

HITOTEXT

**XML FORMAT
DOCUMENTATION**

1	Header	2
1.1	Games	2
1.1.1	Name	2
1.2	Fields	2
1.2.1	Name	2
1.3	Extensions	2
1.3.1	Name	2
1.4	TextParameters	3
1.4.1	Formats	3
1.4.1.1	Name	3
1.4.2	SpecialMapping	4
1.4.2.1	Map	4
1.4.3	Offsets	4
1.4.3.1	Offset	4
1.4.4	SwitchMaps	5
1.4.4.1	SwitchMap	5
1.4.4.1.1	Mapping	5
2	FileStructure	7
2.1	Mapping	7
2.1.1	Entry	7
3	SetStructure	9
3.1	FieldName	9
3.1.1	SpecialUtilization	10
4	DisplayStructure	12
4.1	FieldName	12

Let me preface this whole thing by saying, I have no idea how to format this information well, so it may be a little tough on the eyes, and possibly a little confusing. I'll do my best to ask questions if you've read multiple times and still have no answer. =)

First, this is a link to the XML file: <http://www.the-elitists.com/HiToText/HiToText.xml>

Put this in the same folder as HiToText.exe

So, here's a small tutorial on the XML format. I've added about a dozen drivers to this thing so far, and it works very well. Bagman for example took about 3 minutes to add. In the near future I do plan on making some sort of application that will generate the XML for you, but for now you can manually do it.

The first thing to know is what's currently in it's own class now is separated as entries in the XML. Each entry element always has 4 elements:

- Header
- FileStructure
- SetStructure
- DisplayStructure

Order does not matter.

1 Header

The `Header` element contains:

- 3 required elements: `Games`, `Fields`, and `Extensions`
- 1 optional element: `TextParameters`

Order does not matter.

1.1 Games

List of `Name` elements. Order does not matter.

1.1.1 Name

Each `Name` element is the rom name which corresponds to the mapping. Order does not matter.

```
<Games>
  <Name>005</Name>
</Games>
```

1.2 Fields

List of `Name` elements. Order matters (see below).

1.2.1 Name

Each `Name` element is a field displayed. The order matters here, so put them in the order in which you'd like to see it displayed.

```
<Fields>
  <Name>RANK</Name>
  <Name>SCORE</Name>
  <Name>NAME</Name>
  <Name>ROUND</Name>
</Fields>
```

1.3 Extensions

List of `Name` elements. Order does not matter.

1.3.1 Name

Each `Name` element is a file type associated with the data you're receiving. Order matters. It can be one of these two values:

- `.hi`: The game stores the hiscores in a `.hi` file using the `hiscore.dat` and the `hiscore.c` hack.
- `.nv`: The hiscore data is stored in the `nvr` chip. The `.nv` file is a 'virtual' dump of that memory chip.

It has one semi-optional attribute:

- `NumberOfBytes` allows you to specify the number of bytes you'll be reading for that file. This is a required attribute if you have more than one extension per entry (i.e. Centipede or Asteroids Deluxe). See example below.

Generic example:

```
<Extensions>
  <Name>.hi</Name>
</Extensions>
```

astdelux example:

```
<Extensions>
  <Name NumberOfBytes="64">.nv</Name>
  <Name NumberOfBytes="43">.hi</Name>
</Extensions>
```

1.4 TextParameters

Is used to describe getting text information, mostly for name fields (so far).

It has optional attributes: `ByteSkipAmount`, `ByteStart` and `ByteSkipData`:

- `ByteSkipAmount`: It will have the text functions use the byte for every `ByteSkipAmount`. For example, if there are 6 bytes in the name field, however only 3 are used, a `ByteSkipAmount` value of 2 would use the 1st, 3rd, and 5th, or 2nd, 4th, and 6th bytes for the name function. The default value is 1.
- `ByteStart`: It will allow the text function to start at a specific byte in the name array. For example, if there are 6 bytes in the name field, however only 3 are used and you wish to start at the 2nd byte, a `ByteStart` value of 1 would use the name functions on the 2nd, 4th, and 6th bytes. The default value is 0.
- `ByteSkipData`: It will fill in the bytes that aren't used by the name function. This is only useful if you also use `ByteSkipAmount > 1`. Some games (batsugun) have specific values that need to be placed in the name bytes to show up correctly in-game. The default value is 0x00.

`TextParameters` can contain those elements:

- 1 required element:
 - `Formats` (if `TextParameters` exists)
- 2 potentially required elements, which depends on which `Formats` elements exist:
 - `SpecialMapping`
 - `Offsets`
- 1 optional elements:
 - `SwitchMaps`

Order does not matter.

```
<TextParameters ByteSkipAmount="2" ByteStar="1" ByteSkipData="0x01">
```

1.4.1 Formats

`Formats` contains a list of `Name` elements.

1.4.1.1 Name

`Name` elements can be one of (currently) 5 possible formats:

- `NeedsSpecialMapping`: will require the `SpecialMapping` element
- `ASCIUpper`: will require `Offsets` element

- ASCIILower: will require `Offsets` element
- ASCIINumbers: will require `Offsets` element
- ASCIIStandard: requires nothing

Order does not matter.

```
<Formats>
  <Name>NeedSpecialMapping</Name>
  <Name>ASCIIUpper</Name>
  <Name>ASCIINumbers</Name>
</Formats>
```

1.4.2 SpecialMapping

`SpecialMapping` is a list of mappings with characters to bytes, these are `Map` elements.

1.4.2.1 Map

The `Map` elements have two required attributes:

- `Char`: is a character (only one key allowed here)
- `Byte`: is the byte that it would map too

You can have multiple bytes assigned to a char, but not the other way around, and HiToText will always use the first value assigned to a character when there are multiple options.

```
<SpecialMapping>
  <Map Char="," Byte="0x0a" />
  <Map Char="." Byte="0x0c" />
  <Map Char="" Byte="0x10" />
</SpecialMapping>
```

1.4.3 Offsets

`Offsets` is a list of `Offset` elements

1.4.3.1 Offset

The `Offset` elements have 2 required attributes:

- `Type`: corresponds to the format element earlier, and can only be one of 3 values:
 - `Upper`
 - `Lower`
 - `Numbers`
- `StarByte`: is the byte for which a "A", or "0" will start at when being encoded/decoded in any function involving names. You need an `Offset` element for each of the 3 ASCII formats mentioned earlier (`ASCIIUpper`, `ASCIILower`, `ASCIINumbers`).

Order does not matter.

```
<Offsets>
  <Offset Type="Upper" StarByte="0x11" />
  <Offset Type="Numbers" StarByte="0x00" />
</Offsets>
```

1.4.4 SwitchMaps

Is a list of `SwitchMap` elements that are used for special "switch" functions such as mapping the characters in asterix and batsugun or the ranks in 1941.

1.4.4.1 SwitchMap

Each `SwitchMap` can have multiple `Mapping` elements and has 3 required attributes:

- Name
- DefaultOne
- DefaultMany

Name attribute of the `SwitchMap` element must match the Name attribute (case insensitive) of the `FieldName` element in `DisplayStructure` that is using the `Switch ConversionType` attribute. (Asterix and 1941 are an examples of this).

`DefaultOne` and `DefaultMany` are if nothing can be found matching your `Mapping` elements within the current `SwitchMap` element.

The switchmaps use a one to many relationship so you can have many strings mapped to one byte array. If multiple bytes are required for the byte array they can be separated by pipes. (i.e. `0x01|0x02`).

1.4.4.1.1 Mapping

Each `Mapping` element has 2 attributes:

- One: is a byte array
- Many: is a string

They work just like the default attributes of `SwitchMap` elements (`DefaultOne` and `DefaultMany`) except that `One` and `Many` are checked first, so if a switch function (explained a bit later) is looking for the byte that corresponds to the string "ASTERIX" it will search the mappings first and find `0x01` before it goes to the `DefaultOne`. The `Many` fields are NOT case sensitive, so if you wish for them to be displayed differently, or as was requested a web address to an image or something for a FE to use, then go for it. I prefer all caps, so that's how the file will look.

Asterix example:

```
<SwitchMaps>
  <SwitchMap Name="Character" DefaultOne="0x02" DefaultMany="OBELIX">
    <Mapping One="0x01" Many="ASTERIX" />
    <Mapping One="0x01" Many="A" />
  </SwitchMap>
</SwitchMaps>
```

1941 example:

```
<SwitchMaps>
  <SwitchMap Name="RankDisplay" DefaultOne="0x00" DefaultMany="SECOND LIEUTENANT">
    <Mapping One="0x01" Many="FIRST LIEUTENANT" />
    <Mapping One="0x02" Many="CAPTAIN" />
    <Mapping One="0x03" Many="MAJOR" />
    <Mapping One="0x04" Many="LIEUTENANT COLONEL" />
  </SwitchMap>
</SwitchMaps>
```

```
<Mapping One="0x05" Many="COLONEL" />
<Mapping One="0x06" Many="6" />
<Mapping One="0x07" Many="7" />
</SwitchMap>
</SwitchMaps>
```

batsugun example:

```
<SwitchMaps>
  <SwitchMap Name="Character" DefaultOne="0x00|0x00" DefaultMany="JEENO">
    <Mapping One="0x00|0x01" Many="BELTIANA" />
    <Mapping One="0x00|0x02" Many="ICEMAN" />
  </SwitchMap>
</SwitchMaps>
```

2 FileStructure

The `FileStructure` element is where the byte mapping actually is stored. It contains only `Mapping` elements, and may contain as many as required to fully define each of the bytes in the source file.

2.1 Mapping

The `Mapping` element contains `Entry` elements and has those attributes:

- 2 required attributes:
 - `NumberOfBlocks`: is the number of times to go through the `Entry` elements of that particular `Mapping` element. I'll provide an example shortly that should explain that better.
 - `Ordering`: determines the numerical increment/decrement that will occur as you go through each block.
- 1 optional attribute:
 - `Start`: is the number to start with when going through ordering.

2.1.1 Entry

Each `Entry` has 2 required attributes: `Name` and `Length`.

- `Name`: is the name that will be used for many other portions of the XML from setting scores, to reading scores, and will need to be the same as names for those other elements which will be explained a little later. These names ARE case sensitive, so be careful!
- `Length`: is the number of bytes that make up that entry. These names ARE case sensitive, so be careful!

If none of that made sense, hopefully this example from `arkretrn` helps:

```
<FileStructure>
  <Mapping NumberOfBlocks="5" Ordering="Ascending">
    <Entry Name="Score" Length="4"/>
  </Mapping>
  <Mapping NumberOfBlocks="4" Ordering="Ascending">
    <Entry Name="Name" Length="3"/>
    <Entry Name="Space" Length="1"/>
  </Mapping>
  <Mapping NumberOfBlocks="1" Ordering="Ascending" Start="5">
    <Entry Name="Name" Length="3"/>
  </Mapping>
</FileStructure>
```

So when `HiToText` goes to read the `.hi` file (in this case) it will start with the first map, and it sees an ascending mapping of 5 blocks that are called `Score` of 4 bytes each. So the first 20 bytes make up the first 5 scores (`Score1`, `Score2`, `Score3`, `Score4`, `Score5`). Then it will move to the next mapping, which is 4 blocks of ascending order containing two entries: `Name`, and `Space`. So it will read the next 16 bytes as (`Name1`, `Space1`, `Name2`, `Space2`, `Name3`, `Space3`, `Name4`, `Space4`). Finally, it will move to the last mapping, which is 1 block of ascending order that starts at 5 containing one entry: `Name`. So it will read the last 3 bytes as (`Name5`).

This is what `HiToText` will see for the mapping:

Score1 - Bytes 1-4
Score2 - Bytes 5-8
Score3 - Bytes 9-12
Score4 - Bytes 13-16
Score5 - Bytes 17-20
Name1 - Bytes 21-23
Space1 - Byte 24
Name2 - Bytes 25-27
Space2 - Byte 28
Name3 - Bytes 29-31
Space3 - Byte 32
Name4 - Bytes 33-35
Space4 - Byte 36
Name5 - Bytes 37-39

arkretrn.hi is 39 bytes length, so we have a complete map.

3 SetStructure

All right, now onto the most complicated... `SetStructure`. It contains as many `FieldName` elements as required to set a new score.

3.1 FieldName

The `FieldName` element contains 4 required attributes:

- `Name`:
- `FieldType`:
- `ConversionType`:
- `Position`:

and 2 optional attributes:

- `Constant`:
- `ExternalWrapper`:

The `FieldName` element can also contain as many `SpecialUtilization` elements as needed. These aren't required, but almost certainly will be needed to set something correctly.

Let's explain those attributes a little bit:

`Name` is pretty self-explanatory at this point, but this is what links up with the header names, filestructure names, and display names, so be consistent here.

`FieldType` is the type of the field, such as `string`, `int`, or `long`, etc...

`ConversionType` is the method used to convert the data of type `FieldType` into a byte array to be placed in the file. These are usually constant functions such as `IntToByteArrayHex` which converts an integer 100000 into a byte array that corresponds with the length of the `FileStructure/Mapping/Entry` attribute's name. So for example our 100000 above would convert to a byte array of length 3 for `4dwarrio` as `0x10|0x00|0x00`.

There are other methods such as `IntToByteArrayHexAsHex`, `Name`, or `Switch`. Custom names like those described for `8ballact` can also be used here.

If the field is named `"CustomName"` it will utilize a special function for setting data that uses the name function in addition to data. The function called is a custom function that is named as the parent rom name (the top rom in the XML). An example can be found in `Asteroids Deluxe`.

If the field is named `"PadData"` (It takes two parameters in parenthesis), the first is the value the byte should be if the data is the padded value, the second is the first digit of the byte when it's not. A good example of this can be seen in `circusc` for setting the high score:

```
<FieldName Name="LongHiScore" FieldType="int" ConversionType="PadData(10,2)" Position="1">
  SpecialUtilization>IsHiScore</SpecialUtilization>
  SpecialUtilization>EmptyScores</SpecialUtilization>
</FieldName>
```

`Position` is the position in the arguments array this field will be linked to. So for `4dwarrio`, the score is in the 1 position (0 is the first position). So when you type:

```
HiToText -w 4dwarrio hi 1 40000 NLA
```

the 1 is in the 0 position the 40000 is in the 1 position...

If the fields are separated by pipes it will return data from both of those fields separated by pipes for use in setting this value. This is used mostly for things like checksums, and an example can be found in *Asteroids Deluxe*:

```
<FieldName Name="Checksum" FielType="int" ConversionType="CustomName" Position="1|2">
  SpecialUtilization>IsAdjusted</SpecialUtilization>
</FieldName>
```

`Constant` is just a byte value that will be the value set for that particular `FieldName`. It's currently only used for ajax for the "Unknown" field that is always 0x11 for player created entries.

`ExternalWrapper`, is mostly used for when a byte array needs to be reversed, or if some other functionality needs to occur, like `ByteSwapping` (not currently implemented, but will be soon). In all the examples, only `ReverseByteArray` is currently used or accepted.

3.1.1 SpecialUtilization

The `SpecialUtilization` element can contain one of (currently) 6 values:

- `EmptyScores`
- `ModifyName`
- `IsHiScore`
- `DetermineRank`
- `IsAdjusted`
- `IncomingModified`

There can be more than one type `SpecialUtilization` per `FieldName`.

`EmptyScores` will flag a field for usage in the `EmptyScores` function which will 0 out the field. It's used mostly for `Score`, and `HiScore` fields, but can be used for stages, or coins, or whatever else.

`ModifyName` will flag a field for usage in the `ModifyName` function which will allow that field to have a name changed based on the rank given.

`IsHiScore` will flag a field as the `HiScore` entry for the `SetScore` function. Anything that you wish to only be changed when it's the top score should have this `SpecialUtilization`.

`DetermineRank` will flag a field as being the data used for determining how that value ranks against existing values. It requires an attribute of `Function`, which can be `Standard`, `Reversed`, `Hex`, etc... or a custom function named as described the other custom functions (i.e. `_8ballact`).

`IsAdjusted` will flag a field as being adjustable when a new score comes in. Use this for anything

where you want the data to move down a rank if a new score coming in is higher. Things like HiScores and Unuseds, or separators should NOT have this `SpecialUtilization` whereas things like Name, Round, and Score should.

`IncomingModified` will flag a field as needing to be modified before being used for anything else. It requires a function that is similar to the `Operator` attribute of `FieldName` in `DisplayStructure`. The `Function` attribute should be `"/10"` or `"*45"` or something mathematical.

Certain games save scores without a trailing 0, so 25,000 will be stored as `0x25|0x00` to save space. Using `IncomingModified` will abstract this from the user so they can type `"HiToText -w 005.hi 25000"` which is what the game will show, instead of 2500 which is what's actually stored.

```
<SetStructure>
  <FieldName Name="Score" FieldType="int" ConversionType="_8ballact" Position="1">
    <SpecialUtilization>EmptyScores</SpecialUtilization>
    <SpecialUtilization>IsAdjusted</SpecialUtilization>
    <SpecialUtilization Function="_8ballact">DetermineRank</SpecialUtilization>
    <SpecialUtilization Function="/10">IncomingModified</SpecialUtilization>
  </FieldName>
  <FieldName Name="Name" FieldType="string" ConversionType="Name" Position="2">
    <SpecialUtilization>ModifyName</SpecialUtilization>
    <SpecialUtilization>IsAdjusted</SpecialUtilization>
  </FieldName>
</SetStructure>
```

4 DisplayStructure

The `DisplayStructure` element is how HiToText determines how to display the scores when trying to display them.

It has one optional attribute:

- `NumOfDisplayedEntries`, which is a way to limit the amount of entries to display for scores. An example: 10yard is using this as there are 23 scores/Names in the .hi file, however you only ever see 10. This number can be safely changed to anything from 1-23 (for 10yard) if you wish to see more or less than the 10 the game shows.

The `DisplayStructure` element contains as many `FieldName` elements as required for display.

4.1 FieldName

In the `FieldName` elements order matters, and the first `FieldName` will be the first value per line. The first `FieldName` is almost always going to be Rank.

The `FieldName` element has:

- 2 required attributes:
 - `Name`:
 - `ConversionType`:
- 2 optional attributes:
 - `Operator`:
 - `CustomName`:

`Name` is linked to the `Name` in the `Entry` elements in the `FileStructure/Mapping` element. So if you want to display `Name`, you better have `Entry Name="Name"`, and `FieldName Name="Name"` in the appropriate place. `Name` also matches to the `SwitchMap Name` attribute if it exists.

`ConversionType` is the function used to convert the byte array data into a string. It can be:

- `"Name"`:
- `"CannedDisplay.AscendingFrom1"`: just counts from 1 for that field (perfect for Rank).
- `"Standard"`: converts the byte array as an integer, so `0x10 0x00 0x00` will come back as 100000.
- `"Reversed"`:
- `"Hex"`:
- `"HexReversed"`:
- `"BCD"`:
- `"BCDReversed"`:
- `"Switch"`:

There are also some custom one-off functions, in particular 8ballact has some custom functions for decoding/encoding scores. They should be named as the parent rom name, and if the rom starts with a number, put an underscore before the name. As you can see for 8ballact, it's `"_8ballact"`. Pretty

easy. Name functions (if setup) should just use a conversion type of "Name".

Operator is a mathematical function used to modify the data for display. Usually it's something like "+1" or "*10" so the displayed data more correctly matches what the game would display instead of what the game stores. Often games will not store data with known values and display it with the assumed values. That's basically what this will do.

For example, a game like 005 stores its scores divided by 10, so when we read something like 0x02 0x50 it would normally display as 250, however the game would display that as 2500. So we have a "*10" operator to get it to display how the game would display it. This is used in conjunction with a similar field in SetStructure.

CustomName it hooks into the normal Name function (of ConversionType), but some games (Nichibutsu games as an example) have some special features that don't completely with the standard Name functions. See cclimber example below.

cclimber example:

```
<DisplayStructure>
  <FieldName Name="Rank" ConversionType="CannedDisplay.AscendingFrom1" />
  <FieldName Name="Score" ConversionType="Standard" />
  <FieldName Name="Name" ConversionType="Name" CustomName="cclimber" />
</DisplayStructure>
```

10yard example:

```
<DisplayStructure NumOfDisplayedEntries="10">
  <FieldName Name="Rank" ConversionType="CannedDisplay.AscendingFrom1" />
  <FieldName Name="Score" ConversionType="Reversed" />
  <FieldName Name="Name" ConversionType="Name" />
</DisplayStructure>
```

1941 example:

```
<DisplayStructure NumOfDisplayedEntries="5">
  <FieldName Name="Rank" ConversionType="CannedDisplay.AscendingFrom1" />
  <FieldName Name="Score" ConversionType="Standard" />
  <FieldName Name="Name" ConversionType="Name" />
  <FieldName Name="RankDisplay" ConversionType="Switch" />
</DisplayStructure>
```

8ballact example:

```
<DisplayStructure>
  <FieldName Name="Rank" ConversionType="CannedDisplay.AscendingFrom1" />
  <FieldName Name="Score" ConversionType="_8ballact" Operator="*10" />
  <FieldName Name="Name" ConversionType="Name" />
</DisplayStructure>
```

asterix example:

```
<DisplayStructure>
  <FieldName Name="Rank" ConversionType="CannedDisplay.AscendingFrom1" />
  <FieldName Name="Score" ConversionType="Standard" />
  <FieldName Name="Name" ConversionType="Name" />
  <FieldName Name="Character" ConversionType="Switch" />
</DisplayStructure>
```