

Z Z Z Z Z Z Z Z Z Z

# **Formal Specification and Documentation using Z:**

## **A Case Study Approach**

### **File Storage Service User Manual**

#### **Jonathan Bowen**

**Department of Computer Science  
University of Reading**

Email: *J.P.Bowen@reading.ac.uk*

URL:

*<http://www.comlab.ox.ac.uk/oucl/users/jonathan.bowen/zbook.html>*

Z Z Z Z Z Z Z Z Z Z

# File Storage Service

- Network service
- Abstract state
- Remote procedure calls
- Modelled as operations in Z
- User Manual

## Basic types and definitions

Clients of the service have user numbers;

Operations may return reports

Units of data are byte values:

$$[UserNum, Report, ByteVal]$$

There is a special guest user number:

$$| GuestNum : UserNum$$

Files have a maximum size:

$$| MaxFileSize : \mathbb{N}$$

File data consists of a sequence of byte values up to this maximum size:

$$Data == \{s : seq\ ByteVal \mid \#s \leq MaxFileSize\}$$

A file size may range from zero up to the max size:

$$Size == 0..MaxFileSize$$

# Time

Time is modelled as natural numbers:

$$Time == \mathbb{N}$$

There is a *total order* on time:

$$\left| \begin{array}{l} \_ \leq \_ : Time \leftrightarrow Time \\ \hline (\_ \leq \_) \in total\_order \end{array} \right.$$

**Partial order** – reflexive, antisymmetric and transitive:

$$\begin{aligned} partial\_order[X] == \\ \{ \_ \underline{R} \_ : X \leftrightarrow X \mid (\forall x, y, z : X \bullet \\ x \underline{R} x \wedge \\ (x \underline{R} y \wedge y \underline{R} x) \Rightarrow x = y \wedge \\ (x \underline{R} y \wedge y \underline{R} z) \Rightarrow x \underline{R} z) \} \end{aligned}$$

**Total order** – partial order where all elements are related:

$$\begin{aligned} total\_order[X] == \\ \{ \_ \underline{R} \_ : partial\_order[X] \mid \\ (\forall x, y : X \bullet x \underline{R} y \vee y \underline{R} x) \} \end{aligned}$$

## Files

The service will store this data within a file:

*File*

*owner : UserNum*

*created, updated, expires : Time*

*contents : Data*

*created ≤ updated*

*created ≤ expires*

File identifiers issued by the service:

[*FileId*]

Client's reference to a file.

Special *NullFileId* never issued by the service:

| *NullFileId : FileId*

## File system state

New identifiers not used, *NullFileId* not available:

$FS$
$files : FileId \mapsto File$ $newids : \mathbb{F} FileId$
$newids \cap \text{dom} files = \emptyset$ $NullFileId \notin newids$

*NullFileId* indicates 'no file'.

## Initial state

No files, all ids except *NullFileId* available for use:

$InitFS$
$FS'$
$files' = \emptyset$ $newids' = FileId \setminus \{NullFileId\}$

## Change of state

Operations only performed by an authentic client:

*clientnum* : *UserNum*

The current time is available from the time service:

*now* : *Time*

A report is always provided:

*report!* : *Report*

Any identifier issued become unavailable:

*newids'* = *newids* \ *domfiles'*

Combined:

$\Delta FS$

*FS*

*FS'*

*clientnum* : *UserNum*

*now* : *Time*

*report!* : *Report*

*newids'* = *newids* \ *domfiles'*

## No change of state

Status operations leave the state unaffected:

$$\Xi FS \equiv [\Delta FS \mid \theta FS' = \theta FS]$$

Note: this is the default definition for  $\Xi FS$ .

$\theta FS$  is a schema *tuple* consisting of all the named components of  $FS$  (*owner*, *created*, etc.).

## Partial operations

Some operations require an existing file  $id?$  input:

$\Phi FileId?$
$\Delta FS$
$id? : FileId$
$File$
$\theta File = files\ id?$

Other operations create a new file and output its  $id!$ :

$\Phi FileId!$
$\Delta FS$
$id! : FileId$
$File'$
$clientnum \neq GuestNum$
$owner' = clientnum$
$updated' = now$
$id! \in newids$
$files' = files \cup \{id! \mapsto \theta File'\}$

## Generic definitions

Functions to manipulate file data:

$[B]$	<p> <math>\underline{\textit{after}}</math> ,  <math>\underline{\textit{for}}</math> : <math>(\text{seq } B) \times \mathbb{N} \rightarrow (\text{seq } B)</math>  <math>\underline{\textit{shifted}}</math> : <math>(\text{seq } B) \times \mathbb{N} \rightarrow (\mathbb{N} \twoheadrightarrow B)</math> </p> <hr style="width: 50%; margin-left: 0;"/> <p> <math>\forall s : \text{seq } B; n : \mathbb{N} \bullet</math>  <math>s \underline{\textit{after}} n = (\{0\} \triangleleft \textit{succ}^n) \circ s \wedge</math>  <math>s \underline{\textit{for}} n = (1..n) \triangleleft s \wedge</math>  <math>s \underline{\textit{shifted}} n = \textit{succ}^{-n} \circ s</math> </p>
-------	---

Data *after* a certain position, *for* a given number of bytes, or *shifted* by a specified offset.

Some ‘holes’ may be created in a file’s *contents* if *data* is written after the end of a file.

<p> <math>\textit{Background} : \textit{ByteVal}</math>  <math>\textit{holes} : \text{seq } \textit{ByteVal}</math> </p> <hr style="width: 50%; margin-left: 0;"/> <p> <math>\text{dom } \textit{holes} = 1.. \textit{MaxFileSize}</math>  <math>\text{ran } \textit{holes} = \{\textit{Background}\}</math> </p>
---

## Error reports

The *report!* output indicates success or failure with a number of different reports:

*SuccessReport, NoSuchFileReport,  
NoSpaceReport, NotOwnerReport,  
NotKnownUserReport,  
BadOperationReport : Report*

$\langle \textit{SuccessReport, NoSuchFileReport, NoSpaceReport, NotOwnerReport, NotKnownUserReport, BadOperationReport} \rangle \in \textit{iseq Report}$

Successful report:

$\textit{Success} \equiv$   
 $[\textit{report!} : \textit{Report} \mid \textit{report!} = \textit{SuccessReport}]$

Normally failure leaves the service unchanged.

## File not found:

*NoSuchFile* \_\_\_\_\_

$\exists FS$

$id? : FileId$

$id? \notin \text{dom files}$

$report! = NoSuchFileReport$

## No space left:

*NoSpace* \_\_\_\_\_

$\exists FS$

$report! = NoSpaceReport$

## Not owner of file:

*NotOwner* \_\_\_\_\_

$\exists FS$

$owner : UserNum$

$owner \neq \text{clientnum}$

$report! = NotOwnerReport$

## Not known user:

*NotKnownUser*

*clientnum* : *UserNum*

*report!* : *Report*

*clientnum* = *GuestNum*

*report!* = *NotKnownUserReport*

Also:

*BadOperation*

*report!* : *Report*

*report!* = *BadOperationReport*

*IsKnownUser*

*clientnum* : *UserNum*

*clientnum*  $\neq$  *GuestNum*

## **Service operations**

### **Client operations**

*NewFile* – create a new file of zero length.

*WriteFile* – write data to a stored file.

*ReadFile* – read data from a stored file.

*DestroyFile* – remove a stored file from the service.

*FileStatus* – obtain complete status of a stored file.

*SetFileExpiry* – set the expiry time of a stored file.

*SetFileLength* – set the length of a stored file.

## Manual pages

1. **Abstract** section: procedure heading for the operation, with formal parameters, as it might appear in a programming language.
2. **Definition** section: mathematically defines the operation, by giving a schema which includes every formal parameter of the procedure heading (with a short description).
3. **Reports** section: gives the definition of the *total* operation including all the possible (success or failure reporting) values of *report!*.

## NEWFILE

### Abstract

```
NewFile ( expires? : Time;
          id!       : FileId;
          report!   : Report )
```

### Definition

$NewFile$
$\Delta FS$
$expires? : Time$
$\Phi FileId!$
$created' = now$
$expires' = \max\{expires?, now\}$
$contents' = \langle \rangle$

### Reports

$$NewFile_1 \equiv$$
$$(NewFile \wedge IsKnownUser \wedge Success)$$
$$\vee NoSpace$$
$$\vee (\exists FS \wedge NotKnownUser)$$

## WRITEFILE

### Abstract

```
WriteFile ( id?      : FileId;
            offset?  : Size;
            data?    : Data;
            id!      : FileId; report! : Report )
```

### Definition

*WriteFile*

$\Delta FS$

$\Phi FileId?$

*offset?* : *Size*

*data?* : *Data*

$\Phi FileId!$

$created' = created \wedge expires' = expires$

$contents' = (holes\ for\ offset?) \oplus contents \oplus$   
 $Size \triangleleft (data?\ shifted\ offset?)$

### Reports

$WriteFile_1 \equiv$

$(WriteFile \wedge IsKnownUser \wedge Success)$

$\vee NoSpace \vee NoSuchFile$

$\vee (\exists FS \wedge NotKnownUser)$

# READFILE

## Abstract

```
ReadFile ( id?      : FileId;
           offset?  : Size;
           length?  : Size;
           data!    : Data;
           report!  : Report )
```

## Definition

*ReadFile*

$\exists FS$

$\Phi FileId?$

*offset?*,

*length?* : *Size*

*data!* : *Data*

*data!* = *contents after offset? for length?*

## Reports

*ReadFile*<sub>1</sub>  $\equiv$

(*ReadFile*  $\wedge$  *Success*)

$\vee$  *NoSuchFile*

## DESTROYFILE

### Abstract

```
DestroyFile ( id?      : FileId;  
              report!  : Report )
```

### Definition

$DestroyFile$
$\Delta FS$
$\Phi FileId?$
$clientnum = owner$
$files' = \{id?\} \triangleleft files$

### Reports

$$DestroyFile_1 \equiv$$
$$\begin{aligned} & (DestroyFile \wedge IsKnownUser \wedge Success) \\ & \vee NotOwner \\ & \vee NoSuchFile \\ & \vee (\exists FS \wedge NotKnownUser) \end{aligned}$$

# FILESTATUS

## Abstract

```
FileStatus ( id?      : FileId;  
            owner!   : UserNum;  
            created!, updated!, expires! : Time;  
            length!  : Size; report!   : Report )
```

## Definition

*FileStatus*

$\exists FS$

$\Phi FileId?$

*owner!* : *UserNum*;

*created!*, *updated!*, *expires!* : *Time*;

*length!* : *Size*

*owner!* = *owner*

*created!* = *created*

*updated!* = *updated*

*expires!* = *expires*

*length!* = *#contents*

## Reports

*FileStatus*<sub>1</sub>  $\equiv$

$(FileStatus \wedge Success) \vee NoSuchFile$

## SETFILEEXPIRY

### Abstract

```
SetFileExpiry ( id?      : FileId;  
                expires? : Time;  
                id!      : FileId;  
                report!  : Report )
```

### Definition

*SetFileExpiry* \_\_\_\_\_

$\Delta FS$

$\Phi FileId?$

*expires?* : Time

$\Phi FileId!$

*created'* = *created*

*expires'* =  $\max\{expires?, now\}$

*contents'* = *contents*

### Reports

*SetFileExpiry*<sub>1</sub>  $\equiv$

$(SetFileExpiry \wedge IsKnownUser \wedge Success)$

$\vee NoSuchFile$

$\vee (\exists FS \wedge NotKnownUser)$

## SETFILELENGTH

### Abstract

```
SetFileLength ( id?      : FileId;
                length?  : Size;
                id!       : FileId;
                report!   : Report )
```

### Definition

*SetFileLength*

$\Delta FS$

$\Phi FileId?$

*length?* : *Size*

$\Phi FileId!$

*created'* = *created*

*expires'* = *expires*

*contents'* = (*holes*  $\oplus$  *contents*) for *length?*

### Reports

*SetFileLength*<sub>1</sub>  $\cong$

(*SetFileLength*  $\wedge$  *IsKnownUser*  $\wedge$  *Success*)

$\vee$  *NoSpace*

$\vee$  *NoSuchFile*

$\vee$  ( $\exists FS \wedge$  *NotKnownUser*)

# Implementation operations

## SCAVENGEFILE

*ScavengeFile* – remove an expired file.

### Abstract

ScavengeFile ( id? : FileId )

*FileService* : *UserNum*

*FileService* ≠ *GuestNum*

### Definition

*ScavengeFile* \_\_\_\_\_

$\Phi$ *FileId*?

*clientnum* = *FileService*

*expires* < *now*

*files'* = {*id?*}  $\triangleleft$  *files*

## Costs

$Money == \mathbb{N}$

Operations and errors:

| *NewFileCost, WriteFileCost,*  
| *ReadFileCost, DestroyFileCost,*  
| *FileStatusCost, SetFileExpiryCost,*  
| *SetFileLengthCost, FSErrorCost : Money*

Read/writing costs:

| *StoreByteCost, ReadByteCost,*  
| *GetIdCost : Money*

## Total operations

$[Op]$

$\Phi Op \equiv [op? : Op]$

*NewFileOp, WriteFileOp,  
ReadFileOp, DestroyFileOp,  
FileStatusOp, SetFileExpiryOp,  
SetFileLengthOp : Op*

$\langle$ *NewFileOp, WriteFileOp,  
ReadFileOp, DestroyFileOp,  
FileStatusOp, SetFileExpiryOp,  
SetFileLengthOp* $\rangle \in \text{iseq } Op$

## *Tariff*

$\Delta FS$

$op? : Op$

$\Phi FileId?$

$\Phi FileId!$

$data! : Data$

$idset! : \mathbb{F} FileId$

$cost! : Money$

$report! = SuccessReport \Rightarrow$

$op? = NewFileOp \Rightarrow cost! = NewFileCost$

$op? = WriteFileOp \Rightarrow cost! = WriteFileCost$   
 $+ StoreByteCost * (expires' - updated') * \#contents'$

$op? = ReadFileOp \Rightarrow cost! = ReadFileCost$   
 $+ ReadByteCost * \#data!$

$op? = DestroyFileOp \Rightarrow cost! = DestroyFileCost$   
 $- StoreByteCost * (expires' - updated') * \#contents'$

$op? = FileStatusOp \Rightarrow cost! = FileStatusCost$

$op? = SetFileExpiryOp \Rightarrow cost! = SetFileExpiryCost$   
 $+ StoreByteCost * (expires' - expires) * \#contents'$

$op? = SetFileLengthOp \Rightarrow cost! = SetFileLengthCost$   
 $+ StoreByteCost * (expires' - updated')$   
 $* (\#contents' - \#contents)$

$report! \in \{NoSuchFileReport, NoSpaceReport,$   
 $NotOwnerReport, NotKnownUserReport\} \Rightarrow$   
 $cost! = FSErrorCost$

## Combined operations

$$\begin{aligned} Ops \equiv & Tariff \wedge ((\exists FS \wedge BadOperation) \oplus \\ & ([NewFile_1; \Phi Op \mid op? = NewFileOp] \vee \\ & [WriteFile_1; \Phi Op \mid op? = WriteFileOp] \vee \\ & [ReadFile_1; \Phi Op \mid op? = ReadFileOp] \vee \\ & [DestroyFile_1; \Phi Op \mid op? = DestroyFileOp] \vee \\ & [FileStatus_1; \Phi Op \mid op? = FileStatusOp] \vee \\ & [SetFileExpiry_1; \Phi Op \mid op? = SetFileExpiryOp] \vee \\ & [SetFileLength_1; \Phi Op \mid op? = SetFileLengthOp])) \end{aligned}$$

Note:  $\oplus$  is a non-standard *schema* operator.

$$S \oplus T = (\neg \text{pre } T \wedge S) \vee T$$

# Security

1. Clients may not access a file unless they knows its id, and file ids are hard to guess.
2. Files may be destroyed only by their owners, and user ids are hard to guess.

# Conclusion

‘User Manual’ (cf. ‘Implementor Manual’)

1. Basic types
2. Abbreviation, axiomatic and generic definitions
3. Abstract state
4. Initial state
5. Successful operations
6. Errors
7. Total operations