OpenCS User Manual

OpenMW Team

February 3, 2016

Contents

1 Files and Directories

1.1 Introduction

This section of the manual describes the directories and file types used by OpenCS. A file is a resource for storing data (e.g. .exe, .jpg, .txt), whereas a directory is a folder or file system structure which points to these files (or other directories). You are most likely already familiar with these concepts.

1.2 Used terms

1.3 Basics

Directories OpenMW and OpenCS store their files in multiple directories. Firstly, there is the **user directory** that holds configuration files and several other folders. The location of the user directory is hard coded for each supported operating system.

In addition to the user directory, both OpenMW and OpenCS need a place to store the game's actual data files: for example, the textures, models, sounds and records of in-game objects. We support multiple paths to these files (termed **data paths**), as specified in the configuration. Usually, one data path points to the directory where *Morrowind*^{TM} is installed; however, you are free to specify as many data paths as you would like. In addition, one particular data path, as described below, is used to store newly created content files.

Content files Bethesda Softworks *Morrowind*^{TM} engine uses two file types: ESM (master) and ESP (plugin). The distinction between the two is often confusing. You would expect that the ESM (master) file is used to specify a single master which is modified by the ESP files (plugins), and indeed: this is the basic idea. However, the original expansions are also ESM files, even though they can be described as very large plugins. There were technical reasons behind this decision – somewhat valid in the case of the original engine – but a more logical file system is much preferable. OpenMW achieves this through the creation of our own types of content file.

We support both ESM and ESP files, but, in order to make use of OpenMW's new features, one should consider using new file types designed with our engine in mind: game files and addon files, collectively termed **content files**.

OpenMW content files The distinction between game and addon files is similar to that between ESM and ESP, however their relationship to each other is strictly defined -– the former are always master files, and the latter are always plugins. If you want to make a new game using the OpenMW engine (i.e. a "total conversion"), you should create a game file. If you want to create an addon for an existing game file, simply create an addon file. Nothing else matters: the only distinction you should consider is whether your project involves changing another game, or creating a new one. Simple as that.

Furthermore, our content files' extensions are .omwaddon for addon files and .omwgame for game files.

***Morrowind*™ content files** Using our content files is the recommended solution for projects that employ the OpenMW engine. However, some players will wish to use the original *Morrowind*™ engine, despite its large flaws and lacking features[1]. In addition, since 2002, thousands of ESP/ESM files have been created, some with truly outstanding content. Because of this, OpenCS will support ESP/ESM files, although this will impose limitations on the user. If you do decide to use ESP/ESM files rather than our own content files, you are most likely aiming for original engine compatibility. This subject is covered in the very last section of the manual.

The actual creation of new files is described in the next chapter. Here we are going to focus only on the essential information needed to create your first OpenCS file. For now, let's jut remember that content files are stored in the user directory, in the **data** subfolder (the particular data directory mentioned above).

Dependencies Since addons aim to modify an existing game, it is logical that they also depend on the said game: otherwise they will not function. For example, your modification changes the price of iron swords. But what if there are no iron swords in the game? That is right: it is nonsense. Therefore, it is necessary to make your addon a dependency of other content files. These can be either game files (e.g. an entirely new island), or other addon files (e.g. a house on the island). It is a good idea for addons to depend only on the content files they modify, but this is up to the end user to determine.

Game files do not depend on any other content files, as they act as master files. A player can only use one game file at a time (although this does not apply to the original and dirty ESP/ESM system).

Project files Project files contain data not used by the OpenMW game engine but which are still needed by OpenCS. Good examples of this data type are the record filters (described below). As a mod author, you probably do not need and/or want to distribute project files at all, as they are meant to be used only by you.

Since project files govern how content files are used in OpenCS, they are always used in conjunction with your specific project. In fact, each time work commences on a content file that does not have a corresponding project file, a new project file will be created.

The project file extension is ".project". The name of the project file is the whole name of the content file with appended extensions. For instance, a content file named swords.omwaddon is associated with the project file swords.omwaddon.project.

Project files are stored inside the user directory, in the **projects** subfolder. This is both the location of newly created project files, and the place where OpenCS looks for already existing files.

Resource files The vast majority of modern video games use what we shall term **resource files**: models, textures, icons, sounds and so on. ESPs, ESMs and OpenMW content files do not contain these files, merely instructions on how they are used. It

[1] If this is wrong, we are a very successful project. Yay!

follows that the OpenMW engine must be capable of supporting these resource files in order for them to function. Therefore this section covers ways to add resource files to your content file, and outlines which formats are supported. Later, you will learn how to make use of these files in your content.

Audio OpenMW utilises FFmpeg for audio playback, so we support every audio type supported by this library. This is a huge list. Below is only a small sample of supported file types.

mp3 (MPEG-1 Part 3 Layer 3) A popular audio file format and the *de facto* standard for storing audio. Used by the *Morrowind*^{TM} game.

ogg Open source, multimedia container file which uses the high quality vorbis audio codec. Recommended.

Video As in the case of audio files, we use FFmpeg to decode video files. The list of supported files is long -– only the most significant will be covered.

bik Format used by the original *Morrowind*^{TM} game.

mp4 Multimedia container which use more advanced codecs (MPEG-4 Parts 2,3,10) with a better audio and video compression rate, but which require more CPU intensive decoding – this probably makes it less suited for storing sounds in computer games, but good for videos.

webm A new, shiny and open source video format with excellent compression. It needs quite a lot of processing power to be decoded, but since game logic is not running during cut scenes we can recommend it for use with OpenMW.

ogv An alternative, open source container using theora codec for video and vorbis for audio.

Textures and images *Morrowind*^{TM} uses DDS and TGA files for all kinds of two dimensional images and textures. In addition, the original engine supported BMP files (although BMP is a terrible format for a video game). We also support an extended set of image files – including JPEG and PNG. JPEG and PNG files can be useful in some cases. For instance, a JPEG file is a valid option for a skybox texture and PNG can useful for masks. However, keep in mind that a JPEG can grow large quickly and so are not the best option with a DirectX rendering backend. DDS files are therefore recommended for textures.

2 OpenCS starting dialog

2.1 Introduction

The great day has come. Today, you shall open OpenCS application. And when you do this, you shall see our starting dialog window that holds three buttons that can bring both pain and happiness. So just do this, please.

2.2 Basics

Back to the manual? Great! As you can see, the starting window holds just three buttons. Since you are already familiar with our files system, they come to you with no surprise.

First, there is a **Create A New Game** button. Clearly, you should press it when you want to create a game file. Than, what **Create A New Addon** button do? Yes! You are right! This button will create any addon content file (and new project file associated with it)! Wonderful! And what the last remaining button do? **Edit A Content File**? Well, it comes with no surprise that this should be used when you need to alter existing content file, either a game or addon.

Selecting Files For New Addon As We wrote earlier, both OpenMW and OpenCS are operating with dependency idea in mind. As You remember you should only depend on files you are actually using. But how?
It is simple. When you click either **Create new Addon** you will be asked to choose those with a new dialog window. The window is using vertical layout, first you should consider the the top element, the one that allows you to select a game file with drop down menu. Since we are operating on the assumption that there is only one game file loaded at the time, you can depend only on one game file. Next, choose addons that you want to use in your addon with checkboxes.

The last thing to do is to name your your addon and click create.

Selecting File for Editing Clicking **Edit A Content File** will show somewhat similar window. Here you should select your Game file with drop down menu. If you want to edit this game file, simply click **OK** button. If you want to alter addon depending on that file, mark it with check-box and than click **Ok** button.

2.3 Advanced

If you are paying attention, you noticed any extra icon with wrench. This one will open small settings window. Those are general OpenCS settings. We will cover this is separate section.

And that would be it. There is no point spending more time here. We should go forward now.

3 Windows

3.1 Introduction

This section describes the multiple windows interface of the OpenCS editor. This design principle was chosen in order to extend the flexibility of the editor, especially on the multiple screens setups and on environments providing advanced windows management features, like for instance: multiple desktops found commonly on many open source desktop environments. However, it is enough to have a single large screen to see the advantages of this concept.

OpenCS windows interface is easy to describe and understand. In fact we decided to minimize use of many windows concepts applied commonly in various applications. For instance dialog windows are really hard to find in the OpenCS. You are free to try, though.

Because of this, and the fact that we expect that user is familiar with other applications using windows this section is mostly focused on practical ways of organizing work with the OpenCS.

3.2 Basics

After starting OpenCS and choosing content files to use a editor window should show up. It probably does not look surprising: there is a menubar at the top, and there is a large empty area. That is it: a brand new OpenCS window contains only menubar and statusbar. In order to make it a little bit more useful you probably want to enable some panels[2]. You are free to do so, just try to explore the menubar.

You probably founded out the way to enable and disable some interesting tables, but those will be described later. For now, let's just focus on the windows itself.

Creating new windows is easy! Just visit view menu, and use the "New View" item. Suddenly, out of the blue a new window will show up. As you would expect, it is also blank, and you are free to add any of the OpenCS panels.

Closing opened window is also easy! Simply close that window decoration button. We suspect that you knew that already, but better to be sure. Closing last OpenCS window will also terminate application session.

Multi-everything is the main foundation of OpenCS interface. You are free to create as many windows as you want to, free to populate it with any panels you may want to, and move everything as you wish to – even if it makes no sense at all. If you just got crazy idea and you are wonder if you are able to have one hundred OpenCS windows showing panels of the same type, well most likely you are able to do so.

The principle behind this design decision is easy to see for Bethesda Softworks made editor, but maybe not so clear for users who are just about to begin their wonderful journey of modding.

[2] Also known as widgets.

3.3 Advanced

So why? Why this is created in such manner. The answer is frankly simple: because it is effective. When creating a mod, you often have to work only with just one table. For instance you are just balancing weapons damage and other statistics. It makes sense to have all the space for just that one table. More often, you are required to work with two and switch them from time to time. All major graphical environments commonly present in operating systems comes with switcher feature, that is a key shortcut to change active window. It is very effective and fast when you have only two windows, each holding only one table. Sometimes you have to work with two at the time, and with one from time to time. Here, you can have one window holding two tables, and second holding just one.

OpenCS is designed to simply make sense and do not slowdown users. It is as simple as possible (but not simpler), and uses one flexible approach in all cases.

There is no point in digging deeper in the windows of OpenCS. Let's explore panels, starting with tables.

4 Tables

4.1 Introduction

If you have launched OpenCS already and played around with it for a bit, you have probably gotten the impression that it contains lots of tables. You'd be spot on: OpenCS is built around using tables. This does not mean it works just like Microsoft Excel or Libre Office Calc, though. Due to the vast amounts of information involved with *Morrowind*™, tables just made the most sense. You have to be able to spot information quickly and be able to change them on the fly. Let's browse through the various screens and see what all these tables show.

4.2 Used Terms

4.2.1 Glossary

Record: An entry in OpenCS representing an item, location, sound, NPC or anything else.

Instance, Object: When an item is placed in the world, it isn't an isolated and unique object. For example, the game world might contain a lot of exquisite belts on different NPCs and in many crates, but they all refer to one specific record in the game's library: the Exquisite Belt record. In this case, all those belts in crates and on NPCs are **instances**. The central Exquisite Belt record is called a **object**. This allows modders to make changes to all items of the same type. For example, if you want all exquisite belts to have 4000 enchantment points rather than 400, you will only need to change the **object** Exquisite Belt rather than all exquisite belts **instances** individually.

4.2.2 Recurring Terms

Some columns are recurring throughout OpenCS. They show up in (nearly) every table in OpenCS.

ID Many items in the OpenCS database have a unique identifier in both OpenCS and Morrowind. This is usually a very self-explanatory name. For example, the ID for the (unique) black pants of Caius Cosades is "Caius_pants". This allows you to manipulate the game in many ways. For example, you could add these pants to your inventory by simply opening the console and write: "player->addItem Caius_pants". Either way, in both Morrowind and OpenCS, the ID is the primary way to identify all these different parts of the game.

Modified This column shows what has happened (if something has happened) to this record. There are four possible states in which it can exist.

Base means that this record is part of the base game and is in its original state. Usually, if you create a mod, the base game is Morrowind with optionally the Bloodmoon and Tribunal expansions.

Added means that this record was not in the base game and has been added by a modder.

Modified means that the record is part of the base game, but has been changed in some way.

Deleted means that this record used to be part of the base game, but has been removed as an entry. This does not mean, however, that the occurrences in the game itself have been removed! For example, if you remove the CharGen_Bed entry from morrowind.esm, it does not mean the bedroll in the basement of the Census and Excise Office in Seyda Neen is gone. You're going to have to delete that instance yourself or make sure that that object is replaced by something that still exists otherwise you will get crashes in the worst case scenario.

4.3 World Screens

The contents of the game world can be changed by choosing one of the options in the appropriate menu at the top of the screen.

4.3.1 Regions

This describes the general areas of the gameworld. Each of these areas has different rules about things such as encounters and weather.

Name: This is how the game will show your location in-game.

Map Colour: This is a six-digit hexadecimal representation of the color used to identify the region on the map available in World > Region Map. If you do not have an application with a color picker, you can use your favorite search engine to find a color picker on-line.

Sleep Encounter: These are the rules for what kind of enemies you might encounter when you sleep outside in the wild.

4.3.2 Cells

Expansive worlds such as Vvardenfell, with all its items, NPCs, etc. have a lot going on simultaneously. But if you are in Balmora, why would the computer need to keep track the exact locations of NPCs walking through the corridors in a Vivec canton? All that work would be quite useless and bring your system to its knees! So the world has been divided up into squares we call "cells". Once your character enters a cell, the game will load everything that is going on in that cell so you can interact with it.

In the original *Morrowind*^{TM} this could be seen when you were traveling and you would see a small loading bar at the bottom of the screen; you had just entered a new cell and the game would have to load all the items and NPCs. The Cells screen in OpenCS provides you with a list of cells in the game, both the interior cells (houses, dungeons, mines, etc.) and the exterior cells (the outside world).

Sleep Forbidden: Can the player sleep on the floor? In most cities it is forbidden to sleep outside. Sleeping in the wild carries its own risks of attack, though, and this entry lets you decide if a player should be allowed to sleep on the floor in this cell or not.

Interior Water: Should water be rendered in this interior cell? The game world consists of an endless ocean at height 0. Then the landscape is added. If part of the landscape goes below height 0, the player will see water. (See illustration.)

Setting the cell's Interior Water to true tells the game that this cell is both an interior cell (inside a building, for example, rather than in the open air) but that there still needs to be water at height 0. This is useful for dungeons or mines that have water in them.

Setting the cell's Interior Water to false tells the game that the water at height 0 should not be used. Remember that cells that are in the outside world are exterior cells and should thus *always* be set to false!

Interior Sky: Should this interior cell have a sky? This is a rather unique case. The *Tribunal* expansion took place in a city on the mainland. Normally this would require the city to be composed of exterior cells so it has a sky, weather and the like. But if the player is in an exterior cell and looks at his in-game map, he sees the map of the gameworld with an overview of all exterior cells. The player would have to see the city's very own map, as if he was walking around in an interior cell.

So the developers decided to create a workaround and take a bit of both: The whole city would technically work exactly like an interior cell, but it would need a sky as if it was an exterior cell. That is what this is. This is why the vast majority of the cells you will find in this screen will have this option set to false: It is only meant for these "fake exteriors".

Region: To which Region does this cell belong? This has an impact on the way the game handles weather and encounters in this area. It is also possible for a cell not to belong to any region.

4.3.3 Objects

This is a library of all the items, triggers, containers, NPCs, etc. in the game. There are several kinds of Record Types. Depending on which type a record is, it will need specific information to function. For example, an NPC needs a value attached to its aggression level. A chest, of course, does not. All Record Types contain at least a model. How else would the player see them? Usually they also have a Name, which is what you see when you hover your reticle over the object.

This is a library of all the items, triggers, containers, NPCs, etc. in the game. There are several kinds of Record Types. Depending on which type a record is, it will need specific information to function. For example, an NPC needs a value attached to its aggression level. A chest, of course, does not. All Record Types contain at least a model. How else would the player see them? Usually they also have a Name, which is what you see when you hover your reticle over the object.

Please refer to the Record Types section for an overview of what each type of object does and what you can tell OpenCS about these objects.

5 Record Types

5.1 Introduction

A gameworld contains many items, such as chests, weapons and monsters. All these items are merely instances of templates that we call **Objects**. The OpenCS **Objects** table contains information about each of these template objects, eg. its value and weight in the case of items and an aggression level in the case of NPCs.

Let's go through all Record Types and discuss what you can tell OpenCS about them.

Activator: When the player enters the same cell as this object, a script is started. Often it also has a **Script** attached to it, though it not mandatory. These scripts are small bits of code written in a special scripting language that OpenCS can read and interpret.

Potion: This is a potion that is not self-made. It has an **Icon** for your inventory, Aside from the self-explanatory **Weight** and **Coin Value**, it has an attribute called **Auto Calc** set to "False". This means that the effects of this potion are pre-configured. This does not happen when the player makes their own potion.

Apparatus: This is a tool to make potions. Again there's an icon for your inventory as well as a weight and a coin value. It also has a **Quality** value attached to it: higher the number, the better the effect on your potions will be. The **Apparatus Type** describes if the item is a Calcinator, Retort, Alembic or Mortar & Pestle.

Each has a different effect on the potion the player makes. For more information on this subject, please refer to the UESP page on Alchemy Tools.

Armor: This type of item adds **Enchantment Points** to the mix. Every piece of clothing or armor has a ”pool” of potential Magicka that gets unlocked when you enchant it. Strong enchantments consume more Magicka from this pool: the stronger the enchantment, the more Enchantment Points each cast will take up. For more information on this subject, please refer to the Enchant page on UESP. **Health** means the amount of hit points this piece of armor has. If it sustains enough damage, the armor will be destroyed. Finally, **Armor Value** tells the game how much points to add to the player character's Armor Rating.

Book: This includes scrolls and notes. For the game to make the distinction between books and scrolls, an extra property, **Scroll**, has been added. Under the **Skill** column a scroll or book can have an in-game skill listed. Reading this item will raise the player's level in that specific skill. For more information on this, please refer to the Skill Books page on UESP.

Clothing: These items work just like Armors, but confer no protective properties. Rather than “Armor Type”, these items have a “Clothing Type”.

Container: This is all the stuff that stores items, from chests to sacks to plants. Its **Capacity** shows how much stuff you can put in the container. You can compare it to the maximum allowed load a player character can carry (who will get over-encumbered and unable to move if he crosses this threshold). A container, however, will just refuse to take the item in question when it gets ”over-encumbered”. **Organic Container**s are containers such as plants. Containers that **Respawn** are not safe to store stuff in. After a certain amount of time they will reset to their default contents, meaning that everything in it is gone forever.

Creature: These can be monsters, animals and the like.

6 Record filters

6.1 Introduction

Filters are the key element of OpenCS use cases by allowing rapid and easy access to the searched records presented in all tables. Therefore: in order to use this application fully effective you should make sure that all concepts and instructions written in the this section of the manual are perfectly clear to you.

Do not be afraid though, filters are fairly intuitive and easy to use.

6.1.1 Used Terms

Filter is generally speaking a tool able to “Filter” (that is: select some elements, while discarding others) according to the some criteria. In case of OpenCS: records are being filtered according to the criteria of user choice. Criteria are written down in language with simple syntax.

Criteria describes condition under with any any record is being select by the filter.

Syntax as you may noticed computers (in general) are rather strict, and expect only strictly formulated orders – that is: written with correct syntax.

Expression is way we are actually performing filtering. Filter can be treated as "functions": accepts arguments, and evaluates either to the true or false for every column record at the time.

N-ary is any expression that expects one or more expressions as arguments. It is useful for grouping two (or more) other expressions together in order to create filter that will check for criteria placed in two (again: or more) columns (logical *or*, *and*).

unary is any expression that expects one other expression. The example is *not* expression. In fact *not* is the only useful unary expression in OpenCS record filters.

nullary is expression that does not accepts other expressions. It accepts arguments specified later.

6.1.2 Basics

In fact you do not need to learn everything about filters in order to use them. In fact all you need to know to achieve decent productivity with OpenCS is inside basics section.

6.1.3 Interface

Above each table there is a field that is used to enter filter: either predefined by the OpenMW developers or made by you, the user. You probably noticed it before. However there is also completely new element, although using familiar table layout. Go to the application menu view, and click filters. You should see set of default filters, made by the OpenMW team in the table with the following columns: filter, description and modified.

ID contains the name of the filter.

Modified just like in all other tables you have seen so far modified indicates if a filter was added, modified or removed.

Filter column containing expression of the filter.

Description contains the short description of the filter function. Do not expect any surprises there.

So let's learn how to actually use those to speed up your work.

6.1.4 Using predefined filters

Using those filters is quite easy and involves typing inside the filter field above the table. For instance, try to open referencables table and type in the filters field the following: `project::weapons`. As soon as you complete the text, table will magicly alter and will show only the weapons. As you could noticed `project::weapons` is nothing else than a ID of one of the predefined filters. That is it: in order to use the filter inside the table you simply type it is name inside the filter field.

To make life easier filter IDs follow simple convention.

- Filter ID filtering a specific record type contains usually the name of a specific group. For instance `project::weapons` filter contains the word weapons (did you noticed?). Plural form is always used.
- When filtering specific subgroup the ID starts just like in the case of general filter. For instance `project::weaponssilver` will filter only silver weapons (new mechanic introduced by the *Bloodmon*, silver weapons deal double damage against werewolfs) and `project::weaponsmagical` will filter only magical weapons (able to hurt ghosts and other supernatural creatures).
- There are few exceptions from the above rule. For instance there is a `project::added`, `project::removed`, `project::modyfied`, `project::base`. You would probably except something more like `project::statusadded` but in this case typing this few extra characters would only help to break your keyboard faster.

We strongly recommend to take a look at the filters table right now to see what you can filter with that. And try using it! It is very simple.

6.2 Advanced

Back to the manual? Great.

If you want to create your own filter you have to know exactly what do you want to get in order to translate this into the expressions. Finally, you will have to write this with correct syntax. As a result table will show only desired rows.

Advance subsection covers everything that you need to know in order to create any filter you may want to

6.2.1 Namespaces

Did you noticed that every default filter has `project::` prefix? It is a *namespace*, a term borrowed from the C++ language. In case of OpenCS namespace always means scope of the said object[3]. But what does it mean in case of filters? Well, short explanation is actually simple.

project:: namespace indicates that filter is used with the project, in multiple sessions. You can restart OpenCS and filter is still there.

[3] You are not supposed to understand this at the moment.

session:: namespace indicates that filter is not stored trough multiple sessions and once you will quit OpenCS (close session) the filter will be gone. Forever! Until then it can be found inside the filters table.

In addition to this two scopes, there is a third one; called one-shot. One-shot filters are not stored (even during single session) anywhere and as the name implies they are supposed to be created when needed only once. Good thing about the one-shot filters is that you do not need to open filters table in order to create it. Instead you just type it directly inside the filter field, starting with exclamation mark: "!".

Still, you may wonder how you are supposed to write expressions, what expressions you should use, and what syntax looks like. Let's start with nullary expressions that will allow you to create a basic filter.

6.2.2 Nullary expressions

All nullary expressions are used in similar manner. First off: you have to write it is name (for instance: `string`) and secondly: condition that will be checked inside brackets (for instance `string(something, something)`). If conditions of your expression will be meet by a record (technical speaking: expression will evaluate to true) the record will show up in the table.

It is clear that you need to know what are you checking, that is: what column of the table contains information that you are interested in and what should be inside specific cell inside this column to meet your requirements. In most cases first word inside brackets sets column you want to see, while the second one sets desired value inside of the cell. To separate column argument from the value argument use comma.

String – string("column", "value") String in programmers language is often[4] just a word for anything composed of characters. In case of OpenCS this is in fact true for every value inside the column that is not composed of the pure numbers. Even columns containing only "true" and "false" values can be targeted by the string expression[5]. String evaluates to true, when record contains in the specified column exactly the same value as specified.

Since majority of the columns contain string values, string is among the most often used expressions. Examples:

- `string("Record Type", "Weapon")` – will evaluate to true for all records containing `Weapon` in the `Record Type` column cell. This group contains every weapon (including arrows and bolts) found in the game.
- `string("Portable", "true")` – will evaluate to true for all records containing word true inside `Portable` column cell. This group contains every portable light sources (lanterns, torches etc.).

This is probably enough to create around 90 string filters you will eventually need. However, this expression is even more powerful – it accepts regular expressions (also

[4] Often, not always. There are different programming languages using slightly different terms.
[5] There is no Boolean ("true" or "false") value in the OpenCS. You should use string for those.

called regexps). Regular expressions is a way to create string criteria that will be matched by one than just one specific value in the column. For instance, you can display both left and right gauntlets with the following expression: `string("armor type", ".* gauntlet"))` because `.*` in regexps means just: "anything". This filter says: please, show me "any" gauntlet. There are left and right gauntlets in the *Morrowind*™ so this will evaluate to true for both. Simple, isn't it?

Creating regexps can be a difficult and annoying – especially when you need complex criteria. On the other hand, we are under impression that in reality complex expressions are needed only in sporadic cases. In fact, the truth is: that most of the time only already mentioned `.*` is needed and therefore the following description of regexps can be skipped by vast majority of readers.

Before working with Regular Expressions, you should understand what actually are regular expressions. Essentially, the idea is simple: when you are writing any word, you are using strictly defined letters – that is: letters create a word. What you want to do with regular expression is to use set of rules that will match to many words. It is not that difficult to see what it's needed to do so: first, you will clearly need way to determinate what letters you want to match (word is composed by letters).

Before introducing other ways to choose between characters, I want explain anchors. Anchors allows you to decide where to "look" in the string. You surely should know about `^` anchor and `$`. Putting `^` will tell to OpenCS to look on the beginning of string, while `$` is used to mark the end of it. For instance, pattern `^Pink.* elephant.$` will match any sentence beginning with the word `Pink` and ending with `elephant.`. Pink fat elephant. Pink cute elephant. It does not matter what is in between because `.*` is used.

You have already seen the power of the simple `.*`. But what if you want to chose between only two (or more) letters? Well, this is when `[|]` comes in handy. If you write something like: `^[a|k].*` you are simply telling OpenCS to filter anything that starts with either `a` or `k`. Using `^[a|k|l].*` will work in the same manner, that is; it will also cover strings starting with `l` as well.

What if you want to match more than just one latter? Just use `(|)`. It is pretty similar to the above one letter as you see, but it is used to fit more than just one character. For instance: `^(Pink|Green).* (elephant|crocodile).$` will be true for all sentences starting with `Pink` or `Green` and ending with either `elephant.` or `crocodile.`.

Regular expressions are not the main topic of this manual. If you wish to learn more on this subject please, read the documentation on Qt regular expressions syntax, or TRE regexp syntax (it is almost like in Qt). Above is just enough to use OpenCS effectively.

Value – value("value", ("open", "close")) While string expression covers vast group of columns containing string values, there are in fact columns with just numerical values like "weight". To filter those we need a value expression. This one works in similar manner to the string filter: first token name and criteria inside brackets. Clearly, conditions should hold column to test in. However in this case wanted value is specified as a range. As you would imagine the range can be specified as including a border value, or excluding. We are using two types of brackets for this:

- To include value use [] brackets. For value equal 5, expression `value(something, [5, 10])` will evaluate to true.
- To exclude value use () brackets. For value equal 5, expression `value(something, (5, 10))` will evaluate to false.
- Mixing brackets is completely legal. For value equal 10, expression `value(something, [5, 10)` will evaluate to true. The same expression will evaluate to false for value equal 10.

"true" and "false" Nullary *true* and *false* do not accept any arguments, and always evaluates to true (in case of *true*) and false (in case of *false*) no matter what. The main usage of this expressions is the give users ability to quickly disable some part of the filter that makes heavy use of the logical expressions.

6.2.3 Logical expressions

This subsection takes care of two remaining groups of expressions: binary and unary. The only unary expression present in the OpenCS is logical *not*, while the remaining binary expressions are: *or*, *and*. This clearly makes them (from the user point of view) belonging to the same group of logical expressions.

not – not expression() Sometimes you may be in need of reversing the output of the expression. This is where *not* comes in handy. Adding *not* before expression will revert it: if expression was returning true, it will return false; if it was returning false, it will return true. Parenthesis are not needed: *not* will revert only the first expression following it.

To show this on know example, let's consider the `string("armor type", ".* gauntlet"))` filter. As we mentioned earlier this will return true for every gauntlet found in game. In order to show everything, but gauntlets we simply do `not string("armor type", ".* gauntlet"))`. This is probably not the most useful filter on earth. The real value of *not* expression shines when combined with *or*, *and* filters.

or – or(expression1(), expression2()) *Or* is a expression that will return true if one of the arguments evaluates to true. You can use two or more arguments, separated by the comma.

Or expression is useful when showing two different group of records is needed. For instance the standard actor filter is using the following `or(string("record type", npc), string("record type", creature))` and will show both npcs and creatures.

and – and(expression1(), expression2()) *And* is a expression that will return true if all arguments evaluates to true. As in the case of "or" you can use two or more

arguments, separated by a comma. As we mentioned earlier in the *not* filter, combining *not* with *and* can be very useful. For instance to show all armor types, excluding gauntlets you can write the following: `and (not string("armor type", ".* gauntlet"), string("Record Type", "Armor"))`.

6.2.4 Creating and saving filter

In order to create and save new filter, you should go to the filters table, right click and select option "add record" from the context menu. A horizontal widget group at the bottom of the table should show up. From there you should select a namespace responsible for scope of the filter (described earlier) and desired ID of the filter. After pressing OK button new entry will show up in the filters table. This filter does nothing at the moment, since it still lacks expressions. In order to add your formula simply double click the filter cell of the new entry and write it down there. Done! You are free to use your filter.

6.2.5 Replacing the default filters set

OpenCS allows you to substitute default filters set provided by us, with your own filters. In order to do so you should create a new project, add desired filters, remove undesired and save. Rename the file to the "defaultfilters" (do not forget to remove .omwaddon.project extension) and place it inside your configuration directory.

The file acts as template for all new project files from now. If you wish to go back to the old default set, simply rename or remove the custom file.