PSR

Registration Shuffler

Design: File Formats

Version 1.0

October 7, 2011

Dennis Schulmeister
dennis@developer-showcase.de

Contents

1 Internal file formats

1.1 Registration files for single registrations

1.1.1 Version 1: Keyboard model and registration

Most arranger workstations don't allow to save single registrations into a file. Instead registrations are grouped to fixed-size banks which can be saved and loaded. The newer models save one bank per file but there are still older models which can only save all banks into one file. The purpose of the PSR Registration Shuffler is to import those files and extract the single registrations out of them. This way a data pool is built which can be used to compose new bank files to be loaded into the instrument.

Each extracted registration is stored into a separate file which is totally managed by the program. The user usually doesn't use these files directly. The first version of the file format is very simple as it only contains three fields, including the magic number, the keyboard model and a binary part with the extracted registration data. The magic number is checked in order to recognize registration files. The keyboard model is checked in order to detect the file format of the resulting bank files. The rest of the file contains just the binary registration data which is copied back into the bank files. Typically the file extension is `*.regfile`.

The file structure goes like this:

Position$_{16}$	**Amount**	**Length**	**Content**
00 00 00 00	1	4	Magic number: `RS01`
00 00 00 04	1	16	Keyboard model, e.g. `YAMAHA PSR2000`. Unused bytes at the end are filled with `0x00` bytes.
00 00 00 14	1	variable	Binary registration data

1.1.2 Version 2: Tree structure and additional meta data

Version two extends the file format with user-editable meta data. Therefor the content is organized into a hierarchical tree making great use of block identifiers and length fields. The new file format looks like this:

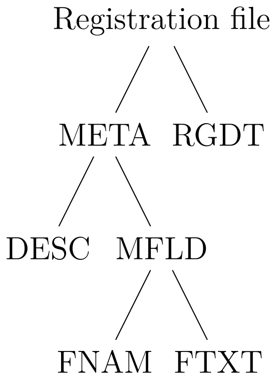
Position$_{16}$	**Amount**	**Length**	**Content**
00 00 00 00	1	4	Magic number: `RS02`
00 00 00 04	1	16	Keyboard model, e.g. `YAMAHA TYROS1`. Unused bytes at the end are filled with `0x00` bytes.
00 00 00 14	0–1	variable	Optional meta data block
. . .	1	variable	Registration data block

Each block consists of three parts: An identifier, a length field and the block content:

Position$_{16}$	**Amount**	**Length**	**Content**
00 00 00 00	1	4	Block identifier. Should be ascii only
00 00 00 04	1	4	Length of the following data
00 00 00 08	1	variable	Block data

Depending on the block type the block data may consist of an arbitrary byte sequence or of more blocks which build up a tree structure. Currently the following block types are supported:

`META`	Meta Data	optional
`DESC`	Description	optional
`MFLD`	Meta Data Field	optional
`FNAM`	Field Name	required
`FTXT`	Field Text	required
`RGDT`	Registration Data	required



As can be seen the `Meta Data` block may contain an optional `Description` block and any number of `Meta Data Fields`. Each `Meta Data Field` consists of a `Field Name` and a `Field Text`. The `Description` block contains latin-1 encoded text which may have line-breaks. The blocks `Field Name` and `Field Text` also contain latin-1 encoded text but without newlines. The `Registration` block contains the raw binary data of a registration.

Here is a complete example:

```
52 53 30 32 59 41 4d 41  48 41 20 39 30 30 30 70  RS02YAMAHA 9000p
72 6f 00 00 4d 45 54 41  00 00 00 1b 44 45 53 43  ro..META....DESC
00 00 00 13 54 68 69 73  20 69 73 20 61 6e 20 65  ....This is an e
78 61 6d 70 6c 65 21 52  47 44 54 00 00 01 d9 50  xample!RGDT....P
75 6e 69 73 68 20 74 68  65 20 6d 6f 6e 6b 65 79  unish the monkey
21 2e 00 40 40 62 70 ...                          !..@@bp.........
```

Position$_{16}$	**Amount**	**Length**	**Content**
00 00 00 00	1	4	Magic number: `RS02`
00 00 00 04	1	16	Keyboard model: `YAMAHA 9000pro.`
00 00 00 14	1	4	Meta Data block: `META`
00 00 00 18	1	4	Length of meta data: `0x1B`
00 00 00 1C	1	4	Description block: `DESC`
00 00 00 20	1	4	Length of description: `0x13`
00 00 00 24	1	19	Description: `This is an example!`
00 00 00 37	1	477	Registration Data block: `RGDT`
00 00 00 3B	1	4	Length of registration data: `0x1D9`
00 00 00 3F	1	473	Registration data

2 Registration banks

2.1 Yamaha PSR-9000 and 9000pro

2.1.1 General notes

All contents of the Flash ROM (including registration banks) can either be saved to disk using the Load/Save or the System Backup function. According to the 9000pro manual all files created with Load/Save can be exchanged between PSR-9000 and 9000pro. System Backups however don't allow to share system settings and registrations between those two models. At the moment no research has been done regarding the differences between the created files of both instruments.

Unlike following models like the PSR-2000 contents of the Flash ROM are not presented in an object-oriented directory/file manner. This means that single objects like registration banks or styles are not treated like files on a disk and thus cannot simply be copied or moved. Instead a functional approach is used where the Function menu contains several options in order to perform tasks like "copy from flash to disk" upon numbered slots within the Flash ROM. (Examples for numbered slots in that sense: Registration Bank 1–64, Flash Style 1–n, User Voice 1–32, . . .)

All text files use a latin-1 enconding and CR LF as line ends. All numbers are unsigned integers in big endian order.

2.1.2 Files created with Load/Save User Data

Directory tree and contained files

When user data is saved it is possible to choose which type of data should be saved, e.g. if only registrations or if registrations and custom voices are to be saved. Acordingly when this data is loaded it is possible to choose which content should be restored. Though all registration banks are stored into one single file it is possible to load either all registration banks (Group) or only selected banks (Individual) back into the instrument.

All contents are stored in a file-system directory which ends in `.usr`, though it is not allowed to see the contained files of that directory. Inside that directory there is a control file called `USERFILE.INI` and at least one of the following data files for Registrations, Multi Pads and so on.

`Regist.reg`	Registration Banks
`*.vic`	Custom Voices
`*.org`	Organ Voices
`*.pad`	Multi Pad Banks
`*.set`	Global Settings
`*.eff`	Effect Settings

> No error message occurs if a user data directory contains more than one file with an `*.reg` extension or if the file isn't called `Regist.reg`. In most cases the instrument tries to load the data but doesn't find it.

Control file USERFILE.INI

The text file `USERFILE.INI` is structured similar to many configuration files in `INI` format and describes the content of a backup. The file is written by the instrument but it's not needed in order to load a backup as long as the data files reside in a directory with a `*.usr` extension.

The general structure of the file goes like this. There are no empty lines inside that file. Empty lines here are only for readability.

Header

```
[TITLE]
9000Pro USERFILE.INI
YAMAHA Corporation
[DISK NO]
DISK000
[INSTRUMENT]
9000Pro
[VERSION]
Ver2.06
[TOTAL USER DATA SIZE]
4712KB
```

Content

...

Footer

```
[DATAEND]
```

The name of the instrument can be found twice. Allowed values are `PSR-9000` and `9000pro`. The disk number is probably a hexadecimal number which is used if the backup doesn't fit to one disk. Interestingly the data size doesn't exactly fit the total file sizes. Maybe it is used for display purposes only. Also there is the OS version with which the backup has been created. It is assumed however that most of these fields are not used.

Not shown here are the content blocks in the middle of the file. They contain exactly one block per data type and enumerate all contained files of a backup. This is, all file names are printed in a numerated list. The only exception to that rule are Registration Banks which are described like any other file but are not saved into single files. Instead there is only one file called `Regist.reg` which contains all banks. Here are some examples, again there are no empty lines:

```
[ORGAN FLUTE]
TOTAL FILE NUM:4
1 = DF0001          00.org
2 = DF0002          01.org
3 = DF3             02.org
4 = DF accomp       03.org

[REGISTRATION]
TOTAL FILE NUM:3
1 = A               00.reg
2 = A               01.reg
3 = A               02.reg

[MULTI PAD]
TOTAL FILE NUM:5
1 = Live! Tom       00.pad
2 = Live! Crash     01.pad
3 = Live! Kit 1     02.pad
4 = Live! Kit 2     03.pad
5 = Live! Kit 3     04.pad

[CUSTOM VOICE]
TOTAL FILE NUM:2
1 = Handel Orch     00.vic
2 = DF Lead 1       01.vic

[SETUP]
```

```
TOTAL FILE NUM:1
setup           00.set

[EFFECT]
TOTAL FILE NUM:1
effect          00.eff
```

Note how file numbers are stored redundant. Each file of a group is numbered consecutively at the beginning of each line. However the same number is also stored as part of the file name. Numbers at the beginning are decimal, numbers in the file name are hexadecimal. Most file names are exactly 20 chars wide not counting the extension. Spaces are used to move the file number to the end of each name. More important the field `TOTAL FILE NUM` always contains the exact amount of files in a group or in case of registrations the amount of Registration Banks.

Binary file Regist.reg

Unlike the other file types registration banks are not stored individually in `*.reg` files even if `USERFILE.INI` suggests that. This simplification hasn't been introduced to the firmware before the next model, the PSR-2000. Prior to that all registration banks were saved into a single file called `Regist.reg`. The general layout is quite simple:

Position$_{16}$	**Amount**	**Length**	**Content**
00 00 00 00	64	48	Index of all contained registration banks. This is always 3072 bytes long and `0x00` bytes are used to fill the list if it is shorter. Entries can be in any order but there can be no gaps between them.
00 00 0C 00	1	16	Padding 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 0C 10	Up to 64	variable	Registration banks

Index entries have the following layout. However the very first two bytes must always be `0xD0 06` even if the first bank is missing.

Position$_{16}$	**Amount**	**Length**	**Content**
00 00 00 00	1	16	Header of the first entry: D0 06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 Header of all other entries: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Position$_{16}$	Amount	Length	Content
00 00 00 10	1	4	Size of registration bank, e.g. 0x12 68
00 00 00 14	1	4	Absolute position of the bank within the file. The position is off by 0x10 which must be added in order to find the bank.
00 00 00 18	1	1	Bank number from 0 to 63.
00 00 00 19	1	22	Bank name. The name always ends in .reg. Spaces 0x20 are used to move the extension to the very end. Otherwise a zero byte indicates the end of the string.
00 00 00 2F	1	1	Final 0x00 byte.

Each bank has the following structure:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	16	Bank name. Filled with 0x00 or spaces at the end
00 00 00 10	1	32	Identification string. Padded with spaces 0x20 at the end: PSR-9000PREGIST Ver1.00
00 00 00 30	0–8	583	Registrations

The registrations seem to follow a flat structure instead of a hierarchical tree. They have the following structure:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	6	Identification string: REG000, ..., REG007
00 00 00 06	1	4	Length of the following data: 0x00 00 02 3D Add 10 in order to get the complete length of the bank including all fields.
00 00 00 0A	1	6	Unknown: 0x08 01 00 00 00 00
00 00 00 10	1	16	Registration name. A zero byte indicates the end if less than 16 characters are used. The other bytes may then contain garbage.
00 00 00 10	1	573	Other registration data

Empty or missing registrations may either be all zero or may be missing. If they are missing the bank size changes accordingly. However neither way is a good idea because the firmware doesn't support missing registrations like the newer models do. A missing registration can still be called like any other registration but all settings are zeroed, then. This is, all volumes are zero, all pan-levels are zero, touch response of all voices is off and even the master scale is set to Arabic tuning. Many settings have to be changed to make the instrument sound right again. A better solution is to save a registration with sane default settings, like Yamaha does with the factory backup disk. Such a registration would basically contain the initial settings which are active after powering the arranger on but with all panel voices off.

Here is an example for PSR-9000:

```
52 45 47 30 30 30 00 00  02 3d 08 01 00 00 00 00  REG000...=......
20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20
00 00 00 08 03 36 00 78  00 00 52 ff 00 64 5a 66  .....6.x..R..dZf
2d 48 26 2e 32 00 40 40  40 2a 58 40 4c 3a 00 14  -H&.2.@@@*X@L:..
20 00 28 32 3c 26 00 00  00 00 12 22 40 00 2e 00   .(2<&....."@...
00 00 00 00 00 14 00 1e  7f 00 7f 00 7d 7f 00 7d  ............}..}
00 70 21 00 71 19 00 74  1b 00 70 30 00 76 1b 00  .p!.q..t..p0.v..
70 12 00 00 03 00 0a 00  00 70 00 00 6e 64 40 00  p........p..nd@.
00 24 02 00 40 00 00 00  00 75 31 00 6e 64 40 00  .$..@....u1.nd@.
00 24 02 00 40 00 00 00  00 72 04 00 6e 64 40 18  .$..@....r..nd@.
00 24 02 00 40 00 00 01  40 00 07 00 00 00 00 07  .$..@...@.......
07 07 00 00 00 00 00 01  10 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 40 42 08 00 00 00  ..........@B....
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 40  ...............@
00 00 00 00 00 00 00 00  00 03 11 00 24 00 01 11  ............$...
01 2e 00 42 10 02 22 7f  46 40 26 00 15 00 1e 04  ...B..".F@&.....
59 00 10 00 00 40 00 00  00 00 40 40 40 40 40 40  Y....@....@@@@@@
40 40 40 40 40 40 00 00  05 07 0f 02 00 01 48 00  @@@@@@........H.
2a 32 40 40 40 40 40 40  00 40 40 40 42 3c 40 3a  *2@@@@@@.@@@B<@:
40 00 40 40 40 40 3a 2c  3e 40 00 40 40 40 40 40  @.@@@@:,>@.@@@@@
40 40 40 00 40 40 40 40  40 40 40 40 00 01 00 00  @@@.@@@@@@@@....
00 12 40 40 44 40 7f 00  01 00 00 00 7f 00 00 00  ..@@D@..........
00 40 40 40 40 00 00 00  22 40 40 40 40 00 00 76  .@@@@..."@@@@..v
12 00 00 64 40 1c 00 7f  02 00 40 00 ff 00 00 00  ...d@.....@.....
00 40 40 28 40 01 01 01  7f 7f 7f 00 63 01 7f 00  .@@(@.......c...
00 00 00 01 7f 01 00 00  00 00 00 30 ff 00 07 00  ...........0....
31 64 05 0f 00 24 64 5f  00 01 07 52 40 40 40 40  1d...$d_...R@@@@
40 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  @...............
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 8f 00 00  00 8f 00 00 00 00 00 00  ................
00 ff 00 00 00 ff 00 00  00 ff 00 00 00 ff 00 00  ................
00 03 00 00 00 00 00 4e  00 00 00 00 00 00 00 00  .......N........
00 a9 00 29 00 00 00 00  00 00 00 00 7f 7f 00 00  ...)............
80 b7 b6 80 00 c0 00                              .......
```

Here is an example for 9000pro. This changes the `GrandPiano` voice to `Live!Grand`:

```
52 45 47 30 30 30 00 00  02 3d 08 01 00 00 00 00  REG000...=......
20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20
00 00 00 08 03 36 00 78  00 00 52 ff 00 64 5a 66  .....6.x..R..dZf
2d 48 26 2e 32 00 40 40  40 2a 58 40 4c 3a 00 14  -H&.2.@@@*X@L:..
20 00 28 32 3c 26 00 00  00 00 12 22 40 00 2e 00   .(2<&....."@...
00 00 00 00 00 14 00 1e  7f 00 7f 00 7d 7f 00 7d  ............}..}
00 70 21 00 71 19 00 74  1b 00 70 30 00 76 1b 00  .p!.q..t..p0.v..
70 12 00 00 03 00 0a 00  00 71 00 00 6e 64 40 00  p........q..nd@.
00 24 02 00 40 00 00 00  00 75 31 00 6e 64 40 00  .$..@....u1.nd@.
00 24 02 00 40 00 00 00  00 72 04 00 6e 64 40 18  .$..@....r..nd@.
00 24 02 00 40 00 00 01  40 00 07 00 00 00 00 07  .$..@...@.......
07 07 00 00 00 00 00 01  10 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 40 42 08 00 00 00  ..........@B....
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 40  ...............@
00 00 00 00 00 00 00 00  00 03 11 00 22 00 01 11  ............"...
01 2e 00 42 10 02 22 7f  46 40 26 00 15 00 1e 04  ...B..".F@&.....
59 00 10 00 00 40 00 00  00 00 40 40 40 40 40 40  Y....@....@@@@@@
40 40 40 40 40 40 00 00  05 07 0f 02 00 01 48 00  @@@@@@........H.
2a 32 40 40 40 40 40 40  00 40 40 40 42 3c 40 3a  *2@@@@@@.@@@B<@:
40 00 40 40 40 40 3a 2c  3e 40 00 40 40 40 40 40  @.@@@@:,>@.@@@@@
40 40 40 00 40 40 40 40  40 40 40 40 00 01 00 00  @@@.@@@@@@@@....
00 12 40 40 44 40 7f 00  01 00 00 00 7f 00 00 00  ..@@D@..........
00 40 40 6c 60 00 00 00  22 40 40 40 40 00 00 76  .@@l‘..."@@@@..v
12 00 00 64 40 1c 00 7f  02 00 40 00 ff 00 00 00  ...d@.....@.....
00 40 40 28 40 01 01 01  7f 7f 7f 00 63 01 7f 00  .@@(@.......c...
00 00 00 01 7f 01 00 00  00 00 00 30 ff 00 07 00  ...........0....
31 64 05 0f 00 24 64 5f  00 01 07 52 40 40 40 40  1d...$d_...R@@@@
40 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  @...............
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 8f 00 00  00 8f 00 00 00 00 00 00  ................
00 ff 00 00 00 ff 00 00  00 ff 00 00 00 ff 00 00  ................
00 03 00 00 00 00 00 4e  00 00 00 00 00 00 00 00  .......N........
00 a9 00 29 00 00 00 00  00 00 00 00 7f 7f 00 00  ...)............
80 b7 b6 80 00 c0 00                              .......
```

2.1.3 Files created with System Backup

Directory tree and contained files

Each backup is stored to one ore more disks and each floppy disk may not contain more than one backup. For that reason each disk contains a folder called `Setup.bXX` where **XX** is the hexadecimal disk number. If a backup fits to one disk there is only one directory called `Setup.b01`. If it takes two disks the directories are called `Setup.b01` and `Setup.b02` and so on.

Similar to user data each backup is stored inside a directory whose name ends in `.buf` and the contents of that directory cannot be seen on the instrument. If a backup is saved to disk it may be splitted to two disks. In that case the extension changes to `*.b01` for the first disk and `*.b02` for the second disk. In order to prevent mistakes the file listings of the backup function don't show directories with other extensions than `*.buf` and `*.b01`.

Each of these directories contains a control file called `BACKUP.INI` which is really needed in order to load the backup. Additionally at least one of the following data files must be present:

`STY.b01`, `STY.bFF`	Flash Styles
`SUP.bup`	Global Setup
`MDB.bup`	Music Database
`MPD.bup`	Multi Pads
`OTS.bup`	One Touch Settings
`REG.bup`	Registration Banks

If the control file is missing the instrument refuses to open the backup directory. If the content of the file is bogus or doesn't meet the exact expectations of the firmware the backup is shown as if it was empty.

Control file BACKUP.INI

The control file `BACKUP.INI` is a simple text file very similar to the `USERFILE.INI` above. In general it follows the same rules, albeit sometimes not in a consistent way.

Header

```
[TITLE]
PSR-9000 BACKUP.INI
YAMAHA Corporation
[DISK NO]
DISK001
[INSTRUMENT]
PSR-9000
[VERSION]
Ver1.12
[TOTAL USER DATA SIZE]
2770276KB
```

Content

...

Footer

```
[DATAEND]
```

Again the keyboard model (`PSR-9000` or `9000pro`) is mentioned twice and also the OS version, disk number and data size are present. Disks are numbered hexadecimal with the first being `DISK000` and the last disk being `DISKFFF`. However backups may not span more than two disks due to the limited size of the backed up Flash ROM. This could be confirmed with a hex-editor. The strings `DISK000` and `DISKFFF` are the only strings which are hard-coded into the firmware.

The following example shows the `BACKUP.INI` of the first disk of a two disk backup. The real file doesn't contain empty lines.

```
[BACKUP SETUP]
TOTAL FILE NUM:0
7 = SUP.bup

[BACKUP STYLE]
TOTAL FILE NUM:0
2 = STY.b01

[BACKUP OTS]
TOTAL FILE NUM:0

[BACKUP MUSIC DB]
TOTAL FILE NUM:0

[BACKUP REGIST]
TOTAL FILE NUM:0

[BACKUP MULTI PAD]
TOTAL FILE NUM:0
```

This is the same file from the second disk:

```
[BACKUP STYLE]
TOTAL FILE NUM:0
2 = STY.bFF

[BACKUP OTS]
TOTAL FILE NUM:0
5 = OTS.bup
```

```
[BACKUP MUSIC DB]
TOTAL FILE NUM:0
3 = MDB.bup

[BACKUP REGIST]
TOTAL FILE NUM:0
4 = REG.bup

[BACKUP MULTI PAD]
TOTAL FILE NUM:0
6 = MPD.bup
```

Several things can be seen. The field `TOTAL FILE NUM` is not used and thus can only be zero. Each data section may only contain one file and files are numbered globally. The following numbering scheme is expected by the firmware:

Section	File number	First disk	Other disks
Setup	7	Present	Missing
Style	2	Present	Present
OTS	5	Present	Present
Music DB	3	Present	Present
Registration	4	Present	Present
Multi Pads	6	Present	Present

System settings and styles are always stored on the first disk. If the style file doesn't fit on disk it is splitted. Only the last disk may contain the other backup files. The file `BACKUP.INI` of the first disk always contains all data sections even if they don't have a file on that disk. In that case the sections are empty. The following disks lack the setup section. Even though a backup must not contain all data types there may be no section missing in the control file.

> For the Yamaha 9000pro the data files are called `SUP_Pro.bup`, `OTS_Pro.bup`, `MTS_Pro.bup` instead of `SUP.bup`, `OTS.bup`, `MTS.bup` and so on. These are the only name recognized by the firmware.

Binary files REG.bup and REG_Pro.bup

All registration banks are saved to the binary file `REG.bup` or `REG_Pro.bup` depending on the keyboard model. No details about that files are known but the general layout is very simple since only fixed-length records and no hierarchical trees are used. Also the files always contain all 8 registrations of all 64 banks.

Position$_{16}$	Amount	Length	Content

00 00 00 00	1	24	Start of file (PSR-9000):
			`00 46 8E 94  00 03 0B 0C`
			`00 00 00 00  00 00 00 00`
			`52 45 47 39  5F 31 30 32`
			Start of file (9000pro):
			`00 46 8E 94  00 03 B0 0C`
			`00 00 00 00  00 00 00 00`
			`52 45 47 39  50 31 30 30`

00 00 00 18	64	3776	Registration banks
00 03 B0 18	1	4	End of file: `E7 97 DD AB`

Each bank consists of a 16 character name and 8 registrations:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	16	Bank name. Filled with `0x00` or spaces `0x20` at the end.
00 00 00 10	8	470	Registrations of the bank. All registrations of a bank must be present even if they are empty. Empty registrations contain only a name followed by `0x00` bytes.

The registrations have a very similar layout:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	16	Registration name. Filled with `0x00` or spaces `0x20` at the end.
00 00 00 10	1	454	Registration Data

No details have been explored about the registration data. There is no length field so all registrations are of exactly the same size. No notable differences have been found between Yamaha PSR-9000 and 9000pro created files.

2.1.4 Control file DISK.MNG

All disks which are formated on the PSR-9000 or 9000pro have a simple text file called `DISK.MNG` in their root directory. It's assumed that the file is not needed by the instrument and instead is a left-over from the firmware of older instruments like the PSR-8000. All lines are exactly 14 characters long with trailing spaces as needed and 8 lines are present!

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	D	I	S	K					M	N	G			
2	P	S	R	-	9	0	0	0						
3	V	e	r	1	.	0	0	R	e	v	1	.	0	0
4														
5														
6														
7														
8														

2.2 Yamaha PSR-2000, PSR-1000, PSR-A1000, PSR-2100

The Yamaha PSR-2000 and all derived models have some interesting characteristics. Sounds, styles and most of the user interface are clearly derived from the PSR-9000 and 9000pro. Yet many reworkings and features typically associated with the Tyros product range have been introduced with this in-between product. Most notable changes are the new file-based access to all Flash ROM contents, the ability to play songs and styles at the same time, dedicated song player buttons and (most important) that registration banks are saved to individual files. Therefor all storage related functions like System Backup and Load/Save User Data are gone in favor of a much simplified handling. The file extension `*.reg` remains though.

> Currently only files created at the Yamaha PSR-2000 are available. Thus no information about differences to the PSR-2100, PSR-1000 and PSR-A1000 is known. It is assumed however that those models are basically identical. Because of that only minimal differences in the file format (like different magic bytes) are expected.

A flat file structure is used for the general layout while registrations are made of a simple block list. All strings are latin-1 encoded and all numbers are unsigned integers in big endian order. Boolean values are stored as 1-byte signed integers like this:

- `0x00`: 0 ⇒ False or Off
- `0x7F`: 127 ⇒ True or On

The general file layout is this:

Position$_{16}$	**Amount**	**Length**	**Content**
00 00 00 00	1	30	Start of file (PSR-2000): 52 45 47 2D 31 30 30 2D 31 30 30 2D 31 30 30 30 50 53 52 32 30 30 30 78

Position$_{16}$	Amount	Length	Content
			00 08 00 40 Most of that resembles the string `REG-100-100-1000PSR2000`. **The other models are likely to have a slightly different start sequence.**
00 00 00 1C	1	4	Amount of contained registrations
00 00 00 20	8	4	Absolute position of each registration or 0xFF FF FF FF for empty slots
00 00 00 40	1	48	Unknown byte sequence or padding: 24 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00
00 00 00 70	0–8	variable, usually 1568	Registration blocks

Each registration has the following structure:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	4	Magic bytes: `RGST`
00 00 00 04	1	2	Unknown fixed value (maybe format number): 0x00 01
00 00 00 06	1	2	Length of the complete registration block. Usually 0x06 20
00 00 00 08	1	104	Unknown byte sequence: 10 70 18 00 00 00 00 00 00 00 1F 80 00 00 1F D0 FF FF FF FF 00 00 1F E0 00 00 20 80 00 00 20 F0 00 00 21 60 00 00 21 D0 00 00 22 70 FF FF FF FF 00 00 22 80 00 00 22 90 FF FF FF FF 00 00 22 A0 FF FF FF FF FF FF FF FF 00 00 22 B0 00 00 23 C0 00 00 24 D0 00 00 24 F0 00 00 25 10 FF FF FF FF FF FF FF FF FF FF FF FF
00 00 00 70	16	variable	GP blocks

All GP blocks share the same basic layout:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	4	Magic bytes, e.g GP00 or GP0B. The last two characters are hexadecimal numbers between 0x00 and 0x14 but not all numbers are used.
00 00 00 04	1	2	Length of the complete block
00 00 00 06	1	2	Fixed value: 0x00 00
00 00 00 08	1	variable	Block data

Some of the blocks have been reverse-engineered so that some information about their content is known.

GP00	Registration name
GP03	Style configuration
GP04	Main voice
GP05	Layer voice
GP06	Left voice
GP08	Transpose values
GP0B	Tempo values
GP0D	Multi pad volume
GP10	Selected style
GP11	Selected multi pads

Registration name:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	4	Magic bytes: GP00
00 00 00 04	1	2	Fixed length: 0x00 50, Other values crash the instrument while loading the registration
00 00 00 06	1	2	Fixed value: 0x00 00
00 00 00 08	1	72	Registration name as zero-terminated string

Style configuration:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	4	Magic bytes: GP03 This block is missing if no style is selected, e.g. if it was tried to save the registration with a floppy style
00 00 00 04	1	2	Fixed length: x00 A0
00 00 00 06	1	2	Fixed value: 0x00 00
00 00 00 08	1	2	Unknown

Position$_{16}$	Amount	Length	Content
00 00 00 0A	1	1	Boolean: Acmp On / Off
00 00 00 0B	1	1	Selected style part: 0x02: Intro, 0x22: Ending, 0x08: Main A, 0x09: Main B, 0x0A: Main C, 0x0B: Main D, 0x10: Fill A, 0x11: Fill B, 0x12: Fill C, 0x13: Fill D
00 00 00 0C	1	2	Unknown
00 00 00 0E	1	1	Style volume
00 00 00 0F	1	1	Style panorama
00 00 00 10	1	2	Unknown
00 00 00 12	1	1	Left split point. This is the midi note number + one octave. e.g. 0x48 (C5) is saved if 0x3C (C4) was selected.
00 00 00 13	1	1	Acmp split point + one octave (see above)
00 00 00 14	1	137	Unknown

Selected panel voices:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	4	Magic bytes: GP04 for Main voice Magic bytes: GP05 for Layer voice Magic bytes: GP06 for Left voice
00 00 00 04	1	2	Fixed length: 0x00 70
00 00 00 06	1	2	Fixed value: 0x00 00
00 00 00 08	1	1	Boolean: Part On / Off
00 00 00 09	1	1	Selected voice MSB
00 00 00 0A	1	1	Selected voice LSB
00 00 00 0B	1	1	Selected voice Program
00 00 00 0C	1	9	Unknown
00 00 00 15	1	1	Volume
00 00 00 16	1	8	Unknown
00 00 00 1E	1	1	Panorama
00 00 00 1F	1	65	Unknown (Maybe voice editor values)
00 00 00 60	1	1	Block GP04: Octave Transpose: -1, 0 or 1 Block GP06: Left hold On / Off
00 00 00 61	1	15	Unknown, maybe only 0x00 bytes

Transpose values:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	4	Magic bytes: GP08
00 00 00 04	1	2	Fixed length: 0x00 10
00 00 00 06	1	2	Fixed value: 0x00 00
00 00 00 08	1	1	Signed integer: Master transpose

Position$_{16}$	Amount	Length	Content
00 00 00 09	1	1	Signed integer: Song transpose
00 00 00 0A	1	1	Signed integer: Keyboard transpose
00 00 00 0B	1	5	0x00 bytes

Tempo values:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	4	Magic bytes: GP0B
00 00 00 04	1	2	Fixed length: 0x00 10
00 00 00 06	1	2	Fixed value: 0x00 00
00 00 00 08	1	2	Fixed value: 0x00 00
00 00 00 0A	1	2	Song tempo
00 00 00 0C	1	2	Style tempo
00 00 00 0E	1	2	0x00 bytes

Multi pad volume:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	4	Magic bytes: GP0D
00 00 00 04	1	2	Fixed length: 0x00 10
00 00 00 06	1	2	Fixed value: 0x00 00
00 00 00 08	1	1	Volume
00 00 00 09	1	2	Unknown
00 00 00 0B	1	1	Panorama
00 00 00 0C	1	4	Unknown

Selected style:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	4	Magic bytes: GP10
00 00 00 04	1	2	Fixed length: 0x01 10
00 00 00 06	1	2	Fixed value: 0x00 00
00 00 00 08	1	264	Style path as zero-terminated string. ROM style D:/STYLE/Pop&Rock/-HeartBeat.S119.sty, flash style in root directory C:/STYLE/HeartBelinda.S119-.STY, floppy Style cannot be saved!

Selected multi pads:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	4	Magic bytes: GP11
00 00 00 04	1	2	Fixed length: 0x01 10

Position$_{16}$	Amount	Length	Content
00 00 00 06	1	2	Fixed value: `0x00 00`
00 00 00 08	1	264	Multi pad path as zero-terminated string, e.g. `D:/MULTI PAD/Samba Show1.S387-.pad`

2.3 Yamaha Tyros, PSR-3000, PSR-S700, PSR-S710, PSR-S900, PSR-S910

Starting with the original Tyros many changes which were introduced earlier have been finished. One important change is that file formats finally have stabilized enough so that new keyboard models introduce only small differences. The registration file format is very similar to the PSR-2000 in that individual banks are saved to files which use a flat structure for the bank and a block list for each registration. Yet besides the general idea all of the inner workings have been reinvented again. Also the file extension has changed from `*.reg` to `*.rgt`.

> The differences between the different arrangers are not totally understood. One reason is that example files are only available for the Tyros range. For the other arrangers there are no or not enough example files available.

All strings are latin-1 encoded and all numbers are unsigned integers in big endian order. Boolean values are stored as 1-byte signed integers like this:

- `0x00`: 0 ⇒ False or Off
- `0x7F`: 127 ⇒ True or On

The general file layout is this:

Position$_{16}$	Amount	Length	Content
00 00 00 00	1	16	Start of file (see below)
00 00 00 10	1	4	File size in bytes
00 00 00 14	1	44	Padding (see below)
00 00 00 40	8	variable	Registration banks
-- -- -- --	1	6	File end: `0x46 45 6E 64 0x00 00` (`FEnd`)

The following file headers (start of file) and are known:

Tyros 1	`53 70 66 46 00 10 0A D9`
	`52 47 53 54 00 00 00 07`

Tyros 2	53 70 66 46	00 10 0B 75
	52 47 53 54	00 02 00 00
Tyros 3	53 70 66 46	00 10 0C 12
	52 47 53 54	00 02 00 02
Tyros 4	53 70 66 46	00 10 0C C1
	52 47 53 54	00 02 00 03
PSR-S900	53 70 66 46	00 10 0B C6
	52 47 53 54	00 02 00 00
PSR-S700	53 70 66 46	00 10 0B C7
	52 47 53 54	00 02 00 00
PSR-3000	53 70 66 46	00 10 0B 20
	52 47 53 54	00 01 00 02

The following paddings are known:

Tyros 1	15 5C 42 48	64 01 00 24
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	
Tyros 2	00 82 42 48	64 01 00 24
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	
Tyros 3	00 65 42 48	64 01 00 24
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	
Tyros 4	00 64 42 48	64 01 00 24
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	FF FF FF FF
	FF FF FF FF	
PSR-S900	00 78 42 48	64 01 00 24
	00 01 FF 04	05 06 07 FF
	00 00 00 00	00 00 00 00
	00 00 00 00	00 00 00 00
	00 00 00 00	00 00 00 00
	00 00 00 00	

```
PSR-S700   00 66 42 48  64 01 00 24
           FF FF FF FF  FF FF FF FF
           FF FF FF FF  FF FF FF FF
           FF FF FF FF  FF FF FF FF
           FF FF FF FF  FF FF FF FF
           FF FF FF FF
PSR-3000   00 00 42 48  64 01 00 24
           FF FF FF FF  FF FF FF FF
           FF FF FF FF  FF FF FF FF
           FF FF FF FF  FF FF FF FF
           FF FF FF FF  FF FF FF FF
           FF FF FF FF
```

Each registration has the following structure. Empty registrations have a length of zero and no GPm blocks.

Position$_{16}$	**Amount**	**Length**	**Content**
00 00 00 00	1	4	Magic bytes: `BHd 0x00`: `0x42 48 64 00`
00 00 00 04	1	2	Length of the following registration data
00 00 00 06	many	variable	GPm blocks

All GPm blocks share the same basic layout:

Position$_{16}$	**Amount**	**Length**	**Content**
00 00 00 00	1	4	Magic bytes, e.g `GPm,` or `GPm-`. The last byte is the block number
00 00 00 04	1	2	Length of the following data
00 00 00 06	1	variable	Block data

Currently no further research has been conducted about the contents of the GPm blocks. It is only known that the first block is always block `01` which contains the registration name:

Position$_{16}$	**Amount**	**Length**	**Content**
00 00 00 00	1	4	Magic bytes: `GPm 0x01`
00 00 00 04	1	2	Length of the following data
00 00 00 06	1	variable	Registration name without trailing spaces or `0x00` bytes