# Script Repository



## Download WebService

http://download.mantidproject.org/scriptrepository/

It contains a cloned version of <https://github.com/mantidproject/scriptrepository>. For every push in git hub, there is a job in Jenkins ( <http://download.mantidproject.org/jenkins/view/All/job/is_sync_scripts_repo/> ) that pull any changes made to the repository and execute the script system/scriptrepositoryparser.py.

This parser is responsible to keep the <http://download.mantidproject.org/scriptrepository/repository.json> up-to-date.

This file keeps the following information for each entry inside the repository:

* Path
* PubDate
* Directory Flag
* Description
* Author

## ScriptRepositoryImpl:

### Check4Updates:

It connects to the WebService to download the lastest version of repository.json file. And Re-ListFiles.

### ListFiles – Core of ScriptRepository:

It parses the copy of repository.json file to get information about the remote repository (parseCentralRepository). Then it inspects the local directory to update its internal information about files already downloaded and new files inside the directory (parseLocalRepository). And finally, it checks a local.json file wich keeps track of downloaded files from the repository (parseDownloadedEntries). When going through these parsing procedures it creates a map ( path->RepositoryEntry ).

The final step is to iterate backwards through the map in order to be able to grasp the status and some information of the directories.

### Info, Description, fileStatus:

Just get the RepositoryEntry from the internal map and returns what is needed.

### Download:

Download the file from the WebService eventually backing up the local file when necessary, and update the local.json with information (pub-date) about the created file, it also update pythonscripts.directories to make python files up-and-running.

## Upload WebService:

Apache was configured to handle requests to <http://upload.mantidproject.org/scriptrepository/payload/X> to the X method of payload.py file using python server pages. The payload.py file is inside the repository: github.com/mantidproject/upload.mantidproject.org.

It supports two requests: publish (to upload) and remove to delete a file from repository.

**Publish** requires author, mail, comment, path, file.

The steps are:

* checks that the inputs are valid, for example, the file is smaller than 1MB
* copies the file to the cloned repository that is pointed to by the environment variable SCRIPTREPOSITORYPATH, which is set in the apache config
* adds the commit to the local git repository and publishes it to github. It returns the success or failure through a json file.

**Remove** requires author, mail, comment, path, file\_n.

The steps are:

* checks the validity of the inputs
* remove the file from the local repository at SCRIPTREPOSITORYPATH
* create a git commit. Then, publish it to github. It returns the success of the operation through json file.

## ScriptRepositoryImpl:

### Upload:

Fill a form and send to <http://upload.mantidproject.org/scriptrepository/payload/publish>. The form has the following fields:

* Author, e-mail, comment
* Path where to place the file
* File to update

In success, it receives from the server a json file with the following information:

* Message – if success or not
* Detail – Detail of the failure
* Pub\_date – The same date that will be inside the repository.json file.
* Shell – command that failed (for debug purpose)

Delete the entry from the repository.json file with the feedback of the server. This is because the file will take sometime to be updated in the remote repository.

### Remove:

Checks the state of the file, only BothUnchanged are possible to remove.

Fill a form and send to <http://upload.mantidproject.org/scriptrepository/payload/remove> with the following field:

* Author
* Mail
* Comment
* File\_n: path to the file.

In success, it receives from the server a json file with the following information:

* Message
* Detail
* Shell

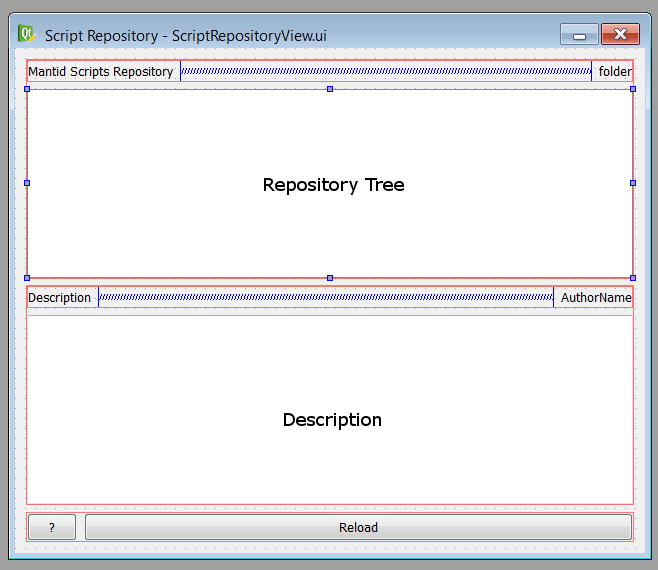
Update the local.json file with the feedback information received from the server.

## Proxy Support

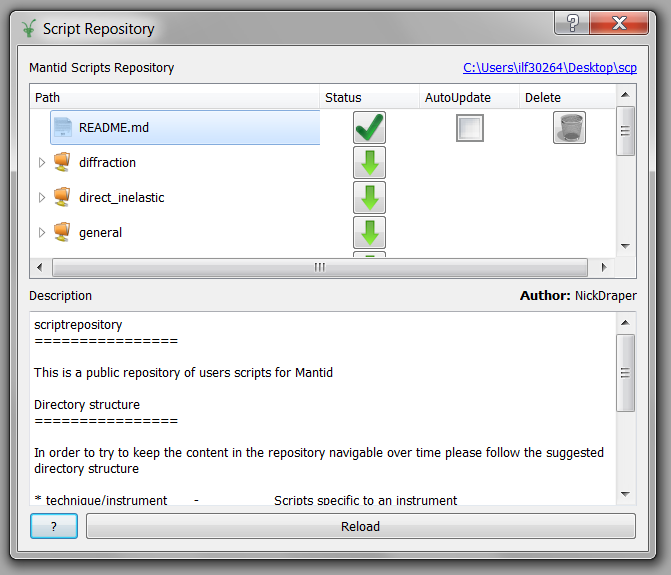
It does support proxy connection.

## User Interface

The user interface is based on the ScriptRepositoryView.ui.



It uses delegates to display the Status, AutoUpdate and Delete Widgets. Which allows having the icons for displaying the actions that are available as download, upload, delete, etc.



It exports loadScript signal to allow MantidUi to open scripts in its FileEditor. The interface is organized mainly in 3 classes:

ScriptRepositoryView:

It is responsible for displaying the description and author name when new entry in the script tree is selected, allow user to open the folder where the local installation is. Display the actions available through the delegates (Status, AutoUpdate, Delete). Allow the user to re-list the entries and to show the documentation about the ScriptRepository.

RepoModel:

Specialization of QAbstractItemModel to fulfil the requirement for the Model View Qt Framework.

It implements data, flags, headerData, index, parent, rowCount, columnCount, setData methods wich allows Qt to support the interaction with the ScriptRepository through the TreeView.

The way it is related to the ScriptRepository Framework is through the **RepoItem** class. This class allows the RepoModel to define the hierarchical structure of the tree and also be able to query the ScriptRepository framework by keeping the path information.

Download, upload, and removal of files are done in a separate thread to improve the user experience.