Tokeneer ID Station
INFORMED Design

S.P1229.50.2
Issue: 1.2
Status: Definitive
19th August 2008

Originator

Janet Barnes (Project Manager)

Approver

David Cooper (Technical Authority)

Copies to:

NSA

Praxis High Integrity Systems
Project File

SPRE Inc

Contents

1	**Introduction**	**3**
1.1	Background	3
1.2	Purpose	3
1.3	Scope	3
1.4	Notation	4
2	**INFORMED Design**	**6**
2.1	Identification of System Boundary	6
2.2	Identification of the SPARK boundary	7
2.3	Identification and Localisation of state	9
2.4	Initialisation of State	18
2.5	Other design issues	20
2.6	The Second SPARK Subsystem	28
3	**Package Summary**	**29**
3.1	Types Packages	29
3.2	Variable Packages	42
3.3	Utility Layers	63
Document Control and References		**68**
Changes history		68
Changes forecast		68
Document references		68

1 Introduction

1.1 Background

In order to demonstrate that developing highly secure systems to the level of rigour required by the higher assurance levels of the Common Criteria is possible, the NSA has asked Praxis High Integrity Systems to undertake a research project to develop a high integrity variant of an existing secure system (the Tokeneer System) in accordance with their own high-integrity development process. The component of the Tokeneer System that is to be redeveloped is the core functionality of the Token ID Station (TIS). This development work will then be used to show the security community that it is possible to develop secure systems rigorously in a cost-effective manner.

1.2 Purpose

The purpose of this design document is to present aspects of the design process that are not covered by the Formal Design. These are non-functional issues concerned with the mechanism by which the system should be implemented. This document does not present a functional description of the core TIS software; the reader is referred to the Formal Design [2] for functional information.

1.3 Scope

This document presents the INFORMED design for the high integrity TIS core development. This document also details the architectural design issues not covered by the Formal Design.

In particular it derives the software architecture in terms of the Ada packages that make up the system, the public operations offered by each of these packages, and their relationship to the Formal Design.

This document also addresses other technical design issues not covered in the formal model, including managing persistent data, file formats and file locations.

1.4 Abbreviations

CA	Certification Authority
CR	Carriage Return character
I&A	Identification and Authentication
LF	Line Feed character
SPARK	SPADE Ada Kernel (Analysable Ada subset from Praxis High Integrity Systems)
TIS	Token ID Station

1.5 Notation

Within the informed design there are several diagrams showing the structure of the system. Within these diagrams the following notation is used.

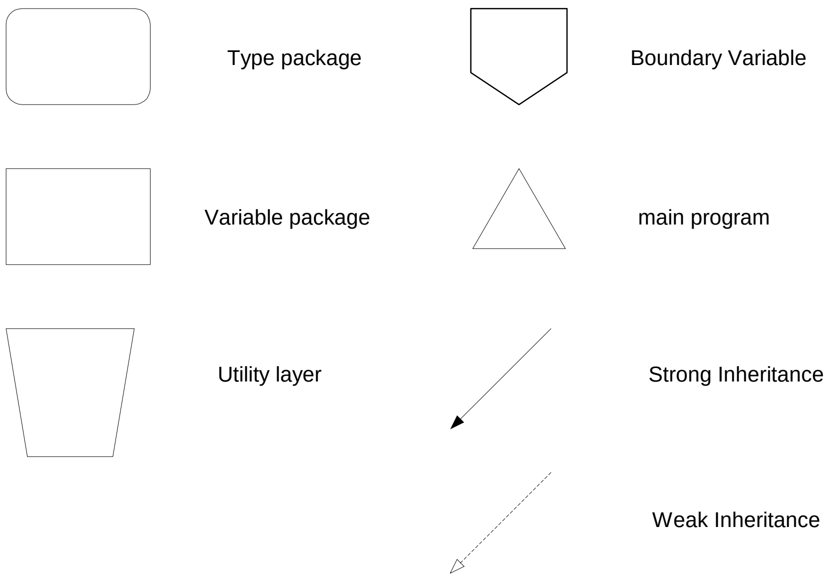

Figure 1 : Key to INFORMED Notation

A **Type package** is a package that introduces types and possibly operations on those types but does not introduce persistent state to the system.

A **variable package** is a package that introduces global variables (persistent state) to the system along with operations that manipulate that state.

A **utility layer** is a package that introduces operations to the system but does not introduce state or types.

A **boundary variable** marks the point of introduction into the SPARK environment of an external variable to the system; this represents either an import into the SPARK system or an export from the SPARK system. Boundary variables are used to represent real world entities.

The **main program** is the point of control of the SPARK system.

Strong Inheritance represents use, either directly or indirectly of the global variables of a package.

Weak Inheritance represents use of types and utilities provided by a package without using or affecting the state of the package.

2 INFORMED Design

The INFORMED design process, as described in [1], progresses through a number of stages, the emphasis being on the location of state in the system. The key stages can be summarised as follows:

1 Identification of System Boundary; inputs and outputs.

2 Identification of the SPARK Boundary.

3 Identification and localisation of system state.

4 Handling of initialisation of state.

5 Handling of secondary requirements.

In order to demonstrate the techniques employed in obtaining the final system architecture these stages are elaborated in this document.

2.1 Identification of System Boundary

This involves determining the entities in the physical real world which influence or are influenced by the behaviour of the core TIS.

These have already been elaborated in the Formal Design [2] and can be summarised as:

Entity	Input / Output	Comment
door	input	The core TIS monitors the status of the door via API calls.
latch	output	The core TIS controls the state of the latch via API calls.
alarm	output	The core TIS controls the state of an (audible) alarm via API calls.
time	input	The core TIS makes use of the current time in its decision making process. Time will be supplied by system calls.
display	output	The core TIS controls the text presented on the display at the portal, this is via API calls.
finger	input	The core TIS makes use of fingerprint data in determining user authentication, this is via API calls to the Biometric Library.
userToken	input /output	Data from the userToken is used by TIS in the authentication process. The userToken may be updated as part of the user authentication process. Access to the userToken is via the CardReader API.
adminToken	input	Data from the adminToken is used by TIS in the administrator logon process.
keyboard	input	The core TIS is controlled by commands entered by the administrator at the console keyboard. Data will be obtained via system calls.
screen	output	The core TIS displays information to the administrator at the console. Screen updates will be performed using system calls
floppy	input/ output	Large volume data is supplied to the core TIS and exported from core TIS via the floppy drive.

Table 1 System boundary, Inputs and Outputs

2.2 Identification of the SPARK boundary

The formal design of TIS shows the TIS system to be a purely sequential system. Although User Entry and Administrative functions could be performed in parallel and be viewed as being performed by separate SPARK systems this will not be done since there are constraints in the specification and design that ensure that user and administrative operations do not occur concurrently.

Considering the Interfaces, the Card Reader does not map well onto the entities that we want to reason about within the main program. The Card Reader does not appear explicitly in the Formal Design and it would be desirable if the annotations in the SPARK system relate closely to the entities in the formal design, this will not be possible if the Card Reader is included in the SPARK system. However the Card Reader itself will be fairly complex and we would benefit from performing static analysis on the processing within the Card Reader.

This suggests introducing a SPARK system to manage the CardReader interface with the tokens, which will process information from and to the tokens via the token reader. This layer will provide operations that answer questions in terms of system entities, such as “is User Token Present”. The implementation of which will require complex interactions with the Card Reader API.

Hence the core TIS will be two SPARK systems:

- one to provide an abstraction of the Card Reader interface which manages token data; and
- one to use token data and all the other inputs to control the TIS system outputs.

This is realised in Figure 2.

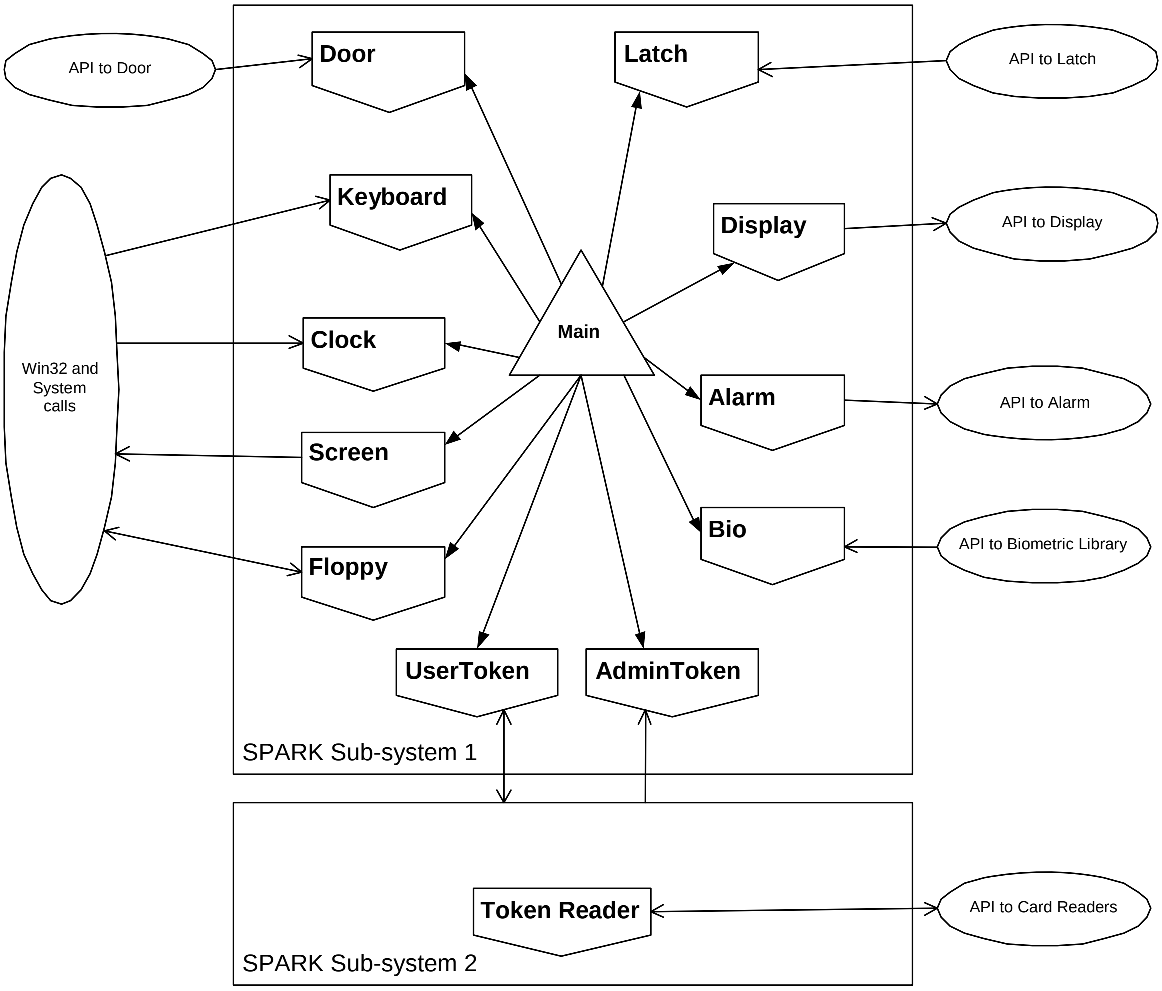

Figure 2 : SPARK System boundary

The core TIS is only a component of the TIS system. The TIS system itself comprises the Crypto Library, Certificate Processing Library, and several drivers that are not included in the core TIS. These will not form part of the SPARK system. The majority of these provide access to the environment. Those that do not provide interfaces to the environment are the Certificate Processing Library and the Crypto Library.

Both of these libraries will be encapsulated within SPARK packages that provide a thin layer access to the routines provided within the library. These SPARK packages will include annotations in their specifications which reflect the flow of information resulting from use of the libraries. The Certificate Processing Library has no persistent state so can be encapsulated within a utility package, CertProcessing. The Crypto Library contains state corresponding to the keys held within the library so this must be encapsulated within a variable package, KeyStore.

2.3 Identification and Localisation of state

Within this stage we need to identify the state that needs to be stored. Again the formal design has identified the system state, so the activity within the INFORMED process is reduced to localisation of state in order to present meaningful system level information flow and ensure that state is not made un-necessarily global. Subsystem 1 captures the core TIS as specified in the formal design.

From the formal design (*IDStationC*) we identify the following state:

State	Constituents	Comments
UserToken	userTokenPresence	Determined through polling the user token via the Card Reader. Associated with the **UserToken** boundary.
	currentUserToken	A local copy of the data on the current user token. Must be held for efficiency reasons. Associated with the **UserToken** boundary. There are four types of certificate that may be utilised from the user token.
AdminToken	adminTokenPresence	Determined though polling the admin token via the Card Reader. Associated with the **AdminToken** boundary.
	currentAdminToken	A local copy of the data on the current admin token. Must be held for efficiency reasons. Associated with the **AdminToken** boundary. Only the authorisation certificate is used from the admin token.
Finger	fingerPresence	Determined through polling the finger via the Bio Interface. This only need be checked when the system is waiting for a finger to authenticate, local state is unnecessary.

State	Constituents	Comments
DoorLatchAlarm	currentDoor	A local copy of the current state of the door, must be held to determine when the door state changes. Associated with the **Door** boundary
	currentLatch	A local copy of the current state of the latch, must be held to determine when the latch state needs to change. Associated with the **Latch** boundary.
	latchTimeout	Indicates when the latch must be locked, associated with the **Latch** boundary.
	doorAlarm	Indicates when the door is in an insecure state. Associated with the **Door** boundary, but depends on the current state of the **Latch**.
	alarmTimeout	Indicates when the door must be closed before the door is considered in an insecure state. Associated with the **Door** boundary
	currentTime	A local copy of the current time. This is associated with the **Clock** boundary.
Floppy	floppyPresence	Indicates whether the floppy is present or not. Associated with the **Floppy** boundary.
	currentFloppy	Indicates the latest value read from the floppy. Associated with the **Floppy** boundary.
	writtenFloppy	Indicates the latest value written to the floppy. Required to enable checks to ensure the audit log is written successfully. Associated with the **Floppy** boundary.
Keyboard	keyedDataPresence	Indicates whether there is data at the keyboard or not. Associated with the **Keyboard** boundary.
Config		Configuration Data used to determine entry permissions and authorisation certificate validity along with a number of durations and thresholds. This configuration data is security critical **essential state**. The configuration data is persistent through power down.
Stats		This is an internal record of operating statistics. It is **not essential** to the function of the system.
KeyStore		The state associated with the KeyStore is entirely maintained by the Crypto Library. Following enrolment this state is constant and is preserved

State	Constituents	Comments
		through power down.
CertificateStore		This contains the next Serial number used for issuing Authorisation certificates. This is persistent through power down. The value of this state impacts the output Authorisation certificate so it should be considered **essential**.
Admin		This is TIS **internal state** that records whether an administrator is logged on or not and the currently performed operation if any.
AuditLog		This represents the record of auditable events held by the system. Its presence is required to ensure the system is demonstrably secure and the state is **essential**. This state is preserved through power down.
Internal	status	**Internal** state associated with the **User Entry** operation. At the system level this need not be visible.
	fingerTimeout	**Internal** state associated with the **User Entry** operation. At the system level this need not be visible.
	tokenRemovalTimeout	**Internal** state associated with the **User Entry** operation. At the system level this need not be visible.
	enclaveStatus	**Internal** state associated with all **enclave** functions of the TIS, ie all functions excluding user entry. At the system level this need not be visible.
currentDisplay		Indicates the latest message on the display, required to enable logging of changes to the display. Associated with **Display** boundary.
currentScreen		Indicates the latest message on the display, screenMsg is required to enable logging of changes to the display. For efficiency reasons it may be useful to retain a record of the other screen components, however it is not necessary. Associated with **Screen** boundary.

Table 2 State Identification

Although the *Internal* state is internal to the way in which the system operates we choose to encapsulate this within two variable packages *Enclave* and *Entry.* Where additional local state is associated with a boundary variable the boundary variable is encapsulated by a variable package containing the additional state. The overall localisation of state is given by Figure 3.

We make the following observations about this design.

1. All boundary variables are encapsulated within a variable package. In many cases this is because there is additional state associated with this boundary. However for *Alarm* and *Bio* this is not the case. The reason for encapsulating these is that the boundary variable packages should have very simple implementations, simply providing calls through to the API. The encapsulating variable packages will also handle other activities such as auditing any system errors.

2. By making *Stats* and *Admin* abstract data types these will not appear in the main program information flow annotations.

3. All the interfaces may use the *AuditLog* for reporting system faults – these dependencies are not shown explicitly to avoid cluttering the diagrams.

4. Some uses of *ConfigData* have been omitted for clarity. These are noted under each figure.

5. There are a number of simple types packages these will be used throughout the system. Again usage is not shown on the diagrams.

6. The *File* utility later provides types and operations for accessing system files.

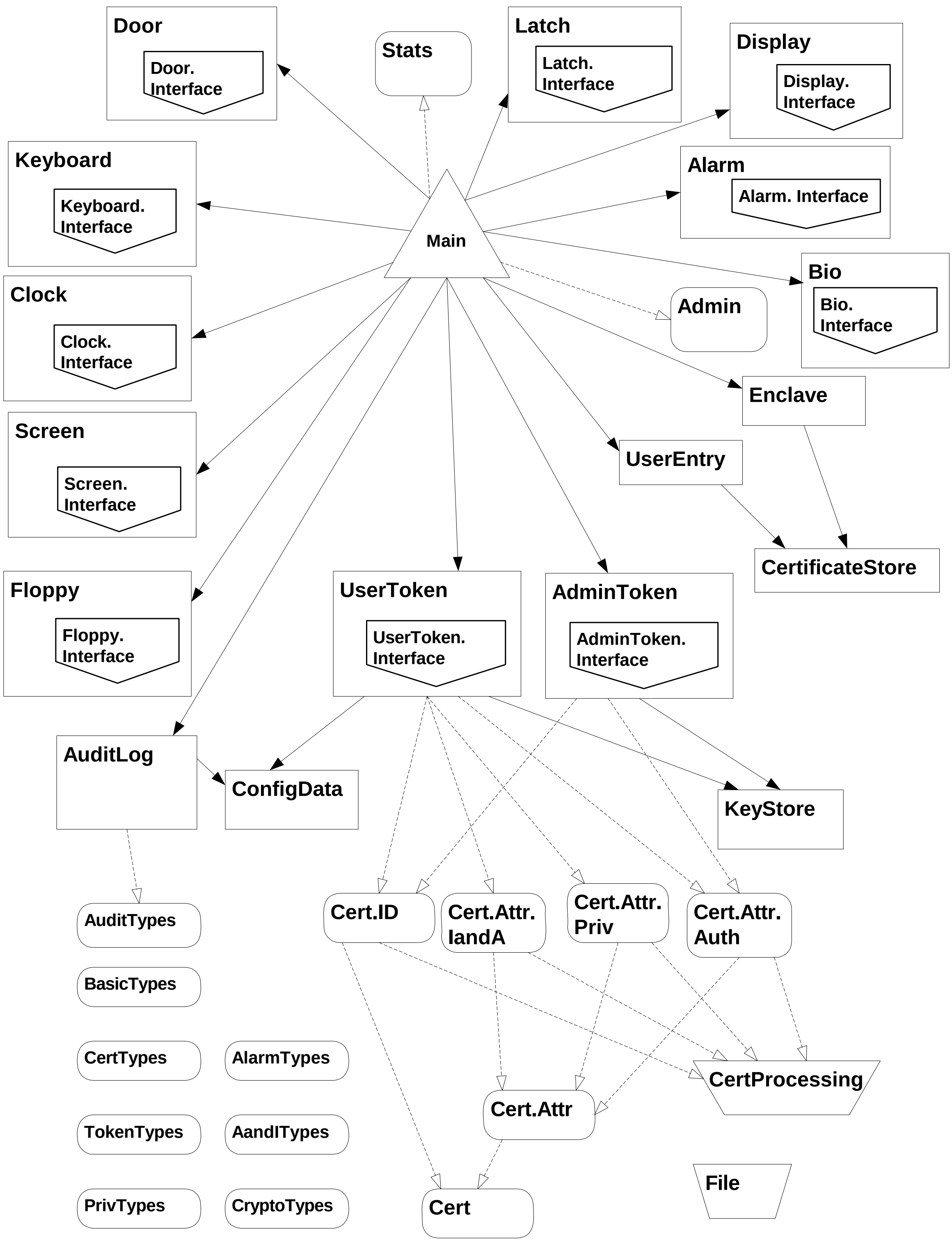

Figure 3 : Localisation of State

The TIS function can be decomposed into a number of key components, which are already apparent in the structure of the formal design. The system structure is shown separately for each of these key components.

1 Polling: this covers the process of obtaining regular updates of system inputs.

2 Updating: this covers the process of performing regular updates of system outputs.

3 Processing user entry: this covers the multi-phase user entry operation.

4 Performing enclave activities: this covers the activities performed at the console, all administrator operations and enrolment.

In the following diagrams the boundary variable is shown only where it is used. So for instance polling makes use of the latch state but not the boundary variable associated with the interface to the latch.

2.3.1 Polling

We introduce a utility layer to control the polling activity. Notice that during polling the internal Door state is updated based on the current state of the latch.

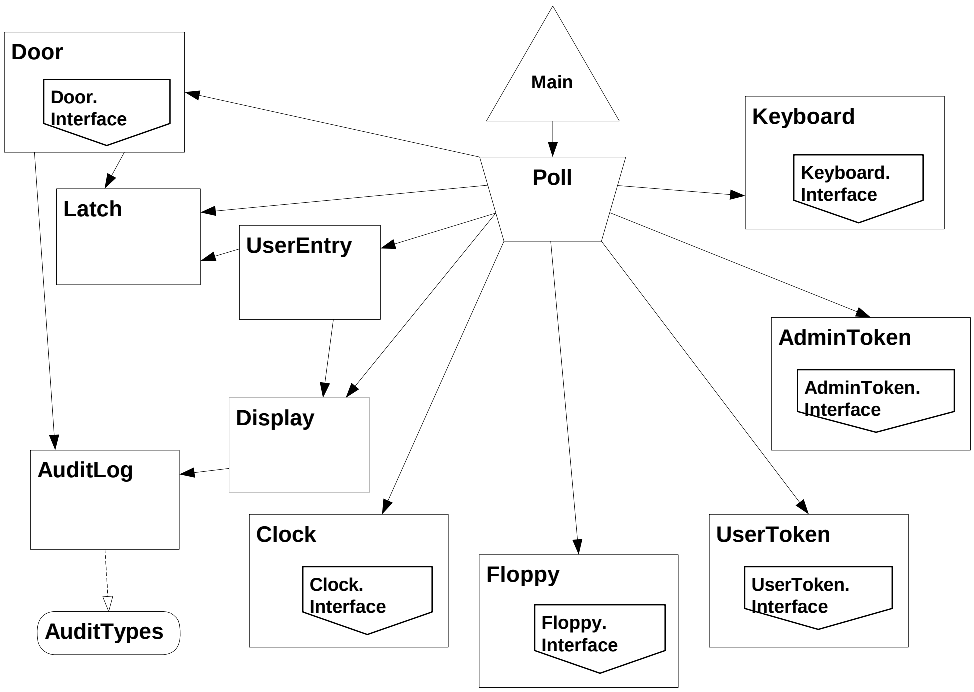

Figure 4 : Poll Context

The AuditLog may be used by the Door, Display, AdminToken and UserToken during polling. The AdminToken and UserToken may raise SystemErrors.
The AuditLog uses ConfigData.

2.3.2 Updating

We introduce a utility layer to manage the update of environmental data controlled by TIS. The screen updates will depend on who is currently logged on and the system statistics, provided as parameters and the configuration data.

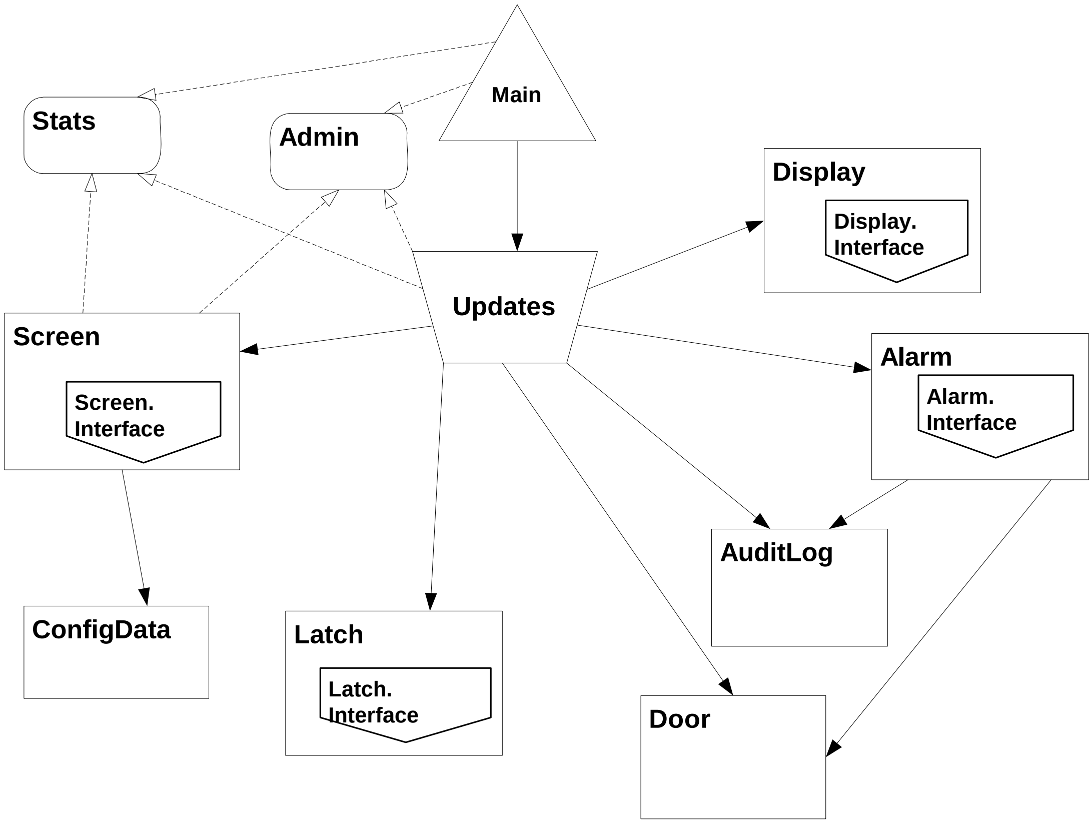

Figure 5: Update Context

The AuditLog may be used by Alarm, Display, Screen, Latch.
The AuditLog also uses ConfigData.

2.3.3 Processing User Entry

The user entry operation is controlled by a state-machine. This state-machine is the state of the Entry package. The Entry package will also provide the top level operations required to support User Entry. The processing of the certificates in the UserToken will make use of the CertificateProcessing utility layer. This CertificateProcessing utility layer is implemented using the **Certificate Processing Library API**.

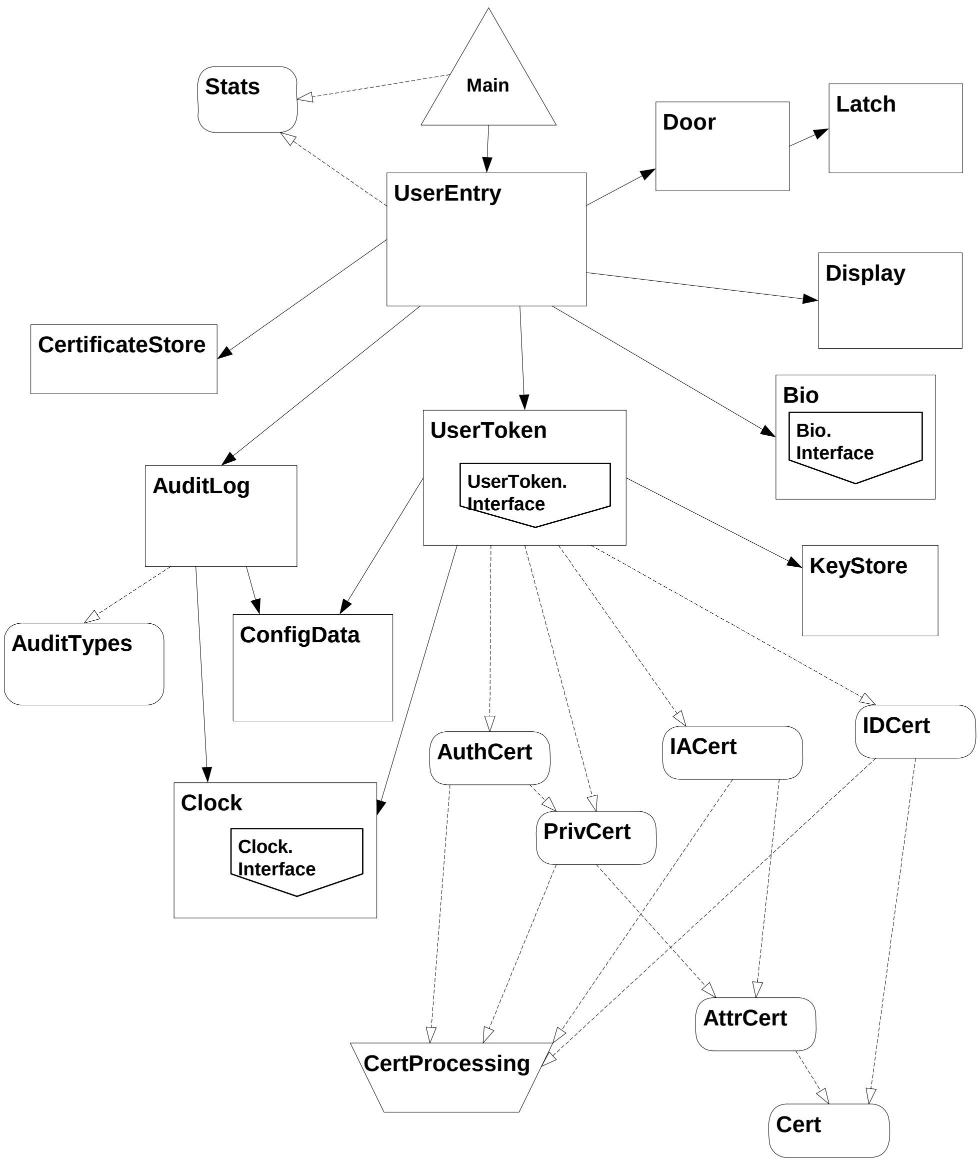

Figure 6 : User Entry Context

The AuditLog may be used by UserEntry, Latch, Display, KeyStore and Bio. Bio and KeyStore may log SystemFaults.
ConfigData is used by AuditLog, Door, UserEntry and UserToken.

2.3.4 Performing Enclave Activities

The activities performed in the enclave are controlled by a state-machine. This state-machine is the state of the Enclave package. The Enclave package must also make use of the Admin state to determine what can be performed. The Enclave package will also provide the top level operations required to support operations performed within the enclave. Utility layers are introduced to perform the configuration data checks and updates and the enrolment checks and updates.

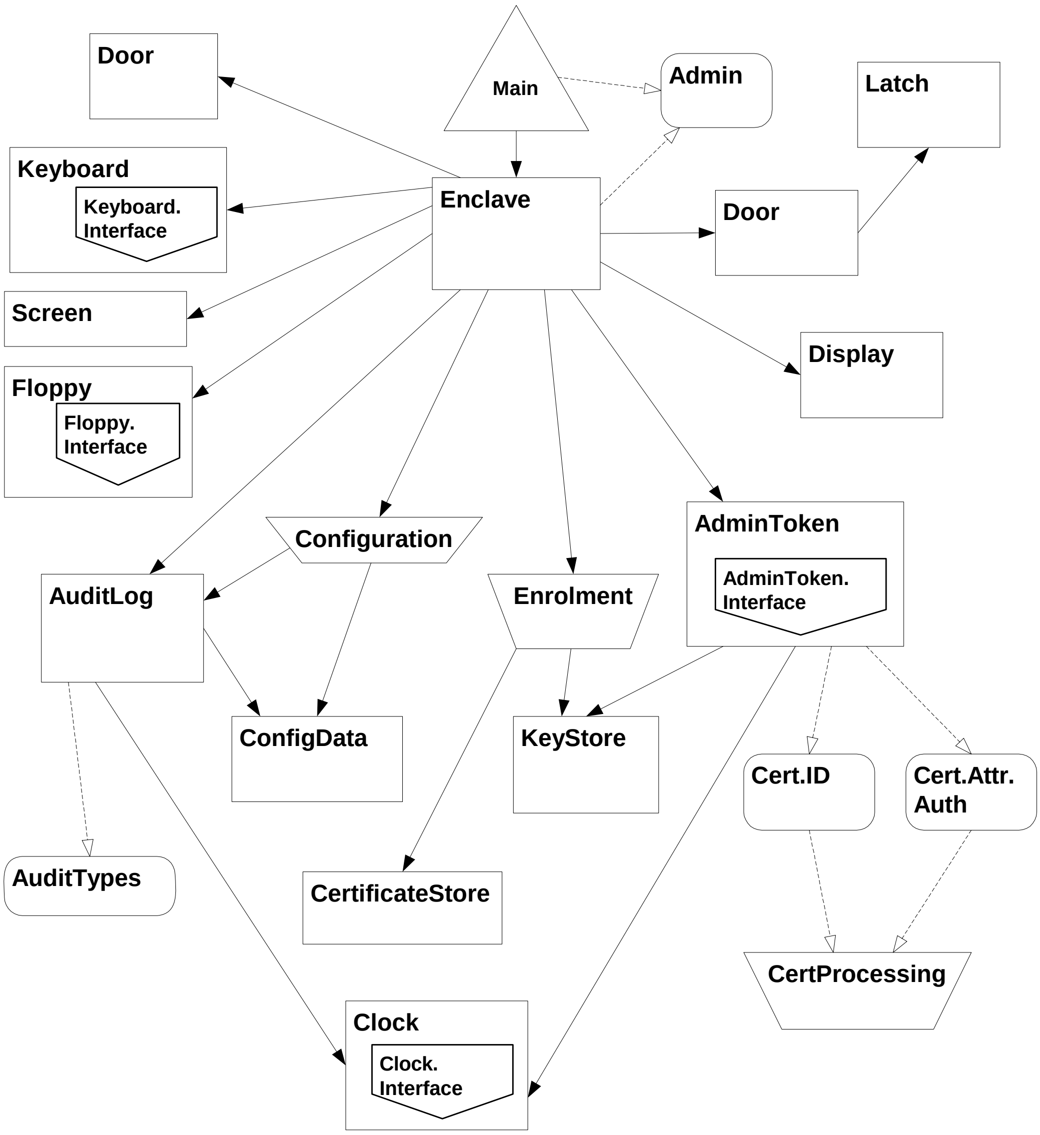

Figure 7 : Enclave Context

AuditLog may be used by Configuration, Display, Enclave, Enrolment, Latch and KeyStore. KeyStore may raise SystemFaults.
ConfigData is used by AuditLog and Configuration.

2.4 Initialisation of State

All boundary variables are assumed to be initialised by the environment.

The remaining state must either be initialised during elaboration, or operations must be made available to initialise the state prior to its use.

Within INFORMED it is considered bad practice to initialise state to “in range” values that are not necessarily consistent since this will prevent the use of static analysis to ensure that the system state is all initialised and consistent. With this in mind we take the following approach to initialisation.

Where *all* components of a package state variable can be initialised at elaboration to appropriate valid values, these will be initialised at elaboration. However the initialisation of much of the state depends on the value of entities persistent across power-down or is dependant on whether the *IDStation* is already enrolled or not. Where this is the case the state must be set by a procedure call to allow adequate static analysis.

State that is persistent across power-down will be stored in persistent memory and will need to be initialised by a procedure call since the data will need to be extracted from the memory. This applies to the state associated with *KeyStore, CertificateStore, ConfigData* and *AuditLog*. The persistent components of the state associated with these packages should be maintained separately from those aspects that are set at power-up. As *CertificateStore, ConfigData* and *AuditLog* maintain local state as well as using files to store persistent data, the data held on file will be recorded by *FileState.*

The initial value of some state depends on whether the *IDStation* is already enrolled or not. Where this is the case the state should be set by a procedure call. This applies to state associated with *Screen, Display, Enclave.*

A summary of the state and the initialisation mechanisms is presented in the Table 3.

Ada Package state variable	mode	initialisation mechanism	Z State
AdminToken.Input	in	assumed	*adminToken*
AdminToken.Status	-	assumed	-
AdminToken.State	-	elaboration	*adminTokenPresence* *currentAdminToken*
Alarm.Output	out	assumed	*alarm*
AuditLog.State	-	procedure call	*AuditLog* (excluding *logFiles*)
AuditLog.FileState	-	elaboration	*AuditLog.logFiles*
CertificateStore.State	-	procedure call	*CertificateStore*
CertificateStore.FileState	-	elaboration	-

Ada Package state variable	mode	initialisation mechanism	Z State
Clock.CurrentTime	-	procedure call	*currentTime*
Clock.Now	in	assumed	*now*
ConfigData.State	-	procedure call	*ConfigData*
ConfigData.FileState	-	elaboration	-
Display.Output	out	assumed	*display*
Display.State	-	procedure call	*currentDisplay*
Door.Input	in	assumed	*door*
Door.State	-	elaboration	*currentDoor* *alarmTimeout* *doorAlarm*
Enclave.State	-	procedure call	*enclaveStatus*
Floppy.Input	in	assumed	*floppy* (for reads)
Floppy.Output	out	assumed	*floppy* (for writes)
Floppy.State	-	elaboration	*currentFloppy* *floppyPresence*
Floppy.WrittenState	-	elaboration	*writtenFloppy*
Keyboard.Input	in	assumed	*keyboard*
Keyboard.State	-	elaboration	*keyDataPresence*
KeyStore.State	-	procedure call	*privateKey*
KeyStore.Store	-	assumed	*keys*
Latch.Output	out	assumed	*latch*
Latch.State	-	elaboration	*currentLatch* *latchTimeout*
Main.TheAdmin	-	elaboration	*Admin*
Main.TheStats	-	elaboration	*Stats*
Screen.Output	out	assumed	*screen*
Screen.State	-	procedure call	*currentScreen.screenMsg*
UserEntry.State	-	elaboration	*status* *fingerTimeout* *tokenRemovalTimeout*

Ada Package state variable	mode	initialisation mechanism	Z State
UserToken.Input	in	assumed	*userToken* (reads)
UserToken.Output	out	assumed	*userToken* (writes)
UserToken.Status	-	assumed	-
UserToken.State	-	elaboration	*currentUserToken* *userTokenPresence*

Table 3 : State Summary

2.5 Other design issues

2.5.1 Managing Persistent data

TIS retains some data that must be preserved over a power cycle. The majority of this data will be stored in TIS System files as described in the following table. The remaining data (KeyStore.Store) is data held within the Crypto library; the preservation of this data is managed by the Library.

Data	FileName	Location
ConfigData	config.dat	./System
CertificateStore	keystore	./System
AuditLog	file01.log ... file***nn***.log	./Log
ConfigData from Floppy	config.dat	./Temp
Enrolment Data from Floppy	enrol.dat	./Temp
Archive for Floppy	archive.log	./Temp

All file locations are relative to the location at which the TIS application is installed.

2.5.2 File Formats

The data received and supplied via the floppy has the following naming convensions:

- Configuration data is supplied in a file named config.dat.

- Enrolment data is supplied in a file named enrol.dat.
- Archives of the audit log are always written to a file named archive.log.

File formats are defined here for all files that are exported or imported from the system.

2.5.2.1 AuditLog

Audit log data is exported during an archive.

The audit file contains a number of audit entries with the following properties.

- each audit entry is terminated by a LF (Line feed) and CR (Carriage return). The entry itself does not contain any LF or CR.
- each field of an audit entry is comma separated. Individual fields do not contain commas.

The fields of an audit entry are presented it the order and format given in the table below:

field	format	comments
time	yyyy-mm-dd hh:mm:ss.s	Time is displayed to 1/10th second accuracy.
severity	n	Allowed values “1” – info; “2” – warning, “3” – critical
id	nn	Numeric representation of the audit element type. This is the offset of the element type in AUDIT_ELEMENT from the first element. *For example startUnenrolledTISElement has value 0.*
type	ASCII text	Text name for the given audit element type see Formal Design [2], values are given in AUDIT_ELEMENT. This field does not exceed 20 characters.
user	ASCII text	Text description identifying user in the form “Issuer: xxx SerialNo: yyy” or “NoUser”. This field does not exceed 50 characters.
description	ASCII text	free text description containing additional information This field does not exceed 150 characters.

Where “n” represents a single ASCII digit.

2.5.2.2 ConfigData

The format of a configuration data file is presented in the next table. Each field is presented on a new line and takes the form of a field identifier followed by at least one space and the value of the field. Each line is terminated by a CR and LF.

The file formats have been selected to provide a user-friendly interface for entering values. This includes restricting the granularity of short timeout durations to 1 second, the granularity of longer times to minutes and the granularity of file sizes to kBytes.

File format	**comments**
ALARMSILENTDURATION nnn	value is in seconds range 00..200
LATCHUNLOCKDURATION nnn	value is in seconds range 00..200
TOKENREMOVALDURATION nnn	value is in seconds range 00..200
FINGERWAITDURATION nnn	value is in seconds range 00..200
ENCLAVECLEARANCE tttttt	values of “ttttt” are unmarked, unclassified, restricted, confidential, secret, topsecret
WORKINGHOURSSTART hh:mm	00:00 represents midnight, max value is 23:59
WORKINGHOURSEND hh:mm	00:00 represents midnight, max value is 23:59
MAXAUTHDURATION hh:mm	max value is 10:00
ACCESSPOLICY tttttt	values of “tttttt” are allhours and workinghours
MINENTRYCLASS ttttt	values of “ttttt” are unmarked, unclassified, restricted, confidential, secret, topsecret *This field must have no higher classification than the value given for ENCLAVECLEARANCE*.
MINPRESERVEDLOGSIZE nnnn	value is in kBytes range 0 .. 4096
ALARMTHRESHOLDSIZE nnnn	value is in kBytes range 0 .. 4096 *This field must be no greater than the value given for MINPRESERVEDLOGSIZE.*
SYSTEMMAXFAR nnnnnnnnnn	INTEGER32 value nnnnnnnnnn/(2^{31} – 1) is the required probability of false acceptance.

2.5.2.3 Enrolment Data

Enrolment data is supplied as a number of ID certificates within a single file. Each ID Certificate takes the same format as a certificate transmitted across the TCP/IP interface to the device drivers (although the outermost braces enclosing the certificate “dictionary” are omitted). Each certificate will appear as a string on a single line within the enrolment data file.

The first ID certificate in the file must be that of the CA that issued the TIS ID Certificate, the second ID certificate in the file must be the ID certificate of the TIS being enrolled. The remaining certificates may be in any order as long as the ID certificate of a CA precedes any ID certificates issued by that CA.

The format of the certificate is as presented in Table 4, non-bold text appears in the file as presented here, bold text is defined in the subsequent table. Return characters should not appear in the string, they are used here to improve layout, similarly space characters are optional and will be ignored within the processing.

ID Certificate format within Enrolment file
'CertLength': '**CertificateLength**',
'CertDataT':
{'CertType': '0',
'SerialNumber': '**SerialNumber**',
'SigAlgID': '**Algorithm**',
'Issuer': {'Text': '**IssuerText**',
'ID': '**IssuerID**',
'TextLength': '**IssuerTextLength**'},
'Validity': {'NotAfter': {'Minute': '**Minute**',
'Month': '**Month**',
'Day': '**Day**',
'Hour': '**Hour**',
'Year': '**Year**'},
'NotBefore': {'Minute': '**Minute**',
'Month': '**Month**',
'Day': '**Day**',
'Hour': '**Hour**',
'Year': '**Year**'}},
'Subject': {'Text': '**UserText**',
'ID': '**UserID**',
'TextLength': '**UserTextLength**'},
'SubjectPublicKeyInfo': {'KeyLength': '**KeyLength**',
'AlgoRithmID': '**Algorithm**',
'KeyID': '**KeyID**'},
'CryptoControlData': {'DigestFinalReturn': '**ReturnValue**',
'DigestLength': '**DigestLength**',
'DigestUpdateReturn': '**ReturnValue**',
'Digest': {'VerifyReturn': '**ReturnValue**',
'SignReturn': '**ReturnValue**',
'DigestID': '**DigestID**'}},
'SignatureData': {'AlgoRithmID': '**Algorithm**',
'SigLength' : '**SigLength**',
'Signature': {'KeyID': '**KeyID**',
'DigestID': '**DigestID**'}}}

Table 4 ID certificate structure within enrolment file

Values taken by entities in bold are presented in Table 5. Constraints on valid ID certificates are detailed in Table 6.

Certificate Entity	**valid values**
Algorithm	string with the following values: RSA, MD2, MD5 SHA_1, RIPEMD128, RIPEMD160 MD2_RSA, MD5_RSA, SHA1_RSA RIPEMD128_RSA, RIPEMD160_RSA
CertificateLength	numeric string range 0 .. 4050
Day	numeric string range 01 .. 31
DigestID	numeric string range 0 .. 2^{32} -1
DigestLength	numeric string range 0 .. 32
Hour	numeric string range 00 .. 23
IssuerTextLength	numeric string range 0 .. 40
IssuerId	numeric string range 0 .. 2^{32} –1
IssuerText	ASCII string with maximum length of 40 characters (no CR or LF characters).
KeyID	numeric string range 0 .. 2^{32} -1
KeyLength	numeric string range 0 .. 128
Minute	numeric string range 00 .. 59
Month	numeric string range 01 .. 12
ReturnValue	string with one of the following values OK, HostMemory GeneralError, FunctionFailed ArgumentsBad, AttributeReadOnly AttributeTypeInvalid, AttributeValueInvalid DataInvalid, DataLenRange DeviceError, DeviceMemory FunctionCanceled, KeyHandleInvalid KeySizeRange, KeyTypeInconsistent KeyFunctionNotPermitted, MechanismInvalid MechanismParamInvalid, ObjectHandleInvalid OperationActive, OperationNotInitialized SignatureInvalid, SignatureLenRange TemplateIncomplete, TemplateInconsistent BufferTooSmall, CryptokiNotInitialized CryptokiAlreadyInitialized
SerialNumber	numeric string range 0 .. 2^{32} -1
SigLength	numeric string range 0 .. 128

Certificate Entity	**valid values**
UserId	numeric string range 0 .. 2^{32} –1
UserText	ASCII string with maximum length of 40 characters
UserTextLength	numeric string range 0 .. 40
Year	numeric string range 1901 .. 2099

Table 5 Valid values of fields

Constraint	**Description**
Date exists	Both the **NotBefore** and **NotAfter** dates must be real dates. Setting Month to 2 and Day to 30 would always be invalid
Certificate length correct	The certificate length is the number of characters in the string starting from the '**{**' following **CertDataT** field to the final '**}**'.
Algorithm consistent	The **SignatureData.AlgorithmID** and **SigAlgID** must match, this algorithm must also be combined algorithm including an encryption mechanism and a digest mechanism.
Signature not too long	The **SigLength** field must be no larger than the length of the key used to sign the certificate.
Returns OK	The crypto control data detailing the return values crypto operations **DigestUpdateReturn**, **DigestFinalReturn** and **VerifyReturn** must be set to OK. This indicates that these functions will succeed.
Digest matches signed digest	The **SignatureData.Signature.DigestID** must match **CryptoControlData.Digest.DigestID**. This represents the digest used to create the signature being the same as the digest calculated from the raw certificate data.
signing key matches issuer	The **SignatureData.Signature.KeyID** must match the key id associated with the **Issuer.ID**. This will be the **KeyID** supplied within the **SubjectPublicKeyInfo** of the Issuer's ID certificate.

Table 6 Constraints on valid certificates

2.5.3 System Faults

The formal design indicates that system faults may be raised. These should be handled as follows:

System faults are warnings except in the following critical cases:

- Failure to control the Latch
- Failure to monitor the Door

- Failure to write to the Audit Log

The system shall continue to function following a system fault categorised as a warning.

The system shall raise an alarm following a critical fault.

2.5.4 Implementation constraints on types

A number of the numeric entities in the Formal Design are permitted to take any value. The following constraints are applied to the values of these entities within the implementation.

Z Entity	State / Type	Implementation range	Units
CertificateId.serialNumber	State	$0 .. 2^{32} - 1$	
TOKENID	Type	$0 .. 2^{32} - 1$	
TIME	Type	1901-01-01 00:00:00.0 to 2099-12-31 23:59:59.9	
ConfigData.alarmSilentDuration	State	0 .. 2000	1/10th sec
ConfigData.latchUnlockDuration	State	0 .. 2000	1/10th sec
ConfigData.tokenRemovalDuration	State	0 .. 2000	1/10th sec
ConfigData.fingerWaitDuration	State	0 .. 2000	1/10th sec
ConfigData.maxAuthDuration	State	0 .. 864000	1/10th sec
ConfigData.minPreservedLogSize	State	$0 .. 2^{22}$	Bytes
ConfigData.alarmThresholdSize	State	$0 .. 2^{22}$	Bytes
CertificateStore.nextSerialNumber	State	$0 .. 2^{32} - 1$	
AuditLog.numberLogEntries	State	$0 .. 2^{14}$	
Stats.successEntry	State	$0 .. 2^{31} -1$	
Stats.failEntry	State	$0 .. 2^{31} -1$	
Stats.successBio	State	$0 .. 2^{31} -1$	
Stats.failBio	State	$0 .. 2^{31} -1$	

The size of an Audit Log entry is 2^8 bytes, this determines the ratio between *AuditLog.numberLogEntries* and *ConfigData.minPreservedLogSize.* The maximum value for *ConfigData.minPreservedLogSize* was selected bearing in mind that

- each log file must fit on the Floppy Drive if it is archivable;
- to archive data we need to have at least a complete log file.

The figure for *ConfigData.minPreservedLogSize* corresponds to 17 log files each with a capacity of 1024 elements. This gives an audit capacity of over 4Mbytes (> 16,000 log elements) and should result in a system where each of the functions is testable.

Certificate serial numbers and token Ids will be represented by Unsigned 32bit words. Other values will be represented by Signed 32bit words.

It should be noted that in the implementation we distinguish between durations and time stamps. All of these appear as type *TIME* in the formal design.

This has the following impact on the implementation:

- Statistics will only report the number of failures/successes up to the implementation maximum, after this point the statistic will not be incremented.

- TIS will not issue authorisation certificates once the maximum certificate serial number has been reached. This will result in TIS raising a system fault.

2.6 The Second SPARK Subsystem

The second spark subsystem is very simple. It consists of a single variable package, which encapsulates a boundary variable representing the interface to the Card Reader API. This subsystem may make use of the audit log facilities of the primary SPARK subsystem. However there is a contextual change made at the interface of the Primary SPARK subsystem at the point at which it uses the TokenReader. This context change ensures that the primary SPARK subsystem distinguishes the interfaces to the User and Admin Token Readers.

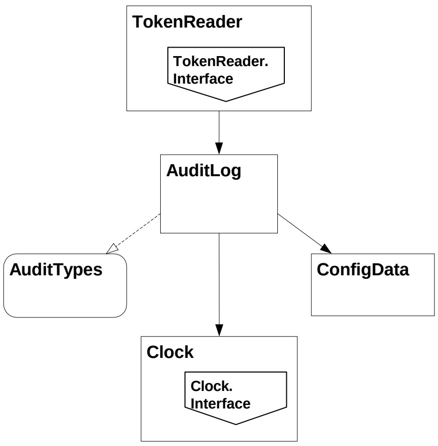

Figure 8 : The Second SPARK Subsystem

The state associated with the TokenReader package is identified as follows. As this SPARK subsystem provides a conversion between the available APIs and the system model there is a less strong correspondence between the Z state and the state encapsulated by this package. Where there is a correspondence it is tabulated below, although in all cases the Z state components are also modelled within the Primary SPARK subsystem.

Ada Package state	mode	initialisation mechanism	represents
TokenReader.Input	in	assumed	data read from tokens (*adminToken* and *userToken*)
TokenReader.Output	out	assumed	data written to tokens (*userToken*)
TokenReader.Status	-	assumed	status information obtained from tokens
TokenReader.State	-	procedure call	persistent state maintained by the SPARK subsystem relating to the token reader (includes *userTokenPresence* and *adminTokenPresence*)

3 Package Summary

The following section summarises the facilities provided by each of the library level packages in the system and references the formal design where the types, state or operations are derived from the formal design.

This does not provide further details of the boundary variables as these have all been made private to a variable package. The state associated with these boundary variables is virtual and will be form part of the appropriate variable package state.

3.1 Types Packages

In the Z there are a number of entities that have optional types. This means that the value may not be defined. Hence, where the underlying type is a free type (implemented by an enumeration type) the implementation will declare a parent type of the enumeration type, which is extended by a value representing the undefined value. Where the underlying type is not implemented by an enumerated type, a Boolean flag will be used to indicate the validity of the optional type.

3.1.1 Admin

Abstract Data Type containing administrator control information.

3.1.1.1 Types

Ada Type	**type classification**	**Z type**
Admin.OpT	enumeration	ADMINOP
Admin.OpAndNullT	enumeration	optional ADMINOP
Admin.T	private	Admin

3.1.1.2 Operations

Ada Operation	**operation type**	**Z Operation**
Admin.Init	procedure	*InitAdmin*
Admin.Logon	procedure	*AdminLogon*
Admin.Logout	procedure	*AdminLogout*
Admin.StartOp	procedure	*AdminStartOp*
Admin.FinishOp	procedure	*AdminFinishOp*
Admin.IsPresent	Boolean function	*AdminIsPresent*
Admin.OpIsAvailable	function returning OpAndNullT	*AdminOpIsAvailable*
Admin.IsDoingOp	Boolean function	*AdminIsDoingOp*
Admin.TheCurrentOp	retrieval function	The *currentAdminOp* value
Admin.SecurityOfficerIsPresent	Boolean function	*the rolePresent = securityOfficer*

3.1.2 AlarmTypes

Types that appear within the context of alarms.

3.1.2.1 Types

Ada Type	type classification	Z type
AlarmTypes.StatusT	enumeration	*ALARM*

3.1.3 AuditTypes

Types that appear within the context of the audit log.

3.1.3.1 Types

Ada Type	type classification	Z type
AuditTypes.ElementT	enumeration	*AUDIT_ELEMENT*
AuditTypes.SeverityT	enumeration	*AUDIT_SEVERITY*
AuditTypes.DescriptionT	string	*TEXT*
AuditTypes.UserTextT	string	*USERTEXT*
AuditTypes.FileSizeT	integer	*N*
AuditTypes.AuditEntryCountT	integer	*N*

3.1.4 BasicTypes

These are types that appear across the TIS implementation

3.1.4.1 Types

Ada Type	type classification	Z type
BasicTypes.Integer32T	numeric	*INTEGER32*
BasicTypes.ByteT	numeric	*BYTE*
BasicTypes.PresenceT	enumeration	*PRESENCE*

3.1.5 Cert

Abstract data type representing the common aspects of certificates.

3.1.5.1 Types

Ada Type	type classification	Z type
Cert.ContentsT	private	*CertificateContents*

3.1.5.2 Operations

Ada Operation	operation type	Z Operation
Cert.TheIssuer	retrieval function	value of *id.issuer*
Cert.TheId	retrieval function	value of *id*
Cert.ExtractUser	function	*thisUser*
Cert.TheMechanism	retrieval function	value of *mechanism*
Cert.IsCurrent	Boolean function	*CertIsCurrent*
Cert.GetData	retrieval function	value of *data*
Cert.GetSignature	retrieval function	value of *signature*
Cert.IssuerKnown	procedure	*CertIssuerKnown*
Cert.IsOK	procedure	*CertOK*

3.1.6 Cert.Attr

Abstract data type representing the common aspects of all attribute certificates.

3.1.6.1 Types

Ada Type	type classification	Z type
Cert.Attr.ContentsT	private	*AttCertContents*

3.1.6.2 Operations

Operations inherited from Cert with the addition of the following.

Ada Operation	operation type	Z Operation
Cert.Attr.TheBaseCert	retrieval function	value of *baseCertId*
Cert.Attr.ExtractUser	function	*thisUser*

Note that ExtractUser overrides the operation inherited from Cert. The user of an attribute certificate is deduced from the *baseCertId* and not the *CertId*.

3.1.7 Cert.Attr.Auth

Abstract data type representing authorisation certificates.

3.1.7.1 Types

Ada Type	**type classification**	**Z type**
Cert.Attr.Auth.ContentsT	private	*AuthCertContents*

3.1.7.2 Operations

Operations inherited from Cert.Attr with the addition of the following:

Ada Operation	**operation type**	**Z Operation**
Cert.Attr.Auth.Contstruct	procedure	*constructAuthCert*
Cert.Attr.Auth.IsOK	procedure	*AuthCertOK*
Cert.Attr.Auth.TheRole	retrieval function	value of *role*
Cert.Attr.Auth.TheClearance	retrieval.function	value of *clearance*
Cert.Attr.Auth.Extract	procedure	*extractAuthCert*
Cert.Attr.Auth.SetContents	procedure	-
Cert.Attr.Auth.Clear	procedure	-

3.1.8 Cert.Attr.IandA

Abstract data type representing the identification and authentication certificates.

3.1.8.1 Types

Ada Type	type classification	Z type
Cert.Attr.IandA.ContentsT	private	*IandACertContents*

3.1.8.2 Operations

Operations inherited from Cert.AttrCert with the addition of the following

Ada Operation	operation type	Z Operation
Cert.Attr.IandA.TheTemplate	retrieval function	value of *template*
Cert.Attr.IandA.Extract	procedure	*extractIandACert*
Cert.AttrIandA.Clear	procedure	-

3.1.9 Cert.Attr.Priv

Abstract data type representing privilege certificates.

3.1.9.1 Types

Ada Type	type classification	Z type
Cert.Attr.Priv.ContentsT	private	*PrivCertContents*

3.1.9.2 Operations

Operations inherited from Cert.Attr with the addition of the following

Ada Operation	operation type	Z Operation
Cert.Attr.Priv.TheRole	retrieval function	value of *role*
Cert.Attr.Priv.TheClearance	retrieval.function	value of *clearance*
Cert.Attr.Priv.Extract	procedure	*extractPrivCert*
Cert.Attr.Priv.Clear	procedure	-

3.1.10 Cert.ID

Abstract data type representing ID Certificates.

3.1.10.1 Types

Ada Type	type classification	Z type
Cert.ID.ContentsT	private	*IDCertContents*

3.1.10.2 Operations

Operations inherited from Cert with the addition of the following

Ada Operation	operation type	Z Operation
Cert.ID.ThePublicKey	retrieval function	value of *subjectPubK*
Cert.ID.TheSubject	retrieval function	value of *subject*
Cert.ID.Extract	procedure	*extractIDCert*
Cert.ID.Clear	procedure	-

3.1.11 CertTypes

Types that appear within the context of certificates.

3.1.11.1 Types

Ada Type	type classification	Z type
CertTypes.RawCertificateT	String	*RawCertificate*
CertTypes.SignatureT	String	*SIGDATA*
CertTypes.RawDataT	String	*RAWDATA*
CertTypes. IDT	record	*CertificateId*
CertTypes.SerialNumberT	integer	*N*

3.1.12 CryptoTypes

Types that appear within the context of encryption.

3.1.12.1 Types

Ada Type	type classification	Z type
CryptoTypes.KeyTypeT	enumeration	*KEYTYPE*
CryptoTypes.KeyPartT	record	*KeyPart*
CryptoTypes.IssuerT	record	*Issuer*
CryptoTypes.AlgorithmT	enumeration	*ALGORITHM*

3.1.13 IandATypes

Types that appear within the context of the Biometric checks.

3.1.13.1 Types

Ada Type	type classification	Z type
IandATypes.FarT	numeric	*INTEGER32*
IandTypes.MatchResultT	enumeration	*MATCHRESULT*
IandATypes.TemplateT	record	*FingerprintTemplate*

3.1.14 PrivTypes

Types that appear within the context of privileges.

3.1.14.1 Types

Ada Type	type classification	Z type
PrivTypes.ClassT	enumeration	*CLASS*
PrivTypes.ClearanceT	record	*Clearance*
PrivTypes.PrivilegeT	enumeration	*PRIVILEGE*
PrivTypes.AdminPrivilegeT	array	*ADMINPRIVILEGE*

3.1.15 Stats

Abstract Data Type for maintaining system statistics.

3.1.15.1 Types

Ada Type	type classification	Z type
Stats.T	private	*Stats*

3.1.15.2 Operations

Ada Operation	operation type	Z Operation
Stats.Init	procedure	*InitStats*
Stats.AddSuccessfulEntry	procedure	*AddSuccessfulEntryToStats*
Stats.AddFailedEntry	procedure	*AddFailedEntryToStats*
Stats.AddSuccessfulBio	procedure	*AddSuccessfulBioToStats*
Stats.AddFailedBio	procedure	*AddFailedBioToStats*
Stats.DisplayStats	procedure	*displayStats*

3.1.16 TokenTypes

Types that appear within the context of tokens.

3.1.16.1 Types

Ada Type	type classification	Z type
TokenTypes.TokenId	INTEGER32	*N*
TokenTypes.TryT	enumeration	*TOKENTRY*

3.2 Variable Packages

3.2.1 Alarm

Variable package providing the interface to the Alarm API.

3.2.1.1 State

The state of this package is solely the boundary variable representing the real world alarm.

3.2.1.2 Operations

Ada Operation	operation type	Z Operation
Alarm.UpdateDevice	procedure	*UpdateAlarm*

3.2.2 AuditLog

The Variable package holding and controlling the audit log.

3.2.2.1 State

Ada Package state	mode	Z State
AuditLog.State	-	*AuditLog* (excluding logFiles)
AuditLog.FileState	-	*AuditLog.logFiles*

The audit log state comprises files holding the audit entries, control state and audit alarm, this is not elaborated here.

3.2.2.2 Operations

Ada Operation	operation type	Z Operation
AuditLog.Init	procedure	*TISStartup* (partial) *InitAuditLog*
AuditLog.AddElementToLog	procedure	*AddElementToLog*
AuditLog.ArchiveLog	procedure	*ArchiveLog*
AuditLog.ClearLogEntries	procedure	*ClearLogEntries*
AuditLog.CancelArchive	procedure	*CancelArchive*
AuditLog.TheAuditAlarm	retrieval function	*auditAlarm* value
AuditLog.SystemFaultOccurred	function	

The operation Init updates the internal state from data preserved on file. In addition to the initialisation detailed in the Formal Design the Init procedure will ensure that the Log directory used by the AuditLog package is present.

The operation AddElementToLog will also construct the element from the parameters.

The operation SystemFaultOccurred indicates whether or not a critical system fault has occurred whilst writing to the log.

3.2.3 AdminToken

Variable package holding and managing the admin token state.

3.2.3.1 State

Ada Package state	type	Z State
AdminToken.State	private	*adminTokenPresence* *currentAdminToken*

3.2.3.2 Operations

Ada Operation	operation type	Z Operation
AdminToken.Poll	procedure	*PollAdminToken*
AdminToken.ReadAndCheck	procedure	*ReadAdminToken* *AdminTokenOK* *AdminTokenNotOK*
AdminToken.IsPresent	Boolean function	*adminTokenPresence = present*
AdminToken.IsCurrent	Boolean function	*AdminTokenCurrent*
AdminToken.ExtractUser	function	*extractUser*
AdminToken.GetRole	retrieval function	value of *AuthCertContents.role*
AdminToken.Clear	procedure	*ClearAdminToken*

The operation ExtractUser makes use of Cert.ID.ExtractUser.

3.2.4 Bio

The variable package holding state representing the biometric device and providing interfaces to the biometric device.

3.2.4.1 State

The state associated with this package is solely state representing the input from and status of the biometric device.

3.2.4.2 Operations

Ada Operation	operation type	Z Operation
Bio.Poll	procedure	*PolFinger*
Bio.Verify	procedure	*verifyBio*
Bio.Flush	procedure	*FlushFingerData*

3.2.5 CertificateStore

The variable package holding and maintaining the certificate store state.

3.2.5.1 State

Ada Package state	type	Z State
CertificateStore.State	private	*CertificateStore*
CertificateStore.FileState	private	-

3.2.5.2 Operations

Ada Operation	operation type	Z Operation
CertificateStore.Init	procedure	*TISStartup* (partial) *InitCertificateStore*
CertificateStore.UpdateStore	procedure	*UpdateCertificateStore*
CertificateStore.SerialNumber	function	*nextSerialNumber* value
CertificateStore. SerialNumberHasOverflowed	function	-

3.2.6 Clock

Variable package maintaining the local time and providing access to the system clock.

3.2.6.1 Types

Ada Type	type classification	Z type
Clock.TimeT	private	*TIME*
Clock.DurationT	integer	*TIME*
Clock.TimeTextT	string	-
Clock.DurationTextT	string	-

3.2.6.2 State

Ada Package state	type	Z State
Clock.CurrentTime	Clock.Time	*currentTime*

3.2.6.3 Operations

Ada Operation	operation type	Z Operation
Clock.Poll	procedure	*PollTime*
Clock.TheCurrentTime	retrieval function	*currentTime* value
Clock.GetNow	procedure, direct read of real world time.	*now* value
Clock.GreaterThan	Boolean function	$>$
Clock.LessThan	Boolean function	$<$
Clock.GreaterThanOrEqual	Boolean function	$\geq$
Clock.LessThanOrEqual	Boolean function	$\leq$
Clock.ConstructTime	constructor function	
Clock.SplitTime	procedure	
Clock.PrintTime	function	
Clock.PrintDuration	function	
Clock.AddDuration	function	$+$
Clock.StartOfDay	function	

3.2.7 ConfigData

Package maintaining the configuration data.

3.2.7.1 Types

Ada Type	type classification	Z type
ConfigData.DurationT	subtype of Clock.Duration	*N*
ConfigData.AccessPolicyT	enumeration	*ACCESS_POLICY*

3.2.7.2 State

Ada Package state	type	Z State
ConfigData.State	private	*ConfigData*
ConfigData.FileState	private	-

3.2.7.3 Operations

Ada Operation	**operation type**	**Z Operation**
ConfigData.Init	procedure	*InitConfig* *TISStartup* (partial)
ConfigData.UpdateData	procedure	-
ConfigData.TheDisplayFields	procedure	*displayConfigData*
ConfigData.ValidateFile	procedure	constraints on *ConfigData*
ConfigData.WriteFile	procedure	
ConfigData.AuthPeriodIsEmpty	function	*authPeriodIsEmpty*
ConfigData.GetAuthPeriod	function	*getAuthPeriod*
ConfigData.IsInEntryPeriod	Boolean function	in *entryPeriod*
ConfigData.TheAlarmThresholdEntries	retrieval function	value of *alarmThresholdEntries*
ConfigData.TheAlarmSilentDuration	retrieval function	value of *alarmSilentDuration*
ConfigData.TheLatchUnlockDuration	retrieval function	value of *latchUnlockDuration*
ConfigData.TheFingerWaitDuration	retrieval function	value of *fingerWaitDuration*
ConfigData.TheTokenRemovalDuration	retrieval function	value of *tokenRemovalDuration*
ConfigData.TheEnclaveClearance	retrieval function	value of *enclaveClearance*
ConfigData.TheSystemMaxFar	retrieval function	value of *systemMaxFar*

In addition to the initialisation detailed in the Formal Design the Config.Init procedure will ensure that the System directory used by the ConfigData package is present.

3.2.8 Display

This variable package maintains a local record of the display state and provides an interface to the Display API.

3.2.8.1 Types

Ada Type	type classification	Z type
Display.MsgT	enumeration	*DISPLAYMESSAGE*

3.2.8.2 State

Ada Package state	type	Z State
Display.State	Display.MsgT	currentDisplay

3.2.8.3 Operations

Ada Operation	operation type	Z Operation
Display.UpdateDevice	procedure	*UpdateDisplay*
Display.SetValue	procedure	set *currentDisplay*
Display.ChangeDoorUnlockMsg	procedure	*DisplayPollUpdate* (partial)
Display.Init	procedure	*TISStartup* (partial)

3.2.9 Door

Variable package maintaining the local state of the door and interfacing to the Door API.

3.2.9.1 Types

Ada Type	type classification	Z type
Door.T	enumeration	DOOR

3.2.9.2 State

Ada Package state	type	Z State
Door.State	private	*currentDoor* *alarmTimeout* *doorAlarm*

3.2.9.3 Operations

Ada Operation	operation type	Z Operation
Door.Poll	procedure	*PollTimeAndDoor* (partial)
Door.UnlockDoor	procedure	*UnlockDoor*
Door.LockDoor	procedure	*LockDoor*
Door.Init	procedure	*InitDoorLatchAlarm* (partial)
Door.TheDoorAlarm	retrieval function	*doorAlarm* value
Door.TheCurrentDoor	retrieval function	*currentDoor* value
Door.Failure	procedure	*DoorLatchFailure* (partial)

3.2.10 Enclave

Variable package maintaining the enclave status and controlling enclave activities.

3.2.10.1 Types

Ada Type	type classification	Z type
Enclave.Status	enumeration	*ENCLAVESTATUS*

3.2.10.2 State

Ada Package state	type	Z State
Enclave	private	*enclaveStatus*

3.2.10.3 Operations

Ada Operation	operation type	Z Operation
Enclave.EnrolmentIsInProgress	Boolean function	*EnrolmentIsInProgress*
Enclave.AdminMustLogout	Boolean function	*AdminMustLogout*
Enclave.CurrentAdminActivityPossible	Boolean function	*CurrentAdminActivityPossible*
Enclave.EnrolOp	procedure	*TISEnrolOp*
Enclave.AdminLogout	procedure	*TISAdminLogout*
Enclave.ProgressAdminActivity	procedure	*TISProgressAdminLogon* *TISAdminOp*
Enclave.StartAdminActivity	procedure	*TISStartAdminLogon* *TISStartAdminOp*
Enclave.ResetScreenMessage	procedure	*ResetScreenMessage* (partial)
Enclave.Init	procedure	*TISStartup* (partial)

3.2.11 Floppy

This variable package maintains local state relating to the contents of the floppy and provides access to the floppy drive via system calls.

3.2.11.1 State

Ada Package state	type	Z State
Floppy.State	File.T	currentFloppy floppyPresence
Floppy.WrittenState	File.T	writtenFloppy

3.2.11.2 Operations

Ada Operation	operation type	Z Operation
Floppy.IsPresent	Boolean function	*PollFloppy* *floppyPresence = present*
Floppy.Write	procedure	*UpdateFloppy*
Floppy.Read	procedure	*ReadFloppy*
Floppy.CheckWrite	procedure	*currentFloppy = writtenFloppy*
Floppy.CurrentFloppy	retrieval function	value of *currentFloppy*
Floppy.Init	procedure	*TISStartup* (partial)

In addition to the initialisation detailed in the Formal Design the Floppy.Init procedure will determine the drive letter for the floppy drive and ensure that the Temp directory used by the Floppy package is present.

3.2.12 Keyboard

This variable package maintains local state associated with the keyboard and accesses the keyboard via system calls.

3.2.12.1 Types

Ada Type	type classification	Z type
Keyboard.DataT	string	KEYBOARD

3.2.12.2 State

Ada Package state	type	Z State
Keyboard.State	private	*keyedDataPresence*

3.2.12.3 Operations

Ada Operation	operation type	Z Operation
Keyboard.Poll	procedure	*PollKeyboard*
Keyboard.DataIsPresent	function	*keyedDataPresence* = *present*
Keyboard.Read	procedure	read *keyboard*

3.2.13 KeyStore

This package provides the TIS core interface to the Crypto Library.

3.2.13.1 State

Ada Package state	type	Z State
KeyStore.State	private	*privateKey*

There is no explicit state, this state models state associated with Cypto Library.

3.2.13.2 Operations

Ada Operation	operation type	Z Operation
KeyStore.Init	procedure	*InitKeyStore* *TISStartup* (partial)
KeyStore.KeyMatchingIssuerPresent	procedure	*keyMatchingIssuer ≠ nil*
KeyStore.PrivateKeyPresent	function	*privateKey ≠ nil*
KeyStore.IssuerIsThisTIS	function	*CertIssuerIsThisTIS*
KeyStore.ThisTIS	function	*(the privateKey).keyOwner*
KeyStore.isVerifiedBy	procedure	*isVerifiedBy*
KeyStore.Sign	procedure	*sign*
KeyStore.AddKey	procedure	*UpdateKeyStore*
KeyStore.Delete	procedure	-

The initialise operation will initialise the underlying crypto library.

3.2.14 Latch

Variable Package maintaining local representation of the state of the Latch and interfacing to the Latch API.

3.2.14.1 Types

Ada Type	type classification	Z type
Latch.T	enumeration	LATCH

3.2.14.2 State

Ada Package state	type	Z State
Latch.State	-	currentLatch latchTimeout

3.2.14.3 Operations

Ada Operation	operation type	Z Operation
Latch.UpdateDevice	procedure	*UpdateLatch*
Latch.UpdateInternalLatch	procedure	*UpdateInternalLatch*
Latch.Init	procedure	*InitDoorLatchAlarm* (partial)
Latch.SetTimeout	procedure	*SetUnlockDoorTimeouts* (partial) *SetLockDoorTimeouts* (partial)
Latch.IsLocked	function	*currentLatch = locked*
Latch.Failure	procedure	*DoorLatchFailure* (partial)

3.2.15 Screen

This variable package maintains the local screen state and interfaces to the screen via system calls.

3.2.15.1 Types

Ada Type	type classification	Z type
Screen.MsgText	enumeration	SCREENTEXT

3.2.15.2 State

Ada Package state	type	Z State
Screen.State	Screen.MsgText	currentScreen.screenMsg

3.2.15.3 Operations

Ada Operation	operation type	Z Operation
Screen.UpdateScreen	procedure	*UpdateScreen*
Screen.SetMessage	procedure	set *currentScreen.screenMsg*
Screen.Init	procedure	*InitIDStation* (partial)

3.2.16 TokenReader

Variable package managing the interface to the token readers.

3.2.16.1 State

Ada Package state	type	represents
TokenReader.State	private	persistent state maintained by the SPARK subsystem relating to the token reader (includes *userTokenPresence* and *adminTokenPresence*)

3.2.16.2 Operations

This is the package within the secondary SPARK system. As such there is no correspondence between Z operations and implemented operations. Here we state the purpose of each operation rather than map it to the Z.

Ada Operation	**operation type**	**Purpose**
TokenReader.Poll	procedure	Polls the specified reader for the status of the token within that reader. Records status information within the TokenReader.State.
TokenReader.TheTokenPresence	function	Retrieves the current presence of a token in the specified reader.
TokenReader.TheTokenID	function	Retrieves the current token Id of a token in a specified reader.
TokenReader.TheTokenTry	function	Retrieves the current status of the token in the specified reader, in terms of *noToken, badToken* or *goodToken.*
TokenReader.GetCertificate	procedure	Reads the required token from the specified reader.
TokenReader.WriteAuthCert	procedure	Writes the supplied auth certificate to the User Token.
TokenReader.Init	procedure	Initialises the token readers, obtaining a list of all the connected readers.

3.2.17 UserEntry

This variable package maintains the state and controls the state-machine managing the User Entry activity.

3.2.17.1 Types

Ada Type	type classification	Z type
UserEntry.Status	enumeration	STATUS

3.2.17.2 State

Ada Package state	type	Z State
UserEntry.State	private	status fingerTimeout tokenRemovalTimeout

3.2.17.3 Operations

Ada Operation	operation type	Z Operation
UserEntry.CurrentActivityPossible	Boolean function	*CurrentUserEntryActivityPossible*
UserEntry.CanStart	Boolean function	*UserEntryCanStart*
UserEntry.InProgress	Boolean function	*UserEntryInProgress*
UserEntry.Progress	procedure	*TISProgressUserEntry*
UserEntry.StartEntry	procedure	*TISStartUserEntry*
UserEntry.DisplayPollUpdate	procedure	*DisplayPollUpdate*

3.2.18 UserToken

Variable package holding and managing the user token state.

3.2.18.1 State

Ada Package state	type	Z State
UserToken.State	private	*currentUserToken* *userTokenPresence*

3.2.18.2 Operations

Ada Operation	operation type	Z Operation
UserToken.Poll	procedure	*PollUserToken*
UserToken.UpdateAuthCert	procedure	*UpdateUserToken*
UserToken.ExtractUser	function	*extractUser*
UserToken.IsPresent	Boolean function	*userTokenPresence = present*
UserToken.ReadAndCheckAuthCert	procedure	*ReadUserToken* *UserTokenWithOKAuthCert*
UserToken.ReadAndCheck	procedure	*ReadUserToken* *UserTokenOK* *UserTokenNotOK*
UserToken.AddAuthCert	procedure	*AddAuthCertToUserToken*
UserToken.GetIandATemplate	retrieval function	value of *iandACertContents.templateC*
UserToken.GetClass	retrieval function	value of *authCertContents.clearance.Class*
UserToken.Init	procedure	-
UserToken.Clear	procedure	*ClearUserToken*

The operation ExtractUser makes use of Cert.ID.ExtractUser and Cert.Attr.ExtractUser.

3.3 Utility Layers

3.3.1 CertProcessing

This utility layer provides operations associated with certificate processing.

3.3.1.1 Operations

Ada Operation	**operation type**	**Z Operation**
CertProcessing.ExtractIDCertData	procedure	*extractIDCert*
CertProcessing.ExtractPrivCertData	procedure	*extractPrivCert*
CertProcessing.ExtractIACertData	procedure	*extractIandACert*
CertProcessing.ExtractAuthCertData	procedure	*extractAuthCert*
CertProcessing.ConstructAuthCert	procedure	*constructAuthCert*
CertProcessing.ObtainSignatureData	procedure	retrieve *signature*
CertProcessing.AddAuthSignature	procedure	sets *signature*

3.3.2 Configuration

This utility layer provides operations associated with managing configuration data.

3.3.2.1 Operations

Ada Operation	**operation type**	**Z Operation**
Configuration.UpdateData	procedure	*FinishUpdateConfigData*
Configuration.Initialise	procedure	*TISStartUp* (partial) *InitConfig*

The initialise operation reads the configuration data from file, if one is available otherwise it sets configuration data to the default value.

3.3.3 Enrolment

This utility layer provides operations associated with the enrolment of the TIS. This will validate file data and insert it into the KeyStore.

3.3.3.1 Operations

Ada Operation	operation type	Z Operation
Enrolment.Validate	procedure	*ValidateEnrolmentData*

3.3.4 File

This utility layer provides types and operations for basic file manipulation.

3.3.4.1 Types

Ada Type	type classification	Z type
File.T	private	FLOPPY

3.3.4.2 Operations

Ada Operation	operation type	Z Operation
File.OpenRead	procedure	-
File.OpenWrite	procedure	-
File.OpenAppend	procedure	-
File.Close	procedure	-
File.SetName	procedure	-
File.GetName	procedure	-
File.Create	procedure	-
File.Exists	function	-
File.Construct	function	-
File.EndOfFile	function	-
File.EndOfLine	function	-
File.GetChar	procedure	-
File.GetString	procedure	-
File.GetInteger	procedure	-
File.PutInteger	procedure	-
File.PutString	procedure	-
File.SkipLine	procedure	-
File.NewLine	procedure	-
File.Compare	procedure	-
File.Copy	procedure	-
File.CreateDirectory	procedure	-

3.3.5 Poll

This utility layer provides operations to manage the polling of external data.

3.3.5.1 Operations

Ada Operation	operation type	Z Operation
Poll.Activity	procedure	TISPoll

3.3.6 Updates

This utility layer provides operations for updating the environment.

3.3.6.1 Operations

Ada Operation	operation type	Z Operation
Updates.Activity	procedure	TISUpdate
Updates.EarlyActivity	procedure	TISEarlyUpdate

Document Control and References

This material was originally developed by Praxis High Integrity Systems Ltd. under contract to the National Security Agency.

Changes history

Issue 0.1 (23 May 2003): Initial issue for internal review.

Issue 0.2 (30 June 2003): Included second SPARK subsystem.

Issue 1.0 (4 July 2003): Updates following internal review, S.P1229.7.9.

Issue 1.1 (22 August 2003): Updates due to fault reports S.P1229.6.11-12, 17, 20, 25-27, 29, 34, 36, 37, 39-40, 45.

Issue 1.2 (19 August 2008): Updated for public release.

Changes forecast

None. This document is now under change control.

Document references

1 INFORMED Design Method for SPARK, S.P0468.42.2, Issue 2.0, October 2001

2 TIS Formal Design, S.P1229.50.1.