReportPassword=xxxxx
   to the [general] section. Replace xxxxx with a password of your choice.
2. Encrypt the script (using the File menu) and give the encrypted script (with the extension of .vmi) to the sub. Do not give the sub access to the original script.

These two actions will encrypt the report and the status file and password protect the report. Note that you will need the program to read the report yourself.

If you're writing a script for yourself:

1. Add the line
   Restrict=1
   to the [general] section.

This will encrypt the status file and it will omit information from the report that the sub should not know, thus making it safe to read the report.

Note: Once you have started the program with Restrict=1 in the script, the program will remain in restricted mode until you delete the status file. Removing Restrict=1 will not change back to non-restricted mode.

To encrypt the status file, Restrict=1 must be in the script first time you use that script. If you later adds Restrict=1, the status file will remain unencrypted. However the program will start to omit sensitive information from the report. And will continue to do so even if you remove Restrict=1.

Autoencrypt the script

If you're writing a script for another person, you may want to automatically encrypt the script every time you start the program. This way you don't need to remember to do it manually, and you are always sure your .vmi file is up to date.

1. Add the line
   AutoEncrypt=1
   to the [general] section.

Not knowing what is going to happen

One of problems about writing a script for yourself is that you know what is in the script. That cannot be changed. But VM contains some methods to help make it unpredictable.

Making extreme values possible

For many of the keywords where you specify a number, you can actually specify two numbers separated by a comma and let the program chose a random value. You can see in the description of each keyword whether it is possible.

The problem with random selection is that the program is just as likely to choose the highest value as the lowest as one in the middle. This prevents you from choosing extreme values as minimum and maximum, as you don't want extreme values chosen all the time.

VM has a solution to this. The keyword CenterRandom will make the program chose mostly from values in the middle of the range and rarely chose the highest and lowest values. Thus you can specify more extreme values, knowing that they are possible, but rare.

CenterRandom=1	Will make the program choose mainly from the middle of the range.
CenterRandom=0	Will make all values in the range appear equally often. If you don't code CenterRandom, it will be the same as CenterRandom=0.

You can use CenterRandom in the [General] section or in the section where you specify the keyword you want it to control. If you specify CenterRandom in the [General] section, it will control the whole program, except the sections that have its own CenterRandom keyword. If you specify CenterRandom in other sections, it will only control the section in which it appears.

Please note that CenterRandom does not affect the generation of random variables with Random# or Random!.

Instructions and clothing

This section describes how to

- give the sub clothing instructions
- give the sub other instructions
- define how you want the sub to report what he is wearing.

This section describes the basics of instructions and clothing. In the section about advanced script writing, you will find more information.

Definitions

First some definitions of the words I will use in this section.

Option An option is a single line in the instructions you want to give the sub.

Choice A choice is a set of options, from which VM will choose one.

Set A set is a collection of choices and other sets that naturally belong together. In a set VM will choose one option from each choice in the set and one option from each choice in the other sets contained in this set.

Instruction An instruction is a collection of all the choices possible in a certain situation. It is what you want the sub to ask instructions for. It can be how to sleep, how to eat or anything else you want the sub to ask instructions for.

Clothing instruction A clothing instruction is exactly the same as any other instruction; it is just about what the sub must wear in a certain situation.

Check Checks can be used to compare clothing instructions with a cloth report to check if the sub is dressed as described.

Checks are described in the section Advanced cloth issues.

Instructions

Instructions can be used for giving instructions for specific situations. The program works with two kinds of instructions, Clothing Instructions and Other Instructions. They work identically, but appear in different menus.

How to use instructions

You can use instructions in two ways. The first is that the sub asks for instructions for a specific situation using the Communication menu. Another is by use of the Clothing and Instructions keywords. The Clothing and Instructions keywords can be used in permissions, reports, confessions, procedures, timers or popups.

Clothing=*name*	This line displays clothing instructions for *name*.
Instructions=*name*	This line displays other instructions for *name*.

Defining the instructions - What must be there

[clothing-*name*]	This line starts a clothing situation definition. Replace *name* with the name of the situation the sub has to ask clothing instructions for.
[instructions-*name*]	This line starts an instruction definition. Replace *name* with the name of the situation the sub has to ask instructions for.

When can the sub ask for instructions?

You can control whether the sub can ask for instructions or instructions are only shown by the script.

Askable=1	The sub can ask for instructions. This is what happens if you don't code Askable.
Askable=0	The sub cannot ask for instructions. Instructions are only shown when the script uses the Instructions= or Clothing= keyword.

Defining a choice

Choice=new	Tells VM that a new choice is starting here
Option=*optionname*	Defines the options to choose between. Add one line for each option. An Option=* means that no instructions will be given from this choice, if it is selected. An option starting with % will not be shown to the sub. This can be useful in combination with sets and the IfChosen / IfNotChosen keywords.

Weight=*n*	Optional. Use this if you do not want an equal chance for each option to be chosen. If you specify weight=2, the last option will be twice as likely to be chosen; weight=3 – three times as likely, etc.
OptionSet=*setname*	Optional. If the last option before this line is chosen, then the set *setname* will be used. Se below about sets.

What if nothing is chosen?

If for some reason nothing is chosen, you can define a text to be shown. This can happen if you choose a * or an item with % in all choices, or if you use sets (see later).

None=*text*	If no choices are shown at all, the program will show *text*.

Examples:
none=You must be naked.
none=There are no special instructions.

When does the program show a new selection?

It is up to you how often the program will change the selection for the same instruction.

Change=daily	Means that the sub will get the same answer all day, but a new answer the next day. This is useful for most purposes. If you do not specify Change, Change=Daily is used.
Change=program	Means that every time you use Instructions= or Clothing= in a report, procedure etc., the program will generate a new answer. If the sub asks later for the same instructions, he will get the same answer.
Change=always	Means that every time the sub presses the Ask button, he will get a new answer. This is mostly useful for testing a script. If you use this option in real life, the sub can keep pressing the Ask button until he gets an answer he likes. A better way to test your script is to use the Test menu. See the chapter "Testing your script".

Using sets to control the choices

Often you want to select choices depending on what has already been chosen. Or you may want your choices to depend on some other circumstance. You can do this by using sets. Instead of a Choice keyword and one or more Option keywords, you add a Set keyword to specify that you have put the choices available in a set. See below how to define a set.

Set=*setname*	Tells that the options will be picked from the set *setname* (see below). Use as many lines as necessary. You can mix set lines and choice/option lines, and the result will be shown in the sequence you use.
Weight=*n*	Optional. Use this if you use Select=random and do not want an equal chance for each set to be chosen. Weight=2 will give the last set twice as much chance to be selected. Weight=3 three times. Etc.

More control of what is chosen

You now know that you build instructions of choices and sets. If you don't add anything further, all choices and sets are used in the sequence they are written. However, you have some further possibilities to control what is chosen. You can use the Select keyword.

Select=all	The program will use all of the set and choice keywords between this select and the next select. (The same as if you don't use select at all.)
Select=first	The program will only use the first possible set or choice between this select and the next select. A choice keyword is always possible. A set may be impossible, if you have used If, IfNot, IfChosen or IfNotChosen. See below.
Select=random	The program will choose a random set or choice keyword between this select and the next select.

Sets

Sets are collections of choices. Sets can be used or not used depending on what else is chosen. Sets are useful for conditional instructions and to avoid repeating the same instructions again and again. The same set may be used in several instructions. Sets can use other sets, but the set that is referred to must be defined before the set that refers to it.

[set-*name*]	Defines the beginning of a set.
Option=new	Works like in instructions (see above).
Set=*setname*	Works like in instructions (see above).
Weight=*n*	Works like in instructions (see above).
OptionSet=*setname*	Works like in instructions (see above).
Select=...	Works like in instructions (see above).

If sets are not always to be used

IfChosen=*precondition,precondition..* If any of the words in the precondition list exist in the items chosen in former choices, the program will use this set. If none of the words exists in the chosen items, the set will not be used. Consider using OptionSet instead of IfChosen, as it will give better control.

IfNotChosen=*precondition,precondition..* If none of the words in the precondition list exists in the items chosen in former sets, the program will use this set. If any of the words exist in the chosen items, the set will not be used. Consider using OptionSet instead of IfNotChosen, as it will give better control.

If=*flagname,flagname...* The set will only be used if all the flags in the list are set. You may have more than one If line. If there is more than one line, the set will be used if any line is true.

NotIf=*flagname,flagname...* If all of the flags in the list are set, the set will not be used. You may have more than one NotIf line. If there is more than one line, the set will not be used if any line is true.

An old way to define choices

This is an older way of defining choices. It is not recommended, as it is less flexible than the way described above. It can be used in instructions, clothing instructions and sets.

Choice=*option,option,option...* This is the simple form. The program will chose one *option* randomly. You can have as many choice lines as you wish, the program will chose one item from each line.

Choice=*option,weight,option,weight..* Use this form if you do not want an equal chance for each *option* to be chosen. The list alternates between items and the weight of that item. E.g.:

"choice=skirt and blouse,2,slacks and blouse,2,dress,1" will tell the program to choose skirt and blouse 2 days out of 5, slacks and blouse 2 days out of 5 and a dress 1 day out of 5. You can omit some or all of the weights, in which case the option will be given a weight of 1. The maximum possible weight for an option is 255.

Cloth reports

The sub can report what he is wearing. The sub can chose to make the report himself via the communications menu, or he can be ordered to make the report.

The principle is that you (the script writer) specify what you want to know, and the sub gives the wanted information. As a script writer you define some cloth types and what you want to know about each. For each type you define a number of attributes that the sub must describe. Each attribute can either be a simple text field, or a list the sub can chose from.

Defining the cloth types

[clothtype-*type*]	This line starts a type definition.
Attr=*attributename*	This line defines an attribute about the type. This could be style, length, color, size etc. Whatever you want to know.
Value=*value*	If you want the sub to choose attribute values from a list, you code a number of Value keywords after the Attr keyword they describe. A Value keyword always refers to the preceding Attr keyword.
Value=?	Means that although the sub can chose values from a list, he is also allowed to write a different value, not on the list.

Ordering the sub to make a cloth report

You can order the sub to make a cloth report in permissions, reports, confessions, procedures, timers, popups or sets. When you order the sub to make a cloth report, the sub is not allowed cancel the report. If the sub makes a cloth report via the menu *Communications*, then the sub is allowed to cancel the report.

ClothReport=*text*	Brings up the cloth report window with *text* as the headline.
ClothReport=1	Brings up the cloth report window with a standard headline.
ClearCloth=1	Removes all the sub's current clothes. The next time the sub is asked to make a cloth report, he will start with nothing selected. ClearCloth does not remove the clothes available, just the current clothes (below "You are wearing").

Importing and exporting clothes

The clothes that the sub enters in the system are stored in the status file, along with everything else about the sub. When the sub deletes the status file, it is all lost. When running seriously, you should not delete the status file and thus restart the program, as

this means that all punishments and assignments are removed and the merit score is reset. However, when testing the program, you might want to keep the clothes you have entered. And there may be situations where you want to allow the sub to delete the status file and restart the program without losing the clothes.

In the test menu you will find the options Export clothes and Import clothes. You can use Export clothes to create a file containing the clothes you have entered. Then, after a restart, you can use Import Clothes to import the clothes to your new file. See The test menu.

If you want to allow the sub to export and import clothes, you can add ClothExport=1 to the [general] section.

ClothExport=1	Adds Export clothes and Import clothes to the File menu.

When the sub adds a cloth type

It's possible for the sub to add a new cloth type, if he needs to add a cloth item that doesn't belong to the existing types. When the sub adds a new type, he can delete it again.

You can "take over" the new type by defining it in the script. When you do, the sub can no longer delete the type. And you can add new attributes.

If you add the new type to the script, and later delete it from the script, it no longer exists, and all items of that type will be removed.

Advanced script writing

This section describes more advanced script writing.

Changing the name of elements

Sometimes you want to give a different name to a permission, report etc. than the one written in the header between the [and] brackets. One reason might be that you want to define more than one element that, for the sub, looks the same. Another reason could be that you want to refer to short names in your script and still give the sub a long and meaningful name.

You change the name with the Title keyword. You can use it in status, reports, permissions, confessions, timers, popups, jobs, punishments and procedures.

Title=*new_name* New_name is what is presented to the sub.

Examples:

```
[permission-sleep-workday]
title=Go to sleep
if=sunday
if=monday
if=tuesday
if=wednesday
if=thursday

[permission-sleep-weekend]
title=Go to sleep
if=friday
if=saturday
```

Procedures

You may want a lot of reports or confessions to behave the same way. Instead of defining a lot of very similar actions, you can define a procedure to do the wanted action and then have the reports call the procedure. This approach makes it much easier to change the behaviour of all the procedures at once. This is only one example of how procedures can be used.

Another reason for using proceduresis conditional actions. You may want something to happen only under special circumstances. You can put the actions in a procedure and then use flags to decide whether you want the actions done.

Procedures can do the same things as reports, but the sub cannot start a procedure. A procedure is started from reports, permissions, timers, popups, jobs, punishments or questions. Almost everywhere you define some action the program must do, you can call a procedure.

Note: A procedure cannot call itself, neither directly nor indirectly. Indirectly means procedure1 calls procedure2 that tries to call procedure1. Succeeding attempts to call the same procedure will be ignored, and you will get a message in the report. The reason for this is to protect you from setting the program in an indefinite loop by accident.

What must be there

[procedure-*name*]	This line starts a procedure definition. Replace *name* with the name of the procedure.

What can you do in a procedure?

You can use the same keywords for procedure as for reports, including Procedure= to call another procedure.

How to use a procedure?

A procedure can be started from reports, permissions, timers, popups, jobs, punishments or questions-almost everywhere you define some action the program must do.

Procedure=*procedurename*	The procedure *procedurename* is called. You can have more than one procedure line.

You can use variables in the procedure name when calling a procedure. Note that if you do, there is no check that the procedure exists. And if you end up calling a procedure that doesn't exist, the procedure call is ignored.

Conditional actions

You can use If, NotIf, NotBefore, NotAfter, NotBetween and PreStatus to specify that the procedure shall not be executed, even if it is called. See chapters about Flags and Status.

You can combine conditional actions with several procedure calls to determine which procedure is called. If you have more than one Procedure keyword in sequence, you can use the select keyword to tell which procedure(s) you want executed. Use select before the first Procedure keyword which must be affected.

Select=All	All the following procedures are called.
Select=First	Only the first possible of the following procedures are called. It is possible to call a procedure, if there are no If, NotIf, NotBefore, NotAfter, NotBetween or PreStatus keywords in the procedure that makes the procedure not callable.
Select=Random	The program will select a random procedure from the following procedures. Only the selected one is called.

If you do not specify Select, all procedures will be called.

Example:

```
Select=All
procedure=procedure1
procedure=procedure2
Select=Random
procedure=procedure3
procedure=procedure4
procedure=procedure5
Select=First
procedure=procedure6
procedure=procedure7
procedure=procedure8
Select=All
procedure=procedure9
procedure=procedure10
```

In the above example, the following will happen in this sequence:

1. Procedure 1 is called
2. Procedure 2 is called
3. Either Procedure3, procedure4 or Precedure5 is called. The program will choose randomly.

4. Either Procedure6, procedure7 or Precedure8 is called. The program will choose the first possible procedure.
5. Procedure 9 is called.
6. Procedure 10 is called

Weighting procedures with Select=Random

If you use Select=Random, you may use some procedure to be chosen more often than others. You can control that by weighting the procedures.

Procedure=procedurename,*weight* The procedure *procedurename* is called. Used with Select=Random, the procedure will have the weight specified by *weight*. *Weight* must be an integer between 1 and 255.

Example:

```
Select=Random
procedure=procedure1,1
procedure=procedure2,3
procedure=procedure3,1
```

In the above example, procedure2 have a weight if 3, which is three times what the others have. Which means that procedure 3 will be called three times as often as procedure 1 or procedure 2.

Popups

Popups define actions to popup randomly. Use popups to check that the sub is home when he/she is supposed to be, to give random instructions to the sub, or for whatever else you want.

How to define a popup

[popup-*name*] This line starts a popup definition.

You can use most of the same keywords that can be used for a report. Use the keyword to give a message or an order to the sub. Use Prestatus, If and Notif, NotBefore, NotAfter and NotBetween to control when a popup can be invoked. Se the chapter Flags later.

How to activate popups

Activate popups by specifying the following keyword on a status definition:

PopupInterval=*min,max* If PopupInterval is present in the definition of the current status, the program will periodically invoke a Popup. *Min* and *max* define the minimum and maximum time between popups. *Min* and *max* are written as hh:mm (hours and minutes) or hh:mm:ss (hours, minutes and seconds).

How to enable and disable popups by use of flags

Normally it's the status that defines whether popups are active. But maybe under certain circumstances you might want to disable popups, even in a status where they are usually enabled. Do this by setting a flag when you want to disable the popups,or by only setting a flag when you want popups enabled.

Use this in a status definition:

PopupIf=*flagname* Only if *flagname* is set, popups will occur.
NoPopupIf=*flagname* If *flagname* is set, no popups will occur.

Use some popups more often than others

When it is time for a popup, the program will search through all possible popups and select one. It is possible to influence this selection and have some popups turn up more often than others. You do this by weighting the popups. The higher weight, the more likely the popup is to be chosen. A popup with a weight of 3 is chosen 3 times more often than one with weight of 1.

Weight=*weight*	Tells the weight of the popup. *Weight* can be a positive number or a counter. Or, it can be two values separated by comma and the program will choose a random value between the two. If you don't code weight, the popup will have a weight of 1.

How to change the sound when a popup occurs

You can change the sound that plays when a popup occurs. You can set up a general sound for all popups, and you can set individual sounds in individual popups. The sound must be in a .wav file.

Use the keyword Alarm in the [general] section to set up an alarm file for all popups which don't have an individual alarm file.

Use the keyword Alarm in the [popup-*name*] section to set up an alarm file for this popup only.

alarm=*wave file*	The name of a sound file to be used as an alarm when a popup occurs.

Use the keyword PopupAlarm in the [status-*name*] section or the [popupgroup-*name*] section to set up an alarm file for this status or group.

PopupAlarm=*wave file*	The name of a sound file to be used as an alarm when a popup occurs.

The program is looking for a sound file in this sequence:

1. The popup
2. The popup group
3. The status
4. The [General] section
5. It uses the default name *alarm.wav*.

The first filename found is used. If the file doesn't exist, there will be no sound.

How to control response times for popup

You may want to ensure that the sub reacts to a popup within a specified time. To do this, see Response time for popups.

How to use popup groups

Normally it's the status that defines how popups are used, but you may have several statuses with the same popup settings. Writing and maintaining the popup keywords for several statuses may be tedious.

You can restrict a popup to be used in a certain status by using the Prestatus keyword. However, if you have a lot of popups which all are to be used in certain statuses, your script may be rather complex.

Popup groups can help you with these problems. You can define popup groups for popups that are to be used in the same way. You can connect statuses and popups to the popup groups. A status can only belong to one popup group, whereas a popup can belong to several groups.

If a status belongs to a popup group, it will use all the popup settings from the status. Popup control keywords in the status itself will be ignored.

If a popup belongs to one or more popup groups, those groups will be compared to the popup group of the current status. Only if the current status belongs to one of the groups, will the popup be used.

Defining popup groups

[Popupgroup-*name]*	Starts a popup group definition.
PopupInterval=*min,max*	Same as if you used it in a status definition.
PopupIf=*flagname*	Same as if you used it in a status definition.
NoPopupIf=*flagname*	Same as if you used it in a status definition.

The following keywords are described in Response time for popups.

PopupMinTime=*hh:mm*
PopupQuickPenalty1=*points*
PopupQuickPenalty2=*ratio*
PopupQuickMessage=*text*
PopupMaxTime=*hh:mm*
PopupSlowPenalty1=*points*
PopupSlowPenalty2=*ratio*
PopupSlowMessage=*text*
PopupQuickProcedure=*procedurename*
PopupSlowProcedure=*procedurename*

Using popup groups

Use this in a status definition:

Popupgroup=*name*	Specifies that this status belongs to the popup group *name*. A status can only belong to one popup group.

Use this in a popup definition:

Group=*name*	Specifies that this popup belongs to the popup group *name*. You can have more Popupgroup keywords, if the popup belongs to more than one group.

Timers

Timers define actions to happen on time. You can use timers give the sub an order on a fixed time (e.g. go to bed) or to do something that the sub doesn't even notice, like setting or removing a flag (see later). You can use timers for anything that must happen on time.

Note that timers do not automatically play an alert sound. You must use the Sound keyword if you want to play a sound. See Playing sounds.

Note that to make something happen after an amount of time has passed, it is better to use a flag with a duration. See more in Using flags to make something happen after a period of time.

What must be there

[timer-*name*]	This line starts a timer definition. Replace *name* with the name of the timer.
start=*hh:mm,hh:mm*	Tells the timer to activate at a random time between the first and the second time. Times are in hours and minutes (24 hour clock).
end=*hh:mm*	If the end time arrives and the timer has not been activated, the timer will not start, but will wait till next day.

Using status with timers

Often you only want the timer be activated when the sub is in a specific status. Use the keyword PreStatus to specify in which status the timer can be activated. If the sub is in a correct status when the time is up, the timer will be activated. If not, the timer will wait until the sub gets into that status. If the time specified by the keyword End is reached without the status having been reached, the timer will wait until the next day.

PreStatus=*status,status...*	When the time arrives, the timer will only be activated if current status is in the list of *status,status.....*

Using timers with flags

You can use If and Notif, NotBefore, NotAfter and NotBetween to control when a timer can be activated. If one of these conditions gives that the timer should not be activated, the timer will wait until the next day.

Timers and weekdays

If you want a timer to be activated only on certain weekdays, you can use the automatic flags (see the chapter Flags).

If=Sunday	The timer will only be activated on Sundays.
NotIf=Saturday	The timer will not be activated on Saturdays.

Writing messages to the sub

There are different ways of giving messages to the sub. You can show a simple message in a popup box with the Message keyword. You can do that in a permission, report, confession, procedure, timer or popup.

Message=*messagetext*	The *messagetext* will appear in a popup box.

If you have more than one message statement following each other, you can control how many are shown to the sub. Use the Select keyword before the first P Message keyword.

Select=All	All the following messages are shown.
Select=Random	The program will select a random message from the following Message keywords.

If you do not code a Select keyword, all messages will be shown.

Example:

```
Select=All
Message=This message will be shown.
Message=This message will also be shown.
Select=Random
Message=This message may be shown.
Message=Or This message may be shown.
Select=All
Message=This message will always be shown.
```

Another way of giving information to the sub is to write it in the message area in the main window. You do this using the Text keyword. While popup boxes generated from the Message keyword are one-time incidents, the message area is static. The text here will stay for a while. For this reason it can only appear in elements that define a period of time. You can use the text keyword in statuses, flags, jobs and punishments.

Text=*textline*	The *textline* will appear in the message area. You can have more than one text line. They will all show.
Text=%instructions	This will show the last instructions given by the Instructions= keyword. It will not show instructions that the sub asked for.
Text=%clothing	This will show the last clothing instructions given by the Clothing= keyword. It will not show clothing instructions that the sub asked for.

You can have a fixed message at the top and/or bottom of the message area. You do that by adding one or both of the following keywords to the [General] section.

TopText=*textline*	The *textline* will appear at the top of the message area. You can have more than one TopText line. They will all show.
BottomText=*textline*	The *textline* will appear at the bottom of the message area. You can have more than one BottomText line. They will all show.

Asking questions to the sub

You may want to ask questions to the sub. There are different ways of doing this, depending on how you want to react to the answer.

Input, the really simple way

The simplest way of asking a question is to use the Input keyword. It can be used in in a permission, report, confession, procedure, timer or popup.

Input=*question*	The sub gets a popup box with headline *question* and a field to type a reply. The reply is recorded in the report.

The sub can choose not to reply to a question phrased by an Input keyword. However, you can catch that situation and react to it. Do that by adding the NoInputProcedure keyword along with the Input keyword.

NoInputProcedure=*procedurename*	If the sub doesn't answer the question, the procedure *procedurename* is called. You can use this procedure to punish the sub or do whatever you wish.

Input to variables: a bit more complex

You can ask the sub to type a text or a number which can be stored in a variable. You do that by using one of the keywords Input$ or Input". Read more in the chapter Variables.

Advanced questions: how to define them

Advanced questions are used when you want to ask the sub a question and have the program react to the answer.

[question-*name*]	This line starts a question definition. Replace *name* with the name of the question
Phrase=*question*	Defines the question the sub is asked.
?*answer*=*procedure*	Each line starting with a question mark (?) defines a possible answer to the question the sub is asked. The sub is presented with a list of answers and must choose one. When an answer is chosen the corresponding procedure will be activated. See the chapter Procedures.
?*answer*=*	If you write an asterisk (*) instead of a procedure name, the program will execute the keywords (statements) between this and the next answer.

Example 1:

```
[question-example]
phrase=Did you do as told?
?Yes=Well-done
?No=Disobedient
```

In this example the procedure Well-done will be executed if the reply is Yes and the procedure Disobedient will be executed when the reply is No.

Example 2:

```
[question-example]
phrase=Did you do as told?
?Yes=*
     message=Well done.
?No=*
     message=You should be ashamed of yourself!
     punish=50
```

This example have the actions to be taken written directly in the question.

Example 3:

```
[question-example]
phrase=Did you do as told?
?Yes=Well-done
?No=*
     message=You should be ashamed of yourself!
     procedure= Disobedient
```

This example is a combination. If the reply is Yes, the procedure Well-done will be executed. If the reply is No, a message will be shown and the procedure Disobedient will be called.

Advanced questions: how to ask them

You ask the question by using the keyword Question in a permission, report, confession, procedure, timer or popup.
Question=*question* Will ask the question.

Duration keywords, being too quick or too slow

These keywords deal with an interval of time. They can be used in statuses, flags, jobs and punishments. See later about response time for popups. You can specify the duration in hours and minutes (hh:mm) or in the special duration syntax. See Specifying intervals and durations later.

MinTime=*hh:mm*	The interval must be at least *hh:mm* hours and minutes. You can use MinTime=*hh:mm,hh:mm* and the program will pick an interval between the two values. You can use a time variable instead of *hh:mm*.
QuickPenalty1=*points*	If MinTime is not reached then the sub is punished with the value of *points*. Use this to give a minimum punishment for being early and still make sure that each minute counts. Used only if MinTime is specified.
QuickPenalty2=*ratio*	The penalty for being early is *ratio* merit points for each minute late. *Ratio* may have decimals, e.g. 0.8 is valid. Used only if MinTime is specified. *Ratio* is reduced when the sub is very early.
QuickMessage=*text*	If you are not satisfied with the standard message when the sub is early, you can write your own message here. If you use a number sign (#) in the text, it will be replaced with the amount of time the sub is early. If you use a ¤ or a percent sign (%) in the text, it will be replaced with the time the sub has used.
QuickProcedure=*procedurename*	The procedure *procedurename* is called when the sub is early. Use it for any action you want to take. The procedure is called after the sub is punished. If you specify QuickProcedure and you do not specify QuickPenalty1 or QuickPenalty2 or QuickMessage, the sub will not receive a message from the program.
MaxTime=*hh:mm*	The interval must not exceed *hh:mm* hours and minutes. You can use MaxTime=*hh:mm,hh:mm* and the program will pick an interval between the two values. You can use a time variable instead of *hh:mm*.
SlowPenalty1=*points*	If MaxTime is exceeded then the sub is punished with the value of *points*. Use this to give a minimum punishment for being late and still make sure that each minute counts. Used only if MaxTime is specified.
SlowPenalty2=*ratio*	The penalty for being late is *ratio* merit points for each minute early. *Ratio* may have decimals, e.g. 0.8 is valid. Used only if MaxTime is specified. *Ratio* is reduced when the sub is very late.

SlowMessage=*text* If you are not satisfied with the standard message when the sub is late, you can write your own message here. If you use a number sign (#) in the text, it will be replaced with the amount of time the sub is early. If you use a ¤ or a percent sign (%) in the text, it will be replaced with the time the sub has used.

SlowProcedure=*procedurename* The procedure *procedurename* is called when the sub is late. Use it for any action you want to take. The procedure is called after the sub is punished. If you specify SlowProcedure and you do not specify SlowPenalty1 or SlowPenalty2 or SlowMessage, the sub will not receive a message from the program.

MinTimeProcedure=*procedurename* The procedure *procedurename* is called when the mininum time specified with MinTime is up. You can use this to give a message to the sub. You can use MinTimeprocedure without MinTime in punishments with ValueUnit=minute or ValueUnit=hour. Then *procedurename* will be called when time is up.

MaxTimeProcedure=*procedurename* The procedure *procedurename* is called when the maximum time specified with MaxTime is up. You can use this to give a message to the sub.

Using MinTime and MinTimeProcedure with jobs and punishments

If you use MinTime with a job or punishment, and the sub finishes before the MinTime specified, the job or punishment will not be completed. The reason for this is to make it more difficult to cheat.

If you use MinTimeProcedure in a punishment with ValueUnit=minute or ValueUnit=hour without specifying MinTime, the program will know the minimum time. You can use this to alert the sub when time is up.

Response time for popups and timers

If you want to measure the response time for popups, you can add the following keywords to the status or popup group, where you put the PopupInterval keyword. Or to the timer definition.

PopupMinTime=*hh:mm*
PopupQuickPenalty1=*points*
PopupQuickPenalty2=*ratio*
PopupQuickMessage=*text*
PopupMaxTime=*hh:mm*

PopupSlowPenalty1=*points*
PopupSlowPenalty2=*ratio*
PopupSlowMessage=*text*
PopupQuickProcedure=*procedurename*
PopupSlowProcedure=*procedurename*

When you use PopupMaxTime in a status, popup group or timer, the program will display the message "Yes, Master". It is the response time of this message that is checked. If you want another message, you can use the keyword PopupMessage to describe your own message. PopupMessage can be used in popups, popup groups or in the [General] section. The message you specify in the [General] section is used for all popups where you don't code PopupMessage.

PopupMessage=*message* Use this message instead of "Yes, Master".

Advanced permission use

This chapter is about advanced ways of controlling whether the sub will be granted permissions,and about how to take action when permission is denied.

Permissions based on time

DenyBefore=*hh:mm*	If the sub asks for permission before *hh:mm*, permission is denied. *hh:mm* is using the 24-hour clock. If you specify two values separated by comma, the program will use a random value in that range. You can use a time variable instead of *hh:mm*.
DenyAfter=*hh:mm*	If the sub asks for permission after *hh:mm*, permission is denied. *hh:mm* is using the 24-hour clock. If you specify two values separated by comma, the program will use a random value in that range. You can use a time variable instead of *hh:mm*.
DenyBetween=*time1*,*time2*	If the sub asks for permission between time1 and time2, permission is denied. *Time1* and *time2* is in the format hh:mm using the 24-hour clock.

Permissions based on flags

DenyIf=*flagname*	Permission is denied if the flag *flagname* is set.
PermitIf=*flagname*	Permission is given if the flag *flagname* is set.

Permissions based on the merit score

DenyBelow=*points*	If the sub's merit score is below *points* points, permission is denied. If you specify two values separated by comma, the program will use a random value in that range. You can use a counter instead of a value.
DenyAbove=*points*	If the sub's merit score is above *points* points, permission is denied. If you specify two values separated by comma, the program will use a random value in that range. You can use a counter instead of a value.

More variation in permission limits

You can generate even more random permissions or you can control the permissions by use of counters.

Pct=*value1*,*value2*	The program will pick a random number between *value1* and *value2* and use that as the permission percentage. *Value1* and *value2* can be numbers or counters.

Variable permissions based on the merit score

Sometimes you want the chance of getting a permission to vary depending on the merit score.

Pct=Var	Tells that you will use a variable percentage of chance to get permission.
HighMerits=*points*	If the sub's merit score is above *points* points, the sub has the chance of getting permission which is specified by HighPct. *) counter instead of a value.
HighPct=*number*	Use together with HighMerits. If the sub's merit score is above the number given in HighMerits, the percentage chance of getting will be *number*%. If you use HighMerits without HighPct, the chance will be 100%. *)
LowMerits=*points*	If the sub's merit score is below *points* points, the sub has the chance of getting permission which is specified by LowPct. LowMerits must be greater than zero. *)
LowPct=*number*	Use together with LowMerits. If the sub's merit score is below the number given in LowMerits, the percentage chance of getting permission will be *number*%. If you use LowMerits without LowPct, the chance will be 0%. *)
*)	For the keywords marked with *), you can If you specify two values separated by comma, the program will use a random value in that range. You can use a counter instead of a value.

Example:
[Permission-Do something]
Pct=Var
HighMerits=800
HighPct=90
LowMerits=300
LowPct=20

The above example works like this:

- If the merit score is above 800, the sub will have 90% chance of getting permission.
- If the merit score is below 300, the sub will have 20% chance of getting permission.
- If the merit score is between 300 and 800, the subs chance of getting a permission will vary between 20% and 90%. The higher merit score, the better chance.

Use a procedure to decide whether permissions should be granted

You can call a procedure when a permission is asked and set a special flag to tell if permission should be granted or denied. You can also choose to cancel the request.

BeforeProcedure=*name*	When permissions is asked, the procedure *name* is called before it is decided whether to give permission.

The procedure can set one of these flags:

zzPermit	Set this flag if permission must be given.
zzDeny	Set this flag if permission must be denied.
zzCancel	Set this flag if you want to cancel the request. Canceling the request will not influence what happens next time the permission is asked, and the sub will not get a permission denied message.

If none of these flags are set, the usual rules determine if permission is given.

Example:

```
[permission-Use the toilet]
beforeProcedure=BeforeToilet

[procedure-BeforeToilet]
question=BeforeToilet

[question-BeforeToilet]
text=Is it really urgent?
?Yes, it is very urgent=ToiletYes
?No, not really yet=ToiletNo
?Sorry, wrong permission=ToiletCancel

[procedure-ToiletYes]
setflag=zzPermit

[procedure-ToiletNo]
setflag=zzDeny

[procedure-ToiletCancel]
setflag=zzCancel
```

Take action when a permission is denied

When a permission is denied, it's possible for you to take some actions.

DenyProcedure=*name*	When permission is denied, the procedure *name* is called.
DenyFlag=*flagname*	When permission is denied, the flag *flagname* is set. You can have more than one flag; separate the names by commas.
DenyMessage=*message*	When permission is denied, display the message.
DenyLaunch=*filename*	When permission is denied, the file *filename* is launched.
DenyStatus=*statusname*	When permission is denied, the program switches to status *statusname*.

Sign in

The sub can be asked to sign in at regular intervals. Sign in means using the program in some way. If nothing else, he can click on a "Reset" button in the lower right corner of the screen.

You can use this to make sure the sub is home and awake, or simply to keep the sub on his/her toes.

Sign in is connected to a status. Use the SigninInterval keyword in a status definition.

SigninInterval=*hh:mm:ss*	The sub must sign in at least once every *hh:mm:ss* (hours, minutes and seconds). Instead of *hh:mm:ss* you can use a time variable. You can use two values separated by a comma. The program will then pick a random time between the two values. The program will pick a new value every time the sub signs in.

Punishing the sub for late sign in

SigninPenalty1=*number*	If the sub is late signing in, he is punished with a severity of *number*.
SigninPenalty2=*number*	If the sub is late signing in, he is punished with a severity of *number* for each minute he is late.

If you specify both SigninPenalty1 and SigninPenalty2, then both are used.

Example:

```
[status-Normal]
    SigninInterval=00:30:00
    SigninPenalty1=40
    SigninPenalty2=3
```

The sign in time is 30 minutes. If the sub signs in after 34 minutes, he is 4 minutes late. He will be punished with a severity of 40 + 4*3 = 52.

Time keywords, being late or early

These keywords deal with the time of day. They specify the latest or earliest time the sub is allowed to do something, like make a report or ask a permission. They can be used in reports, confessions, procedures and permissions.

Latest=*hh:mm*	The report etc. should be made before the time given in *hh:mm* (hours and minutes, 24 hour clock).
LatePenalty1=*points*	Use this to give a minimum punishment for being late. Used only if Latest is specified.
LatePenalty2=*ratio*	The penalty for being late is *ratio* merit points for each minute late. *Ratio* may have decimals, e.g. 0.8 is valid. *ratio* is reduced when the sub is very late. Used only if Latest is specified.
LateProcedure=*procedure*	When the sub is late, the procedure *procedure* will be called.
Earliest=*hh:mm*	The report etc. should be made after than the time given in *hh:mm* (hours and minutes, 24 hour clock).
EarlyPenalty1=*points*	Use this to give a minimum punishment for being early. Used only if Earliest is specified.
EarlyPenalty2=*ratio*	The penalty for being late is *ratio* merit points for each minute too early. *Ratio* may have decimals, e.g. 0.8 is valid. *Ratio* is reduced when the sub is very early. Used only if Earliest is specified.
EarlyProcedure=*procedure*	When the sub is early, the procedure *procedure* will be called.

The severity of the punishment given is calculated by multiplying penalty2 by the number of minutes and adding penalty1. The penalty is reduced when the sub is very late to avoid huge punishments.

Specifying Latest and Earliest

Latest and Earliest can be specified by using two values. The program will then pick a random value between the two, each time it is needed.

Latest and Earliest can be specified by using a time variable instead of a constant time. If the value of the time variable is less than 24 hours, it is treated as a time of day. However, if the value is above 24 hours, it is treated as a date and time.
See Time variables.

Controlling menus based on time

Sometimes you want to control which permissions, reports and confessions are available in a menu, based on time. You may want to control in the same way whether procedures can be called or timers or popups are allowed to spring.

Use these keywords in permissions, reports, confessions, procedures, timers or popups.

NotAfter=*hh:mm*	The report etc. is not available after the time given in *hh:mm* (hours and minutes, 24 hour clock).
NotBefore=*hh:mm*	The report etc. is not available before the time given in *hh:mm* (hours and minutes, 24 hour clock).
NotBetween=*hh:mm,hh:mm*	The report etc. is not available between the time given in the first *hh:mm* and the time given in the last *hh:mm*.

Prevent simultaneous assignments

If the sub has several long-running assignments (jobs or punishments) of the same kind in the assignment list, the program will prevent him from starting them simultaneously. Examples: Wear a gag for 4 hours and Wear a gag for 3 hours.

However, if the sub has different assignments, that can't possibly be carried out simultaneously, the program has no chance of knowing this, unless you tell it. Examples: Wear a ball gag for 4 hours and Wear a ring gag for 3 hours. In this case, you must tell the program that the punishments are mutually exclusive. You do this by using the Resource keyword with same resource name on both jobs or punishments.

Resource=*name*	*Name* is the name that you chose to describe the "resource". You can have several lines with the Resource keyword.

Example:
[punishment-Wear a ball gag for # hours]
resource=Gag
[punishment-Wear a ring gag for # hours]
resource=Gag
[job-Wear a gag for 2 hours]
resource=Gag

Events

Events are things that happen, which you may want to react to and which are not controlled in other ways. You can specify a procedure to start on each event.

[events] This line starts the event section.

Events about program start, stop etc.

FirstRun=*procedure* Activates *procedure* the very first time a new script is used.
OpenProgram=*procedure* Activates *procedure* each time the program is opened.
CloseProgram=*procedure* Activates *procedure* each time the program is closed.
StartFromPause=*procedure* Activates *procedure* when the program is opened after a pause activated by PgmAction=Pause. This procedure is called before the OpenProgram event.
DeleteStatus=*procedure* Activates *procedure* just before the status file is deleted.
Minimize=*procedure* *procedure* is called when the program is minimized.
Restore=*procedure* *procedure* is called when the mimimized program is restored.

Events about the report file

BeforeNewReport=*procedure* Activates *procedure* just before a new report is made. Use this to send an email with the last part of the old report before a new report is started.
AfterNewReport=*procedure* Activates *procedure* just after a new report is made.

Events about reports and confessions

AfterReport=*procedure* *procedure* is called when the sub reports or confesses something. It is called after the report or confession have been finished, and after call to the events ForgetConfession and IgnoreConfession. The variable $zzName will contain the name of the report or confession.

ForgetConfession=*procedure* *procedure* is called when the sub makes a confession that he has forgotten to ask a permission. See Automatic confessions. You can use the variables $zzName or $zzTitle to get the name of the forgotten confesstion. Se Predefined string variables.

IgnoreConfession=*procedure* *procedure* is called when the sub makes a confession that he has done something though permission was denied. See Automatic confessions. You can use the variables $zzName or $zzTitle to get the name of the forgotten confesstion. Se Predefined string variables.

Events about permissions

PermissionGiven=*procedure* *procedure* is called when the sub is given permission to something. The variable $zzName will contain the name of the permission.

PermissionDenied=*procedure* *procedure* is called when the sub is denied permission to something. The variable $zzName will contain the name of the permission.

Events about punishments

PunishmentGiven=*procedure* Activates *procedure* just after a punishment is given, except when it is asked for by the sub. The counter #zzpunishment will contain the severity of the punishment.

PunishmentAsked=*procedure* Activates *procedure* just after the sub asks for a punishment. The counter #zzpunishment will contain the severity of the punishment.

PunishmentDone=*procedure* Activates *procedure* just after a punishment is done (finished).

Events about jobs

JobAnnounced=*procedure* Activates *procedure* just after a job is announced to the sub.

JobDone=*procedure* Activates *procedure* just after a job is done (finished).

Events about assignments

BeforeDelete=*procedure* Activates *procedure* when the sub tries to delete an assignment. If the *procedure* sets the flag zzDeny, the assignment will not be deleted.

Events about clothes

BeforeClothReport=*procedure* Activates *procedure* just before a cloth report is made.

AfterClothReport=*procedure* Activates *procedure* just after a cloth report is made.

CheckOn=*procedure* Called every time VM finds that the sub is not wearing something he should. Use the string variable $zzCheck to tell what it is. See Check the sub's cloth report.

CheckOff=*procedure*	Called every time VM finds that the sub is wearing something he is not supposed to wear. Use the string variable $zzCheck to tell what it is it is. See Check the sub's cloth report.
CheckAll=*procedure*	Called after a cloth report is made if the checks showed a difference between what the sub is wearing and what he is supposed to wear. Use the counter #zzClothFaults to tell how many violations there were. See Check the subs cloth report.

Other events

Signin=*procedure*	Activates *procedure* whenever the sign in button is pressed.
MeritsChanged=*procedure*	*procedure* is called every time the merits are changed. Note: If you change the merits in *procedure*, it will not be recalled. When in *procedure*, you can use the variables #zzBeforeMertis to show the merits before change, #zzAddMerits to show the change and #zzMerits to show the merit points after the change. *Procedure* is called after the merits are changed.
AutoAssignEnd=*procedure*	*procedure* is called when an autoassign period ends. It is not called immediately when the time is up; it is called when the active assignment is finished and you're back in a status with autoassign=1.
AutoAssignNone=*procedure*	*procedure* is called when there are no assignments available to do and the automatic assignment period is not finished.
MailFailure=*procedure*	Activates *procedure* if for some reason a mail cannot be sent.
NewDay=*procedure*	*procedure* is called at midnight if the program is running. If the program is not running at midnight, *procedure* is called first time the program is started. If the program is closed for more than one day, *procedure* is only called once.

Starting assignments - Keeping the sub busy

You may have periods where you want the program to keep the sub busy, to tell him exactly what to do. In VM this is called “automatic assignments”.

In a period of automatic assignments the program will pick the first assignment (the one with the first deadline) that can be done in the time available. It will order the sub to do it now. When the sub reports it finished, the VM will choose the next one. This will go on until the time is up or there are no more assignments that fit into the time available.

VM will never start an assignment that is marked as long running, as the long running assignments are meant to be simultaneous with something else.

How to do it

There are three or four things you need to do:

1. You need to make sure that your jobs and punishments have an estimate. That's the only way VM can now how long the assignment will take. You need not specify an estimate on punishments with ValueUnit=minute or ValueUnit=hour. If VM can not estimate an assignment, it will not use it.
2. (Optional) You may define some jobs that VM can use, if no other assignments are available.
3. You need to tell in which status VM is allowed to do automatic assignments.
4. You need to start the period and specify how long it will be.

Estimating jobs and punishments

In jobs and punishments:

Estimate=*hh:mm*	Tells how long time a job is estimated to take. Estimate is not necessary in punishments with ValueUnit=minute or ValueUnit=hour.

Defining which status to use

In status:

AutoAssign=1	Tells that in this status is allowed to use automatic assignments if automatic assignments are started.

Defining extra jobs

In jobs:

AutoAssign=*n*	Tells that this job may be started if no other assignments are available. *N* is usually 1, but may be a higher number. The

higher the number, the more likely the job is to be chosen. A job with AutoAssign=2 is chosen twice as often as a job with AutoAssign=1.
If you specify two values separated by comma, the program will use a random value in that range. You can use a counter instead of a value.

Starting automatic assignments

Use one of these keywords to start a period with automatic assignments. They can be used in reports, permissions, confessions, procedures, timers, or popups.

StartAutoAssign=time,*hh:mm* Starts automatic assignments and lets them run until a specific time of day. *hh:mm* specifies the time when automatic assignments are to stop. *hh:mm* is hours and minutes, using the 24-hour clock.

StartAutoAssign=interval,*hh:mm* Starts automatic assignments and lets them run for a fixed period. *hh:mm* specifies how long automatic assignments will be used.

StartAutoAssign=ask,*text* Starts automatic assignments and ask the sub how much time is available. *text* is the message shown to the sub. If *text* contains a comma, it must be enclosed in ". If you omit *text*, a standard text is used.

Knowing what happens

You can set up events that tells you when the period is over or when there are no assignments to do.

In [events]:

AutoAssignEnd=*procedure* *procedure* is called when an autoassign period ends. It is not called immediately when the time is up; it is called when the active assignment is finished and you're back in a status with autoassign=1.

AutoAssignNone=*procedure* *procedure* is called when there are no assignments available to do and the automatic assignment period is not finished.

Examples

```
[Status-Normal]
AutoAssign=1

;Work from the time you get home from work until 6 PM
[report-Home from work]
```

```
prestatus=On job
newstatus=Normal
StartAutoAssign=time,18:00

;Work one hour after dinner
[report-Finished diner]
prestatus=Dinner
newstatus=Normal
StartAutoAssign=interval,01:00

[report-I have some time available]
prestatus=Normal
StartAutoAssign=ask,"Then do some work for me. How much time do you have?"

;Define some events
[events]
autoassignEnd=autoEnd
autoassignNone=autoNone

[procedure-autoEnd]
message=OK, you may relax now.

[procedure-autoNone]
message=Nothing to do right now, I will be back.

; Define some extra jobs in case there are no assignments left
[job-Clean the toilet]
AutoAssign=1
Estimate=00:15

[job-Stand in the corner 10 minutes]
AutoAssign=1
Estimate=00:10

[job-Kneel on the floor 5 minutes]
AutoAssign=1
Estimate=00:5

[job-Write 50 times obey]
title=Write 50 times: "I will always obey my Master"]
AutoAssign=1
Estimate=00:30
```

Advanced cloth issues

Check the sub's cloth report

You can let VM check if the sub is wearing what she is supposed to wear. Do this with the Check and CheckOff keywords.

The concept

- In a clothing instruction you can use Check=stockings to tell that the sub must wear stockings.
- In a clothing instruction you can use CheckOff=panties to tell that the sub is not allowed to wear panties.
- In the cloth type stockings, you use Check=stockings.
- In the cloth type panties, you use Check=panties.
- To set up the requirements, you must use Clothing=*xxxx* to enable the requirements.

If you have done as above, and the sub makes a cloth report that does not include the cloth type stockings, then VM will display a message saying "You are supposed to wear stockings". If the sub makes a cloth report that includes the cloth type panties, then VM will display a message saying "You are not supposed to wear panties".

If you want to define your own actions instead of just a message, you can do so. You may want to punish the sub, lower the merit points or do something else.

The requirements are only activated when you use the Clothing keyword in a report, permission, procedure or anywhere else. If the sub asks for instructions himself, no requirements are set up. The reason for this is simple: The sub can ask for any instructions at any time. This is not to say that he is suddenly required to wear what he asks for. Only when the master orders a specific clothing are the requirements activated.

How to define the cloth report.

In an instruction or set definition, you can use the following keywords.

Check=*name*	If the last option before this line is chosen, then the sub is supposed to wear *name*. *Name* is the name displayed to the sub in the message "You are supposed to wear *name*".
CheckOff=*name*	Means that if the last option before this line is chosen, then the sub is not supposed to wear *name*. *name* is the name displayed to the sub in the message "You are not supposed to wear *name*".

Examples:

```
[set-underwear]
    choice=new
        option=panties
            check=panties
        option=string panties
            check=panties
            check=string panties
        option=no panties
            check=no panties
    choice=new
        option=pantyhose
        option=stockings
            check=stockings
        option=bare legs
            checkOff=stockings
            checkOff=pantyhose
```

How to define the cloth type

In a clothtype definition you can use the Check keyword. Naturally, there is no CheckOff keyword in cloth types. Here you only tell what the sub is wearing, not what she is not wearing.

Check=*name*	Means that the cloth type is equal to *name*. *name* must be the same name as used in a Check or CheckOff. If you use Check after a value, it applies only to that value. If you use it before the first Attr keyword, it applies to the whole cloth type.

Examples:

```
[clothtype-panties]
    check=panties
    attr=color
    attr=style
       value=normal
       value=string
           check=string panties
       value=?
    attr=description

[clothtype-stockings]
    check=stockings
    attr=color
    attr=description
```

How to activate / deactivate the requirements

The requirements are only activated when you use the Clothing keyword in a report, permission, procedure or anywhere else. If the sub asks for instructions himself, no requirements are set up. The reason for this is simple: The sub can ask for any instructions at any time. This is not to say that he is suddenly required to wear what he asks for. Only when the master orders specific clothing are the requirements activated.

You can deactivate the requirements by using a new Clothing keyword or by using the keyword ClearCheck. The Clothing and ClearCheck keywords can be used in permissions, reports, confessions, procedures, timers or popups

Clothing=*name*	This line displays clothing instructions for *name*. It also activates checking of required clothes.
ClearCheck=1	Deactivates the checking of the clothes.

Examples:

```
[permission-Go to work]
Pct=100
Clothing=On job
ClothReport=What are you wearing?
Newstatus=On job

[report-Home from job]
Prestatus=On job
NewStatus=Home
ClearCheck=1
```

How do define your own actions

Basically the program just informs the sub that he/she is not correctly dressed. If you want to take further action, like punishing the sub, you can use three events. See Events.

CheckOn=*procedure*	Called every time VM finds that the sub is not wearing something he should. Use the string variable $zzCheck to tell what it is.
CheckOff=*procedure*	Called every time VM finds that the sub is wearing something he is not supposed to wear. Use the string variable $zzCheck to tell what it is it is.
CheckAll=*procedure*	Called after a cloth report is made if the checks showed a difference between what the sub is wearing and what he is supposed to wear. Use the counter #zzClothFaults to tell how many violations there were.

Example 1:

```
[events]
CheckOn=CheckOn
CheckOff=CheckOff

[procedure-CheckOn]
PunishMessage=You are supposed to wear {$zzCheck}.
Punish=30

[procedure-CheckOff]
PunishMessage=You are not supposed to wear {$zzCheck}.
Punish=30
```

Example 2:

```
[events]
CheckAll=CheckAll

[procedure-CheckAll]
PunishMessage=You are wearing {#zzClothFaults} wrong items.
set#=#punishment,#zzClothFaults
multiply#=#punishment,10
Punish=#punishment
```

Setting flags in cloth reports

You can set flags when the sub makes a cloth report. This gives you the opportunity to code your own checks of what the sub is wearing. The flags are set when the sub makes a cloth report, and removed the next time a cloth report is made. Then of course new flags are set.

Use this in clothtype definitions.

Flag=*flagname*	Tells that you want the flag *flagname* raised when this is selected. If the flag keyword is before the first attribute, it is raised if the cloth type is selected. If the flag keyword comes after a Value keyword, it is raised when this value is selected.

You can also let the program automatically set flags depending on the cloth types the sub is wearing. The flags are the name of the cloth type suffixed with "_on". If clothtype "panties" is chosen, the flag "panties_on" is set.

In the [general] section you add

AutoClothFlags=1	Tells that you want automatic flags for cloth types.

Examples:

```
[clothtype-panties]
   flag=underwear_on
   attr=color
   attr=style
      value=normal
      value=string
          flag=string_on
      value=?
   attr=description
```

If the sub reports wearing string panties, the flags "underwear_on" and "string_on" are set. If the sub reports wearing normal panties, only the flag "underwear_on" is set.

If you have AutoClothFlags=1 in the [general] section, the flag "panties_on" will also be set in both cases.

If the sub reports to be naked, the flag zzNaked will be raised.

Setting flags in instructions and clothing instructions

You can set flags when you give instructions to the sub. This gives you the opportunity to react on the instructions later in the program. The flags are set when you give instructions

to the sub using the Instructions or Clothing keywords. the sub makes a cloth report, and removed the next time instructions are given. Then of course new flags are set.

Note: Flags are not removed and set when the sub asks for instructions. The reason is that the sub can ask for any instructions at any time, and you do not want this to mess with the flags you have set.

Use these keywords in sets:

Flag=*flagname*	Tells that you want the flag *flagname* raised when this set is selected.
OptionFlag=*flagname*	Tells that you want the flag *flagname* raised when this option is selected. Must be coded immediately after the option you want to react to.

Example:

```
[set-pants]
flag=Pants
select=all
Choice=new
        Option=Slacks
                OptionFlag=Slacks
        Option=Jeans
                OptionFlag=Jeans
        Option=Shorts
                OptionFlag=Shorts
```

When the sub is instructed to wear pants, the flag Pants is always set and one of the flags Slacks, Jeans or Shorts is set.

Send email from the program

You can send email from the program in two ways. You can send a message to the master or to the sub when necessary, and you can let the program send reports to the master automatically.

Note: To send mail you need an SMTP server (outgoing mail server). Usually your ISP (Internet Service Provider) offers you the service of an SMTP server. The problem is that, because of the growing spam threat, your ISP will probably have some security requirements. Some of these requirements can be provided by VM, and some cannot. You will have to test it, and there is no guarantee that you will succeed.

To begin with you must have some information for the SMTP server. Your ISP can provide you with this information. All email programs (including Outlook Express and Windows Mail) use the same information, so you should be able to find the relevant information in your email program (except for the password).

What must be there

In the [general] section you must add these keywords:

smtp=*outgoing-mail-server*	The name of the SMTP server or outgoing mail server.
MasterEmail=*email-address*	The email address you want the messages sent to.
SubEmail=*email-address*	The sub's email-address to be used when sending mails to the sub, and as the sender and reply-to address of all mails. Note: Some ISPs require this address to be an address the server knows. In those cases you can't use a webmail address like Hotmail, Yahoo mail or GMail. You may need to use the senderEmail keyword.

If authentication is required

Most ISPs require the email program to authenticate itself. If yours does, you must use the following keywords (still in the [general] section).

smtpUser=*user*	The user name the server expects
smtpPassword=*password*	The password for the user

Optional

masterEmail2=*email-address*	An extra email-address you want all mails sent to. This address will receive a copy of all mails sent to the master.
smtpPort=*port-number*	If your ISP requires you to use a non-standard port-number for outgoing mail, you can write it here.

SenderEmail=*email-address*	If you code SenderEmail, it will become the sender and reply-to address of all mails. If you don't code SenderEmail, SubEmail will be used.
ShowMailWindow=0	Normally the program shows a pop up window while sending the mail. If you code ShowMailWindow=0, the program will not show this window.

How to send a mail to the Master

You can send messages from permissions, reports, confessions, procedures, timers or popups.

MasterMail=*subject*	This keyword will send a message to the master with the subject *subject*. Included in the email will be an extract from the report.
MasterAttach=*filename*	If you want to attach one or more files to the email, use this keyword. Write one MasterAttach keyword for each file you want to attach.

How to send a mail to the sub

You can send messages from permissions, reports, confessions, procedures, timers or popups. You cannot attach files to messages to the sub.

SubMail=*textline*	This keyword will send a message to the sub.

You can have more than one SubMail keyword. Each will become a line in the message. The first line of the message will become the subject of the mail. The mail is sent at the end of the section (report, procedure etc.) where the keyword is used.

Messages to the sub do not include any part of the report.

How to automatically send a created report

You can have the program send the reports automatically when the sub chooses "Make new report file" from the menu, or when the program creates a new report because of MakeNewReport in the Script. See Make a new report.

Do it by adding this keyword to the [general] section:

AutoMailReport=1	This keyword will send a message to the master with the report attached every time a new report is made.

Line writing as jobs or punishments

You can order the sub to write lines. You can do that either as a punishment or a job. The sub will be given a number of lines to write. It can be one line to be written a number of times, or the program can choose between a number of lines.

For jobs, you must define the number of lines to write. For punishments, you can let the program decide based on the value keyword and the severity of the punishment.

Line writing is a punishment that the sub cannot avoid by lying about it.

What must be there

Use these keywords in either a job or a punishment definition. A # in the header or the title is replaced by the number of lines to write.

Type=Lines	This keyword specifies that the job or punishment is a line writing assignment.
Line=*textline*	This keyword defines the line the sub must write. Replace *textline* with the phrase the sub must write. You can have as many lines as you like.

Required in jobs:

Linenumber=*number*	The number of lines to write.

The same or different lines, or a whole text

If you define more than one line, you must tell the program if you want the sub to alternate between the different lines, if you want him to write the same line over and over again or if you want him to write all of the lines in the sequence you have written them.

Select=Random	The program will pick a random line when the job is announced or the punishment is given, and the sub must write that line the specified number of times. Note that this choice will change the name of the punishment or job that is announced to the sub to "Write # times: *textline*" where # is the number of times and *textline* is the line to write.
Select=All	The program will alternate randomly between the possible lines, so the sub will never know which line to write next.
Select=Sequence	The sub must write all of the lines in sequence a number of times. Use this if you want the sub to write a poem or another text. The page size will change to the number of lines in the text. **NOTE**: When you use Select=Sequence, the sub must always write the whole text a number of

times. For jobs the Linenumber keyword will specify the number of repetitions of the text, not the number of lines written. For punishments the Value keyword will specify the value for the whole text, not for a single line. This means that if you have 10 lines in the text, value must be 10 times larger than when you use Select=All or Select=Random.

What should be in the Value parameter

The Value keyword determines how many lines to write. A large value gives a small number of lines. A small value gives a large number of lines.

If you use Select=Sequence, use the value keyword to give the value of on repetition of the whole text.

For other values of select, use the value keyword to give the value of one line.

This means that if you have Select=Sequence and 10 lines at all, the number in the Value parameter must be 10 times larger than if not using Select=Sequence.

Examples:

Line writing job examples:

```
[job-lines1]
Title=Write # lines about your Master
NewStatus=Writing lines
Type=Lines
Linenumber=20
Select=All
Line=I am devoted to my Master.
Line=I love how my Master dresses me.
Line=I have no free will.
Line=I have no secrets for my Master.
Line=Life is good when being controlled.
Line=I hate to disappoint my Master.
Interval=8,16
```

In this example the sub must write 20 lines alternating between the lines.

```
[job-lines2]
NewStatus=Writing lines
Type=Lines
Linenumber=20
Select=Random
Line=I am devoted to my Master.
```

Line=I love how my Master dresses me.
Line=I have no free will.
Line=I have no secrets for my Master.
Line=Life is good when being controlled.
Line=I hate to disappoint my Master.
Interval=8,16

This example the sub must write 10 identical lines. A title is not necessary, as the program will supply one.

Changing the page size

The page size (number of lines per page) is usually 10. You can change that if you wish.

PageSize=*number*	The number of lines per page will be *number*. *Number* can be a counter variable. You can specify two values separated by a comma. Then the program will select a random number in that interval.

If you use Select=Sequence, PageSize is ignored, as it is always the size of the whole text.

If you specify PageSize as a variable or as two numbers, the page size used for the punishment will be determined when the assignment is announced. Once it is in the assignment list, the page size will not change.

Detention as punishment

You can give your sub detention as a punishment. When in detention, the sub must look at a blank computer screen, ready to click a button when it appears. If the sub fails to click the button, the detention is aborted and the sub must start it all over.

Detention is a boring punishment, which the cub cannot avoid by lying about it.

What must be there

Use these keywords in either a job or a punishment definition. A # in the header or the title is replaced by the number of minutes.
Type=Detention This keyword specifies that the punishment is a detention.

Detention is given in minutes. Use value to specify how much one minute counts.

Adding additional text

You can use the Text keyword to add lines that will be displayed on the screen.

Examples:

Example 1:

```
[punishment-Detention for # minutes]
   type=Detention
   value=1
   max=60
   NewStatus=Punishment
```

Example 2:

```
[punishment-Detention for # minutes]
   type=Detention
   value=2
   max=90
   NewStatus=Punishment
   text=Use the time to think about what you have done.
   text=I hope this teaches you a lesson.
```

Punish=50 will give the sub 50 minutes in detention in example 1 (each minute has a value of 1) and 25 minutes in example 2 1 (each minute has a value of 2). In example 1 the detention can be no longer than 60 minutes, in example 2 the maximum is 90 minutes.

Use FTP

If you have an FTP server (File Transfer Protocol), you can use it to let the program deliver reports to the FTP server and to have the program retrieve new versions of the script or the program itself (VirMst3.exe) from the FTP server.

Using FTP requires a bit more knowledge than using email, but you can avoid all of the problems involved when using email.

If you use FTP, the program will connect to the FTP server in these situations:

- When the program starts.
- When a new report file is made, either by the sub or by the script.

When the program is connected to the FTP server, it will perform the actions you have specified. See below.

Each time the program connects to the FTP server, it will also

- Upload a backup copy of the status file.
- If the sub has added, updated or deleted cloth descriptions, it will upload a file containing all the registered cloth items.

What must be there

You must define a special section for the FTP keywords.

[ftp]	This defines the start of the FTP section.
URL=*address*	The address of the FTP server. Often it looks this: ftp.xxxxxx.com
ftpUser=*userid*	The user-id that the program shall use to log onto the server.
ftpPassword=*password*	The password that the program shall use to log onto the server.
ftpServerType=*type*	The type of server you are connecting to. *Type* must be **windows**, **dos**, **unix**, **multinet** or **vms**. Use unix for a Linux server. Usually your server will be Windows or Unix (Linux).

Using a subdirectory

You may not want to keep the files in the root of your server. You can specify a directory path where your files must be placed.

ftpDir=path	The path where the files should be placed.

Automatically update the script and/or the program

VM can automatically retrieve new versions of the script and the program itself from the server. Each time the program connects to the FTP site, it checks if a new version is available. When you have tested a new script, you upload it to the FTP site using a FTP program, and VM will find it and retrieve it to the sub's machine.

UpdateScript=Restart	Tells the program to retrieve new scripts and to restart VM if a new script is found, to make it active immediately.
UpdateScript=Update	Tells the program to retrieve new script, but not restart VM. The script will become active the next time VM is restarted.
UpdateScript=No	Tells the program not to retrieve new scripts. This is the same as not coding the UpdateScript keyword.
UpdateProgram=Restart	Tells the program to retrieve new programs and to restart VM if a new version of the program is found. When retrieving a new program, you must restart. The program cannot be replaced without a restart.
UpdateProgram=No	Tells the program not to retrieve new program versions. This is the same as not coding the UpdateProgram keyword.

Uploading reports

You can let VM upload the report to the FTP server whenever the sub uses the "Make new program" menu or you use MakeNewProgram=1 in the script.

SendReports=1	This will upload all reports to the FTP server.

Other FTP options

FtpLog=1	This will show all FTP communication in the report. This is useful for testing the FTP-connection and identifying problems, but should be turned off when you are using the script for real. Turn it off by changing to FtpLog=0 or removed the line.
TestFtp=0	This will suppress uploading and downloading via FTP unless you run from an encrypted script or have Restrict=1 in the [General] section. This is useful when you are testing other parts of the script.

An FTP example:

```
[ftp]
   URL=ftp.virmst.eu
   ftpUser=myuser
   ftpPassword=mypassword
   ftpDir=/virmst/slaves
   ftpServerType=unix
   UpdateScript=Restart
   UpdateProgram=Restart
   SendReports=1
   ftpLog=0
   testFtp=0
```

Using a web cam

The program can control a web camera. Using a web camera can increase accountability and gives a new way of monitoring the sub when Virtual Master is used in a long distance relationship. The web cam can be controlled with a few but powerful keywords.

You can use email or FTP to send the pictures to yourself.

Use of a web cam requires DirectX 9.0 to be installed.

The program has been tested with a Logitech Quickcam, but I imagine that most cameras can be used.

Monitoring a status

One way to use a web cam is to check randomly that the sub is doing what he/she is supposed to do. If the sub is ordered to stand in the corner, write an essay, scrub the floor or any other activity that occurs in a room with a web cam, you can let the program take pictures at random intervals to make sure that the order is followed.

Use these keywords in a status definition:

CameraInterval=*min,max*	Tells how often the program should take pictures. Specify a minimum and a maximum interval on the form hh:mm:ss or just hh:mm.
PointCamera=*text*	Write a text that directs the sub to point the camera in the desired direction. A live picture will be shown to the sub, to make him/her able to see what the camera is pointing at.

Example:

[status-Standing in the corner]

PointCamera=Point the camera at the corner

CameraInterval=00:03,00:10

This will ask the sub to point the camera at the corner when the status starts. And it will take pictures at random intervals with a minimum of 3 minutes and a maximum of 10 minutes between each picture.

Having the sub pose for the camera

You can have the sub pose for the camera. Use this to verify that the sub is dressed as ordered or for any other reason you wish.

When the sub is ordered to pose, he/she will be shown a live picture from the camera and a countdown from 6 seconds. This gives the sub 6 seconds to pose. When the 6 seconds has passed, a picture is taken. The sub is asked whether the picture is OK. If not, a new countdown is given. This continues until the sub reports that the picture is OK. All pictures will be sent to the master, if automatic send is used. The reason for the short time frame is that the sub will not have time to change clothes.

I suggest you use posing in two ways:

- Let the sub pose after changing clothes or maybe after each cloth report.
- Have popups at random intervals that orders the sub to pose.

This keyword can be used in reports, confessions, permissions, procedures, timers and popups.

PoseCamera=*text*	Orders the sub to pose. *Text* is the order that is shown to the sub.

Example 1:

[report-I have changed clothes]

ClothReport=What are you wearing now?

PoseCamera=Pose for the camera.

Example 2:

[popup-pose]

PoseCamera=Pose for the camera.

How to send pictures

You can send pictures taken with the PoseCamera keyword, as mail attachments or upload them via FTP. Even pictures marked as “Not OK” will be sent. This is to give you as much information as possible.

If you do send emails from the script, all pictures will automatically be attached to the next mail sent.

To upload pictures via FTP, you must add SendPictures=1 to the [FTP] section of the script.

SendPictures=1	If you put this keyword in the [FTP] section of the script, all pictures will be uploaded via FTP next time a connection is made.

Where to save the pictures

You can control where VM saves the pictures by using the keyword CamaraFolder in the [General] section. If you do not code CameraFolder, the pictures will be saved in the same folder as the report files.

CameraFolder=*foldername*	If you put this keyword in the [General] section of the script, all pictures will be saved in the folder you name.

Other options

Playing sounds

You can let the program play sound files using the Sound keyword. The sound files must be in the Wave format (.WAV). You can use the Sound keyword in reports, confessions, permissions, procedures, timers and popups.

sound=*wave file*	The name of a sound file to be played

Showing pictures

Virtual Master has two ways of showing pictures. You can show a picture when in a certain status, or you can open pictures in separate windows.

Only JPG files are supported, and the file extension must be .jpg or .jpeg. Some GIF files will work, and must have an extension of .gif. But there is no guarantee, and a .gif file may cause the program to fail.

Show a picture when in a status

You can show a picture to the right of the text in the main window. The status controls if a picture is shown and which picture. Use the Picture keyword in a status definition.

Picture=*picture file*	The name of an image file. Files with extensions other than .jpg, .jpeg and .gif are ignored. You can have a number of picture keywords for a status. If you have, the program will select a random picture. You can use string variables in the *picture file* name. You can use wild cards (*) in the file name. The program will select a random picture that matches.

Examples:

```
[status-cleaning]
Picture=cleaning1.jpg
Picture=cleaning2.jpg
Picture=cleaning3.gif
Picture=work*.jpg
Picture=c:\images\cleaning\*.jpg
Picture={$picname}
Picture={$picture}.jpg
```

Display a picture in a separate window

You can open a picture in a separate window and close the window again. Use one of the following keywords in reports, confessions, permissions, procedures, timers or popups.

ShowPicture=*filename*	Will open a new window which displays the picture with the name *filename*. You can use string variables in the *picture file* name. You can not use wild cards in the file name.
Select=All	Used before a number of ShowPicture lines, it indicates that all the following pictures will be shown.
Select=Random	Used before a number of ShowPicture lines, it indicates that the program should select a random picture from the following ShowPicture lines.
RemovePicture=*filename*	Will close the window that displays the picture *filename* if it has not already been closed.

Starting files and other programs

You can launch a file or start a program from within VM.

Launch=*filename*	Will open the *filename* using the program that Windows has registered as the program to open this file type.
Launch=*program,parameters*	Will start the program *program* with the parameters *parameters*. You can omit *parameters* if they are not relevant. Parameters can be a file to be opened by *program* or whatever *program* needs.

Writing to the report file

It is possible to write your own lines to the report file. You can do it in reports, confessions, permissions, procedures, timers and popups.

WriteReport=*text*	Writes *text* to the report. Use this if you want to make a note to yourself in the report, or to add an eye-catcher. You can have more than one WriteReport line. If you do, all lines will be shown in the report.

Make a new report file

It is possible to make a new report, just as if the sub had chosen *Make new report file* from the File menu. You can do it in reports, confessions, permissions, procedures, timers and popups.

MakeNewReport=1	Does exactly the same as if the sub had chosen *Make new report file* from the File menu.
MakeNewReport=2	Does the same as MakeNewReport=1, except it will not show a message box to the sub.

If you use the mail facility, you can have the program send the reports automatically when the sub chooses "*Make new report file*" from the menu or when the program creates a new report because of MakeNewReport in the Script.

Do it by adding this keyword to the [general] section:

AutoMailReport=1	This keyword will send an email to the master with the report attached every time a new report is made.

See also Send mails from the program.

The rules menu

The Rules menu, where the sub can see which rules he must obey, is maintained automatically. You can, however, control it in a few ways.

Omitting permissions, reports and instructions from the Rules menu

There may be reports, permissions or instructions that you don't want to appear in the Rules menu.

Rules=0	Use this in a report, permission or instruction to indicate that this report/permission/instruction should not appear in the Rules menu.

Creating your own rules

You can add your own rules to the Rules menu. These rules are only instructions to the sub, they have no effect on how the program functions.

[rule-*name*]	Use this header to define a rule.
Text=*textline*	Use this to write extra text lines to explain the rule to the sub. You can have many lines with the Text keyword.

Omitting the Rules menu completely

If you don't want the Rules menu shown at all, you can specify Rules=0 in the [General] section. This means that the Rules menu will never be shown, not even if you have Rules=1 in a status definition.

Rules=0	Coded in the [General] section, this will hide the Rules menu completely. Rules=1 or not coding the Rules keyword will show the Rules menu.

Disabling the Rules menu in a specific status

If you don't want the Rules menu used in a specific status, you can specify Rules=0 in the status definition. You cannot use Rules in a status definition if you have Rules=0 in the [General] section.

Rules=0	Coded in a status definition, this will disable the Rules menu when in that status. Rules=1 or not coding the Rules keyword will enable the Rules menu.

Minimizing or closing the program

You can minimize or close the program by use of the PgmAction keyword. You can use it in reports, confessions, permissions, procedures, timers and popups.

PgmAction=Close	When this keyword is encountered, the program will close.
PgmAction=Minimize	When this keyword is encountered, the program will be minimized.

Setting the program on standby

You can set the program on standby. When the program is on standby, all deadlines will be pushed ahead. You can use this to allow the sub to go on vacation or close the program for some time for other reasons.

Note that standby does not affect started activities. Long running assignments, statuses with a MinTime or MaxTime, flags with an ExpireProcedure etc. will not be affected.

To set the program on standby, use this keyword. You can use it in reports, confessions, permissions, procedures, timers and popups.

PgmAction=Pause	When this keyword is encountered, the program will close and it will be on standby until it is opened again.

Quick reports

A quick report is a report that the sub can make quickly, either by clicking a large button on the screen or by pressing a function key (F9-F12). You can define one global quick report and up to 2 local quick reports attached to a status. You will automatically get one quick report when you have a started task that is not long running.

The global quick report

The global quick report is defined in the [General] section. You can use it to quickly close the program in case of unexpected guests or for whatever purpose you want. The global quick report can be called by pressing the F12 key.

Add these keywords to the [General] section:

QuickReport=*reportname*	Indicates that *reportname* should be used as the quick report. *Reportname* must be defined as a normal report.
QuickLabel=*text*	Optional. Defines the text shown on the quick report button. If you omit QuickLabel, the name of the report will be used.

The assignment quick report

You will automatically get an assignment quick report button when you have started an assignment that is not long running. The assignment quick report can be called by pressing the F11 key.

You need not do anything. However, you may find that the generated label for the button is too long, and may want to define a shorter one.

Add this keyword to the job or punishment definition:

DoneLabel=*label*	Specifies that the assignment button should be labelled *label.*

Sometimes you may want to hide the assignment quick report button and disable the F11 functionality.

DoneButton=0	This will hide the button and disable F11.

The local quick reports

You can have 1 or 2 local quick reports for each status. The local quick reports are defined in the status sections. Use it in statuses where the sub always or most of the time will be using the same report to end the status.

Add this keyword to the status definition:

QuickReport=*reportname*	Indicates that *reportname* should be used as the quick report. *Reportname* must be defined as a normal report.

If you use EndReport to define a report to end a status, it will automatically become a quick report.

Changing font size

If you wish, you can change the font size of some of the text boxes and buttons. You need to use the [Font] section like this:

[font]

TextSize=*size*	Sets the general font size in the program to *size*. If you do not code this keyword, the size will be 8.
ButtonSize=*size*	Sets the size of the labels of buttons to *size*. If you do not code this keyword, the size will be 8.

The size of menus will follow your Windows setting, and cannot be changed here.

Show buttons that are outside the window

If you have increased the font in Windows, you may find that some buttons are not visible because they are outside the window. Usually this happens to the reset and quick report buttons in the main window and to the buttons on the cloth report. It doesn't help to resize the window, as the buttons move as you resize.

You may be able to increase the window size by using the Border keyword in the [Font] section.

[font]

Border=*size*	Increases the empty space on the right and bottom side of the window with the number of pixels you specified as *size*. Try Border=50 to begin with and increase it if you still can't see the buttons.

Hiding the program

You may need to hide the program on the computer, even when it is active, so that passersby will not see it. Use this if others have access to your computer and you don't want them to know that you are using Virtual Master.

Setting a password on the program

You can set a password on the program, so that every time it is opened it will ask for a password before displaying anything.

Another way of using the password is to make the slave feel more submissive. If he has to enter “i am a slave” every time he starts the program, it may help making him feel more like a slave.

In the [general] section add

OpenPassword=*password*	Sets the password for the program to *password*.

Example:

```
[general]
OpenPassword=i am a slave
```

Minimize to the system tray

When you minimize the program, you may want it to go to the system tray (the icons on the right side of the task bar) and not to the task bar itself. Add these keywords to the [General] section.

UseIcon=1	Minimizes to the system tray and not to the task bar. If you use OpenPassword, the program will ask for the password when it is restored.
StartMinimized=1	If you use this keyword, the program will always start minimized.

How to handle popups when minimized

MinimizePopup=Ignore	If you use this keyword, popups are not possible when the program is minimized.
MinimizePopup=Restore	If you use this keyword, a minimized program will be restored if a popup occurs.

Variables

Variables are used as the program’s "memory". They provide you with a tool to save some information and use it later.

There are four kinds of variables:

- **Flags** are conditions that can be true or false.
- **String variables** are text strings, that is, words and sentences.
- **Counters** are numbers.
- **Time variables** can be a date, a time of day, date+time or an interval (e.g. 10 minutes).

Flags

A flag is a condition, that can be true (the flag is set) or false (the flag is removed). Flags can (like status) be used to control which permissions, reports etc. are possible. They can also be used to control how long a condition is true (e.g. how long the sub may wear panties).

Set and remove flags

You can set and remove flags in reports, confessions, procedures, timers and pop ups. You can also set a flag when a permission is granted.

SetFlag=*flagname*	Sets the flag *flagname* when the status begins. You can specify several flags separated by commas. You can have more than one SetFlag line.
RemoveFlag=*flagname*	Removes the flag *flagname* when the status begins. You can specify several flags separated by commas. You can have more than one SetFlag line.

Use flags

Just like PreStatus, you can use flags to decide when something is possible. Sometimes you can also use them to decide what happens, like if a permission is granted or denied. You can use If and NotIf in permissions, reports, confessions, punishments, procedures, timers, pop-ups and sets. Combined with procedures, you can control almost anything the program can do.

If=*flagname,flagname...*	This is only possible if all the flags in the list are set. You may have more than one If line. If there is more than one line, it is possible if any line is true.
NotIf=*flagname,flagname...*	If all of the flags in the list are set, this is not possible. You may have more than one NotIf line. If there is more than one line, it will be impossible if any line is true.
DenyIf=*flagname,flagname...*	Can be used in permissions. If all the flags in the list are set, the sub is denied permission. There may be more than one DenyIf line. If any line is true then the sub is denied permission.
PermitIf=*flagname,flagname...*	Can be used in permissions. If all the flags in the list are set, permission is given. There may be more than one PermitIf line. If any line is true then permission is given.

Define flag attributes

You need not define flags to use them. You can use the above keywords without having defined the flag specifically. But you can define a flag with some special characteristics (attributes), if you need to.

[flag-*name*]	This line starts a flag definition.
text=*text*	When the flag is set, show *text* in the instruction window along with the status instructions. You can have more than one *text*= line.
Duration=*hh:mm*	When *hh:mm* hours and minutes has passed, the flag is automatically removed. You can add two values, Duration=*hh:mm,hh:mm*, and the program will pick a random value between the two. You can use a time variable instead of *hh:mm*.
ExpireMessage=*text*	If the flag is removed because the duration is up, the message *text* will be shown.
ExpireProcedure=*procedure*	If the flag is removed because the duration is up, the procedure *procedure* will be called.
SetProcedure=*procedure*	When the flag is set, the procedure *procedure* will be called.
RemoveProcedure=*procedure*	When the flag is removed, the procedure *procedure* will be called. The RemoveProcedure is not called when a flag expires. Instead the ExpireProcedure is called.
ReportFlag=0	Normally it will be registered in the report when a flag is set, removed or expired. You can suppress this by using ReportFlag=0.

Using flags to make something happen after a period of time

Sometimes you want something to happen at a specific amount of time after another event. For example you can let a popup order the sub to kneel until he is allowed to rise. It can be done like this:

```
[popup-Kneel]
   message=Kneel down until told otherwise.
   NewSubStatus=Kneeling
   setflag=Kneeling

[status-Kneeling]
   SubStatus=1

[flag-Kneeling]
   text=Stay kneeling until told otherwise.
```

```
   Duration=00:00:05,00:00:10
   ExpireProcedure=StopKneeling

[Procedure-StopKneeling]
   sound=alarm.wav
   message=You may raise now.
   NewStatus=&LastStatus
```

Predefined flags

Some flags are set automatically. They are the names of the day and month: monday, tuesday, january, february etc. Note that it is always the English names that are used. They are not translated. A flag indicating the day of the month is also set, it is called "day*nn*", where *nn* is the day. Examples: day01 is the first day of the month, day15 is the 15th etc. In the report you can see which flags are set automatically when the program starts and after midnight.

Other predefined flags you can use:

zzNaked	is set when the sub has reported to be naked.

String variables

String variables are another kind of memory than flags. Where flags are simply yes/no's, string variables are words to be remembered. You can ask the sub for some text, or you can use some of the predefined variables.

The name of a string variable must start with $.

You can set, input and drop variables flags in reports, confessions, permissions, procedures, timers and pop ups.

Setting a string variable

set$=$*name*,*text*	The value of the variable $*name* is set to *text*.
input$=$*name*,*question*	The sub is asked the question *question*, and the answer is placed in the variable $*name*.

Using a string variable

You can use string variables in texts, messages etc. you write in the script. Simply write the variable name enclosed in { }. Remember the $ sign in the beginning of the name.

You can compare two string variables and use the result as if it were a flag. See Using variables in If and NotIf.

Removing a string variable

drop$=*$name*	Removes $*name* completely.

Examples of string variables

```
input$=$FirstName,What is your first name?
input$=$LastName,What is your last name?
set$=$name,{$FirstName} {$LastName}
message=Hello {$name}.
input=Where do you live, {$name}?
```

Predefined string variables

There are some predefined variables you can use. Predefined variables always start with $zz.

$zzSubname	gives the name of the sub as specified with the Subname keyword in the [General] section. If you have specified

	more than one subname, $zzSubname will choose one at random.
$zzMaster	gives the name of the master as specified with the Master keyword in the [General] section.
$zzDate	gives the date.
$zzTime	gives the time.
$zzReportFile	gives the filename of the last report made.
$zzPVersion	gives the version of the program.
$zzSVersion	gives the version of the script.
$zzName	gives the name of the object*. Name is what is in the header record between [and]. Usually you should not use $zzName in text to the sub. Use $zzTitle instead.
$zzTitle	gives the title of the object*. Title is the name that the sub sees. If there is no Title keyword in the object, the name is used.
$zzReport	gives the name of the last or current report given. Consider using $zzTitle instead of this.
$zzPermission	gives the name of the last or current permission asked. Consider using $zzTitle instead of this.
$zzStatus	gives the name of the current status. Consider using $zzTitle instead of this.
$zzAnswer	gives the name of the last answer from a question
$zzCheck	gives the name of the check that caused a CheckOn or CheckOff event
$zzL	gives a newline. Use it to force a new line in a text.

* An object is a status, report, permission, assignment, job or whatever you are working on right now, where the variable needs to be translated.

<u>These variables are only available when in a relevant context:</u>

$zzAssignment	gives the name of an assignment as the sub sees it.
$zzAssignmentType	gives the type of an assignment. It can be ‘job’ or ‘punishment’
$zzAssignmentId	is the section header for the job or punishment that generated an assignment.
$zzPunishReason	gives the reason for a punishment.

Counters

Counters are used to store numbers. You can do simple calculations on counters.

The name of a counter must start with #.

You can set, calculate on and drop counters flags in reports, confessions, permissions, procedures, timers and pop ups.

Setting a counter

set#=*#counter,value*	Sets the counter *#counter* equal to *value*. *Value* can be a number or another counter.
input#=*#name,question*	The sub is asked the question *question*, where the answer must be a number. The answer is placed in the variable *#name*.

Calculating on counters

set#=*#counter,value*	Sets the counter *#counter* equal to *value*. *Value* can be a number or another counter.
add#=*#counter,value*	Adds *value* to *#counter*. *Value* can be a number or another counter. The result is stored in *#counter*.
subtract#=*#counter,value*	Subtracts *value* from *#counter*. *Value* can be a number or another counter. The result is stored in *#counter*.
multiply#=*#counter,value*	Multiplies *#counter* with *value*. *Value* can be a number or another counter. The result is stored in *#counter*.
divide#=*#counter,value*	Divides *#counter* by *value*. *Value* can be a number or another counter. The result is stored in *#counter*.

Generating a random value

You can assign a random value to a counter.

random#=*#counter,maxvalue*	Sets the counter *#counter* to a random value between 1 and *maxvalue* (both included). *Maxvalue* can be a number or another counter.
random#=*#counter,minvalue,maxvalue*	Sets the counter *#counter* to a random value between *minvalue* and *maxvalue* (both included). *Minvalue and Maxvalue* can be a number or another counter.

Removing a counter

drop#=*#counter* Removes *#counter* completely.

Using a counter

You can use counter variables in texts, messages etc. you write in the script. Simply write the variable name enclosed in { }. Remember the # sign in the beginning of the name.

You can compare two counters or a counter with a number and use the result as if it were a flag. See Using variables in If and NotIf.

Examples of counters

```
set#=#MonthPerYear,12
input#=#age,How old are you?
set#=#month,#age
multiply#=#month,#MonthPerYear
message={#age} years equals {#month} months.
set#=#left,100
subtract#=#left,#age
message=In {#left} years you will be 100 years old.
```

Predefined counters

There are some predefined counters you can use. Predefined counters always start with #zz.

#zzMerits	gives the sub's merit points.
#zzMaxMerits	gives the maximum possible merit number (top of the bar).
#zzMinMerits	gives the minimum merit number (bottom of the bar).
#zzYellow	gives the number of merits below which the bar turns yellow.
#zzRed	gives the number of merits below which the bar turns red.
#zzBlack	gives the same as #zzMinMerits.
#zzBeginMerits	gives the merit points at the beginning of the current day.
#zzDay	gives the day of the month (1-31).
#zzMonth	gives the month (1-12).
#zzYear	gives the year.
#zzLeapYear	gives 1 if you're in a leap year and 0 if not. Add this value to 28, and you have the number of days in February.
#zzHour	gives the hour of the day (0-24).
#zzMinute	gives the minutes (0-59).
#zzSecond	gives the seconds (0-59).
#zzSecondsPassed	gives the seconds passed since midnight.

#zzDaysPassed	gives the number of days passed since 1899-12-31.
#zzClothFaults	gives the number of checks that failed and have caused a CheckAll event.
#zzPunishmentSeverity	Gives the severity of the last punishment given.

<u>These variables are only available when in a relevant context:</u>

#zzPunishmentNumber	Gives the number of iterations, minutes, hours or days for a specific punishment. In other words, what # in the punishment name is translated to.
#zzLate	gives the number of minutes late in a SlowProcedure.
#zzEarly	gives the number of minutes early in a QuickProcedure.
#zzBeforeMerits	gives the sub's merit points before a change. Can only be used in a MeritsChanged event.
#zzAddMerits	In a MeritsChanged event, gives the amount by which the merit points have changed. Or gives the number of merit points given when an assignment is done, a report is made etc. In a punishment gives the number of merit points that will be added when the punishment is done.
#zzSeverity	Gives the severity of a punishment.

Time variables

Time variables are used to store dates, times and time intervals. You can do simple calculations on time variables.

The name of a time variable must start with !.

You can work on variables flags in reports, confessions, permissions, procedures, timers and pop ups.

Time constants

You will also encounter time constants. A time constant can be written in one of the following ways:

- hh:mm
- hh:mm:ss
- nd

where hh is hours in the 24 hour clock, mm is minutes, ss is seconds, n is the number of days and d is the letter "d".

Examples:

- 08:00 8 am or 8 hours.
- 13:00 1 pm or 13 hours.
- 14:30 2:30 pm or 14 hours 30 minutes.
- 00:02 2 minutes.
- 00:02:30 2 minutes 30 seconds.
- 2d 2 days

Setting a time variable

set!=!*variable,value*	Sets the time variable *!variable* equal to *variable2*. V*alue* must be a time constant or another time variable.
inputDate!=!*name,question*	The sub is asked the question *question*, where the answer must be a date. The answer is placed in the variable !*name*. The date entered cannot be lower the current date.
inputTime!=!*name,question*	The sub is asked the question *question*, where the answer must be a time of day without a date. The answer is placed in the variable !*name*.
inputInterval!=!*name,question*	The sub is asked the question *question*, where the answer must be an amount of time. The answer is placed in the variable !*name*. The interval cannot exceed 23 hours 59 minutes and 59 seconds.

Calculating on time variables

set!=!*variable,value*	Sets the time variable *!variable* equal to *value*. V*alue* must be a time constant or another time variable.
add!=!*variable, value*	Adds *value* to *!variable*. V*alue* must be a time constant or another time variable. The result is stored in *!variable*.
subtract!=!*variable, value*	Subtracts *value* from *!variable*. *Value* must be a time constant or another time variable. The result is stored in *!variable*.
multiply!=!*variable, value*	Multiplies *!variable* with *value*. *Value* must be a number or a counter. The result is stored in *!variable*.
divide!=!*variable, value*	Divides *!variable* by *value*. *Value* must be a number or a counter. The result is stored in *!variable*.

Converting a time variable to a counter

days#=#counter,*!timevar*	Extracts a whole number of days from the time variable *!timevar* and puts it in the counter *#counter*. Example: If *!timevar* is 1 day, 2 hours 3 minutes and 4 seconds, *#counter* will be set to 1.
hours#=#counter,*!timevar*	Extracts a whole number of hours from the time variable *!timevar* and puts it in the counter *#counter*. Example: If *!timevar* is 1 day, 2 hours 3 minutes and 4 seconds, *#counter* will be set to 26 (1day+2hours).
minutes#=#counter,*!timevar*	Extracts a whole number of minutes from the time variable *!timevar* and puts it in the counter *#counter*.
seconds#=#counter,*!timevar*	Extracts a whole number of seconds from the time variable *!timevar* and puts it in the counter *#counter*.

Converting a counter to a time variable

days!=!timevar,*#counter*	Sets the time variable *!timevar* to *#counter* days.
hours!=!timevar,*#counter*	Sets the time variable !*timevar* to *#counter* hours.
minutes!=!timevar,*#counter*	Sets the time variable *!timevar* to *#counter* minutes.
seconds!=!timevar,*#counter*	Sets the time variable *!timevar* to *#counter* seconds.

Rounding a time variable

You round a time variable.

round!=!*variable,base*	Rounds the time variable *!variable* to size specified by *base*. *base* must be a time variable or constant.

Examples:

- round!=!hours,01:00 Rounds !hours to a whole number of hours.
- round!=!used,00:10 Rounds !used to a multiply of 10 minutes.

Generating a random value

You can assign a random value to a time variable.

random!=!*variable,!mintime,!maxtime* Sets the time variable *!variable* to a random value between !*mintime* and !*maxtime*. *!mintime and !maxtime* must be time variables or constants.

Removing a time variable

drop!=*!variable* Removes *!variable* completely.

Using a time variable

You can use counter variables in texts, messages etc. you write in the script. Simply write the variable name enclosed in { }. Remember the ! sign in the beginning of the name.

There are different ways of showing a time variable in a text. It can be formatted as a date, a time, date and time or an interval. VM will try to guess how you wished it formatted depending on the size of the variable. However, you may wish to decide the format yourself. You decide the format like this:

- {!variable,d} will always be shown as a date.
- {!variable,t} will always be shown as a time of the day.
- {!variable,i} will always be shown as an interval.

You can compare two time variables or a time variable with a time constant and use the result as if it were a flag. See Using variables in If and NotIf.

Predefined time variables

There are some predefined time variables you can use. Predefined time variables always start with !zz.

!zzDate	gives the date.
!zzTime	gives the time of the day.
!zzDateTime	gives the date and time of the day (same as zzNow).
!zzNow	gives the date and time of the day (same as zzDateTime).
!zzBeginTime	gives the date and time for the first run of this script. If the status file is made by a version prior to version 3.3, it will give the first time version 3.3 was used.
!zzOpenTime	gives date and time when the current run of the program started.

!zzCloseTime	gives date and time the program was last closed. If the program have never been closed, it will give the same time as !zzOpenTime.
!zzResponseTime	gives response time for the last Message or Input statement. Response time is the time from the message or input window was shown and until the sub clicks OK.

These variables are only available when in a relevant context:

!zzMinTime	gives the minimum time of an assignment.
!zzMinTimeEnd	gives the minimum date and time when an assignment to be done (!zzStartTime + !zzMinTime).
!zzMaxTime	gives the maximum time of an assignment.
!zzMaxTimeEnd	gives the maximum date and time when an assignment must be done (!zzStartTime + !zzMaxTime).
!zzInitTime	gives the time when an assignment was created. That is the time a punishment was given or a job was announced.
!zzStartTime	gives the time an assignment is started.
!zzRunTime	gives the time passed since an assignment was started.
!zzDeadline	gives the time when an assignment must be finished.
!zzRemindTime	gives the time when the sub next time will be reminded of an assignment.
!zzLate	gives the time late in a SlowProcedure.
!zzEarly	gives the time early in a QuickProcedure.

These variables for flags are available in in a SetProcedure, RemoveProcedure or ExpireProcedure or immediately after a SetFlag keyword.

!zzDuration	gives the duration of the flag. That is value of the Duration keyword.
!zzExpireTime	gives the time the flag with duration will expire.

Using variables in If and NotIf

You can compare two string variables, counters or time variables and use the result as if it were a flag.

Read more in Use flags. You can also use variables in Case statements. Read more in Conditional execution (case).

You can compare two counters or a counter with a number and use the result as if it were a flag.

If the comparison is true the program will act as if a flag was set. If the comparison is false the program will act as if a flag was not set.

Examples:

```
if=#try<3
if=$answer=yes
if=!AllowedWhen<!zzNow
if=!TimeUzed>00:10:00
DenyIf=#zzmerits<=40
```
(same as DenyBelow=#zzmerits,40)

You can use the following operators:

=	Equal to (for string variables ignoring case)
<>	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

You can use the following extra operators vith string variables only:

==	Exactly equal to (case sensitive)
[	Part of (ignoring case)
[[	Part of (case sensitive)

Examples:

set$=$Master,Master
set$=$line,master
if=$Master=$Line returns true
if=$Master==$Line returns false

set$=$line,Yes master
if=$Master[$Line returns true ($Line contains “master”)
if=$Master[[$Line returns false ($Line does not contain “Master”)
if=$Line[$Master returns false ($Master does not contain “Yes master”)

Divide your script in parts

If you have a very large script, you may want to divide it into different parts, each containing a specific subject. You can do that by using the **%include** keyword. You can add %include anywhere in the script.

%include=*filename*	copies the file named by *filename* into the script at the position of the %include line. %include must always begin a new line.

The file name can be specified with or without a path. If you don't specify a path, the program will look in the folder where the script is located. If you specify a relative path, the program will look for it relative to the folder where the script is located.

Note: If you encrypt a script containing %include, the program will generate one encrypted script containing all the files from the original script. Thus you need only distribute one file to the sub. However, if you chose to run the unencrypted script, all the files must be available to the sub. Otherwise, the sub will not be able to start Virtual Master.

Examples:

If your script is located in the folder C:\VirMst\scripts, then you can use %include like this:

%include cloth.inc	will look for the file C:\VirMst\scripts\cloth.inc.
%include includes\cloth.inc	will look for the file C:\VirMst\scripts\includes\cloth.inc.
%include ..\cloth.inc	will look for the file C:\VirMst\cloth.inc.
%include C:\VirMst\includes\cloth.inc	will look for the file C:\VirMst\includes\cloth.inc.

Conditional execution (case)

Conditional execution is code that is executed sometimes depending on the circumstances. Previously you have learned how to make conditional execution using procedures with If and NotIf. With that method either all of a procedure or none of a procedure is executed. This chapter will describe how you can make conditional execution inside a procedure (or report, timer, popup or anywhere else you can use executable keywords).

You write your code using the keywords Case, When and WhenNot. These are powerful statements, but they are also more complicated to use than most of the keywords.

A case statement looks like this:

```
Case=All   or   Case=First
   When=condition
        statements
   When=condition
        statements
   WhenNot=condition
        statements
   When=All
        statements
   When=NotAll
        statements
   When=Any
        statements
   When=None
        statements
Case=End
```

You always start with Case=All or Case=First. Case=All means that all statements where the *condition* is true are executed. Case=First means that only the statements after the first true *condition* are executed.

Then you have a When=*condition*. The *condition* can be a flag or a comparison. Exactly the same as you can write after an If=. See Using variables in If and NotIf. If the *condition* is true, all executable keywords (here called statements) until the next When is executed. You can have as many statements as you like between two When.

WhenNot=*condition* works like When, except that statements are executed when *condition* is false.

You can have as many When=*condition* and WhenNot=*condition* as you like, each followed by as many statements as you like.

You can add When=All. It means that the following statements will be executed if all the preceding When are true.

You can add When=NotAll. It means that the following statements will be executed if not all of the preceding When are true. It can be that none of them are true, or that some of them, but not all of them, are true.

You can add When=Any. It means that the following statements will be executed if at least one of the preceding When are true.

You can add When=None. It means that the following statements will be executed if none the preceding When are true.

The statements after When=All, When=NotAll, When=Any and When=None is executed even if you started with Case=First.

You must always finish with Case=End. After Case=End, all statements are executed as normal.

Example 1:

```
Case=First
   When=TV
        Message=You were supposed to turn the TV off.
        Punish=50
   When=None
        Message=It's good that you remembered to turn the TV off.
Case=End
```

In this example the sub is punished if the flag TV is on and praised if not.

Example 2:

```
Case=First
   When=#zzmerits>900
        Message=You are doing very well.
   When=#zzmerits>700
        Message=You are doing all right, but there is room for improvement.
   When=#zzmerits>500
        Message=You are doing poorly.
   When=#zzmerits>200
        Message=You need to improve.
   When=None
        Message=You are a useless scum.
Case=End
```

In this example a different message is given depending on the subs merit points.

Example 3:

```
Case=All
    When=#zzmerits<500
    When=HaveBeenBad
    When=#errors>5
    When=Any
        SetFlag=#zzDeny
Case=End
```

This example is meant for a BeforeProcedure for a permission. If any of the listed conditions are true, the permission is denied.

Foreign language translation

The program can be translated to a language of your own choice. To translate do the following steps:

1. Write your script file in your preferred language. Remember not to translate the keywords, only the text that the sub sees may be translated.
2. Choose a name for your language file, where you will do the translations. I suggest your use the language followed by **.txt**. Examples: french.txt, francais.txt, german.txt, deutsch.txt.
3. In the [general] section of your script, add the line *language=filename* (where *filename* is the name you chose).
4. Run the program. It will still be in English. Try as many different situations as possible.
5. Close the program.
6. Edit your language file. You will find only one section with the name [missing].
7. Change [missing] to [language].
8. Change the text on the right side of the equal sign (=) to your own language.
9. When you have finished translating, run the program. You will find that it is translated. If the program discovers a new phrase, that hasn't been translated, it will add a new [missing] section to the language file and add the new phrases. Your job is to translate the phrase and move the lines from the [missing] to the [language] section. You can leave the empty [missing] section if you want.

Notes:

- There may only be one [language] section in the language file. So you only change the name once.
- You will notice an ampersand (&) in many of the texts that are menu entries and button labels. This indicates that the letter after the ampersand (&) will be underlined and used as a windows hot key (used with the Alt key). Move the ampersands to the suitable letters in your language.
- Sometimes I add new functionality and forget to call the language module,which means that some phrases will not be translated. If you find some phrases that are not translated, email me, and I will correct the mistake.
- I know that I have a problem with Yes and No buttons, which are not translated. There's no need to write me about these. Maybe someday in the future I will solve this problem.

Advice for self scripters

If you are making a script for yourself, there are things you can do to make the script more interesting.

You can use Restrict=1 in the [General] section. This will remove information that you don't want to know, from the report. See Prevent cheating.

For many keywords you can specify two values and let the program chose a random value. Thus not knowing which value will be used. If you add the keyword CenterRandom to the [General] section, the program will mainly chose from the middle of the range, rarely from the upper and lower ends. This way you can specify minimum and maximum values close to your limits. You will then know that they are possible if the program is in a "bad mode" or "god mood", but they will not be chosen all the time. See Making extreme values possible.

Compability with older versions

Version 3.3 and earlier versions didn't check if there were misplaced or misspelled or otherwise invalid keywords in the script. Version 3.4 does. However to be able to use older scripts, these checks are enabled or disabled depending on the script.

In the [General] section these keywords will be used to control whether the program validates all keywords:

ValidateAll=1	The program will fail to load the script if it contains misplaced or misspelled keywords.
ValidateAll=0	The program will ignore misplaced or misspelled keywords. This is the behaviour of version 3.3 or earlier.

If the ValidateAll keyword is not present, the program will use the MinVersion keyword to determine the behaviour.

MinVersion=3.4 or higher	The program will fail to load the script if it contains misplaced or misspelled keywords.
MinVersion=3.3 or lower	The program will ignore misplaced or misspelled keywords. This is the behaviour of version 3.3 or earlier.

Testing your script

The test menu

You can add a Test menu to the menu line, by adding the keyword TestMenu=1 to the [General] section. The Test menu will help you test your scripts.

If you run from an encrypted script or if you have Restrict=1 in the [General] section, the test menu will always be hidden.

NOTE: When you test a script, always use a different folder from when you run the program, if you make scripts for yourself. Once you have manipulated the time, you cannot turn it back without deleting the status file.

TestMenu=1	Adds the Test menu to the menu line, unless you run from an encrypted script or have Restrict=1.

The test menu

Add	Simulate that time passes. Note: This simulation does not work for popups. To test popups, use Quick popups (see below). It will work for almost anything else.
Randomize	If this is checked, all instructions and clothing instructions will generate a new result, even if you have Change=Daily or Change=Program.
Quick popups	Will make popups popup every 5 seconds, if you are in a status that allows popups. Use this to test your popups.
Always permit	Permissions will always be permitted.
Always deny	Permissions will always be denied.
Change status	Can be used to change to any status possible.
Launch job	Will add a specific job to the assignment list, thus giving you the possibility to test it.
Give punishment	Let you test a punishment.
Export clothes	Export the sub's entered clothes to a file.
Import clothes	Import clothes from a file made by Export clothes.

Avoiding Internet connection when testing

You may not be interested in sending email or using FTP when testing your script.

Add the line
TestMail=0
to the [General] section of your script. This will suppress sending of email unless you run from an encrypted script or have Restrict=1 in the [General] section.

Add the line
TestFtp=0
to the [FTP] section of your script. This will suppress uploading and downloading via FTP unless you run from an encrypted script or have Restrict=1 in the [General] section.

Avoiding entering the open password when testing

You may not be interested in typing the open password when testing your script. Add the lineTestPassword=0
to the [General] section of your script. This will ignore the openpassword unless you run from an encrypted script or have Restrict=1 in the [General] section.

Making reports only for test usage

You can make reports that only works when not in sub mode.

Add the line
Test=1
to the report. This will ignore the report if you run from an encrypted script or have Restrict=1 in the [General] section.

Debug help

This is not for common use. In the event you have problems with the status file, you may be asked to supply some statistical information about it to help me find the problem. In that case, add the line

Statistics=1

to the [General] section of your script. This will give you a new menu entry call Statistics.

Trouble shooting

This chapter contains description of some common problems. The chapter is still very new, but will be expanded as I find it necessary.

I cannot see the buttons on the Cloth Report form

This can happen if you have set Windows up to use a larger font than the default font. You may be able to solve the problem by using the Border keyword in the [Font] section.

See Show buttons that are outside the window.

What's new

What's new in 3.1

A lot! Version 3 has a complete rewrite of syntax compared to versions 2 and 1. Thus it is not possible to use scripts made for versions 1 or 2 with version 3 of the program. Because of the great changes, I have not made a list of what is changed.

What's new in 3.2

- Use of FTP.
- Use of a webcam.
- Line writing as a job or punishment.
- QuickReport in status definitions. Automatic quick reports for assignments.
- Weight can be used in punishments and popups.
- You can define your own rules to show in the Rules menu.
- You can now use Latest and Earliest in permissions, confessions and procedures.
- You can now use special flags in a BeforeProcedure on a permission to tell the program whether to permit, allow or cancel the permission. The special flags are zzDeny, zzPermit and zzCancel.
- New keyword for status and popup groups:
  - PopupAlarm
- New keywords for jobs:
  - FirstInterval to make first interval shorter than the normal interval.
  - OneTime to make the job run only once.
  - AnnounceProcedure is called when the job is announced.
- New keywords for permissions:
  - ForgetProcedure
  - IgnoreProcedure
  - ForgetConfession=0 and IgnoreConfession=0 in the [General] section-
- New keyword for punishments
  - AnnounceProcedure is called when the punishment is given.
  - MaxSeverity. Actually not new, it has just not been documented.
- New keywords for instructions and clothing instructions:
  - Rules to control whether the instructions are shown in the Rules menu.

- Askable to control whether the sub can ask for instructions.

- New keyword for flags:
  - ReportFlag=0 to suppress information about flags in the report.
- Use %include to divide your script into parts.
- Text=%instructions and Text=%clothing will show latest (clothing) instructions in the message area.
- New events:
  - StartFromPause
  - JobAnnounced
  - ForgetConfession
  - IgnoreConfession
  - MeritsChanged
- Variable improvements:
  - Time variables
  - A new Random# keyword to generate random values.
  - New string operators: ==, [and [[.
  - New standard variables.
- Several keywords have been changed so you can specify one or two values. If you specify two values the program will choose a random value between the two. One or both values can be replaced by variables.
- You can now display pictures in two ways. As before you can use the Picture keyword in a status definition,or you can open and close windows with the ShowPicture and RemovePicture keywords.
- The restriction that a procedure cannot be called twice has been changed a bit. A procedure can still not call itself, but a procedure can call another procedure twice. If it happens, you will now get a message in the report.
- Test menu improvement:
  - Change status
  - Launch job
  - Give punishment

About Foreign language translation:

The way that foreign language translation works internally has been changed between 3.1 and 3.2. This means that you may have to retranslate some phrases. The reason for this change is to make it easier for myself to maintain the foreign language facility in the future.

What's new in 3.3

- New chapter with advice for self scriptors.
- Unfinished punishments now decrease the maximum possible merit number. See Setting the limits.
- When you can specify two values in a keyword to let the program choose a random value, a new keyword, CenterRandom, allows the program to use the values in the middle very often and the values near the minimum and maximum rarely. See Making extreme values possible.
- New punishment type: Detention.
- New keyword, Select=Sequence, for line writing assignments to allow you to specify a text the sub has to write.
- New keyword, PageSize, for line writing assignments allows you to set the number of lines pr. page.
- More use of variables in keywords. See the description of the individual keywords.
- Converting time variables to counters (days#, hours#, minutes# and seconds#).
- Converting counters to time variables (days!, hours!, minutes! and seconds!).
- inputDate!, inputTime! and inputInterval! to ask the user for a date, a time or an interval.
- Variables zzStartTime and zzRunTime now works in a DoneProcedure and AbortProcedure.
- New predefined variables:
  - $zzL contains a newline. Use it to force a new line in a text.
  - $zzMonth contains the name of the current month.
  - $zzDayOfWeek contains the day of the week (Monday etc.).
  - #zzDayOfWeek contains the day of the week (1=Sunday, 2=Monday etc.).
  - !zzBeginTime, !zzCloseTime and !zzOpenTime gives times when the program have started or ended.
  - !zzResponseTime gives the subs response time for a Message or Input statement.
  - !zzDuration gives the duration of a flag.

- !zzExpireTime gives the time a flag with duration will expire.
- #zzAssignments contains the number of assignments in the assignment list.
- #zzJobs contains the number of jobs in the assignment list.
- #zzPunishments contains the number of punishments in the assignment list.
- #zzAssignmentsLate contains the number of assignments where the deadline have been passed.
- #zzJobsLate contains the number of jobs where the deadline have been passed.
- #zzPunishmentsLate contains the number of punishments where the deadline have been passed.
- # zzSeverity contains the severity of a punishment.
- !zzInitTime gives the time when an assignment was created. That is the time a punishment was given or a job was announced.
- $zzAssignment gives the name of an assignment as the sub sees it.
- $zzAssignmentType gives the type of an assignment. It can be ‘job’ or ‘punishment’.
- $zzAssignmentId is the section header for the job or punishment that generated an assignment.
- $zzPunishReason gives the reason for a punishment.
- The flag zzNaked is raised when the sub reports to be naked.

- New events:
  - AfterReport
  - PermissionGiven
  - PermissionDenied
  - BeforeDelete
- New keyword for jobs and punishments:
  - BeforeDeleteProcedure. Use it to prevent deletion of assignments.
- Rules=0 keyword in the [General] section hides the Rules menu.
- Rules=0 keyword in a status definition disables the Rules menu when the sub is in that status.
- Improvements when Restrict=1 is used.
- Notify=2 in permissions notifies the sub without telling the time to wait.
- Keyword CameraFolder in the [General] section lets you control where pictures are saved.

- Keyword Away in status definitions tells that the sub is not home. This is not a new feature, but it has not been documented until now.
- Everytime the program starts, a backup copy of the status file is made in the file vm3stat.vms.bak or vm3stat.vmt.bak.
- The size of the status file is reduced by removing obsolete data.
- If you use FTP, the program will automatic make a backup copy of the status file and the cloth list to the FTP server.
- The sub can now add cloth types.
- The sub can no longer cancel a cloth report which the script demands. Only if the sub makes a cloth report via the menu *Communications*, it can be canceled.
- You can now use MarkDone in a MinTimeProcedure.
- You can now send mails to the sub.
- PopupMinTime now works for timers.

What's new in 3.4

- **Important!** The program now validates that only valid keywords are used in the script. See Compability with older versions.
- Conditional execution with the Case keyword. See Conditional execution (case).
- Improved questions to allow inline action instead of calling procedures. See Advanced questions: how to define them.
- BeforeDoneProcedure for jobs and punishments. See Assignment keywords (both jobs and punishments).
- You can prevent an assignment from being started or completed by setting the flag zzDeny in a BeforeProcedure or BeforeDoneProcedure. See Preventing an assignment from being started or marked done.
- New event: NewDay. See Other events.
- The camera window is now only shown when needed.
- You can now use variables in the keywords Respite and Estimate.
- A lot of bug fixes.