IPSO Chart

|  |  |
| --- | --- |
| Input | Process |
| * USB Drive information when drive is inserted * Files on USB drive when inserted * Add backup interface contents * Request for drive to be backed up * Insertion backup of drive * Drive amendments from modification window * Request to refresh connected drives list * Right clicks on notification icon | * Check drive capacity of location for enough space * Copy files to system / system to drive * Check if user wants to compress (and or encrypt data) * Compress data from drive backup * Detect the user logged on * Check drive has backup record for the entered device * Obtain drive details from USB for checks * Check database exists * Store data backup statistics into database on each backup * Validate backup before execution * Add backup record |
| Storage | Output |
| CREATE TABLE [Backups](  [Drive\_ID] NTEXT,  [Start\_Date] BIGINT,  [Backup\_Name] NTEXT,  [Hash] NTEXT,  [Duration] INT);  CREATE TABLE [Drive](  [ID] NTEXT,  [Name] NTEXT,  [Capacity] BIGINT,  [File\_System] NTEXT,  [Type] NTEXT);  CREATE TABLE [Properties](  [Property] NTEXT,  [Value] NTEXT);  CREATE TABLE [Recordset](  [Name] NTEXT,  [Drive\_ID] NTEXT,  [Creation\_Date] BIGINT,  [Backup\_Location] NTEXT,  [Automatic] BIT,  [Compression] BIT,  [Previous\_Backups] INT); | * Backup of Data in compressed generic file * List of backups to UI * Progress bar for current backup state |

Pseudo Algorithms:

## Check if user wants compression (DriveName, BackupName, DriveCapacity)

SQL("SELECT isCompressed FROM backupTable WHERE driveName = {} AND BackupName = {} AND DriveCapacity = {}", DriveName, BackupName, DriveCapacity)

## Has drive changed since backup? (backupName, driveLetter)

driveHash <-- calculateHash(driveLetter)

backupHash = ExecuteSQL("SELECT Hash FROM Backups WHERE backupName = {}", backupName)

if backupName is driveHash then return False

Else return True

## Check Drive Capacity (USB\_Location, Drive\_Location)

freeSpace <-- getCapacity(Drive\_Location)

driveFiles <-- getCapacity(USB\_Location)

if driveFiles > freeSpace then return True

else return False

## Check if drive has a record (DriveFormat, DriveName, DriveCapacity)

result <-- ExecuteSQL("SELECT BackupName FROM backups WHERE driveFormat = {} AND DriveName = {} AND DriveCapacity = {};", DriveFormat, DriveName, DriveCapacity)

if results is None then return False

Else Return True

## Store Drive backup statistics in database

SQL(“INSERT INTO Backups VALUES (?,?,?,?,?)”, Drive\_ID, Start\_date, Backup\_Name, Hash, Duration)

## Create backup record in database

SQL(“INSERT INTO Drive (ID, Name, Capacity, File\_System, Type) VALUES (?,?,?,?,?)”, Drive\_ID, Drive\_Name, Capacity, File\_System, Type)

Process Algorithms

The algorithms that are not listed as pseudo can be found in the technical documentation.

System integrity

# Storage Requirements

The system will store data primarily in its database, located in the install directory. The program will also store data in the directories of the user’s choice. This is where it will store the backups of the drives. The locations of these directories are stored in the database.

# Security and integrity of data

The database is encrypted using a strong password, however is hard coded into the program, and doesn’t change per install. The database is inside the install directory, so can be easily copied out to make a backup, provided the user can access the install directory. The data inside the database conforms to 1st normal form, to make sure that storage is efficient, this is especially necessary due to the size restriction on the database.

# System Security

Basic security flaws such as SQL injection are protected against thanks to all values being escaped. The database is encrypted to protect the backup directories and information, and against someone adding malicious statements into the database.