The Okteta Handbook

Friedrich W. H. Kossebau
Alex Richardson

Contents

Abstract

Okteta is a simple editor for the raw data of files. This type of program is also called hex editor or binary editor.

Chapter 1

Introduction

Okteta is a simple editor for the raw data of files.

The data is displayed in two variants: as the numeric values of the bytes and as the character assigned to the values. Values and characters can be shown either in two columns (the traditional display in hex editors) or in rows with the value on top of the character. Editing can be done both for the values and for the characters.

Besides the usual editing capabilities Okteta also brings a small set of tools, like a table listing decodings into common simple data types, a table listing all possible bytes with their character and value equivalents, an info view with a statistic, a checksum calculator, a filter tool and a string extraction tool.

All modifications to the data loaded can be endlessly undone or redone.

Chapter 2

Basics

2.1 Starting Okteta

Type **okteta** at a command prompt or select **Hex Editor** from the **Applications** → **Utilities** group in the application launcher.

The standard Qt™ and KDE Frameworks 5 command line options are available, and can be listed by entering **okteta --help**.

Command line options specific to Okteta are:

<URL(s)> - open file(s) from the specified URL(s)

2.2 Usage

The main Okteta window has the following components: a menu bar, a toolbar, a status bar, one or several sidebars with tools, and the main view with the tabbed data views.

When a file is opened or a new byte array is created, the bytes contained are listed consecutively in lines with a given number of bytes per line. They are displayed in two variants: as the numeric values of the bytes and as the characters assigned to the values. Values and characters can be shown either separated in two columns or next to each other with the value on top of the character. On the left side the offsets of the first byte in each line are shown.

The handling is similar to that of most text editors: the data can be edited, cut, copied, pasted, dragged and dropped much as text in these can. A cursor marks the current position. Pressing the **Insert** key toggles between the overwrite and insert modes. The overwrite mode is stricter than in text editors, as it doesn't allow any operation which changes the size of the byte array.

Other than in text editors the content is displayed in two variants. Only one of these is active with regard to new input. There are two linked cursors shown for the value and the character display, the cursor of the active one is blinking. With the characters active, characters can be entered as known from text editors. With the values active, typing a digit will open a minimal editor to enter the rest of the value.

The search dialog allows the user to search for a specific string of bytes, definable as values (hexadecimal, decimal, octal, binary) or text (current 8-bit encoding or UTF-8).

Multiple byte arrays can be open at the same time, but only one can be active. Use the **Windows** menu to select which byte array will be active.

Chapter 3

Tools

3.1 Overview

Okteta brings some tools, some to analyze and manipulate the byte arrays and some with more general purpose. These tools can be activated or deactivated from the **Tools** entry in the menu bar. Each tool has a small view, which docks either in one of the sidebars or freely floats as a window. You can dock, undock, rearrange and also stack the tool views with the mouse, by pressing the left mouse button on the title bar of a tool view, moving it as you like and releasing the left mouse button to complete the action, otherwise cancel it by pressing the **Esc** key.

3.1.1 Analyzers and Manipulators

Value/Char Table

The table lists all possible byte values, both as character and in the different numerical codings.

The selected value can be inserted at the cursor position for a defined number of bytes. This can be achieved by using the **Insert** button or double-clicking the line in the table.

Binary Filter

The filter performs binary operations on the selected bytes. After choosing the operation (AND, OR, ROTATE..) the parameters, if any, can be set in the box below. The filter is executed on the use of the **Filter** button.

Strings

This tool locates the strings in the selected bytes. After choosing the minimum string length, the strings are grepped for on the use of the **Extract** button. The list of the strings displayed can be narrowed by entering a filter term.

Statistics

This tool builds a statistic for the selected bytes. The statistic gives the frequency of the occurrence of each byte value in the selection. It can be calculated by using the **Build** button.

Checksum

This tool calculates various checksums or hashsums for the selected bytes. After choosing the algorithm and setting the parameter, if any, the sum is computed on the use of the **Calculate** button.

Decoding Table

The table displays the values of the byte or the bytes starting at the cursor for some common simple data types like Integer or Float, but also UTF-8. Double-clicking on a line in the table opens an editor, so the value can be edited and changed.

Structures

This tool enables investigating and editing of byte arrays based on user-creatable structure definitions. Detailed instructions are in an own section.

3.1.2 General tools

Filesystem

This tool offers an embedded file browser which can be used to select files to open.

Documents

This tool shows all currently created or loaded files. Symbols mark the file with the currently active view and also show which files have unsaved changes or which storage copy has been modified by another program.

Bookmarks

This tool can be used to manage the bookmarks, alternatively to the **Bookmarks** menu.

> NOTE
> Bookmarks are currently only transient and not saved if you close a byte array or the whole program.

File Info

This tool displays some information about the current file, including its type, the location of storage and the size.

Terminal

An embedded Terminal, the working directory is not coupled with the active file.

Charset Conversion

The tool rewrites the bytes so the respective chars are the same as with the other charset. Only 8-bit charsets are supported, and unmatched chars are currently substituted with a value hardcoded to 0.

3.2 Structures Tool

3.2.1 General

The Structures tool enables analyzing and editing of byte arrays based on user-creatable structure definitions, which can be built from arrays, unions, primitive types and enum values.

It has an own settings dialog, which can be reached by using the **Settings** button. There are various options that can be configured, like the style (decimal, hexadecimal or binary) in which the values are displayed. Moreover it is possible to choose which structure definitions get loaded and which structures are shown in the view.

Structures are defined in Okteta Structure Definition files (based on XML, with the file extension `.osd`). Additionally a `.desktop` file contains metadata about that structure description file, such as author, homepage and license.

Currently there is no built-in support for creating or editing structure definitions, therefore this must be done manually as described in the next sections.

3.2.2 Installing structure definitions

3.2.2.1 Installing using KNewStuff

The easiest way of installing new structure definitions is by using the built-in KNewStuff support in Okteta. To install an existing structure open the settings dialog of the Structures tool. There select the **Structures Management** tab and press the **Get New Structures...** button. The dialog that shows up now allows you to install and uninstall structures.

3.2.2.2 Installing structure definitions manually

The Structures tool looks for structure descriptions in the subdirectory `okteta/structures/` of the user's directory for program data (find that by executing **`qtpaths --paths GenericDataLocation`**). You may need to create this directory if there are no structure definitions installed yet.

Two files exist for every structure definition: One file for the actual definition and a `.desktop` file for the metadata (author, version, etc.).

In that directory there is a subdirectory for each structure definition, which contains both the `.desktop` file and the `.osd` or `main.js` file of that definition.

For example, with the program data directory `qtpaths --paths GenericDataLocation` and a structure definition named ExampleStructure there is the directory `okteta/structures/ExampleStructure` which contains a file `ExampleStructure.desktop` and a file `ExampleStructure.osd`.

3.2.2.3 Using the newly installed structures

After having installed a new structure definition you need to restart Okteta before you can use it. Once Okteta has started, open the settings dialog of the Structures tool. There select the **Structures Management** tab and make sure the relevant structure definition is checked. Then switch to the **Structures** tab and make sure the desired element is listed on the right-hand side.

3.2.3 Sharing structure definitions

For common structures you may not need to create a definition yourself, but instead can reuse an already existing definition from places like store.kde.org.

You also may want to share a definition yourself. To do so, create a file archive (e.g. a zipped tar archive, `.tar.gz`) containing just the subdirectory with the `.desktop` file and the structure definition file. Looking at the example in the last section this would be the directory `ExampleStructure` with all its contents. Using this format for sharing the structure definitions allows installing them inside Okteta and requires no manual installation.

3.2.4 Creating structure definitions

> NOTE
> A more up to date, but not completed guide to writing structure definitions can be found on the KDE UserBase Wiki.

There are two different ways of creating structure definitions. The first is writing the definition in XML the other is using JavaScript. The JavaScript approach allows you to create more complex structures with features like e.g. validating the structure. Using XML gives you less features but if a static structure is all you need this may be the easiest approach. If you need a dynamic structure i.e. where array lengths depend on other values in the structure or the structure layout is different when some member value changes, then you will have to write the structure definition in JavaScript. There is one exception to that rule: if you have an array where the length is supposed to be **exactly** the same as another value in the structure, then you can use XML. But if it is something like *length - 1* it has to be JavaScript.

3.2.5 Structure definition XML file format

> NOTE
> A more up to date, but not completed guide to writing structure definitions can be found on the KDE UserBase Wiki.

The `.osd` XML file has one root element: <*data*> with no attributes. Inside this element there must be one of the following elements:

<primitive>

To create a primitive data types like e.g. *int* and *float*. This element accepts no subelements and can have the following attributes:

type

The type of this primitive type. It must be one of the following:

- *char* for a 8 bit ASCII character
- *int8*, *int16*, *int32*, *int64* for a signed integer of that size
- *uint8*, *uint16*, *uint32*, *uint64* for an unsigned integer of that size
- *bool8*, *bool16*, *bool32*, *bool64* for an unsigned boolean (0 = false, any other value = true) of that size
- *float* for a 32 bit IEEE754 floating point number
- *double* for a 64 bit IEEE754 floating point number

<bitfield>

To create a bitfield. This element accepts no subelements and can have the following attributes:

width

The number of bits used by this bitfield. Must be between 1 and 64.

type

The type of this bitfield. It must be one of the following:

- *unsigned* for a bitfield where the value will be interpreted as an unsigned value (value range from 0 to $2^{width} - 1$)
- *signed* for a bitfield where the value will be interpreted as a signed value (value range from $-2^{width - 1}$ to $2^{width - 1} - 1$)
- *bool* for a bitfield where the value will be interpreted as a boolean value

> NOTE
> Always remember to add padding after a <*bitfield*>, since otherwise the next element (except for strings and arrays, since they add padding automatically) will start in the middle of a byte. Obviously padding is not necessary if you want this behavior.

<enum>

To create a primitive type, but where the values are displayed as members of an enumeration if possible. This element accepts no subelements (however you will need an <*enumDef*> tag in the file to reference it). It has the following attributes:

enum

The underlying enum for this value. Must match the *name* attribute of one of the <*enumDef*> tags in this file.

type

The type of this enum. See type attribute of <*primitive*>. Only difference is that *Double* and *Float* make no sense.

<flags>

This is the same as <*enum*> with the only difference being that values are represented as a *bitwise-or* of all the values of the enumeration.

<struct>

To create a structure. All other elements (including <*struct*>) can be a child of this and will then be part of the resulting structure

<union>

To create a union. Basically the same as <*struct*> except for the fact that all child elements will start from the same offset. Useful for interpreting the same sequence of bytes in various ways.

<array>

To create an array. This element accepts exactly one child (the underlying type of the array), which can be any of the elements, even <*array*> itself. It also has the following attributes:

length

The number of elements in this array as a decimal number. Alternatively it can also be a string which matches the name attribute of a previously defined <*primitive*>, <*enum*> or <*flags*> element. Then the length will be the value of that element. Currently it is limited to 10000, because larger arrays would use too much memory and slow down the tool too much.

<string>

To create a string in various encodings. By default you get a *NULL*- terminated C-style string. However different types of string can be created with the following attributes:

terminatedBy

This attribute determines what unicode codepoint the string is terminated by. It must be a hexadecimal number (optionally with leading *0x*). When encoding is ASCII, only values up to 0x7f are meaningful. If neither this nor *maxCharCount* nor *maxByteCount* are set, this is assumed to be set to 0 (C-style string)

maxCharCount

The maximum number of chars this string can have. If *terminatedBy* is set too then whatever is reached first terminates the string. This is mutually exclusive with *maxByteCount*

maxByteCount

The maximum number of bytes this string can be long. If *terminatedBy* is set too then whatever is reached first terminates the string. This is mutually exclusive with *maxCharCount*. With encodings like *ASCII* this is the same as *maxCharCount*.

type

The encoding of this string. Can be one of the following:

- *ASCII*

- *LATIN-1*
- *UTF-8*
- *UTF-16-LE* or *UTF-16-BE*. If neither *-LE* or *-BE* suffix is given, little endian is assumed.
- *UTF-32-LE* or *UTF-32-BE*. If neither *-LE* or *-BE* suffix is given, little endian is assumed.

Every element also accepts an attribute *name* which is then visible in the structures view.

3.2.6 An example structure definition in both XML and JavaScript

> NOTE
> A more up to date, but not completed guide to writing structure definitions can be found on the KDE UserBase Wiki.

3.2.6.1 The common step shared by both approaches

Our metadata file looks like this:

```
[Desktop Entry]
Icon=arrow-up<:\coref{1}{icon}:>
Type=Service
ServiceTypes=KPluginInfo

Name=Simple test structure
Comment=A very simple test structure containing only two items

X-KDE-PluginInfo-Author=Alex Richardson
X-KDE-PluginInfo-Email=foo.bar@email.org
X-KDE-PluginInfo-Name=simplestruct
X-KDE-PluginInfo-Version=1.0
X-KDE-PluginInfo-Website=https://www.plugin.org/
X-KDE-PluginInfo-Category=structure
X-KDE-PluginInfo-License=LGPL
X-KDE-PluginInfo-EnabledByDefault=false
```

❶ The icon displayed in Okteta for this structure can be anything found by executing **`kdialog --geticon`** or a path to an icon

These fields should all be pretty much self-explanatory, except for `X-KDE-PluginInfo-Name`. The value of this field must match the name of the directory containing the file as well as the name of the `.desktop` file. When creating XML structure definitions the name of the `.osd` file must also match the name.

In this example we would have a directory named `simplestruct` containing the file `simplestruct.desktop`. When defining structures in XML the directory would also contain a file named `simplestruct.osd`. Using JavaScript we would have a file named `main.js` instead.

3.2.6.2 A simple XML structure definition

To start we create a definition for a very simple test structure containing only integral data types (one char, one 32-bit signed integer, and a bitfield). This would be expressed in C/C++ as:

```
struct simple {
  char aChar;
  int anInt;
  bool bitFlag :1;
  unsigned padding :7;
};
```

The first step is writing the `.osd` file according to the file format defined in the previous section. We will call `simplestruct.osd`:

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <struct name="simple">
    <primitive name="aChar" type="Char"/>
    <primitive name="anInt" type="Int32"/>
    <bitfield name="bitFlag" type="bool" width="1"/>
    <bitfield name="padding" type="unsigned" width="7"/>
  </struct>
</data>
```

which is fairly similar to the C/C++ definition.

Now create a directory `simplestruct` inside the structure installation directory (see manually installing structure definitions) and copy the two files to this directory. Now you can restart Okteta and use the new structure.

3.2.6.3 The simple structure in JavaScript

To implement the structure above in JavaScript, create a file named `main.js` instead of `simplestruct.osd` and change X-KDE-PluginInfo-Category=structure to X-KDE-PluginInfo-Category=structure/js. The contents of that file should be:

```
function init() {
  var structure = struct({
    aChar : char(),
    anInt : int32(),
    bitFlag : bitfield("bool", 1),
    padding : bitfield("unsigned", 7),
  })
  return structure;
}
```

The structure displayed by Okteta is always the return value of the `init` function.

The following functions can be called to create a primitive type:

- char()
- int8(), int16(), int32() or int64()
- uint8(), uint16(), uint32() or uint64()
- bool8(), bool16(), bool32() or bool64()
- float()
- double()

The bitfield function takes two parameters, the first being a string consisting of `bool`, `signed` or `unsigned`. The second parameter is an integer which sets the width in bits.

3.2.6.4 More complex structures

Next we create a definition of a more complex structure which we will call "complex" and save in a file named `complex.osd`. This structure will contain two arrays (one with fixed length and one where the length is determined at runtime) as well as a nested structure and a union.

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <struct name="complex">
    <primitive name="size" type="UInt8" />
    <union name="aUnion">
      <array name="fourBytes" length="4">
        <primitive type="Int8" />
      </array>
    </union>
    <struct name="nested">
      <array name="string" length="size"> <!-- references the  ↩
          field size above -->
        <primitive type="Char" />
      </array>
    </struct>
  </struct>
</data>
```

This would correspond to the following in pseudo-C/C++

```
struct complex {
  uint8_t size;
  union aUnion {
    int8_t fourBytes[4];
  };
  struct nested {
    char string[size] //not valid C++, references value of the  ↩
        uint8 size
  };
};
```

> **NOTE**
> You can obviously only have dynamic length arrays reference fields before the array.

Next we create the `complex.desktop` file just as in the example before (make sure you set `X-KDE-PluginInfo-Name` correctly) and also do the same to install both files.

3.2.6.5 Further information

A few example structure definitions can be found in the Git repository. This includes for example the file header for PNG files and the ELF file header. An XML schema describing the structure of the `.osd` file can be found here. If more information is needed feel free to contact me at arichardson.kde@gmail.com

Chapter 4

Interface Overview

4.1 Menu Items

Apart from the common KDE menus described in the Menu chapter of the KDE Fundamentals documentation Okteta has these application specific menu entries:

4.1.1 File Menu

File → New (Ctrl+N)

Create a new byte array...

- **Empty**: ... as Empty one.
- **From Clipboard**: ... by the current content of the clipboard.
- **Pattern...**: ... with a given pattern.
- **Random Data...**: ...with random data.
- **Sequence**: ... with all the bytes from 0 to 255.

File → Export

Export the selected bytes to a file...

- **Values**: ... encoded as byte values. By default the values are separated with one whitespace. The **Separation** characters can be changed in the **Export** dialog.
- **Characters**: ... encoded as plain text.
- **Base64**: ... encoded in the Base64 format.
- **Base32**: ... encoded in the Base32 format.
- **Ascii85**: ... encoded in the Ascii85 format.
- **Uuencoding**: ... encoded in the Uuencoding format.
- **Xxencoding**: ... encoded in the Xxencoding format.
- **Intel Hex**: ... encoded in the Intel Hex format.
- **S-Record**: ... encoded in the S-Record format.
- **C array**: ... defined as an array in the programming language C.
- **View in Plain Text**: ... as in the data view with offset, byte values and characters.

File → Permissions → Set Read-only

When set, changes may not be made to the loaded byte array.

File → Close All Other

Close all besides the current byte array.

4.1.2 Edit Menu

Edit → Copy as

Copy the selected bytes in one of different formats to the clipboard. For a list of available formats see the menu item **File → Export**

Edit → Insert

Insert Pattern...

Insert a specified string of bytes at the cursor.

Options in the dialog box allow you to specify the number of insertion of the pattern and it's format (Hexadecimal, Decimal, Octal, Binary, Character(s) or UTF-8).

Edit → Deselect (Ctrl+Shift+A)

Deselect the current selection.

Edit → Select Range... (Ctrl+E)

Opens an embedded dialog to enter the range to select.

Edit → Overwrite Mode (Ins)

Switch between Insert mode and Overwrite mode.

> NOTE
> Overwrite mode is implemented to be very strict, it is not possible to change the size of the data (no appending or removing of bytes).

Edit → Find... (Ctrl+F)

Find a specified pattern in the document. Hexadecimal, decimal, octal, binary or text patterns can be searched for.

Options in the dialog box allow you to specify the starting point, direction and range of the search.

Edit → Goto Offset... (Ctrl+G)

Move the cursor to a specified offset.

4.1.3 View Menu

View → Show Line Offset (F11)

Toggle display of the line offset on a pane to the left on and off.

View → Show Values or Chars

Select which of the byte interpretations are shown. Possible are:

- **Values**
- **Chars**
- **Values and Chars**

View → Value Coding

Select the coding of the values from:

- **Hexadecimal**
- **Decimal**
- **Octal**

- **Binary**

View → **Char Coding**

Select the coding of the chars from the submenu.

View → **Show Non-printing Chars**

Toggle display of non-printing chars on and off. If the display is turned off, at the corresponding places in the character column a substitute char is shown instead.

View → **Set Bytes per Line**

Select the displayed bytes per line from the dialog, the default value is 16 bytes.

View → **Set Bytes per Group**

By default the hexadecimal values are displayed in groups of 4 bytes. Using this menu item you can adapt this to your preferences in a dialog.

View → **Dynamic Layout**

Set the rules for the layout of the data display. This defines how many bytes are displayed per line, depending on the width of the view. Possible rules are:

- **Off**: The layout is fixed to the current number of bytes per line and not adapted on the change of the view size.
- **Wrap only complete byte groups**: Puts as many bytes per line as possible, as long as groups of bytes are complete.
- **On**: Same as previous, but allows also incomplete groups of bytes.

View → **View Mode**

Select the layout for the view from:

- **Columns**: The values and chars interpretations are shown in the classic layout with each listed in a separate column.
- **Rows**: The char interpretation of a byte is directly shown under the value interpretation.

View → **Split Horizontal (Ctrl+Shift+T)**

Split the view area with the currently focused view into two parts and add a copy of the current view to the new, lower area.

View → **Split Vertically (Ctrl+Shift+L)**

Split the view area with the currently focused view into two parts and add a copy of the current view to the new, right area.

View → **Close View Area (Ctrl+Shift+R)**

Close the view area with the currently focused view.

View → **View Profile**

View settings can be separately stored as view profiles. The currently selected profile can be updated directly from the current view settings, or a new one can be created from them. All view profiles can be managed in a dialog available from **Settings** → **Manage View Profiles...**.

4.1.4 Windows Menu

Provides a list of the current views. Select the active window.

4.1.5 Bookmarks Menu

Multiple bookmarks can be set for a single byte array. Each byte array has its own set of bookmarks, and the appropriate set is displayed at the bottom of the **Bookmarks** menu. Choose a bookmark from the menu to move the cursor and the view to it.

> NOTE
> Bookmarks are currently only transient and not saved if you close a byte array or the whole program.

Bookmarks → Add Bookmark (Ctrl+B)

Bookmark a location within the byte array.

Bookmarks → Remove Bookmark (Ctrl+Shift+B)

Remove the current bookmark. This command is only available if the cursor is at a bookmarked location.

Bookmarks → Remove All Bookmarks

Clear the bookmark list.

Bookmarks → Goto Previous Bookmark (Alt+Up)

Move the cursor to the previous bookmark.

Bookmarks → Go to Next Bookmark (Alt+Down)

Move the cursor to the next bookmark.

4.1.6 Tools Menu

Provides a list of installed tools. Toggle the display of each tools on or off. A detailed description of each tool you find in the Tools section.

4.1.7 Settings Menu

Settings → Manage View Profiles...

Open a dialog to create, edit, delete and set a default view profile.

Chapter 5

Credits and License

Okteta