GRID CARTOGRAPHER 4

MAPPING MADE EASY

GAME LINK AND LIBRETRO

Version 4.1.2

APPENDIX A – GAME LINK PROFILE REFERENCE

Table of Contents

Introduction

Game Link profiles are XML documents that contain all of the information needed to support automatic tracking of the player (or party) position in a game, within the editor.

A profile is made up of four components:

- Information to automatically identify the game (or games) that this profile is for.
- A list of memory addresses to read from the emulator that is hosting the game. The data at these addresses forms the data packet – as described below.
- Setup information for a compatible map (region names, floor sizes, etc.)
- A set of 'views' that translate the data read from the emulated game into area, local co-ordinate and facing angle outputs that are used to control the Grid Cartographer avatar marker in the editor view.

The next section describes the full hierarchy of possible elements in a profile document and the section after describes the elements in detail.

Note: this appendix is not meant as a tutorial for creating new profiles. To see examples of existing profiles, open the file `base0.zip` found in the installation directory. Inside this compressed archive is a `gamelinkprofiles` folder where you will find all of the built-in profiles.

The Data Packet

A data packet is raw data read from emulator memory at the address(es) specified by the profile. One byte of data will be read from each address and is added to the packet in the order specified in the list. The elements of the packet are indexed by an offset value, starting from zero.

For example, `<peek bytes="d9c6 d9c4 d9c0 d9c2" />` might generate the following packet:

Address	d9c6	d9c4	d9c0	d9c2
Offset	0	1	2	3
Data	7	2	1	1

NOTE: While the data may change as the player moves around, the address list and offsets are fixed.

The packet itself is a block of raw data, but is interpreted and assigned meaning by the `<views>` section of the profile, described below.

Element Hierarchy

This structure lists all of the possible elements and their relationship to one-another.

```
<gamelink>
|---  <card>
|---  <libretro>
|     \---  <detect>
|           |---  <content_hash>
|           \---  <peek>
|---  <dsub>
|     \---  <detect>
|           \---  <peek>
|---  <regions>
|     \---  <region>
|           \---  <grid>
|---  <views>
      \---  <class> or <packetview>
            |---  <check>
            |---  <check_or>
            |     \---  <check>
            |---  <seq>
            |---  <face>
            |---  <regname>
            |---  <regw>
            |---  <regh>
            |---  <xpos>
            |---  <ypos>
            |---  <xypos>
            |---  <match>
            |---  <mask>
            |---  <scalex>
```

```
|---  <scaley>
|---  <move>
|---  <const_floor>
\---  <floor>
```

Element Reference

This section describes each element, including any attributes that can be used. Note that a small number of elements share the same name but are differentiated by their place in the hierarchy.

<gamelink>

This is the root container element of the profile document.

<card>

This element provides user-facing information about the game. The following attributes are required. If the <card> element is not valid, the profile will be ignored and an error written to the log file.

Attribute	**Meaning**
`title`	The full title of the game, in upper case. e.g. `"WIZARDRY: PROVING GROUNDS OF THE MAD OVERLORD"`
`short`	A shortened version of the game title, in upper case. e.g. `"MIGHT AND MAGIC III"`
`titlelo`	The title of the game, in sentence case. e.g. `"Dragon Wars"`
`beta`	A Boolean value – '`true`' or '`false`' to indicate whether the profile is not fully complete or still in an early stage of development.

NOTE: In older versions of the software, other attributes were present in this element. These have now been removed from the specification and are no longer necessary.

<libretro>

This element allows the profile to be compatible with LibRetro game emulation and is a container for multiple `<detect>` elements to allow for different regional variations (and in some cases, ports across multiple platforms) without needing to create multiple profiles.

<detect>

(child of <libretro>)

Each `<detect>` element contains the information required for the game (including variants) on a single system. In order to support multiple systems, add additional `<detect>` elements.

The following attributes are required:

Attribute	**Meaning**
`system`	An abbreviated value to represent the system that the game is running on. This is used for user facing display only and not validated by the detection system. To maintain a consistent user experience, please consult the *System Naming Convention* table in the next section for recommended system naming.
`tag`	This optional attribute allows this detect element to be tagged with an arbitrary string. This is used to make minor adjustments to how the packet data is interpreted, based on system.

Within a `<detect>` element should be one or more child elements to provide rules to match the profile to the game being played, plus a single `<peek>` element to describe the memory addresses to read from to form the 'packet' of data interpreted by the rest of the profile.

<content_hash>

Detect a game by matching a computed hash of the game's binary data with the value provided.

NOTE: In order to obtain the hash, the easiest way is to load the game in question and then look in the log file (accessible via the start menu program group) where it will be written. The hash is usually 64 characters long and comprised of hexadecimal digits.

The hash value itself should be added a child element of this one, for example:

```
<content_hash>
      68a5c90239730c6fb729089740ff6f338122af48ea6b710e4f4e547b6942cf19
</content_hash>
```

Multiple `<content_hash>` elements can be listed under the parent `<detect>` to handle minor variations of games released in multiple territories or after being patched.

<peek>

(child of <libretro>)

This element describes a list of memory addresses to read bytes from in order to form a 'packet' of input data processed by the 'view' elements described later in this section.

Only one `<peek>` element should be specified as a child of `<detect>` and should have the following (required) attribute:

Attribute	**Meaning**
`bytes`	A list of addresses in hexadecimal within the work ram of the system. Multiple address values should be separated by whitespace.

NOTE: Address values are obtained using tools such as the *LRMEM* utility available on the Grid Cartographer forum, operation of that software is beyond the scope of this document.

<dsub>

This element allows the profile to be compatible with the older 'DSUB' protocol used by *Custom DOSBox*. It is a container for multiple `<detect>` elements to support different variations of a game without needing to create multiple profiles.

<detect>

(child of <dsub>)

A detect element under a `<dsub>` parent has the following required attributes:

Attribute	Meaning
`sys`	The hash of the system running the game. For Custom DOSBox this will always be the value: `"e9b551c5"`

`prg`
`ph3`
`ph2`
`ph1`
`ph0`
	Program hash values to identify the specific game being played.
	
	These values can be obtained by running the game within Grid Cartographer and then activating the 'debug HUD' by selecting to the Options tab, Editor page and clicking "Show Debug HUD (Advanced)". Then looking at the white text overlaid onto the first editor viewport.
	
	The `prg` attribute should be set to the hexadecimal value next to `PROGRAM`
	
	The `ph3` – `ph0` attributes should be set to the four, colon-delimited hexadecimal values next to `PROGRAM HASH`. The order of values corresponds to `ph3`, `ph2`, `ph1` and `ph0` respectively.
	
	Wildcards
	
	Any value used for these attributes can optionally be replaced with a single asterisk character `"*"` to represent a 'don't care' or wildcard value. These values will automatically match successfully and can be used for games that self-modify their own executable (e.g. Dragon Wars) resulting in a change to one or more components of the hash value and preventing successful auto-detection.

<peek>

(child of <dsub>)

This element describes a list of memory addresses to read bytes from in order to form a 'packet' of input data processed by the 'view' elements described later in this section.

Only one `<peek>` element should be specified as a child of `<detect>` and should have the following (required) attribute:

Attribute	Meaning
bytes	A list of addresses in hexadecimal within the work ram of the system. Multiple address values should be separated by whitespace.

<regions>

A container element for a bank of <region> elements that are used to setup a compatible map setup for the game within Grid Cartographer.

<region>

Each of these elements describes a named region of the map needed for the supported game.

The following attributes are supported:

Attribute	Meaning
id (required)	A numerical identifier for the region, referenced by <packetview> elements described below. Any positive integer greater than zero is valid, but must be unique among the other <region> elements.
name (required)	The name of the region. This will be shown on the region tab bar in Grid Cartographer. It is one of the key fields used to determine whether the current map is setup correctly.
prefix	This string attribute is used as a prefix when a region is named dynamically by the game itself. See the <regname> below for more details.
auto_create	The Boolean attribute specifies whether this region should be created automatically when the map is initially setup. If set false the region will only be created once the player has reached that location – this is recommended to maintain the element of discovered. At least one region must have this value set to true, it is recommended that this be the starting area.

`ground_floor`	A Boolean value specifying whether the region should have a ground floor present as well as floors and basements. For games which have no vertical element, this value is recommended to be set to true. By default, this value is false.
`start_floor`	Specify the starting floor when the region is created. Above ground floors are specified with `'F'` followed by a positive integer (e.g. `"F2"` for floor 2), basements with a `'B'` prefix (e.g. `"B3"` for basement 3). Ground floor is specified with `"G"`. By default, this setting will either the `"G"` if a ground floor is present, or `"F1"` if not.

<grid>

This element describes the grid setup for the region. It's a requirement that this element is present. It has the following attributes:

Attribute	**Meaning**
`width` (required)	Specifies the width of the major grid size in tiles. It must be an integer value between `2` and `128`.
`height` (required)	Specifies the height of the major grid size in tiles. It must be an integer value between `2` and `128`.
`infinite`	A Boolean value that specifies whether the map should have a fixed maximum size, or be an infinite plane. This setting is typically recommended for large overworld maps.
`tilex`	Specifies the width, in major tiles, of the grid. By default, this value is 1. For infinite maps the value is ignored.
`tiley`	Specifies the height, in major tiles, of the grid. By default, this value is 1. For infinite maps the value is ignored.
`origin_tl`	A Boolean value that specifies a top-left map origin, if `true`. By default, the origin is located at the bottom-left of the grid.

one_base	If "true" the grid will be labelled with values starting at one, rather than the default of zero.
label_major	If "true" the major tiles will be labelled with axis values, rather than the default of values drawn next to each tile.
major_lines	If "false" the darker grid lines on major tiles will not be drawn. By default, these lines are visible.
minor_lines	If "false" the minor grid lines within the major tiles will not be drawn. By default, these lines are visible. The major lines are not affected by this setting.
natural_rows	If "true" activates the 'natural rows' setting where axis values on the vertical axis are drawn so that they increase from top to bottom on the screen. This setting requires that the label_major option is enabled. (This feature is used for the world map in *Might and Magic*.)
x_letters	If "true" the axis values on the horizontal axis are written as letters instead of numbers.
y_letters	If "true" the axis values on the vertical axis are written as letters instead of numbers.
repeating	If "true" the axis values on both axes will restart from their original value at the beginning of each major tile.

<views>

This element is a container for all of the <class> and <packetview> elements that process the raw data extracted from the game's memory into a set of output values used by the avatar marker.

<packetview> and <class>

The <packetview> is the core element for processing the data packet into parameters needed by the avatar marker in the editor view.

The <class> element is a supporting feature to allow common elements to be shared among multiple <packetview> elements. A <class> provides default values for any <packetview> that 'inherits' from it without the need to specify them each time.

Processing of the data packet works by iterating through each <packetview> element in the order specified in the profile. Any checks in the view are evaluated against the packet data, if all pass then the avatar marker is updated and processing stops.

The following attributes are recognized:

Attribute	**Meaning**
region (required)	This attribute specifies the numerical id of the <region> element that this view corresponds to. If the view is used, the editor will automatically switch to the corresponding region tab. Multiple views can specify the same region.
name	A unique string name assigned to a <class> to allow for referencing by a child <packetview>. This attribute is optional for <packetview> elements.
if	This optional attribute is used to allow classes to be included or ignored based on the regional or platform variant of a game. When a class is included it is possible for a <packetview> to inherit from it. This feature gives a limited ability to adapt to minor variations in the data (e.g. different values may be used to specify the current facing direction on SNES vs. Mega Drive.) by specifying multiple classes with the same name but different if attributes. When processing the <class> for use by a LibRetro game, the tag attribute specified in the <detect> element (child of <libretro>) is considered. When processing for use by Custom DOSBox the tag "dsub" will be used instead. The if attribute should specify a semi-colon delimited list of tags such that if any element matches with the <detect> tag (or "dsub"), the class will be used, otherwise the class will be rejected and cannot be used as a parent to any <packetview>. If this attribute is omitted, the class is always included.

`extends`	This attribute is used by `<packetview>` elements and should specify the `name` of a `<class>` parent view. The parent class will be used as the base for this view and provide defaults for any properties or checks specified. If multiple classes of the same `name` attribute are specified, the first valid one from the top of the document will be used. See the `if` attribute above for a mechanism to select which `class` is used as a parent based on the system being emulated. Classes cannot extend other classes and a `<packetview>` cannot extend more than one class.
`tag`	Views can be tagged with a fixed string for debugging purposes. The string is shown on the Debug HUD in the editor viewport. It is intended to help verify which packet view is being used to provided information for the avatar marker.

<check>

Adds a check to compare the data packet against a specific value or some specific condition. If the check fails, the view will be immediately rejected and processing of subsequent views will continue. Multiple checks can be included but all must pass for the view to be valid.

The following attributes are supported:

Attribute	**Meaning**
`offset` (required)	Specifies a byte offset into the data packet to begin reading. Offsets are given in hexadecimal and start from `0` (zero). The offset must be less than the size of the packet.
`length` (required)	The number of bytes to consider for this check between 1 and 4. It is a requirement that `offset` plus `length` is less than the size of the data packet.
`value` (required)	A specific constant value, written in hexadecimal notation, to check against the number read from the packet. Note that when length is greater than one, the packet value is read as little endian.

<table>
<tr><td>mask</td><td>An optional bitmask that is “bitwise and” with both the value and the packet value to selectively ignore specific bits, if necessary.</td></tr>
<tr><td>op</td><td>Specifies the mathematical operation to perform to evaluate the check. By default, the operation is “equals”. Other operations can be performed as follows (where P is the value read from the packet and V is the constant value attribute.):
<table><tr><td>"EQ"</td><td>P equals V (default)</td></tr><tr><td>"NE"</td><td>P doesn’t equal V</td></tr><tr><td>"LT"</td><td>P is less than V</td></tr><tr><td>"LTE"</td><td>P is less than or equal to V</td></tr><tr><td>"GT"</td><td>P is greater than V</td></tr><tr><td>"GTE"</td><td>P is greater than or equal to V</td></tr></table>.</td></tr>
</table>

<check_or>

This element allows multiple checks to be performed but only one of the checks needs to pass for the whole <check_or> element to pass. Note that other <check> elements within the packet view must still pass for the view itself to be valid.

<xpos>

This element reads the X ordinate of the player (or party) in the game from the data packet. It does this by reading a value from the packet and performing a validation that it is between a minimum and maximum range.

If the value read is within range the avatar marker position is updated, if not then processing of the packet view is aborted and the next view is considered.

The following attributes are supported by this element and are required unless otherwise stated:

Attribute	Meaning
`offset`	Specifies a byte offset into the data packet to begin reading. Offsets are given in hexadecimal and start from `0` (zero). The offset must be less than the size of the packet.
`length`	The number of bytes to read from the packet, between 1 and 4. It is a requirement that `offset` plus `length` is less than the size of the data packet.
`mask` (optional)	An optional bitmask that is "bitwise and" with the packet value to selectively ignore specific bits, if necessary.
`min` `max`	These attributes define the minimum and maximum allowable values for the position. The constraint is as follows: `min <= value < max` The values must be given in hexadecimal notation.

<ypos>

This element reads the Y ordinate of the player (or party) in the game from the data packet. It does this by reading a value from the packet and performing a validation that it is between a minimum and maximum range.

If the value read is within range the avatar marker position is updated, if not then processing of the packet view is aborted and the next view is considered.

The following attributes are supported by this element and are required unless otherwise stated:

Attribute	Meaning
`offset`	Specifies a byte offset into the data packet to begin reading. Offsets are given in hexadecimal and start from `0` (zero). The offset must be less than the size of the packet.
`length`	The number of bytes to read from the packet, between 1 and 4. It is a requirement that `offset` plus `length` is less than the size of the data packet.

`mask` (optional)	An optional bitmask that is “bitwise and” with the packet value to selectively ignore specific bits, if necessary.
`min` `max`	These attributes define the minimum and maximum allowable values for the position. The constraint is as follows: `min <= value < max` The values must be given in hexadecimal notation.

<xypos>

This element reads a combined X and Y co-ordinate from the packet when those values are stored in the form: `value = x + y * stride`. This format is used by *Eye of the Beholder*.

When the value is processed the x and y positions of the avatar marker are extracted as follows:

```
x = value mod stride
y = floor( value / stride )
```

Note that unlike the `<xpos>` and `<ypos>` elements, no additional range checking is performed.

The following attributes are supported by this element and are required unless otherwise stated:

Attribute	**Meaning**
`offset`	Specifies a byte offset into the data packet to begin reading. Offsets are given in hexadecimal and start from `0` (zero). The offset must be less than the size of the packet.
`length`	The number of bytes to read from the packet, between 1 and 4. It is a requirement that `offset` plus `length` is less than the size of the data packet.
`mask` (optional)	An optional bitmask that is “bitwise and” with the packet value to selectively ignore specific bits, if necessary.
`stride`	This attribute sets the stride value of the equation above, specified in hexadecimal.

<match>

This element allows complex non-rectangular areas to be isolated and used for matching the current view. A position list is specified and if the extracted x/y co-ordinate is found within the list, the view will be accepted, otherwise processing will stop and the next `<packetview>` considered.

Attribute	Meaning
`pts`	Specifies a list of comma-separated point pairs "`x,y`", each separated by whitespace. It is recommended to use whitespace to arrange the pairs into the shape of the area to be isolated to avoid missing values.

<mask>

This element allows complex non-rectangular areas to be isolated and rejected from matching the current view. A position list is specified and if the extracted x/y co-ordinate is found within the list, processing will stop and the next `<packetview>` considered.

Attribute	Meaning
`pts`	Specifies a list of comma-separated point pairs "`x,y`", each separated by whitespace. It is recommended to use whitespace to arrange the pairs into the shape of the area to be isolated to avoid missing values.

<face>

This element extracts the facing direction of the player (or party) in the game from the data packet. It does this by reading a value from the packet and checking it against four constant values that represent north, east, south or west.

If the packet value doesn't match any of the constants, or if this element is not included in the `<packetview>`, the facing angle of the avatar marker is set to 'unknown' and the lines on the marker are hidden. This may be desired when creating a view for an overworld map.

The following attributes are supported by this element and are required unless otherwise stated:

Attribute	Meaning
`offset`	Specifies a byte offset into the data packet to begin reading. Offsets are given in hexadecimal and start from `0` (zero). The offset must be less than the size of the packet.
`length`	The number of bytes to read from the packet, between 1 and 4. It is a requirement that `offset` plus `length` is less than the size of the data packet.
`mask` (optional)	An optional bitmask that is "bitwise and" with the packet value to selectively ignore specific bits, if necessary.

`n`
`e`
`s`
`w`
	These four attributes (lower case) define the constant values that are checked for to determine the facing direction.
	
	The values must be given in hexadecimal notation.

<scalex>

Applies a scaling to the extracted x position before being sent to the avatar marker. This is useful to compress a larger map into a smaller space. The following attributes are required:

Attribute	Meaning
`mul`	Specifies an integer multiplier to be applied to the position.
`div`	Specifies an integer divisor to the applied to the position. Note that after dividing any fractional component of the result is discarded.

The values are applied using the following method:

```
x' = floor( ( x * mul ) / div )
```

<scaley>

Applies a scaling to the extracted y position before being sent to the avatar marker. This is useful to compress a larger map into a smaller space. The following attributes are required:

Attribute	Meaning
`mul`	Specifies an integer multiplier to be applied to the position.
`div`	Specifies an integer divisor to the applied to the position. Note that after dividing any fractional component of the result is discarded.

The values are applied using the following method:

```
y' = floor( ( y * mul ) / div )
```

<move>

Applies a constant offset to the x and y position before being sent to the avatar marker.

Here are the supported attributes, note that both are optional and default to zero if not specified.

Attribute	Meaning
`x`	Specifies an integer offset to be applied to the final x position. Both positive and negative values can be used.
`y`	Specifies an integer offset to be applied to the final y position. Both positive and negative values can be used. Note that positive offsets move the position up the screen for bottom-left origin regions and down the screen for top-left origin regions.

Note that this offset is applied after any scaling is applied using the `<scalex>` or `<scaley>` elements.

<const_floor>

This element generates a constant floor index output for the view without reading data from the packet. This is useful for areas of the game world that don't have well defined floors, or exist on a flat plane.

The floor is specified as child text of the element, e.g. `<const_floor>F2</const_floor>`

The value can be either `F` followed by a positive integer to represent above-ground floors, a single `G` to represent the ground floor or `B` followed by a positive integer to represent a basement.

<floor>

This element reads a floor value from the data packet. It is designed to read either a linear series of above-ground floors or basements. If a `<const_floor>` is specified this element is ignored.

Attribute	**Meaning**
`offset`	Specifies a byte offset into the data packet to begin reading. Offsets are given in hexadecimal and start from `0` (zero). The offset must be less than the size of the packet.
`length`	The number of bytes to read from the packet, between 1 and 4. It is a requirement that `offset` plus `length` is less than the size of the data packet.
`mask` (optional)	An optional bitmask that is "bitwise and" with the packet value to selectively ignore specific bits, if necessary.
`min` `max`	These attributes define the minimum and maximum allowable values for the floor. The constraint is as follows: `min <= value < max` The values must be given in hexadecimal notation.
`dir`	The direction of the floor value is controlled using this attribute and must be specified. Two values are supported: `"up"` and `"down"`. The up mode treats the value read as a positive floor index, the down mode treats the data as a negative floor index.

`adjust` (optional)	This optional value allows a positive or negative offset to be applied to the floor index before being output to the avatar marker. Note that the value is not affected by the `dir` attribute and positive value will always raise the floor up, and a negative value will always lower the floor down.

<regname>

This element is used for games which feature a dynamic region name, typically those with a built-in editor such as *Forgotten Realms: Unlimited Adventures*.

When added to the view an ASCII character string is read from the packet and applied to the region name specified by the view. If the <region> declaration for this region in the <regions> bank contains a prefix attribute, that string is placed before the name read from the packet.

The following attributes are supported by this element and are required unless otherwise stated:

Attribute	**Meaning**
`offset`	Specifies a byte offset into the data packet to begin reading. Offsets are given in hexadecimal and start from `0` (zero). The offset must be less than the size of the packet.
`length`	The number of character bytes to read from the packet. It is a requirement that `offset` plus `length` is less than the size of the data packet.
`op` (optional)	An operation can be performed on the text string. If this attribute is omitted, no transformation is applied. `"UP"` — Text is transformed into upper case. .

<regw>

This element is used for games which feature a dynamic region size, typically those with a built-in editor such as *Forgotten Realms: Unlimited Adventures*.

When added to a view, a value is read from the packet, compared against a minimum and maximum range and then used to update the width of the current region. When the region size is set the region is also automatically configured to have a single major tile.

The following attributes are supported by this element and are required unless otherwise stated:

Attribute	Meaning
`offset`	Specifies a byte offset into the data packet to begin reading. Offsets are given in hexadecimal and start from `0` (zero). The offset must be less than the size of the packet.
`length`	The number of bytes to read from the packet, between 1 and 4. It is a requirement that `offset` plus `length` is less than the size of the data packet.
`mask` (optional)	An optional bitmask that is "bitwise and" with the packet value to selectively ignore specific bits, if necessary.
`min` `max`	These attributes define the minimum and maximum allowable values for the width. The constraint is as follows: `min <= value < max` The values must be given in hexadecimal notation. The largest maximum value is 255 (`FFh`).

<regh>

This element is used for games which feature a dynamic region size, typically those with a built-in editor such as *Forgotten Realms: Unlimited Adventures*.

When added to a view, a value is read from the packet, compared against a minimum and maximum range and then used to update the width of the current region. When the region size is set the region is also automatically configured to have a single major tile.

The following attributes are supported by this element and are required unless otherwise stated:

Attribute	Meaning
`offset`	Specifies a byte offset into the data packet to begin reading. Offsets are given in hexadecimal and start from `0` (zero). The offset must be less than the size of the packet.
`length`	The number of bytes to read from the packet, between 1 and 4. It is a requirement that `offset` plus `length` is less than the size of the data packet.
`mask` (optional)	An optional bitmask that is “bitwise and” with the packet value to selectively ignore specific bits, if necessary.
`min` `max`	These attributes define the minimum and maximum allowable values for the height. The constraint is as follows: `min <= value < max` The values must be given in hexadecimal notation. The largest maximum value is 255 (`FFh`).

System Naming Conventions

Note that not all systems listed below are fully supported and are provided here for future reference only. The international or originating country of the console is used where variations exist (e.g. *Mega Drive* rather than *Genesis*).

Console / Computer System	Abbreviation
Atari ST	`"ST"`
Amiga series	`"AMI"`
Commodore 64	`"C64"`
Commodore Plus/4	`"PLUS4"`
Commodore VIC-20	`"VIC20"`
Game Boy	`"GB"`
Game Boy Advance	`"GBA"`
Game Boy Color	`"GBC"`
IBM-PC compatible DOS	`"DOS"`
Mega-CD / Sega CD	`"MCD"`
MSX series	`"MSX"`
Nintendo DS	`"DS"`
Nintendo Entertainment System	`"NES"`
PC Engine / TurboGrafx-16	`"PCE"`
PC-8800 series	`"PC88"`
PC-9800 series	`"PC98"`
Sega Game Gear	`"GG"`
Sega Master System	`"SMS"`
Sega Mega Drive / Genesis	`"MD"`

Sega Saturn	"SAT"
Sharp X68000	"X68"
Sinclair ZX Spectrum	"ZX"
Sony PlayStation	"PS1"
Super Nintendo	"SNES"

.