

Project Code: **BS1**

## **Technical Manual for:**

### **S3ONTHEGO**

**A secure offsite file backup & accessibility solution**

**Prepared by:** Cormac Redmond  
CASE 4  
53478637

**Date:** 01/05/2007

**Supervisor:** Dr. Brian Stone

# TABLE OF CONTENTS

TABLE OF CONTENTS.....	2
<b>1 INTRODUCTION.....</b>	<b>4</b>
1.1 OVERVIEW .....	4
1.2 DOCUMENT CONVENTIONS/GLOSSARY.....	4
<b>2 SYSTEM ARCHITECTURE.....</b>	<b>7</b>
2.1 S3ONTHEGO WEB SERVICE SERVER .....	8
2.2 INTERNAL SQL SERVER DATABASE (S3ONTHEGO DATABASE).....	8
2.3 AMAZON S3 WEB SERVICE .....	8
2.4 USER APPLICATION .....	8
<b>3 HIGH-LEVEL DESIGN - S3ONTHEGO DATABASE .....</b>	<b>9</b>
3.1 TABLES .....	9
3.1.1 <i>S3OnTheGoUsers Table</i> .....	9
3.1.2 <i>Stored Procedures</i> .....	9
<b>4 HIGH-LEVEL DESIGN - S3ONTHEGO WEB SERVICE.....</b>	<b>10</b>
4.1 FUNCTIONS.....	10
4.1.1 <i>User Authentication</i> .....	10
4.1.2 <i>Encryption Key Retrieval</i> .....	10
4.1.3 <i>Salt Value Retrieval</i> .....	10
4.1.4 <i>Signature Creation</i> .....	10
<b>5 HIGH LEVEL DESIGN - USER APPLICATION .....</b>	<b>11</b>
5.1 APPLICATION COMPONENTS.....	11
5.1.1 <i>Global Engine</i> .....	11
5.1.2 <i>Model Layer</i> .....	12
5.1.3 <i>Presentation Layer</i> .....	13
5.2 NAMESPACES & CLASSES DESCRIPTION .....	13
5.2.1 <i>S3OnTheGo.Backup</i> .....	14
5.2.2 <i>S3OnTheGo.Backup.Model</i> .....	15
5.2.3 <i>S3OnTheGo.Backup.Engine</i> .....	17
5.2.4 <i>S3OnTheGo.Backup.Presentation</i> .....	18
5.3 FUNCTIONAL DESCRIPTION .....	22
5.3.1 <i>Files and Objects on Amazon S3</i> .....	22
5.3.2 <i>Initial User Authentication</i> .....	24
5.3.3 <i>Signature Creation</i> .....	25
5.3.4 <i>Backup Item Retrieval &amp; Storage</i> .....	26
5.3.5 <i>Encryption within S3OnTheGo</i> .....	27
5.3.6 <i>Backing up a Backup Item</i> .....	28
5.3.7 <i>Restoring from a Backup Item</i> .....	32
5.3.8 <i>Backup Scheduling</i> .....	35
5.3.9 <i>Proxy Settings within S3OnTheGo</i> .....	36
<b>6 PROBLEMS AND RESOLUTION .....</b>	<b>37</b>
6.1 AMAZON S3 RELIABILITY .....	37
6.2 SECURITY ISSUES .....	37
<b>7 INSTALLATION GUIDE .....</b>	<b>38</b>
7.1 REQUIREMENTS.....	38
7.2 INSTALLATION .....	38
7.2.1 <i>Setting up the S3OnTheGo Database</i> .....	38
7.2.2 <i>Setting up the S3OnTheGo Web Service</i> .....	40
7.2.3 <i>Setting up the S3OnTheGo Registration Web Application</i> .....	41
7.2.4 <i>Deploying the S3OnTheGo User Application</i> .....	42
<b>8. APPENDICES .....</b>	<b>43</b>
A. AMAZON S3 OVERVIEW.....	43
A.1. <i>Interacting with Amazon S3</i> .....	44

A.2. *Common Elements* ..... 44

# 1 INTRODUCTION

## 1.1 Overview

This document describes the technical specifications of the “S3OnTheGo” system, a secure off-site backup solution for Microsoft Windows, based on the Amazon S3 web service. The aim of S3OnTheGo is to provide an organisation with the ability to allow its employees to backup and synchronise a selection of their files, from any internet-connected location.

Current backup solutions generally rely on separate pieces of hardware, such as tapes, hard drives, REV™ drives, CD/DVDs, and NASs (Network-Attached Storage). In the event of a server crash or other disaster, files and system state can be restored from the backup media. These conventional solutions can be costly, always entail the deployment and maintenance of extra hardware and can often over-complicate a simple set-up. These restraints can preclude their use by the “small-time” operator, such as a small business.

S3OnTheGo removes these extra layers of complication to provide a robust and secure backup solution, which also acts as a type of “revision control”, whereby users can rollback to an older “snapshot” of a particular backup, if necessary. On top of that, it provides the ability for users to retrieve and restore backups from any internet-connected location, thus allowing greater flexibility for the travelling user.

The system is intended for use by small-to-medium sized businesses as an alternative or reinforcement to conventional backup systems, and also as a tool to increase accessibility amongst its employees/users.

## 1.2 Document Conventions/Glossary

### ***S3OnTheGo server/service:***

Any organisation/entity that holds an Amazon S3 account, and offers the relevant S3OnTheGo web services through an internet-accessible server is an S3OnTheGo service provider.

### ***User:***

Anyone who is authorised as a user with the S3OnTheGo service is a ‘user’.

### ***SOAP:***

SOAP (originally Simple Object Access Protocol) is a protocol for exchanging XML-based messages over computer network, normally using HTTP.

### ***WSDL:***

The Web Services Description Language is an XML format published for describing Web services.

**XML:**

XML (Extensible Mark-up Language) is a W3C initiative that allows information and services to be encoded with meaningful structure and semantics that both computers and humans can understand. XML is ideal for information exchange, and can easily be extended to include user-specified and industry-specified tags.

**Amazon S3:**

Amazon S3 (Simple Storage Service) provides a web service that can be used to store and retrieve any amount of data, at any time, from anywhere on the web - for a small price. See Appendix A for more details.

**Web service:**

Web Services are application components that use:

- WSDL - Web Services Description Language, for self-description
- TCP/IP, for transport.
- HTTP, for interaction.
- SOAP - Simple Object Access Protocol, for requesting and granting actions
- XML, for underlying representation

**Advanced Encryption Standard (AES):**

Advanced Encryption Standard (AES), also known as Rijndael, is a block cipher adopted as an encryption standard by the U.S. government.

**Hash function:**

A hash function takes a string of any length as input and produces a fixed length string as output, sometimes termed a “message digest” or a “digital fingerprint”. The SHA (Secure Hash Algorithm) hash function, considered to be the successor to MD5 (an earlier, now compromised hash function), will be used in this system.

**Salting:**

Salting is the process of adding some additional random data to the front of a password before it is hashed, so that in the event that two users happen to have the same password, it isn't obvious by looking at the resulting hash values. The plaintext salt has no value to an attacker (or a malicious system admin) because its function is solely to remove the ability to detect duplicate passwords.

**Backup Item:**

A `BackupItem` is an object containing a reference to a chosen directory (for backup), a schedule time to perform the backup (if any), a name, and any number of “snapshots”.

**Snapshot:**

Each time a backup is performed from a `BackupItem`, a “snapshot” is created. It holds the information identifying what files were backed up at that particular time, and what time the backup occurred, etc.

**Serialization:**

Serialization is the process of saving an object onto a storage medium (such as a file, or a memory buffer) or to transmit it across a network connection link, either as a series of bytes or in some human-readable text format such as XML

**GUID:**

Globally Unique Identifier or GUID is a pseudo-random number used as a unique identifier. While each generated GUID is not guaranteed to be unique, the total number of unique keys ( $2^{128}$  or  $3.40282366 \times 10^{38}$ ) is so large that the probability of the same number being generated twice is very small.

**Web.Config:**

The `Web.Config` file is the main settings and configuration file for an ASP.NET web application. The file is a text based XML document that defines such things as connection strings to any databases the application uses, etc.

**Web Services Description Language Tool (wsdl.exe):**

The Web Services Description Language tool generates code for XML Web services and XML Web service clients from WSDL contract files, XSD schemas, and .discomap discovery documents.

**Windows Forms:**

Windows Forms is the name given to the graphical user interface application programming interface (API) included as a part of Microsoft's .NET Framework. When mentioned, a 'form' represents a window or dialog box that makes up an application's user interface.

**BackgroundWorker:**

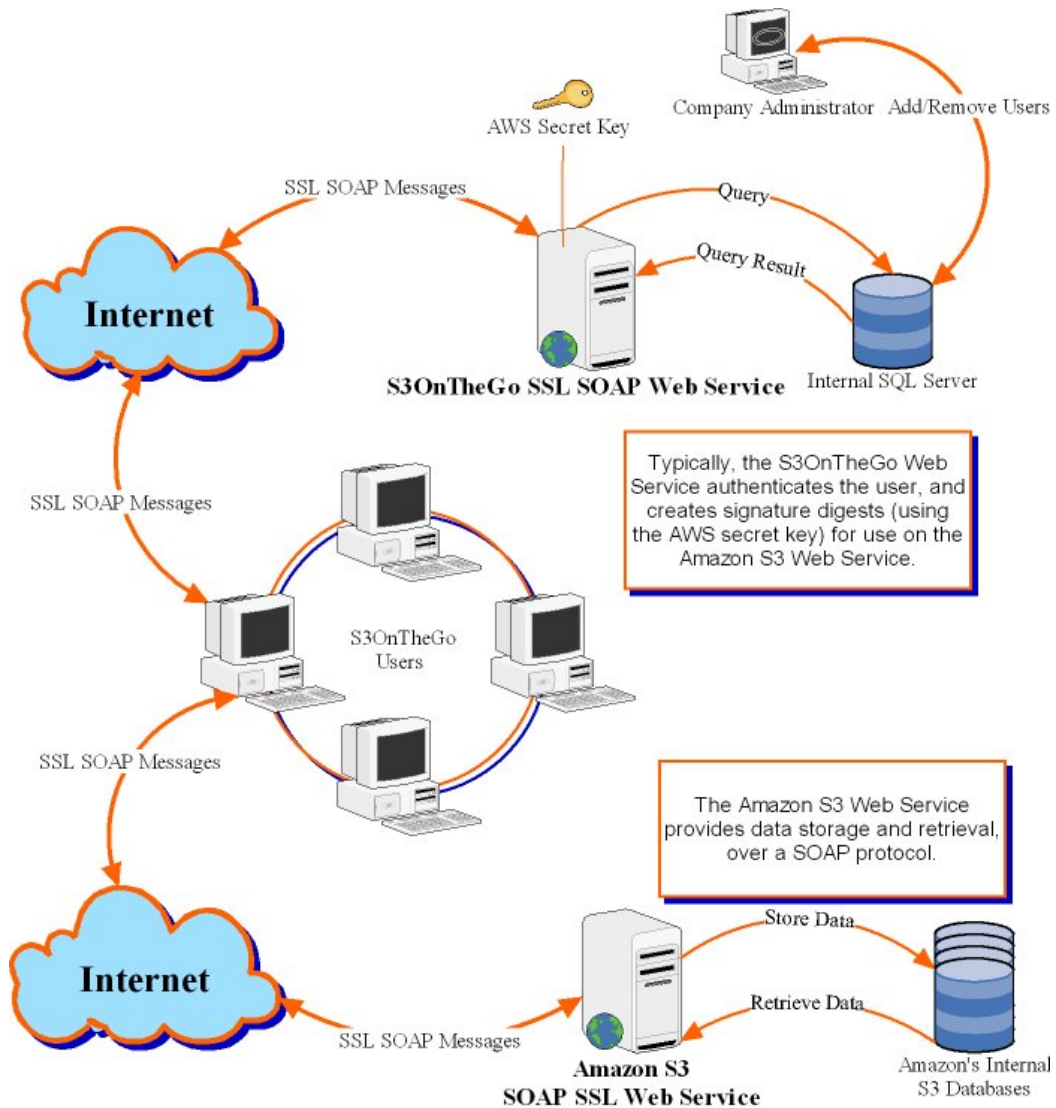
The `BackgroundWorker` is a .NET component which gives you the ability to execute time-consuming operations asynchronously ("in the background"), on a thread different from your application's main UI thread.

**Timer:**

Provides a mechanism for executing a method at specified intervals.

## 2 SYSTEM ARCHITECTURE

This section describes the system's context and architecture in relation to the below diagram:



As evident from the diagram, the architecture of the system can be broken down into three main components:

- An S3OnTheGo Server (inclusive of an internal SQL Server)
- The Amazon S3 web service
- The Users (people who use the user application)

## **2.1 S3OnTheGo Web Service Server**

The S3OnTheGo server provides an SSL-secured SOAP web service which is responsible for authenticating users upon logging into the system (via the user application) and subsequently creating Amazon S3 signatures thereafter, for use by the user application. Steps have been taken to prevent a malicious user from maliciously using the service (i.e., create their own application).

Typically, any company implementing the S3OnTheGo system must create an Amazon S3 account and setup this server (and expose the S3OnTheGo web service). Additionally, this server must be available from anywhere on the Internet to provide world-wide use of the service.

## **2.2 Internal SQL Server Database (S3OnTheGo Database)**

An SQL Server is also installed (either locally on the S3OnTheGo server, or preferably, nearby on the intranet) in order to hold user information - such as usernames, salt values, password digests, encryption keys, etc. It is only accessible to the S3OnTheGo server.

## **2.3 Amazon S3 Web Service**

The Amazon S3 web service is a third-party service which provides a simple SSL-secured SOAP web service interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. In order to store data, the user application uses this service. See Appendix A for more information.

## **2.4 User Application**

Any registered user of the S3OnTheGo system uses the user application in order to make use of the service. The application authenticates itself with the S3OnTheGo server, and subsequently requests signatures, keys, etc, from it.

## 3 HIGH-LEVEL DESIGN - S3ONTHEGO DATABASE

This section outlines the high-level design of the S3OnTheGo SQL Server database.

The S3OnTheGo database is small (SQL Server 2000/2005) database, consisting of (currently) one table, and a number of stored procedures. It is not exposed to the internet, and is only accessible to the S3OnTheGo web service and local internal network.

### 3.1 Tables

The S3OnTheGo currently only contains one table.

#### 3.1.1 S3OnTheGoUsers Table

S3OnTheGoUsers			
	Column Name	Condensed Type	Nullable
🔑	UserName	varchar(20)	No
	PasswordHash	varchar(100)	No
	Salt	varchar(8)	No
	EncryptionKey	varchar(8)	No

#### 3.1.2 Stored Procedures

A number of stored procedures are available to the calling application:

***CreateUser:***

A simple stored procedure used by an administration/registration application (See Section 7.2.3) to register a user on the S3OnTheGo system.

***GetCryptoKey:***

Used to retrieve a user's encryption/decryption 'key'.

***LookupUser:***

Retrieves information on a user, given their username.

***Ping:***

Responds to verify that the service is available.

## 4 HIGH-LEVEL DESIGN - S3ONTHEGO WEB SERVICE

The S3OnTheGo web service is developed using ASP.NET, with a C#.NET back-end, which retrieves data, when necessary, from the S3OnTheGo database. It is exposed to the internet through an SSL enabled web server, such as IIS 6.0. As mentioned in Section 2.3, its main purpose is to aid the user application in authenticating the user, and thereafter, to create Amazon S3 signature digests, supply the user's salt value and encryption key, etc, as required.

As described in Appendix A, a signature is created using, amongst other things, an AWS secret key – which is held on this server in the `web.config` file, along with a connection string to the S3OnTheGo database.

### 4.1 Functions

The S3OnTheGo web service provides the following functionality:

#### 4.1.1 User Authentication

The response represents the success/failure in authenticating a user, given a username and hashed password.

#### 4.1.2 Encryption Key Retrieval

Returns the user's unique symmetric encryption key if given a valid username and hashed password.

#### 4.1.3 Salt Value Retrieval

Returns the user's unique salt key when given a valid username and password.

#### 4.1.4 Signature Creation

Creates and returns an Amazon S3 signature (as described in Appendix A) for a given operation, when given valid a username, hashed password and the rest of the 'string to sign'.

The details of where and why this service is used are described later.

## 5 HIGH LEVEL DESIGN - USER APPLICATION

The user application is developed using C# .NET. Being the most complicated component of the system, it is necessary to demonstrate, from a few perspectives, how the system fits together. In order of higher-to-lower level, this section describes the user application in relation to:

- Application Components
- Namespaces & Classes Description
- Functional Descriptions

“Application Components” takes a very high-level look at how the application is divided into components.

“Namespaces & Classes Description” describes the purpose and main functionality of each namespace and its members.

Finally, in order to communicate what is actually occurring at a lower level, “Functional Descriptions” describes, with reference to the other sections, the more intricate parts of the system, and what is ‘actually going on’ for some of the major operations that are of interest.

### 5.1 Application Components

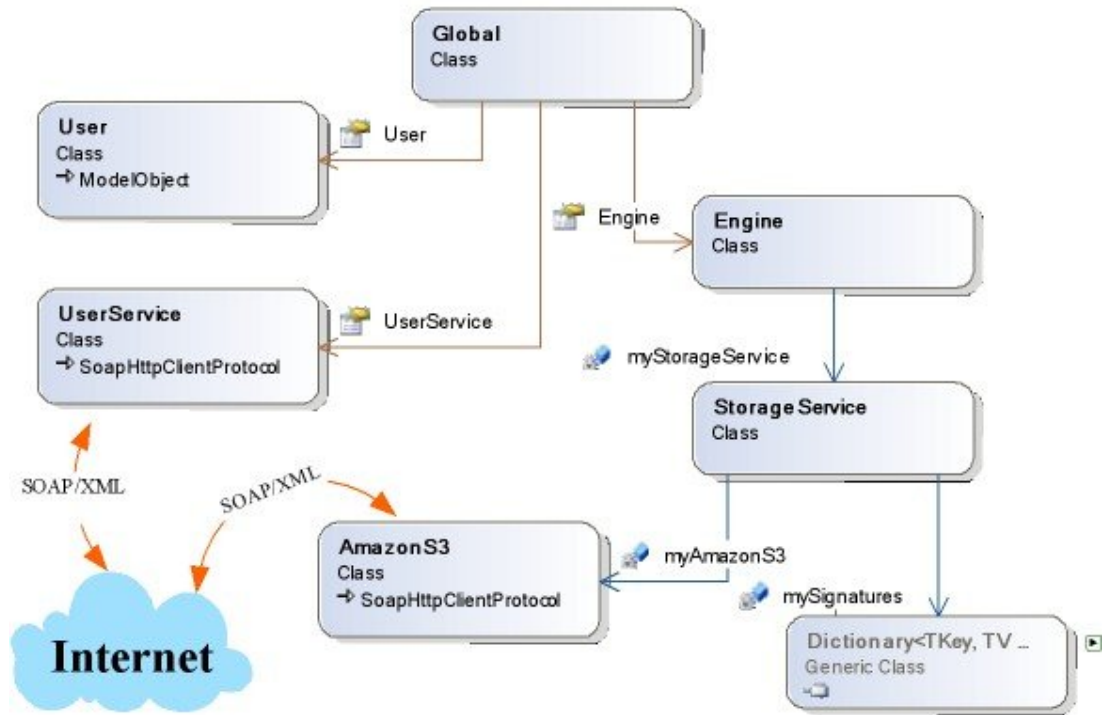
The user application can be broken down into rational components, as described in this section.

It may be advisable to reference this section in order to comprehend later sections.

#### 5.1.1 Global Engine

The Global Engine is accessible to each namespace throughout the application. Amongst other things, it references the S3OnTheGo web service proxy, the StorageService (which references the Amazon S3 web service proxy), the current User, and the ‘Engine’ to use, etc, and is utilised extensively from the Presentation Layer (see below).

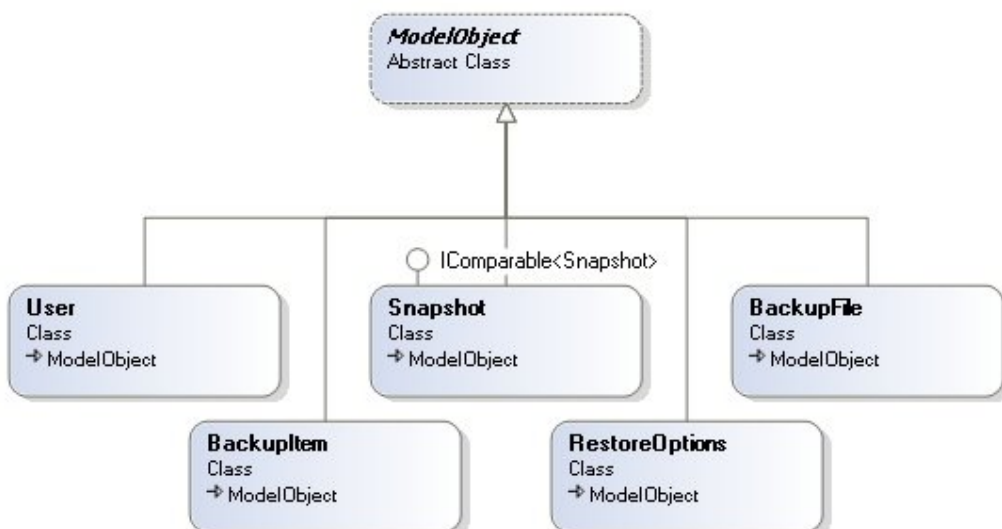
*A simplified model of the Global Engine:*



**5.1.2 Model Layer**

The Model Layer is contained within the `S3OnTheGo.Backup.Model` namespace, and represents ‘things’ in the S3OnTheGo system, which are used throughout each namespace.

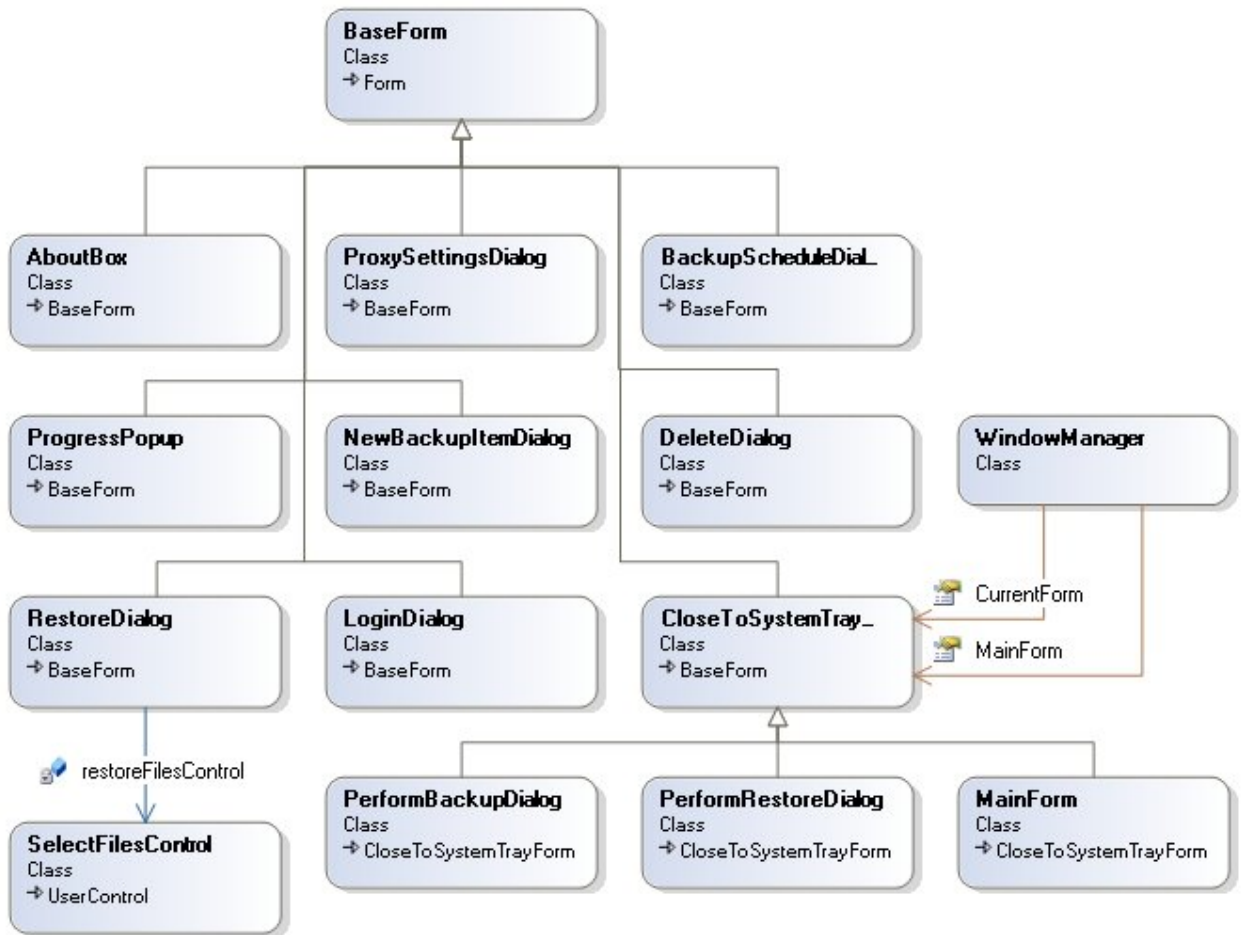
*A reduced view of the components in the Model Layer:*



### 5.1.3 Presentation Layer

This component is completely contained within the `S3OnTheGo.Backup.Presentation` namespace and is responsible for managing the graphical user interface, and interacting with the Global Engine and Model Layer components to carry out the required operations.

*A simplified diagram of the Presentation Layer:*



## 5.2 Namespaces & Classes Description

The S3OnTheGo user application, as somewhat illustrated above, is broken down into a number of namespaces. This section describes the central functionality of each namespace and its members.

### 5.2.1 S3OnTheGo.Backup

`S3OnTheGo.Backup` consists of the application's entry-class and numerous (mostly static) helper classes which are utilised throughout the entire application.

#### **Program**

Extends:      Nothing

`Program` is the application's entry-class, which does little apart from catch any unhandled exceptions that may make their way back down the hierarchy (this should be rare as errors should be handled at the place of error). It allows the application to start in minimised mode; if for example, it is invoked from the registry, or the startup folder, etc. Then, it creates a new instance of `MainForm` (see later) which it hands to `SingleInstance` (see below).

#### **SingleInstance**

Extends:      Nothing

`SingleInstance` is a static class which maintains a global mutex, ensuring that only one instance of the S3OnTheGo application is running on the machine. As long as the mutex is available, it will proceed to run the form `MainForm`.

#### **ProgressInfo**

Extends:      Nothing

Throughout the application, there are several times when the user needs to be informed on the progress of a certain operation. `ProgressInfo` is made up of a group of variables which represent the current state of an ongoing operation, and is passed back down layers to the user interface.

#### **CountDownInfo**

Extends:      Nothing

A tiny class used as a countdown label for reporting the progress of an operation.

#### **Utility**

Extends:      Nothing

`Utility` consists of a bunch of commonly used static methods that are used throughout the application

#### **WindowManager**

Extends:      Nothing

`WindowManager` is a static class written purely to help manage any form (that extends `CloseToSystemTrayForm`). For example, it deals with hiding, activating, minimising forms to system tray, maintaining a record of the ‘current’ and ‘parent’ forms, and handling certain form settings depending on whether the application is minimised, or in the system tray, or maximised, etc.

### **Global**

Extends:      `Nothing`

`Global` (described as the Global Engine in Section 5.1.1) contains data which is accessible to the whole user application, such as the machine name, the chosen `S3OnTheGo` web service to use, the `Engine` to use, the current `User` using the application, etc. Some other variables, such as the product name and product version are retrieved from the compiled assembly (using information which originates from the auto-generated `AssemblyInfo.cs`, etc) and utilised throughout the application.

## **5.2.2 S3OnTheGo.Backup.Model**

The `S3OnTheGo.Backup.Model` namespace consists of classes which represent ‘things’ within the backup system, which are required throughout the other namespaces.

### **ModelObject**

Extends:      `Nothing`

Being the base (super) class for all the other classes in the `S3OnTheGo.Backup.Model` namespace, `ModelObject` is a serializable, abstract class which maintains a unique identifier variable (GUID), which is passed in at by all subclasses at creation time.

### **BackupFile**

Extends:      `S3OnTheGo.Backup.Model.ModelObject`

`BackupFile` is a serializable class used to represent any file which is to be backed-up or restored, etc. It stores information regarding the size and path of the actual file, as well as the number of parts the file is broken into within the `S3OnTheGo` system. See Section 5.3.1 for more information.

### **BackupItem**

Extends:      `S3OnTheGo.Backup.Model.ModelObject`

A `BackupItem` is a serializable object containing a reference to a chosen directory, a backup schedule frequency (if any), a name, and a bunch of helper methods. The application performs a backup “from a Backup Item” (i.e., it backups the folder

pointed to by it). Each `BackupItem` is associated with one or more `Snapshots`, but there is no connection (cohesion) from the `BackupItem` class,

### **Snapshot**

Extends: `S3OnTheGo.Backup.Model.ModelObject`

A `Snapshot` represents the state of a directory when a backup was performed. It contains other information such as to which `BackupItem` it belongs, the time of backup, and the overall size of transfer, etc.

### **ScheduleFrequency (Enumeration)**

A simple enumeration containing the frequencies possible for backup scheduling. Currently, a backup can be scheduled for any time on a daily, weekly, or monthly basis.

### **WeekOfMonth (Enumeration)**

An enumeration representing the weeks of a month. For example, “first”, “second”, “third”, “fourth” and “last” (if applicable).

### **User**

Extends: `S3OnTheGo.Backup.Model.ModelObject`

`User` is a (mostly) serializable class which represents the currently logged-in user of the `S3OnTheGo` user application. It maintains the user’s password and hashed password, as well as the user’s proxy settings (`ProxySettings`). A list of the user’s `BackupItems` is also maintained, which is the non-serializable part of `User` (as this list is stored separately on Amazon S3).

### **RestoreOptions**

Extends: `S3OnTheGo.Backup.Model.ModelObject`

A simple class used to wrap a bunch of user-chosen options (custom download location, a list of chosen download files, etc) when the user chooses to perform a restore of a backup.

### **ProxySettings**

Extends: `Nothing`

A serializable class used to hold all the information necessary if the user chooses to use a proxy server.

### 5.2.3 S3OnTheGo.Backup.Engine

The `S3OnTheGo.Backup.Engine` namespace contains classes responsible for dealing with send and receiving data to the Amazon S3 and the S3OnTheGo web services. Recall the (simplified) diagram in Section 5.1.1.

#### AmazonS3

Extends:

```
System.Web.Services.Protocols.SoapHttpClientProtocol
```

`AmazonS3` is a Web Services Description Language tool (`wsdl.exe`) generated proxy responsible for communicating with the Amazon S3 web service. Extending `System.Web.Services.Protocols.SoapHttpClientProtocol`, it supports both asynchronous and synchronous method calls, automatic XML generation and automatic sending and getting – thus, creating a simple communication medium between the user application and Amazon S3.

#### UserService

Extends:

```
System.Web.Services.Protocols.SoapHttpClientProtocol
```

`UserService` is a Web Services Description Language tool (`wsdl.exe`) generated proxy responsible for communicating with the Amazon S3 web service. Extending `System.Web.Services.Protocols.SoapHttpClientProtocol`, it supports both asynchronous and synchronous method calls, automatic XML generation and automatic sending and getting – thus, creating a simple communication medium between the user application and the S3OnTheGo web service.

#### Engine

Extends:      Nothing

`Engine` is a huge wrapper to `StorageService` (which itself is a wrapper to the `AmazonS3` proxy). Being a layer above the `StorageService`, it operates on a higher level, analysing and dealing with mostly `ModelObjects`, etc, before sending/receiving from the lower layer. Classes from the `S3OnTheGo.Backup.Presentation` namespace use this `Engine` extensively (through the `Global` class/engine).

The main function of `Engine` is to:

- Analyse files before backup to prevent unnecessary uploading
- Apply proxy settings to the system
- Encrypt and decrypt data
- Get & Put `BackupItems` & `Snapshots`
- Delete `BackupItems`
- Backup `BackupFiles`
- Restore actual files

- Detect locked (in-use) files
- Check what files exist locally
- Apply proxy settings to system
- Find system's default proxy settings
- ....and provide many other helper services

### **StorageService**

Extends:      `Nothing`

`StorageService` is a large class which acts as a wrapper to the `AmazonS3` proxy class. It operates on a lower level than `Engine`, and its main purpose is to:

- Initialise and setup the `AmazonS3` proxy
- Delete data on Amazon S3
- Send/receive data to/from Amazon S3
- Request and then maintain a dictionary of valid signatures to prevent unnecessary calls to the `S3OnTheGo` web service
- Handle SOAP exceptions gracefully
- Pass more serious exceptions up a layer (to `Engine`)

#### **5.2.4 S3OnTheGo.Backup.Presentation**

This subsection describes the `S3OnTheGo.Backup.Presentation` namespace (the Presentation Layer). When referred to, a 'form' represents a window or dialog box – a component of 'Windows Forms' (see glossary).

### **BaseForm**

Extends:      `System.Windows.Forms.Form`

The `BaseForm` is the base (super) class for many other forms in the user application. It deals with standard issues such as setting up the application, where to position on the screen, etc. Any form which should not be minimised and *can* be 'exited' extends `BaseForm`, whereas any form which can be minimised and where an 'exit' should cause the application to be sent to the system tray, should extend `CloseToSystemTrayForm`.

### **CloseToSystemTrayForm**

Extends:      `S3OnTheGo.Backup.Presentation.BaseForm`

Any form which needs to be able to be maximised, minimised, sent to system tray, sent to taskbar, etc, inherits `CloseToSystemTrayForm`. In conjunction with `WindowManager`, it handles all the details to prevent any windowing problems or inconsistencies.

## ProgressPopup

Extends: `S3OnTheGo.Backup.Presentation.BaseForm`  
Invoked by: `MainForm`, `LoginDialog`, `PerformBackupRestore`,  
`PerformRestoreDialog`, `ProxySettingsDialog`,  
`RestoreDialog`

`ProgressPopup` uses a `System.Windows.Forms.ProgressBar` control to create a pop-up which indicates to the user, through this basic progress bar, that the application is currently carrying out a blocking operation. For example, it is used when the application cannot continue until it retrieves certain data, etc. This however, is rare, and most background operations are multi-threaded and utilise a `ProgressBar` and `CountDownInfo` label which are interacted through non-blocking operations, as described briefly in Sections 5.3.6 and 5.4.7.

## MainForm

Extends: `S3OnTheGo.Backup.Presentation.CloseToSystemTrayForm`

Being the largest class in the system, `MainForm`'s jobs are to:

- Setup initial user interface screen
- Initialise the authentication progress (`LoginDialog`)
- Provide user controls
- Populate main `TreeList`
- Refresh `BackupItems` from other computers
- Invoke relevant forms based on user choices
- Invoke relevant forms based on scheduled backups, etc
- Inform user of different circumstances at different times, through notify icons, pop-ups, etc.
- Handle drag and dropping from outside-to-inside the application
- Handle creation and storage of `BackupItems`
- Serialize/de-serialize 'User' objects from local file system
- Handle scheduled backups
- Handle user error and higher-level exceptions
- ...offer many more behind-the-scenes functionality

## LoginDialog

Extends: `S3OnTheGo.Backup.Presentation.BaseForm`  
Invoked by: `MainForm`

Invoked on each application run, `LoginDialog`'s main role is to:

- Request a username and password
- Allow user to save password
- Request a company's AWS Access Key and an `S3OnTheGo` web service URL
- Invoke proxy settings dialog (`ProxySettingsDialog`)
- Authenticate users and accordingly return control to `MainForm`

### **ProxySettingsDialog**

Extends: BaseForm  
Invoked by: LoginDialog

The job of ProxySettingsDialog is to:

- Allow user to automatically detect proxy settings
- Allow user to define an anonymous or authenticating proxy
- Allow user to test connection/proxy settings
- Return control to LoginDialog

### **NewBackupItemDialog**

Extends: BaseForm  
Invoked By: MainForm

This form allows the user to create a Backup Item. It provides functionality to:

- Allow user to choose a folder for backup
- Give a name to the Backup Item
- Create a backup schedule for the new Backup Item which becomes effective immediately

### **BackupScheduleDialog**

Extends: S3OnTheGo.Backup.Presentation.BaseForm  
Invoked by: MainForm

BackupScheduleDialog allows a user to create a backup schedule for an already-existing Backup Item.

### **DeleteDialog**

Extends: S3OnTheGo.Backup.Presentation.BaseForm  
Invoked by: MainForm

DeleteDialog is a 'yes or no' pop-up which confirms a user to delete a Backup Item and all associated previous backups and snapshots of it.

### **PerformBackupDialog**

Extends: S3OnTheGo.Backup.Presentation.CloseToSystemTrayIcon  
Invoked by: MainForm

PerformBackupDialog is a large class which is invoked when a user chooses to perform a backup from a BackupItem, or when it is time to backup a scheduled Backup Item (not visible if the user application is minimised or in system tray).

Its job is to:

- Analyse files, while displaying its overall progress
- Check for locked files
- Once analysed, upload only necessary files, while displaying overall progress
- Display estimated completed time
- Handle a cancellation gracefully
- Allow user to send the process to the system tray

### **RestoreDialog**

Extends: `S3OnTheGo.Backup.Presentation.CloseToSystemTrayIcon`

Invoked by: `MainForm`

`RestoreDialog` is invoked when a user chooses to restore a backup from a chosen Backup Item.

It must:

- Present the user with a list of ‘snapshots’ from which they can choose one to restore from
- After choosing a snapshot, allow user to choose what files from that snapshot they would like to restore (uses a populated `SelectFilesControl`)
- Allow user to restore to original or custom location

### **SelectFilesControl**

Extends: `System.Windows.Form.UserControl`

Invoked by: `RestoreDialog`

`SelectFilesControl` is a complicated custom `UserControl` which is used inside the `RestoreDialog` form to allow the user to choose from a tree, the filenames associated with the currently-selected snapshot. When done, this control returns this list to the `RestoreDialog`. The control must:

- Populate the tree
- Detect when users click the ‘state’ images and act accordingly
- Recursively select/deselect parent and child nodes, depending on what the user clicked and the current state of the tree

### **PerformRestoreDialog**

Extends: `S3OnTheGo.Backup.Presentation.CloseToSystemTrayIcon`

Invoked by: `RestoreDialog`

`PerformRestoreDialog` is invoked after the user has chosen a snapshot and associated files for restoration (which is wrapped in a `RestoreOptions`). This dialog must:

- Check if files exist locally
- Discover locked files
- Restore necessary files

- Display estimated completion time
- Display current and past files being dealt
- Handle cancellation gracefully

### **AboutBox**

Extends: `S3OnTheGo.Backup.Presentation.BaseForm`

Invoked by: `MainForm`

A brief form to display information ‘about’ the S3OnTheGo application.

## **5.3 Functional Description**

Sections 5.1 and 5.2 give a description of the main components of the user application. This section attempts to convey how the system’s components behave to complete the tasks at hand, on a lower level. Rather than give a brief description, it attempts to build on Sections 5.1 and 5.2 to give the reader a deeper understanding of how the major operations of the system behave. Many not-so-important (but not necessarily simple) features of the system have been omitted. Many of the features associated with the graphical user interface are deemed unimportant as part of a Technical Manual, but it should be noted that a similar amount of time went into developing most of these.

Also, please note that while still at a low-level, this section simplifies many details in order to assist the reader.

### **5.3.1 Files and Objects on Amazon S3**

As described in Appendix A, Amazon S3 stores “buckets” and “objects”. A “bucket” could be viewed as a directory, and an “object” could be viewed as a file. Buckets are limited to 100 per account holder and so cannot be a substitute for a directory, and so the Amazon S3 developer is left with the task of creating their own “file system” and naming standards. The S3OnTheGo system only uses one bucket:

```
S3OnTheGo.Backup
```

This section disregards the issues involved with actually using Amazon S3, and only describes the naming standard adopted by the S3OnTheGo system.

Recall, the whole idea of the S3OnTheGo system is so that registered users can backup a collection of their files and restore/synchronise them from *any* location. Also, a user can be logged into S3OnTheGo system on more than one machine simultaneously. Also implemented is the notion of a type of “revision control” where a user can restore a “snapshot” of an older backup. In order to achieve all this, the following information must be stored on Amazon S3, so as to be always available:

## Backup Items

A user may create one or more Backup Items (`BackupItems`) per computer. For any computer used by a user, there is a separate serialized 'List' (containing all the `BackupItem` objects created on that computer, if any) stored as the value of an object on Amazon S3, with the naming convention:

```
<Username>.BackupItems.<Computer Name>
```

“Username” refers to the user’s registered username, and “Computer Name” is the name of the computer which that set of Backup Items belongs. For example, presume a user with the username “cormac” uses S3OnTheGo at three locations, and has created one or more Backup Items on each of these. Then there might be three such objects stored on Amazon S3, i.e.:

```
cormac.BackupItems.workComputer1  
cormac.BackupItems.homeComputer1  
cormac.BackupItems.otherComputer1
```

Everytime a `BackupItem` is created immediately stored on Amazon S3. The reason for storing these on Amazon S3 is to provide the user application with the details of the user’s Backup Items from any location.

## Files

The application breaks large files into numerous objects, in order to store them on Amazon S3.

The naming convention used for file-part objects is as follows:

```
<Username>.BackupItem<BackupItem GUID>.  
file<SHA1 Hash Digest>.<Part>
```

“BackupItem GUID” refers to the `BackupItem` to which the object/file belongs. “SHA1 Hash Digest” is a SHA1 hash created from the concatenation of the file path and the last time that file was written to. Where a file is split into multiple objects, “Part” refers to which part it is.

Data from the file is stored as the object’s value and the object’s metadata includes the number of parts the file is split into, and also the overall size of the whole file.

## Snapshot Information

Each time a backup is performed, a snapshot object is created on Amazon S3, containing information on that particular backup. The naming convention used is:

```
<Username>.<BackupItem GUID>.snapshot<Snapshot GUID>
```

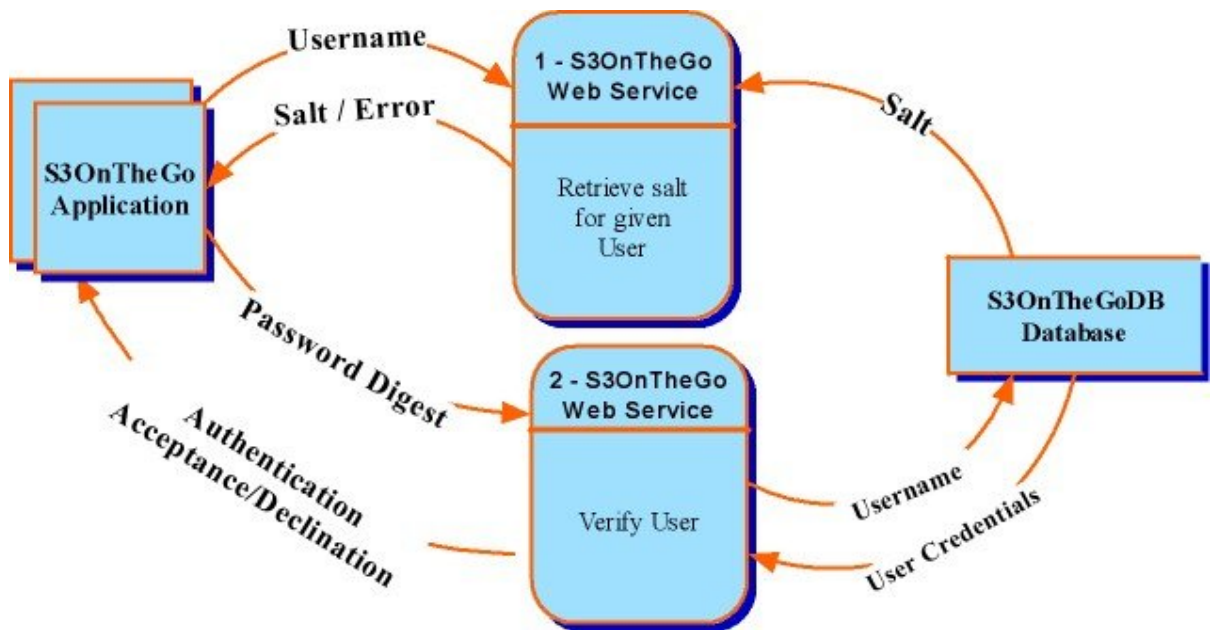
“Username” and “BackupItem GUID” refer to the same as above, where “Snapshot GUID” is the GUID associated with that `Snapshot`. These objects hold metadata containing the date and time of the backup, and the size of the transfer involved. The object’s value is a serialized 'List' of the `BackupFiles` associated with that

backup.

### 5.3.2 Initial User Authentication

The subsection describes the steps involved in authenticating a user upon execution of the application.

*DFD describing the behaviour in dealing with “Initial User Authentication”:*



The following is a list of the operations involved:

1. MainForm checks the user’s “application data path” for a configuration file
2. A LoginDialog is presented to user, with user settings (if any) already set
3. User fills in credentials (if not already filled in) and clicks “Login”
4. LoginDialog parses and verifies input
5. A ProgressPopup is invoked and displayed to user
6. A LoginDialog background worker requests a salt from S3OnTheGo web service, handing it the username
7. The S3OnTheGo web service invokes a stored procedure on S3OnTheGo requesting the salt, and returns this value, or null if user doesn’t exist
8. On return of a valid salt, LoginDialog creates a SHA1 hash digest of the concatenation of the user-given password and the salt
9. LoginDialog requests authentication from S3OnTheGo web service, handing it the user’s username and hashed password
10. S3OnTheGo web service, through a stored procedure, gets and compares the given username and hashed password from that in the S3OnTheGo database
11. S3OnTheGo web service returns results
12. LoginDialog’s background worker closes the ProgressPopup and parses result
  - a. Verified: Hand control to MainForm

- b. Declined: Inform user, and re-request login
- 13. MainForm serializes user chosen settings to configuration file

The reason for authenticating like this is to:

1. Prevent sending plaintext passwords ‘over-the-wire’ (Although SSL is being used at the moment, there is nothing stopping a company from *not* using it)
2. Prevent a malicious administrator from looking at the S3OnTheGo database and retrieving a user’s password

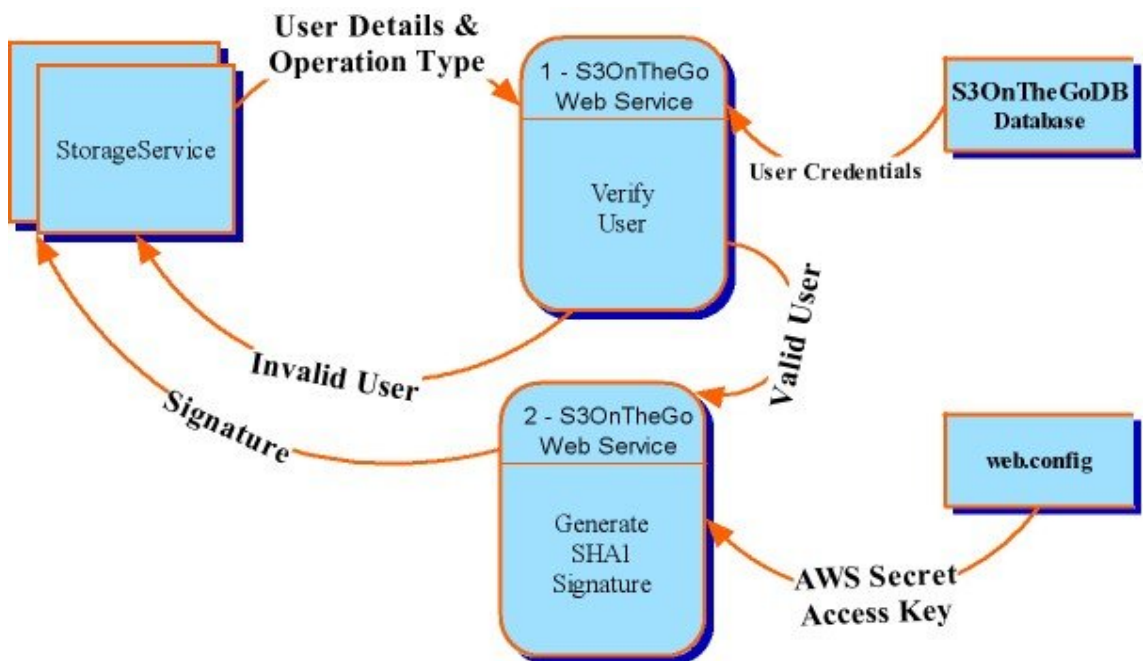
### 5.3.3 Signature Creation

This subsection describes the operations involved with creating an Amazon S3 signature as described in section TODO.

Creating a signature is an expensive process, as it means requesting it from the S3OnTheGo web service each time, and so therefore `StorageService` maintains a cache of signatures in order to reduce to need to re-create a signature for similar operations. When a signature expires, `StorageService` simply requests that the cache updates itself (i.e., re-request a signature from the S3OnTheGo web service).

The following describes the steps involved when a signature is requested for the first time, or when the cache is re-requesting a signature (for any particular operation).

**DFD describing the behaviour of dealing with “Signature Creation”:**



1. `StorageService` requests a signature from S3OnTheGo web service, handing it user credentials (username and hashed password)

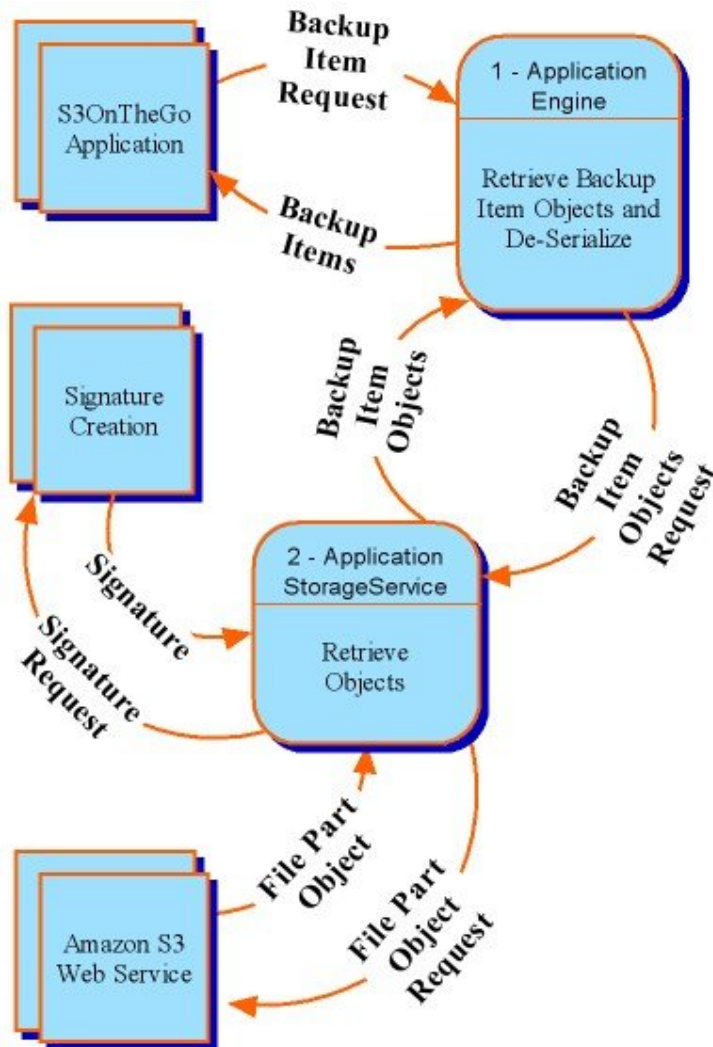
2. S3OnTheGo web service calls a stored procedure and compares credentials
3. And if valid, retrieves the AWS Secret Access Key and creates the SHA1 signature, as described in Appendix A.
4. S3OnTheGo web service returns the signature
5. StorageService caches the signature

The need to include the user credentials as part of a request is to prevent a malicious application from attempting to utilise the web service.

### 5.3.4 Backup Item Retrieval & Storage

Upon logging into the system, a `ListView` is created to display all of the user's Backup Item information, such as which computer each Backup Item was created on, when it was last backed up or restored, and its schedule time, etc.

*DFD describing the behaviour in dealing with "Backup Items Information":*



This is a relatively simple process:

1. A BackgroundWorker in MainForm invokes a ProgressPopup and requests Engine to get user Backup Items
2. Engine generates a key and asks StorageService to retrieve the object
3. Engine deserializes object's value and returns the list to MainForm
4. The ListView gets updates
6. The ProgressPopup is closed

Additionally, when the user creates a Backup Item, creates a schedule, or has just finished a backup, then the same as described above occurs, except that instead of Engine *receiving* Backup Items, it *sends* them (through StorageService) to be stored on Amazon S3. If the user is creating a Backup Item, they are also prompted to perform a backup afterwards.

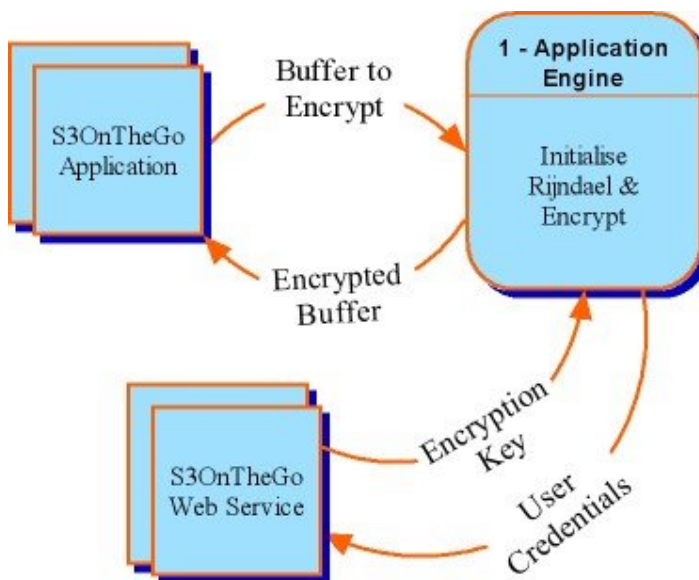
Similarly, if a user chooses to delete a Backup Item, a similar process occurs whereby the Backup Item and all stored snapshot objects and file objects are deleted from Amazon S3.

### 5.3.5 Encryption within S3OnTheGo

This subsection describes the encryption technique used within S3OnTheGo.

As mentioned earlier, a large file is broken up into smaller objects (a small file is just one object) before being uploaded. Each of these objects is encrypted just before uploading, and decrypted when downloaded.

***DFD describing the behaviour of dealing with "Encrypting Data":***



The Engine uses the Rijndael encryption algorithm to encrypt/decrypt data.

If it is the first time the application requires encryption/decryption, the following occurs:

1. `Engine` (through `UserService`) requests the user's encryption key from the S3OnTheGo web service, handing in the user's credentials
2. S3OnTheGo verifies the user as described in the previous sections, and returns the encryption key/salt – or else denies user
3. If verified, `Engine` creates a 384-bit (48 byte) SHA hash from encryption key and splits this into a 256-bit (32 byte) encryption and a 128-bit (16 byte) initialisation vector
4. `Engine` then initialises the Rijndael cryptographic object with these variables

The application only carries the initialisation process out once per application run.

Once Rijndael is initialised, `Engine` encrypts and decrypts data without the need to re-request the encryption key from the S3OnTheGo web service.

Obviously, the encryption keys stored on the S3OnTheGo database are not actually used to encrypt data, but are used as something to create a 384-bit SHA hash from, which is then broken into a key and initialisation vector. This is not an attempt to heighten security in the event of a malicious database administrator – but is used more so for convenience.

That said, it does provide a barrier, to some extent – but relying on confusion to protect data is considered bad practice in a cryptography sense.

For example, an administrator who has access to the S3OnTheGo database and the company's AWS Secret Access key, could potentially grab and decrypt a user's files, if they also knew the exact semantics of the Rijndael initialisation process implemented in the S3OnTheGo user application.

The idea of using the user's password as part of the key-making process is tempting, as it would provide 100% security for users' data. But, then a user could forget their password and be incapable of decrypting their currently encrypted data.

So, there is no “perfect” solution.

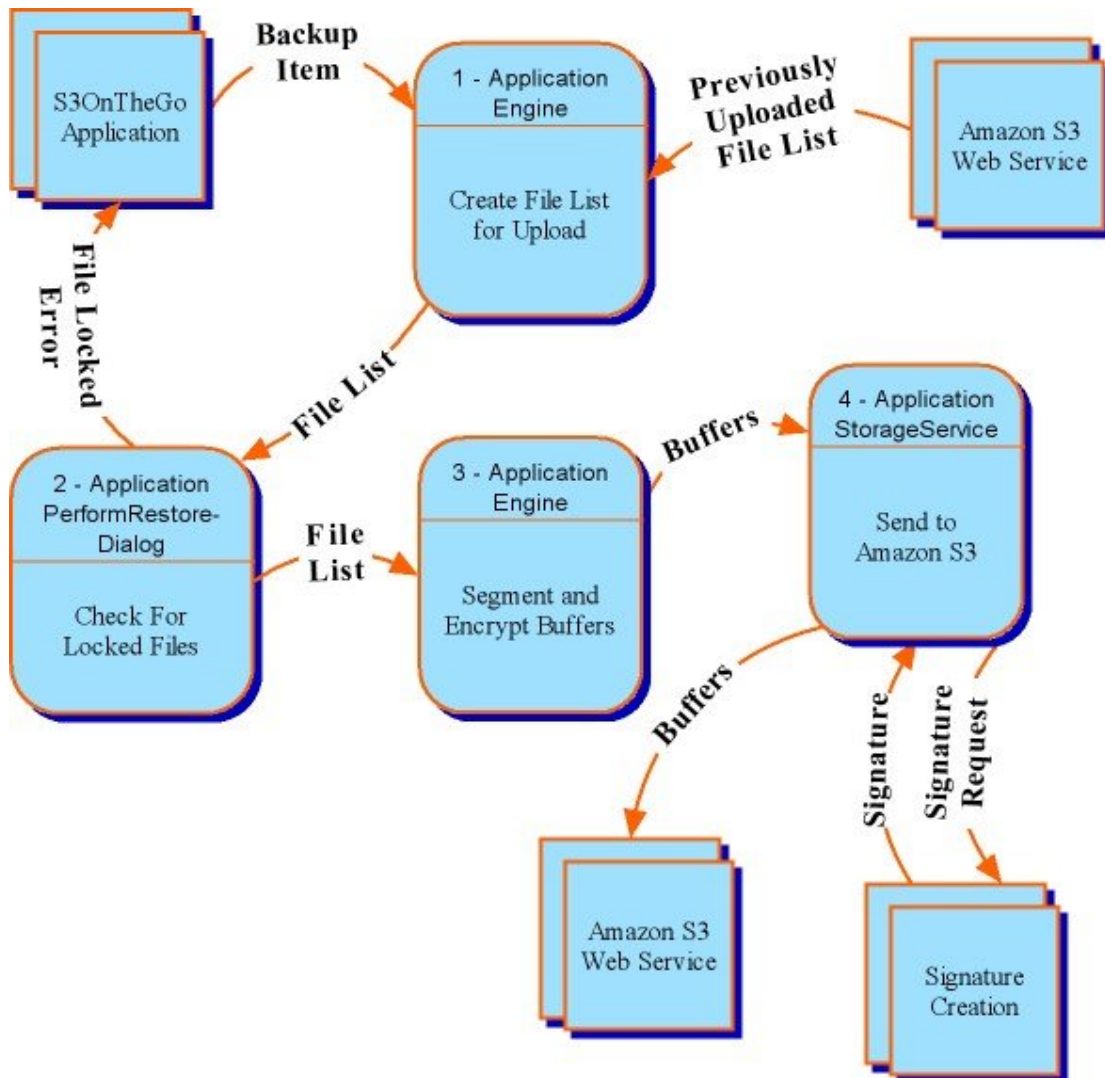
A further step could be taken by encrypting the encryption keys which are stored in the database using DPAPI (which offers key protection) or similar technology. This, however, can still be violated by a malicious administrator, as it must rely on user/password credentials (i.e., the machine account under which it runs).

### 5.3.6 Backing up a Backup Item

This subsection describes the steps carried out in “performing a backup from a Backup Item”. It assumes the Backup Item has already been created.

**Note:** For obvious reasons, a user is not given the choice to backup from a Backup Item which was created on another computer (as it is specific to that computer). They can, however, restore from such a Backup Item.

*DFD describing the behaviour of the application in dealing with “Performing a Backup from a Backup Item”:*



**Note:** MainForm maintains a Timer (backupFromQueueTimer) which “ticks” once a second. Its job is to continuously check a queue (myBackupQueue) and take action accordingly.

The following occurs after a user chooses to backup a Backup Item, or occurs from Step 3 if the scheduler adds a BackupItem to myBackupQueue:

1. Displays a “waiting cursor” and ping the web services through Engine
  - a. If either web service is unavailable, inform the user and cancel operations
2. In a synchronous block of code (to prevent other threads accessing the queue), it adds the selected BackupItem to myBackupQueue

3. In less than a second, the `backupFromQueueTimer` detects this addition and will check if the application is currently backing up a different `BackupItem`
4. If so, it will simply check every other “tick” until it is not
5. Next, in a synchronised block of code, it marks the application as “currently backing up” and “peeks” at `myBackupQueue` (meaning, it gets the `BackupItem` from it, but does not remove it from the queue)
6. It then disables the “Exit” command from the system tray icon’s context menu (the operation can only be cancelled by clicking ‘Cancel’)
7. Next, it displays a system tray balloon popup, informing the user that it is currently backing up
8. Then, it invokes a `PerformBackupDialog`, and hands it the `BackupItem`
9. Then it sets `PerformBackupDialog` as the “current form”

So, now the `MainForm` is no longer visible, and `PerformBackupDialog` is the visible form. This form consists of several components:

- A “checking” `ProgressBar` and `CountDownInfo` label, to display the progress of files are being analysed
- A “upload” `ProgressBar` and countdown label to display upload progress
- A `ListView` of details showing the files currently being dealt with and their progress

The user can “Hide” this window (send to the system tray) or “Cancel”.

Once invoked, `PerformBackupDialog` will:

1. Create a `BackgroundWorker` (`backgroundWorker`), which will:
  - a. Create a `Snapshot` from the current `BackupItem`, and handle any error (i.e., file system errors, etc)
  - b. Then proceed to hand `Engine` this `Snapshot`, and a reference to itself (`performBackupBackgroundWorker`), in order to analyse the files to determine what files are necessary to backup

**Note:** The `performBackupBackgroundWorker` has a delegate called `ReportProgress`, which points to a handler in `PerformBackupDialog` which updates the progress bars and countdown labels accordingly. This handler uses populated instances of `ProgressInfo` to determine the progress of an operation. The delegate is called throughout the file analysing and uploading stages, by both `PerformBackupDialog` and `Engine`.

Also, `PerformBackupDialog` detects if a user clicks “Cancel”, and marks `performBackupBackgroundWorker` as “cancelled”, which is also checked on both layers regularly, and acts accordingly. `PerformBackupDialog` also maintains a list of the objects which have just been uploaded so that in the event that the user does decide to cancel a backup, the `PerformBackupDialog` can request `Engine` to remove this clutter from Amazon S3 (clutter equals wasted money for the AWS account holder).

Next, Engine requests from StorageService, a List of all files which are associated with this BackupItem that have already been uploaded to Amazon S3. Recall that StorageService is a wrapper to the AmazonS3 proxy, and is used as a medium between the application and Amazon S3 proxy.

For each entry in this List, Engine creates a List of what files from the current Snapshot do not already exist on the Amazon S3 server. It also ‘reports progress’ to the ‘checking’ bar by calling performBackupBackgroundWorker’s ReportProgress delegate (with a new instance of ProgressInfo which is initialised accordingly) regularly.

This List is returned to PerformBackupDialog, and each file-to-be-backed-up is checked to make sure it is not ‘locked’ (locked either by the Operating System or another application). If this occurs, users are presented with a list of the files which are locked, and given a chance to close the offending applications.

Next, for each BackupFile in this List, PerformBackupDialog requests that Engine “backs it up”. So for each BackupFile, Engine proceeds to read the actual file and split it up into FILE\_PART\_LENGTH sizes, encrypts them, and requests StorageService to upload them, while implementing the naming standard described in Section 5.3.1, etc.

At the time of writing, FILE\_PART\_LENGTH is 500000 bytes, or half a megabyte. There are a number of reasons for segmenting a file— one of them being purely for error recovery. Doing this, for example, makes it possible to deal with a serious error such as a network problem, allowing the application to simply resume where the uploading left off, when the connection is restored. For example, it is possible to disconnect from the network, and reconnect without the application appearing to have, or reporting any errors.

Another major reason is described in Section 6.1.

During these operations, if StorageService experiences any exception which it cannot handle (i.e., it is not a signature exception or basic SOAP exception), it will throw the exception back to Engine. Engine will detect WebExceptions, and will usually wait WEB\_EXCEPTION\_WAIT\_TIME before retrying. If the WebException is a timeout, it will retry again immediately. For non-WebExceptions it will simply retry MAX\_NON\_WEB\_EXCEPTION\_RETRIES times, before throwing the error up a layer. Typically however, this is rare, and overcoming network problems is easily handled without any notice to the user.

Upon completion, PerformBackupDialog then requests that Engine upload a snapshot of the completed backup, using the naming standards described in Section 5.3.1, before returning control to MainForm.

MainForm now:

1. Informs WindowManager that it is the “current form”
2. Marks the backup item’s “last backup time”
3. Tells Engine to store the user’s BackupItems on Amazon S3
4. Refreshes the applications visual ListView of Backup Items

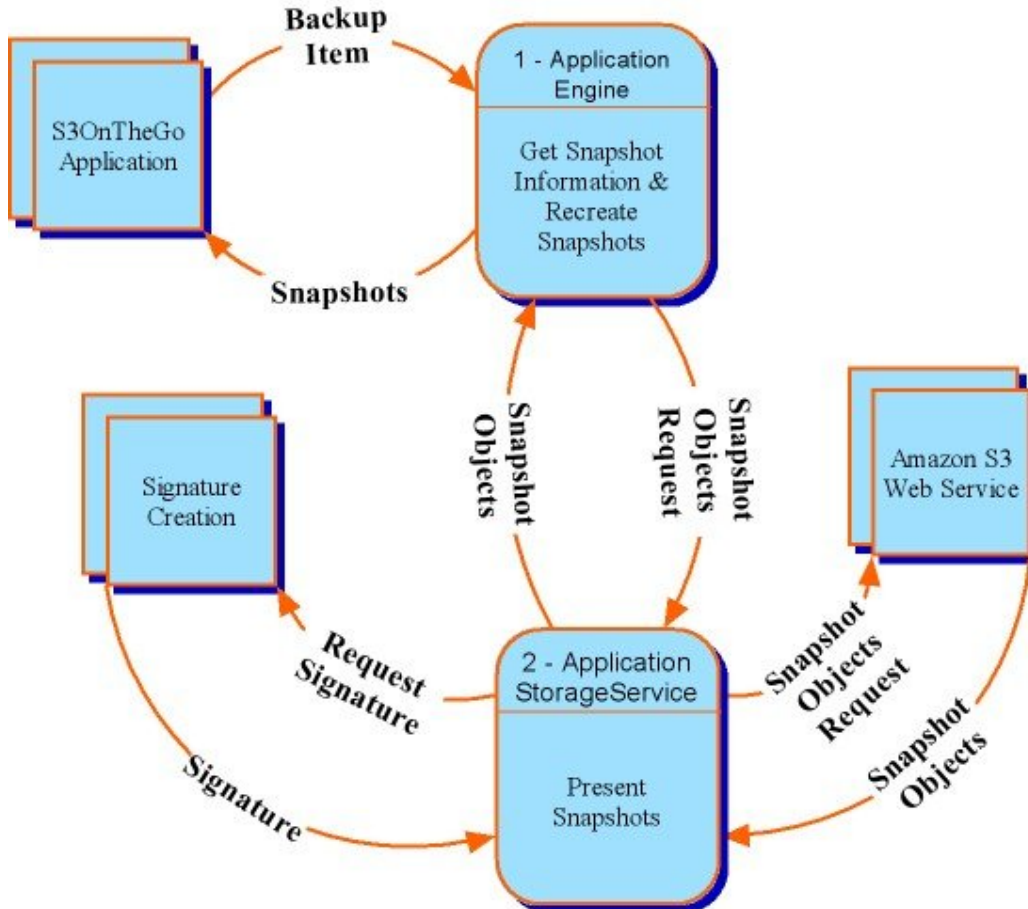
5. Informs user through a balloon popup of the completion of the backup
6. In synchronised code, removes the BackupItem from myBackupQueue, and marks the application as “not backing up” to unblock any waiting operations

It should also be noted, that if an unrecoverable error or cancellation occurred, it would make its way back to MainForm, which would also then inform the user.

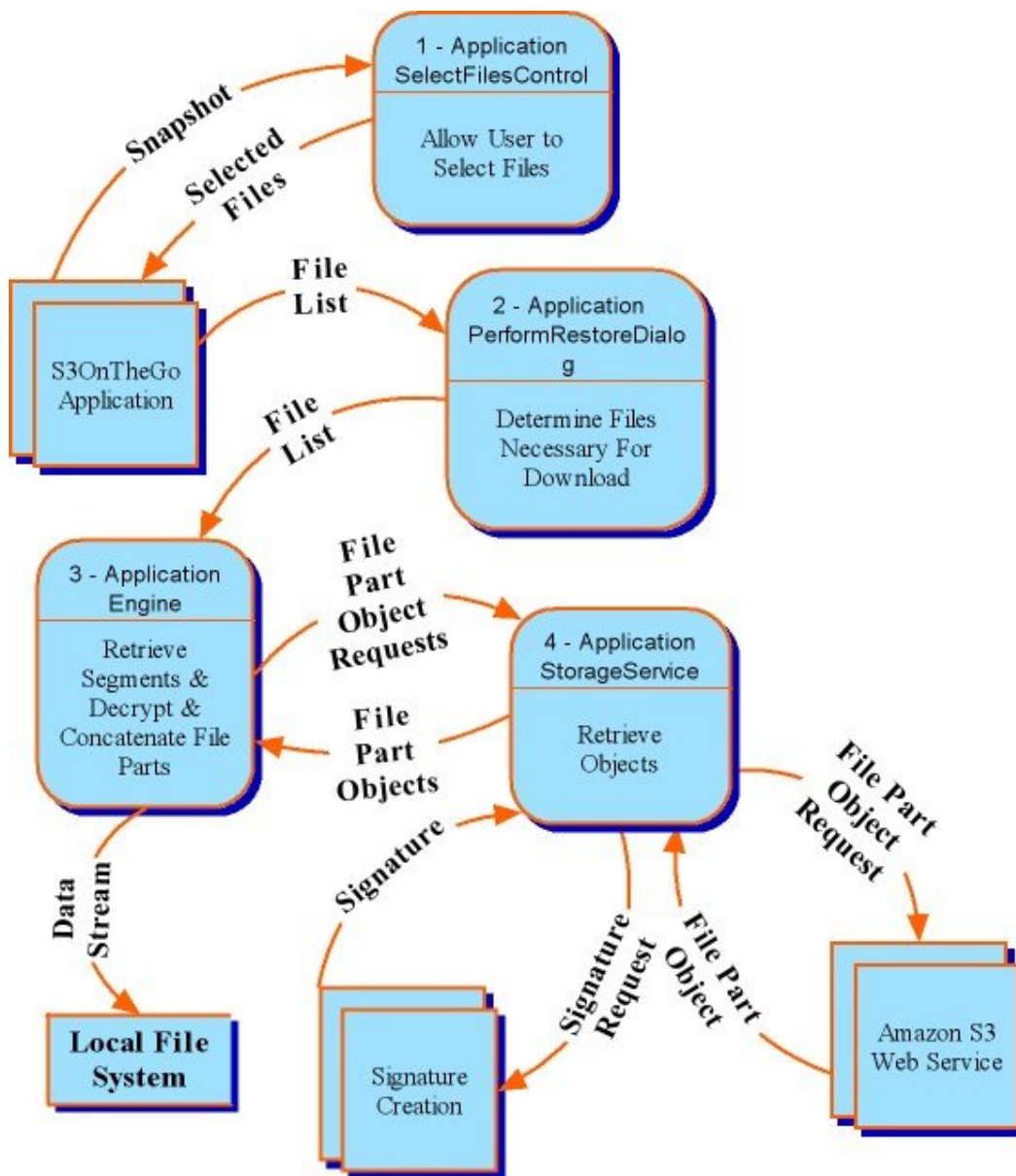
### 5.3.7 Restoring from a Backup Item

This subsection describes the steps carried out in “restoring from Backup Item”. A user is only given the option to restore from a Backup Item when that Backup Item has been backed up at least once. It does not matter where or when.

*DFD describing the behaviour of the application in dealing with “Retrieving Snapshot Information for a given Backup Item”:*



*DFD describing the behaviour of the application in dealing with “Restoring Files from a Snapshot”:*



The following occurs in MainForm after a user chooses to restore from a Backup Item:

- Displays a “waiting cursor” and ping the web services through Engine
  - a. If either web service is unavailable, inform the user and cancel operations
- Invokes a RestoreDialog and hands it the currently selected BackupItem

RestoreDialog then creates a BackgroundWorker (restoreBackgroundWorker) whose first job is to initialise a ProgressPopup,

informing the user that it is requesting snapshot information. Next, it requests `Engine` to retrieve all the Snapshots associated with this `BackupItem` from Amazon S3.

Then `Engine`, one by one, re-creates the original `Snapshot` objects from the retrieved object's values and metadata, and returns a list to `RestoreDialog`, which then closes the `ProgressPopup`.

Some basic checks are carried out to ensure that snapshots exist. This should always be the case, as a user cannot attempt to restore a Backup Item which has never been backed up.

Next, the `Snapshot` list gets sorted and is added to the dropdown list presented to the user. The user then chooses a snapshot, which fires an event which passes the file list from the `Snapshot` to a `SelectFilesControl` (`restoreFilesControl`). `restoreFilesControl` proceeds to populate the tree viewable by the user to create a filesystem-like structure that mimics the exact one at time of the backup.

The user may view several snapshots before deciding on one.

Then, they select the files/directories they wish to restore. Each time a user selects a directory or file, `restoreFilesControl` must recursively iterate through the tree, and check or uncheck different file nodes as necessary. This is to prevent inconsistencies, such as having a file/directory checked, but not having its parent directory checked, etc. Many other conveniences are introduced to aid the user – for example:

- Checking a directory will automatically check all its children file/directory nodes
- A double click will not cause something to be selected and deselected
- Double clicking 'directories' expands and dettracts that structure
- The user can use keys and spacebar to select files, if they wish
- ...etc

The `RestoreDialog` also provides the user with the means to specify where to restore the backup (original or to a custom location). Upon clicking 'OK', `RestoreDialog` verifies that the user input is valid, and then wraps the user selected options in a `RestoreOptions`, which contains a `BackupFile List`, location and the `BackupItem` involved, and some more minor parameters.

The `RestoreDialog` is then closed, and `MainForm` is restored and:

- Notifies the user that a restore is starting through a balloon popup
- Creates a `PerformRestoreDialog` and hands it the `RestoreOptions`
- Marks `PerformRestoreDialog` as the "current window".
- It then disables the "Exit" command from the system tray icon's context menu (it can only be cancelled by clicking 'Cancel')

`PerformRestoreDialog` is a form with two main components:

- A `ListView` to inform the user of completed files and file being dealt with and its current progress
- An overall progress `ProgressBar` and `CountDownInfo` label

`PerformRestoreDialog` has a `BackgroundWorker` (`restoreBackgroundWorker`) which:

- Checks if a file already exists locally (compares the SHA1 digests of the concatenation of the file path and the “last write time”, as described in Section 5.3.1.) to prevent unnecessary download requests.
- Determines the overall transfer size - to be used in progress reporting
- For each `BackupFile` in the list, requests `Engine` to restore it while handing in the `BackupFile`, location to restore to, `restoreBackgroundWorker`, and overall size of transfer, etc

`Engine` firstly creates the root directory, if it does not exist, and proceeds to request (through `StorageService`) metadata from Amazon S3, on the current file-in-question to determine how many parts it is broken up into. It then requests it to get the data for each particular file part, while decrypting them and writing them out to the file stream, thus restoring the whole file.

During these operations, if `StorageService` experiences an exception which it cannot handle (i.e., it is not a signature exception or basic SOAP exception), it will throw the exception back to `Engine`. `Engine` will detect `WebExceptions`, and will usually wait `WEB_EXCEPTION_WAIT_TIME` before retrying. If the `WebException` is a timeout, it will retry again immediately. For non-`WebExceptions` it will simply retry `MAX_NON_WEB_EXCEPTION_RETRIES` times, before throwing the error up a layer. Typically however, this is rare, and overcoming network problems is easily handled without any notice to the user.

Also, just like in the “Performing a backup from a Backup Item” operation, `restoreBackgroundWorker` has a `ReportProgress` delegate, which is called from `PerformRestoreDialog` and `Engine` layers regularly. They populate a `ProgressInfo` according to the current situation and call `ReportProgress` which calls a handler to report progress to the user through the `ProgressBar`, `ListView` and `CountDownInfo` label, etc.

Once the whole operation is complete, the dialog is closed and control is returned to `MainForm`, which:

- Informs `WindowManager` that it is the “current form”
- Refreshes the list of Backup Items
- Informs user through a balloon popup of the completion of the backup

### 5.3.8 Backup Scheduling

This subsection describes how `S3OnTheDeal` handles scheduling a backup.

As described in Section 5.2.2, each `BackupItem` may have a schedule time and frequency associated with it, which the user may have set at creation time, or afterwards.

A `BackupItem` can be scheduled for backup for any minute on a daily, weekly or monthly basis.

`MainForm` maintains a `Timer` (`scheduleTimer`) which becomes active as soon as a user has successfully logged in and the application has retrieved the user's `BackupItems` from Amazon S3.

`scheduleTimer` "ticks" every 60 seconds and examines all of the user's `BackupItems`. If they were created locally and *not* already in `myBackupQueue` it checks to see if the current minute and day/week/month is that which is set in the `BackupItem`. If so, it, in synchronised code, adds the `BackupItem` to `myBackupQueue`, and the `backupFromQueueTimer` will deal with it as described in Section 5.3.6.

### 5.3.9 Proxy Settings within S3OnTheGo

Before communicating over a network, `S3OnTheGo` must know which proxy, if any, to use. As mentioned in Section 5.2.4, `S3OnTheGo` can automatically detect proxy settings. Typically, the standard way to do this is by retrieving Internet Explorer's proxy settings and using the application's (.NET's) default credentials.

Or, the user can define an anonymous or authenticating proxy to use. `S3OnTheGo` creates the user's credentials depending on whether it is authenticating or not. .NET handles the issues of passing the given credentials to the proxy server.

Please note that this section attempts to describe the more important features of the system and is not an extensive analysis of the `S3OnTheGo` user application.

## 6 PROBLEMS AND RESOLUTION

### 6.1 Amazon S3 Reliability

Unfortunately, the Amazon S3 service is temperamental at times. This seems to be an on-going problem, judging from experience and by several complaints received on the developer forums. The most common problem is lack of availability, whereby sometimes the Amazon S3 web service just isn't available. Other times there is latency - but overall this is rare and performance is good and reliable enough.

Furthermore, there are some file size limitations according to the Amazon S3 release notes: "A load balancer bug causes the connection to close whenever an upload request with content-length between 2 GB and 4 GB is received. Amazon has received a fix for the issue from the load balancer vendor, and is in the process of testing the fix." This was part of the incentive to segment files when storing them on Amazon S3.

Because the Amazon S3 web service is no longer beta, such problems are surprising. However, the Amazon S3 team maintain that they are working quickly to eliminate any residing problems, and also declare that there are plans to install data centres in Europe, which would speed the S3OnTheGo system up noticeably (as it will be mostly used in Europe).

### 6.2 Security Issues

While all efforts have been made to develop a fully secure system, there remains one problem whereby it is possible, in theory, for a registered user to develop their own application and utilise the S3OnTheGo web service, using their own credentials to create signatures for use by their own application.

To somewhat get around this, the S3OnTheGo web service is setup to ignore all HTTP GETs or POSTs (but will accept SOAP), thus not exposing the WSDL, or even appearing "online" to any requests sent using HTTP GET/POST - therefore it renders proxy generation tools useless and convinces the online browsing user that no such web service exists. However, it is still possible that someone with exact knowledge of the S3OnTheGo web service could manually create SOAP messages to fool the S3OnTheGo web server; unlikely, but possible.

A perfect solution would be for the S3OnTheGo web service to somehow make sure the calling application is *always* the S3OnTheGo user application. There should be no risk of a different application/user emulating this authenticating procedure.

This sort of security may be possible with Web Services Enhancements (WSE) 3.0, an add-on to the .NET 2.0 Framework. This will be investigated at a later stage to help eliminate the aforementioned risks.

More information on this is available at:

<http://msdn.microsoft.com/msdnmag/issues/06/02/WSE30/>

## 7 INSTALLATION GUIDE

This section describes the steps a company takes to implement the S3OnTheGo system.

### 7.1 Requirements

1. A company internet-accessible web server with the following installed:
  - Microsoft .NET 2.0 Framework
  - SQL Server 2000/SQL Server 2005
  - IIS 5.0 or later
  - Certificate Server (For SSL authentication)

Alternatively, the SQL Server could be installed on a separate machine, internally.

2. An Amazon Web Services Account

This can be purchased with a credit card via <http://aws.amazon.com> and provides an AWS Access Key and an AWS Secret Access Key (See Appendix A).

### 7.2 Installation

The following should be carried out in the order presented:

#### 7.2.1 Setting up the S3OnTheGo Database

The S3OnTheGo database must be installed on any version of Microsoft's SQL Server. For improved security, install it on a separate machine from the web server (which is described below). It is expected that the reader has database administration experience.

**Note:** Administrator access is required.

#### Create Database

Firstly, create a new database called:

```
S3OnTheGo
```

Next, add the following database users, choosing whichever passwords you wish:

```
S3OnTheGoAdmin  
S3OnTheGoUser
```

Both users should be granted 'connect' access. S3OnTheGoUser requires db\_datareader access to the S3OnTheGo database, while S3OnTheGoAdmin requires db\_datawriter access.

### Create S3OnTheGoUsers Table

Run the following script on the S3OnTheGo database:

```
IF EXISTS (SELECT * FROM sysobjects WHERE type = 'U' AND name =
'S3OnTheGoUsers')
    BEGIN
        DROP Table S3OnTheGoUsers
    END
GO

CREATE TABLE S3OnTheGoUsers (
    [UserName] [varchar] (20) NOT NULL ,
    [PasswordHash] [varchar] (100) NOT NULL ,
    [Salt] [varchar] (8) NOT NULL ,
    [EncryptionKey] [varchar] (50) NOT NULL ,
    CONSTRAINT [PK_S3OnTheGoUsers] PRIMARY KEY CLUSTERED
    (
        [UserName]
    ) ON [PRIMARY]
) ON [PRIMARY]
GO

USE master
GO

GRANT SELECT ON S3OnTheGoUsers TO [S3OnTheGoUser]
GO
```

**Warning:** This will remove the S3OnTheGoUsers table if it already exists.

### Install Stored Procedures

Next, run the following script to install the stored procedures:

```
IF EXISTS (SELECT * FROM sysobjects WHERE type = 'P' AND name =
'CreateUser')
    BEGIN
        DROP Procedure CreateUser
    END
GO

CREATE PROCEDURE CreateUser
@userName varchar(20),
@passwordHash varchar(100),
@salt varchar(8),
@encryptionKey varchar(50)
AS
INSERT INTO S3OnTheGoUsers VALUES(@userName, @passwordHash, @salt,
@encryptionKey)
```

```

GO
GRANT EXEC ON CreateUser TO [S3OnTheGoAdmin]
GO

IF EXISTS (SELECT * FROM sysobjects WHERE type = 'P' AND name =
'GetCryptoKey')
    BEGIN
        DROP Procedure GetCryptoKey
    END

GO
CREATE PROCEDURE GetCryptoKey
@userName varchar(20)
AS
SELECT EncryptionKey
FROM S3OnTheGoUsers
WHERE UserName = @userName
GO
GRANT EXEC ON GetCryptoKey TO [S3OnTheGoUser]
GO

IF EXISTS (SELECT * FROM sysobjects WHERE type = 'P' AND name =
'LookupUser')
    BEGIN
        DROP Procedure LookupUser
    END

GO
CREATE PROCEDURE LookupUser
@userName varchar(20)
AS
SELECT PasswordHash, Salt
FROM S3OnTheGoUsers
WHERE UserName = @userName
GO
GRANT EXEC ON LookupUser TO [S3OnTheGoUser]
GO

```

**Warning:** This will overwrite any existing stored procedures of the same name.

The database is now setup.

## 7.2.2 Setting up the S3OnTheGo Web Service

Firstly, locate the “PrecompiledWeb\S3OnTheGoWebService” directory on the installation CD and copy to a temporary directory.

Browse to this directory and open up Web.Config in a text editor and locate the line:

```

<add key = "awsSecretAccessKey" value = "<AWS Secret
Access Key Goes Here>" />

```

Replace <AWS Secret Key Goes Here> with the purchased AWS Secret Access Key.

Next, locate the line:

```
<add name = "S3OnTheGoServiceDB" connectionString =  
"Server = <computerName>\<SQLServerName>; Database =  
S3OnTheGo; uid = S3OnTheGoUser; pwd = <S3OnTheGoUser's  
password goes here>;" />
```

Replace <computerName> and <SQLServerName> with the computer and SQL Server name, where the S3OnTheGo database was installed.

Now replace <S3OnTheGoUser's password goes here> with the password which was supplied when creating the S3OnTheGoUser account.

The web service (everything in the directory) is now ready to be deployed to IIS (5.0 or later).

Refer to the IIS documentation on “Deploying ASP.NET applications” and “How to set up SSL by using IIS and Certificate Server” in order to provide the service over SSL.

**Note:** `Web.Config` contains other security settings, which should only be altered by an experienced person.

### 7.2.3 Setting up the S3OnTheGo Registration Web Application

For the company’s benefit, the source code of a registration/administration web application has been provided in the directory “AdminApp”. Wrapped in a Visual Studio 2005 project and written in ASP.NET and C#, it provides a foundation for an application which is used to allow user registration for the S3OnTheGo system. Such an application may be potentially exposed through the web, or used internally by administrators, depending on a company’s needs.

To install, locate the “AdminApp” directory on the installation CD and copy to a temporary directory.

Open up `Web.Config` in a text editor and locate the line:

```
<add key = "awsSecretAccessKey" value = "<AWS Secret  
Access Key Goes Here>" />
```

Replace <AWS Secret Key Goes Here> with the purchased AWS Secret Access Key.

Next, locate the line:

```
<add name = "S3OnTheGoServiceDB" connectionString =  
"Server = <computerName>\<SQLServerName>; Database =
```

```
S3OnTheGo; uid = S3OnTheGoUser; pwd = <S3OnTheGoAdmin's  
password goes here>;"/>
```

Replace <computerName> and <SQLServerName> with the computer and SQL Server name, where the S3OnTheGo database was installed.

Now replace <S3OnTheGoAdmin's password goes here> with the password which was supplied when creating the S3OnTheGoAdmin database account.

The web service (the whole directory) is now ready to be deployed to IIS (5.0 or later).

Refer to the IIS documentation on “Deploying ASP.NET applications” and “How to set up SSL by using IIS and Certificate Server” in order to provide the service over SSL.

**Note:** This sample is provided as a foundation to create a “user registration” web application, or an internal administration application. When extending this application (or building one from scratch), it is essential to sustain the procedure involved in creating the `passwordHash` variable, as the same implementation is used in the S3OnTheGo user application in order to compare hash digests. Also, to avoid potential error, it is recommended to maintain the process used to call the stored procedure (`CreateUser`) in the S3OnTheGo database.

#### 7.2.4 Deploying the S3OnTheGo User Application

The S3OnTheGo user application is located in the “UserApplication/S3OnTheGo” directory on the CD and can be deployed to users “as is”.

The `setup.exe` file will start an installation process whereby the user can choose a destination installation directory. If the user’s machine does not have .NET 2.0 installed they will be prompted to install it and directed to a download location.

Thereafter, the user can access the application from either the Start menu, or a desktop or ‘quicklaunch’ shortcut. The application will also run each time Windows is launched, and is minimised to the system tray by default (handy for scheduled backups).

Upon executing the application, users are prompted to enter the company’s AWS Access Key and S3OnTheGo web service URL. These should be provided to users at time of deployment, and will automatically be saved after entering.

**Note:** The AWS Access Key is not the AWS *Secret* Access Key, and is safe to distribute.

## 8. APPENDICES

### A. Amazon S3 Overview

It is crucial to understand how the Amazon S3 web service works in order to understand the major topics in this document:

Amazon S3 has a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. It is intentionally built with a minimal feature set:

- Write, read, and delete objects containing from 1 byte to 5 gigabytes of data each, with accompanying metadata. The number of objects which can be stored is unlimited
- A straightforward flat object store model, where each object is stored and retrieved via a unique developer-assigned key.
- Authentication mechanisms are provided to ensure that data is kept secure from unauthorized access. Objects can be made private or public, rights granted to specific users (AWS account holders).
- A SOAP interface designed to work with any Internet-development toolkit.

#### ***Bucket:***

A bucket is simply a container for objects stored in Amazon S3. Each S3 account holder can create up to 100 buckets. Every object is contained within a bucket.

#### ***Object:***

Amazon S3 is used to store objects. An object has four parts: value, key, metadata, and an access control policy.

There is no notion of a file system, and it is left up to the developer to define their own by using keys and metadata:

#### ***Key:***

The key is the handle that is assigned to an object so that it can be fetched later. No two objects in a bucket may have the same key. A key is a sequence of Unicode characters whose UTF-8 encoding is at most 1024 bytes long.

Keys can be listed by prefix. By choosing a common prefix for the names of related keys, and marking these keys with a special character that delimit hierarchy, you can use the list operation to select and browse keys hierarchically. This idea is similar to how groups of files are organized into directories in a file system.

Keys are often given a suffix that indicates something about the type of data in the object, though this is not required. For example, often keys will use a suffix of ".jpg" to indicate that an object is an image.

**Metadata:**

An Amazon S3 object's metadata is a set of key-value pairs associated with the object. The idea is to store information about the object in its metadata. There are two kinds of metadata: system metadata, and user metadata.

- System metadata is utilized by Amazon S3, and is sometimes computed by Amazon S3
- User metadata entries are specified by the developer, the user of Amazon S3.
- Amazon S3 does not interpret this metadata - it simply stores it, and then passes it back when it is requested
- Metadata keys and values can be any length, but must conform to UTF-8

**A.1. Interacting with Amazon S3**

This system will interact with S3 using SOAP 1.1 over HTTP. The S3 WSDL, which describes the S3 API in a machine-readable way, is available at: <http://s3.amazonaws.com/doc/2006-03-01/AmazonS3.wsdl>.

**SOAP Endpoint:**

This system will send S3 SOAP messages to an SSL secured endpoint. Note that authenticated SOAP requests are only accepted over SSL. The available S3 SOAP endpoints are <http://s3.amazonaws.com/soap> and <https://s3.amazonaws.com/soap> (SSL).

**A.2. Common Elements**

The account holder is assigned an AWS Access Key ID and Secret Access Key when they register for an AWS account at <http://aws.amazon.com>. The following are authorization-related elements necessary with *every* SOAP request:

**AWSAccessKeyId:**

The account holder's AWS Access Key ID.

**Timestamp:**

This must be a dateTime (<http://www.w3.org/TR/xmlschema-2/#dateTime>) in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as 2005-01-31T23:59:59.183Z. Authorization will fail if this timestamp is more than 15 minutes away from the clock on Amazon S3 servers.

**Signature:**

The signature is a RFC 2104 HMAC-SHA1 digest (described in glossary or at <http://www.ietf.org/rfc/rfc2104.txt>) of the concatenation of "AmazonS3" + OPERATION + Timestamp, using the AWS Secret Access Key as the key. For example, in the following CreateBucket sample request, the signature element would contain the HMAC-SHA1 digest of the value "AmazonS3CreateBucket2005-01-31T23:59:59.183Z":

```
<CreateBucket xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Bucket>quotes</Bucket>
  <Acl>private</Acl>
  <AWSAccessKeyId>1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
```

<Timestamp>2005-01-31T23:59:59.183Z</Timestamp>  
<Signature>SZf1CHmQ/nrZbsrC13hCZS061yws</Signature>  
</CreateBucket>