Tiedot Specification

version 0.1.0

Chris *Kwpolska* Warrick

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

Contents

Part I. Rationale

The Tiedot project is a data storage system by Chris *Kwpolska* Warrick, built on top of python and the following elements:

1. **The Tiedot Rationale**, this paragraph, which describes the need for this whole thing;
2. **Engine**, including:
   (1) **Python OOP Engine**, providing specific classes for every Element and its needs;
   (2) **TDP**, for storing the data;
   (3) **TDR**, alias references for the data;
   (4) **Authentication**, used for access control.
3. **Network Access**, used to access data from the network, in the following ways (*Actions*):
   (1) **Sync**, syncing data between the client and the server, allowing to use the data by the client without a network connection;
   (2) **Live Network Access** (LNA), used to access the data one-by-one, and send them in the same way;
   (3) **Web Client**, used to access data through a web browser.

   Additionally, all data access must happen through encrypted protocols.

Part II. Engine

1 Python OOP Engine

The main engine of the Tiedot is the Python programming language. Its technologies and awesomeness are used to store, create and make use of the objects. Main module used is the `pickle` module, explained further in section **??** on page ?? (TDP—Tiedot Data Pickles).

Since 2012-12-01, Python must be in version 2.7.3. Otherwise, the Sync protocol will not work.

2 TDP

Tiedot Data Pickles are the files storing all objects of the System. They are the standard Python pickles, with protocol version 2. They contain the `Object()`s and other objects used by the system (authentication data, TDRs)

3 TDR

TDR is an additional, optional reference to an object. Created by something as easy as `TDR = full.object.path`, they are providing an easy way to refer to objects that are referred to a lot. Although the method of creation is very easy, to ensure safe performance of all authorized activities, they should be modified using the `addtdr()` and `rmtdr()` functions and with `obj.tdr` set to the TDR. Otherwise, they will not be remembered for future sessions.

The suggested length of a TDR is **4**, although it doesn't matter to the Tiedot. This is a suggestion by a human, for humans.

4 Authentication

All authentication stuff is done by `tiedot.AUTH`, an instance of `tiedot.auth.Auth`. It should be used with credential providers, eg. `tiedot.ui.cli.auth`.

Account modification actions require synchronization with the sync server, using the `USERMOD` action.

Part III. Network Access

Tiedot has multiple Network Access features. They are using Twisted, Python and other solutions to provide a consistent interface.

1 BP

BP stands for *Base64-Pickle*. It is a method of transporting data. Pickle protocol 2 applies.

2 Data Dict

The Data Dict is a dict of file SHA1 sums (`s`) and timestamps (`t`), transported as a BP.

3 Data Comparison Procedure

ITMS	All items.
HAVE	Items to be sent to the client.
WANT	Items to be received from the client.
NONE	Items that are in sync on both sides.

1. Compare items.
   (a) $=$ Continue.
   (b) $\neq$ Add missing items to dicts with sha1sums and timestamps set to 0.
2. Iterate over items.
   (a) Add the item to →**ITMS**.
   (b) Compare checksums.
      i. $=$ →**NONE**.
      ii. $\neq$ Compare timestamps.
         A. $S > C$ →**HAVE**.
         B. $S < C$ →**WANT**.
         C. $S = C$ Kill somebody and assume →**HAVE**.
3. Remove `__auth__.tdp` from the **WANT** list (if it is there), issue a warning (using the dedicated command of `AUTHTR` and force a replace by `HAVE`.

4 Requirements for implementations

1. SSL encryption.
2. Python + Twisted.
3. Proper security measures: authentication, file permissions, network config etc.
4. Communicate everything important to humans.

5 Actions

The following actions exist:

Action Name		
Full	**Internal**	**Description**
Sync	`SYNC`	A standard sync operation.
Live Network Access (LNA)	`LNA`	A protocol for *live* data access.
User Modification	`USERMOD`	A user modification operation.

Tab. 1: Actions.

5.1 Sync

The Sync action is used to synchronize data between the server and a client. Afterwards, the client can disconnect from the Internet, cut the cable in half and make a figurine of Chris Warrick. If your cable is long enough.

5.2 LNA

The LNA action is used to interact with the Tiedot without storing the data anywhere, in a live connection fashion. You cannot cut your cable and make a figurine of yours truly. Unfortunately.

5.3 User Modification

The User Modification action provides access to exactly that: user modification. Request are placed in a queue, and are fulfilled after issuing `QUIT`.

6 Commands

Note that this list may change in the future. The following are valid for the version of the Tiedot distributed alongside this document (0.1.0).

Command	Arguments	C	S	Purpose
HELLO	last sync timestamp	+		Handshake.
HOWDY	current server timestamp		+	Handshake.
AUTH	username password	+		Authentication. Password is sha512.
AUTHSTATUS	auth status (0/1)		+	Authentication status. ⊖BYE is mandatory after a failed authentication.
ACTION	action to use	+		Action to use. More details in section 5 on the previous page.
ADDUSER	username; password	+		Add an user.
DELUSER	username; password	+		Delete an user.
CHANGEPWD	old; new	+		Change current user's password.
USERMOD	user modification status (0/1)		+	User modification status. (*Sync*, *User Modification*: Immediately followed by mandatory AUTHTR, FILES HAVE *__auth__.tdp* and ⊖BYE.)
AUTHTR			+	A warning informing that the __auth__.tdp file cannot be transferred due to security policies and a replacement of that file will be forced.
FILES	HAVE/WANT (*Sync*), GET/PUT (LNA) *BP(files)*	+	+	File transmission.
REFUSE	refused item			The item is not accepted by the server.
QUIT	reason (for humans to read)	+		Request to close the connection.
BYE	⊕ current server TS ⊖ HELLO TS		+	Connection closed.

Tab. 2: Global NA commands.

Command	Arguments	C	S	Purpose
DIRS	existing directories		+	BP of a list of all existing directories.
CURDATA	current data	+		Data Dict—client's data.
HAVE	data to send to the client		+	Data Dict—client's outdated data.
WANT	data to get from the client		+	Data Dict—server's outdated data.
DATAEQ			+	Both sides have the same data.

Tab. 3: Sync NA commands.

Command	Arguments	C	S	Purpose
TREE	BP(existing files and dirs)		+	A tree of files on the server.
MKDIR	path	+		Directories to create.
GET	filename/__TREE__	+		Request a file to get.
PUT	filename	+		Put a file up.
REPR	filename	+		Get a __repr__() of the requested data, in plaintext.

Tab. 4: LNA commands.

Part IV. Meta

1 License

2 Revisions

Timestamp	Changes
2012-03-28T19:00:00Z	initial version
2012-03-29T14:00:00Z	[deleted] introduced
2012-04-06T07:30:00Z	[deleted] introduced; values: [deleted]
2012-04-18T19:00:00Z	paper document type, other style changes.
2012-04-25T09:00:00Z	RDS can happen in 1000 years; reference updates.
2012-06-11T18:14:00Z	Updating to reflect [deleted], [deleted] introduced, style modifications.
2012-06-11T17:00:00Z	[deleted] introduced.
2012-11-20T20:00:00Z	A very big update to reflect all the changes that were made in the recent days: (a) a *working* version of the RDS itself; (b) better mechanisms; (c) retirement of [deleted]; (d) making RDSQT much less important.
2012-11-24T20:00:00Z	An update to cover the Sync mechanism.
2012-11-25T20:00:00Z	(a) rename to KDS/Kw Data System (originally RDS/Rapid Data System); (b) more sync coverage; (c) retirement of RDSQT.
2012-12-01T20:00:00Z	Sync updates, Py2k requirement.
2012-12-06T20:00:00Z	Even more sync updates.
2012-12-08T20:00:00Z	Rename to Tiedot, going up to git.