Updating with the masterserver

Specification draft

Revision 1.2

OpenClonk will have an automatic update system like Clonk Rage. However, the update logic will not be in the engine anymore.
The masterserver acts as the facilitator between the update packages on the server and the engine. Basically, the process looks like this: The engine asks for an update by sending his version information to the masterserver. The masterserver responds with the new version information and path to the right update package for the engine. The engine will then download that update package and apply it.

Masterserver interface to the Engine

The engine will provide two additional parameters when requesting the list of running games:

version	May only contain numbers, letters, ".", "_" and "-"
platform	May also only contain numbers, letters, ".", "_" and "-". The string will normally be in this format: `win-x86`, `win-x86_64`, `linux-x86`, ...

An example query string would look like this

```
http://boom.openclonk.org/server/?version=5.0.1.0&platform=win
```

Additionally to the list of running games, more information will be shown by the masterserver. This includes the most recent engine version, the message of the day and the URL to the update package the requesting client will have to download in order to update. If the client has the newest version, the field is not shown.

```
[Info]
Version=[Most recent engine version]
MOTD=[A message of the day]
UpdateURL=[URL to the update package if there is one]
```

If the client does not specify the version information or platform information, no *UpdateURL* is shown.

If the parameter *action* is specified with the value `version`, only the above [Info]-section will be displayed, not the list of running games. Example:

```
http://boom.openclonk.org/server/?action=version&version=5.0.1.0&platform=win
```

Constructing the UpdateURL

The masterserver will have a database table with the following coloums: *old version*, *new version*, *platform, file*. (The actual row names and the table name can be chosen freely.)

old version and *platform* together form the primary key of the table. *platform* holds the information about OS and architecture, e.g. `win-x86`. *old version* is the version to update from, the version of the engine that is requesting an update. *new version* is the version to which it can be updated and

file denotes the relative file path to the update package from the build directory on the server (the build directory will be on www.openclonk.org/builds/ but this should be configurable).
Thus, the construction of the update url is a simple lookup in the table.

If the version information the engine stated in the request cannot be found under *old version* in the table but the *platform* parameter is valid, the *update url* will then always point to the newest full installation file which should be found under *old version* = [none] and *platform* = [the one specified] in the table.

Interface to the build scripts

The build script compiles the source, pack the game data, pack the update packages and installation files and finally upload those packages on the server. For each uploaded package, the build script will authenticate itself and command the masterserver to write the information for that package into the table.

Following parameters will be given for each uploaded update package:

action	Will be `release-file` for that action.
old_version	A comma seperated list of versions to update from. If not specified, this file is the full installation. Format as described earlier.
new_version	Version to update to. Format as described earlier.
platform	The platform as described earlier. Format as described earlier.
file	The path to the uploaded file. The path is given relative to the builds directory on the server (which will be /www/builds/ in the file system, but this too should be configurable). Check that this is a valid filename.
hash	The hash of the uploaded file signed with a common HMAC. The string only contains numbers plus letters from a-f (hexadecimal). If the unencrypted hash is not equal to the hash of the specified file or the file does not exist, the authentication failed and the command is rejected.
delete_old_files	If "yes", referenced files from deleted rows are also deleted.

The task of the masterserver is to

1. Verify the authentication of the build server and validate the parameters.
2. If *old_version* has not been specified, only the row containing *old_version* = [none] and the given platform is deleted from the table (the old installation file). Otherwise, all rows which have the same *platform* as the one specified in the command and update to an older version than the one given in the *new_version* parameter are deleted from the table (the old update package) except the *old_version* = [none] row.
   All files that have been referenced by deleted rows must be deleted by the masterserver in both cases if *delete_old_files* is "yes".
3. Write the new row(s) with the data into the table: For every value in old_version, make a new row with otherwise the same values (*new_version, platform, file*).
   Also, note down the *new version* somewhere as the most recent engine version so it can be displayed in the server's [Info]- section.

Example:

An excerpt of this table could look like this:

old_version	new_version	platform	file
	1.3.0	win-x86	oc-win32-1.3.0.msi
1.0.1	1.3.0	win-x86	oc-win32-1.0.1-1.3.0.c4u
1.0.2	1.3.0	win-x86	oc-win32-1.0.1-1.3.0.c4u
1.2.4	1.3.0	win-x86	oc-win32-1.0.1-1.3.0.c4u
1.2.6	1.3.0	win-x86	oc-win32-1.0.1-1.3.0.c4u

This call is made:

```
?action=release-file&old_version=1.2.4,1.2.6&platform=win-
x86&new_version=1.4.0&file=oc-win32-1.2.4-1.4.0.c4u&hash=3981842046
```

The table will look like this after that:

old_version	new_version	platform	file
	1.3.0	win-x86	oc-win32-1.3.0.msi
1.2.4	1.4.0	win-x86	oc-win32-1.2.4-1.4.0.c4u
1.2.6	1.4.0	win-x86	oc-win32-1.2.4-1.4.0.c4u

Another call is made:

```
?action=release-file&platform=win-x86&new_version=1.4.0&file=oc-win32-
1.4.0.msi&hash=1238754345
```

And the table looks like this:

old_version	new_version	platform	file
	1.4.0	win-x86	oc-win32-1.4.0.msi
1.2.4	1.4.0	win-x86	oc-win32-1.2.4-1.4.0.c4u
1.2.6	1.4.0	win-x86	oc-win32-1.2.4-1.4.0.c4u

Download page

The download page will be a static page which will call certain template functions from the masterserver to get each the links to the most recent installation (and archive) files for a given platform. The template function's header could look like this:

```
function getDownloadURL(platform);
```

The masterserver has all the information it needs for that and can supply that information. For the download page, the masterserver web frontend does not need to query the list of running games as the download page only needs the urls to the installation files. There should be a way to tell the masterserver web frontend to not query that list automatically if there isn't yet.

Versioning and updates

The versioning will be A.B.C, whereas...

A = major release (Back to the Rocks, etc.)

B = update of a major release / minor release

C = bugfix release / minor changes

The build scripts will generate c4u update packages that can be applied to the last X released versions, independent of which numbers in the version string did change. (While X will probably a number around 10-20. But this number can easily be changed by the build script with this design.) For any versions that are not covered by this update anymore, the whole most current installation package is loaded. The exact logic when to and for which versions to supply an update packageis be decided by the build script server. It is possible to always force a reinstallation for e.g. when the A version has changed or when there occur problems with update packages.

With this update infrastructure, there will always be only one c4u update package per platform which always updates to the newest version. Certain installation files (for example each the last installation package if A or B is changed) can be kept by decision of the build script server by delete_old_files = 'no'.

RPMs, DEBs etc. are not covered by this update system because updates don't work if Clonk is installed system-wide (with a package manager) under linux. It also wouldn't make sense, because package-managers have their own update mechanism. Whenever the time comes that RPMs, DEBs etc. will be maintained, we need a preprocessor-switch in the source to disable the update system for package-manager-users.