Tokeneer ID Station
Overview and Reader's Guide

S.P1229.81.8
Issue: 1.7
Status: Definitive
4th October 2011

Originator

Rod Chapman (Principal Engineer)

Copies to:

Altran Praxis Limited
Project File

Contents

1 Introduction

This document is intended as an introduction, overview and "reader's guide" to the public release of the Tokeneer ID Station software and documentation.

Section 2 contains a brief description of the content of the release package, and the licences under which the material has been released. It goes on to recommend specific entry points into the project documentation for readers interested in particular topics.

Section 3 details changes that have been made between the original release of the Tokeneer material to our customer in 2003 and this release.

Sections 4 and 5 go into more detail, describing how the Tokeneer software components are built and analysed using the SPARK toolset for those interesting in re-producing the system binaries, analysis results, or proofs of correctness.

Section 6 illustrates the use of the demonstration user-interface for the system that Praxis developed following the original project.

Section 7 sets a few opening "challenges" for researchers to tackle.

Finally, Appendix A lists the tools required to build and analyse the Tokeneer documents and software.

1.1 Tokeneer project time-line

In early 2002, Praxis published the results of our work in developing the MULTOS CA system[1]. This led to an invitation to attend the NSA's High Confidence Systems and Software (HCSS) conference in Baltimore where the idea of a "demonstrator" project in secure software engineering was first proposed.

The remainder of 2002 was spent proposing and negotiating the terms of the contact, with the main development project taking place over 9 months in 2003. Results were presented at the HCSS conference in 2004, but we felt that the work deserved a wider audience beyond the "closed shop" of the NSA and its research partners. In 2005, we prepared a conference paper that was eventually cleared for publication and appeared in the 2006 IEEE International Symposium on Secure Software Engineering (ISSSE)[2].

At that time, the so-called "Grand Challenge" in Dependable System Evolution[3] had come to our attention calling for a repository of verified software to be used as a basis for scientific experiment and research. This led to the idea of releasing the Tokeneer project material under a suitable licence that would permit its inclusion in the repository.

This licence (in the form of an NSA "Technology Transfer Agreement") was signed by the NSA and Praxis in July 2008.

1.2 Acknowledgements

Many people have been involved in the development and release of the Tokeneer system.

Firstly, we would like to thank the NSA staff that had the vision and budget to make the project happen in the first place. In particular, Randolph Johnson worked tirelessly as our "customer" during the development work, and in the long process of securing approval to release the material in its current form.

The original development team at Praxis—Janet Barnes, David Cooper, and David Painter–deserve special mention for producing such a remarkable piece of work given such a limited budget.

Finally, we would like to thank the principals of the Verified Software Initiative, in particular Tony Hoare, Jay Misra, and Jim Woodcock, for their encouragement and patience during the approval and release process.

1.3 List of Acronyms

NSA	National Security Agency
SPRE	Software Process and Reliability Engineering Inc. (see www.spre-inc.com)
TIS	Tokeneer ID Station
TTA	Technology Transfer Agreement

2 TIS Public Release Navigation Guide

This section contains a brief tour of the released material, a reader's guide, and notes on the licences under which the material is supplied.

Important – Where to begin

This section does not intend to reproduce background information about the project that is available in other documents. In particular, for a thorough introduction to the Tokeneer system, the redevelopment project, and the results of the work, we strongly recommend that all readers begin by reading the Project Summary Report (document 81.1) or the paper from the ISSSE 2006 conference [2], which is available from www.altran-praxis.com.

2.1 Release Structure and Content

The release material is stored below five top-level directories. The content of each directory is as follows:

2.1.1 Documents – directory "docs"

This directory contains further subdirectories named for each document, consisting of the document's part number in the project filing system and its name. For example, document 50.1 "Formal Design" appears in subdirectory "50_1_Formal_Design."

Where possible, the source format of each document is supplied. Most are in Microsoft Word format. Documents containing the Z notation are in the LaTeX format. Documents where we are not able to supply the source are in PDF format.

2.1.2 Tools – directory "tools"

This directory contains the Praxis Z styles and tools for LaTeX. The use of these is described in appendix A.

2.1.3 Software source code – directory "code"

This directory contains a further six sub-directories as follows:

- Core. The source-code of the TIS Core software. Further subdirectories within here contain user-defined proof rules for the Simplifier and "PRV" files.
- Support. The source-code of the support software.
- Simulators. The source-code of the peripheral simulator.

- GuiDemo – the source code of the demonstration GUI.
- Test – 35 sub-directories each containing a test case from the System Test Specification. Also contains the source-code of the "makekey" utility.
- Testkeys – contains test token and enrolment data needed to start up and use the system.

2.1.4 Discovery – directory "discovery"

Provides a tutorial introduction to the SPARK Toolset, using Tokeneer as a worked example. The material is provided in HTML format. The tutorial is self-explanatory – to begin, simply open the file discovery/index.html in any standard web browser.

2.2 Licences

2.2.1 Material developed by Praxis under contract

The material generated by Praxis under contract to the NSA is distributed under the terms of the Technology Transfer Agreement (TTA) agreed by Praxis and the NSA. A copy of this agreement is included with (and must always accompany) the release. This material consists of

- The "Core" TIS Software (directory code/core)
- The "Support" TIS Software (directory code/support)
- The project documents (directory docs, except document 31.2 (see below))
- The test cases derived from the system test specification (directory code/test)
- The test tokens and biometric data (directory code/testkeys)

2.2.2 Material supplied by NSA

The Protection Profile (document 31.2) was developed by SPRE Inc under a separate contract, and is supplied "as is." As such, the Protection Profile is not subject to copyright protection as provided in 17 U.S.C. section 105, in line with section 4iii of the TTA.

2.2.3 Other material developed by Praxis not under contract to the NSA

The demo GUI (directory "code/guidemo"), tools (directory "tools"), peripheral simulator (directory "code/simulators") and Discovery Tutorial (directory "discovery") were developed by Praxis, but not under contract to the NSA.

These items are supplied under the terms of the “Two-Clause Simplified BSD Licence” (also known as the “FreeBSD Licence”) See http://www.freebsd.org/copyright/freebsd-license.html for more details.

2.3 Document Structure and Recommended Reading

The relationship between the various documents and the project phases that they relate to is detailed in section 3 of the Project Summary Report (document 81.1).

3 Changes from the original TIS

This section documents changes to TIS that have been made in preparing the public release since the original release of the material to NSA in 2003.

In general, we have tried to keep changes to an absolute minimum, so the material is an accurate reflection of that delivered within the original project's timescale and budget.

The following sections give a brief summary of changes:

3.1 Documents

All documents have been updated as follows:

- The names of now-retired NSA employees have been removed.
- The copyright notice(s) have been updated to comply with the NSA's TTA.
- Spelling mistakes discovered en-passant in reviewing these documents have been corrected.
- The company name has been changed from "Praxis Critical Systems" to "Altran Praxis Limited".

Finally, the "Code Verification Summary" (document 52.1) has been updated to reflect the results of proving the code using release 8.1 of the SPARK toolset, as described in section 5.

3.2 Code

The code has been updated as follows:

- Copyright notices have been updated to comply with the NSA's TTA.
- The core TIS components were re-analysed using Release 8.1 of the SPARK toolset. This resulted in a few minor improvements:
  - A single defect (detailed in section 5.1 below) has been corrected.
  - Explicit loop invariant assertions have been strengthened where necessary to add the "A = A%" conjunct where a "for" loop has a dynamic range.
  - Comments justifying expected flow errors and warnings have been replaced by the equivalent "accept" annotations. Analysis of the core software with SPARK Examiner 8.1 now passes with zero unjustified warnings or errors.

- Base-type assertions have been added to signed integer type declarations where necessary. These are correct for the compilers mentioned in Appendix A. For other compilers and target platforms, these might require modification.
- Redundant “with” and “use type” clauses identified by the compiler have been removed.
- Improvements to contracts and proof rules contributed by Phil Thornley (www.sparksure.com) have been incorporated.

- Subsequently, the core software has been re-analysed with the GPL 2011 Edition of the SPARK Toolset.
- The GPL 2011 Edition of GNAT has also been used to compile the system. This resulted in a single warning of a redundant type conversion, which has therefore been removed.
- The support software was also updated as follows:
  - Copyright notices have been updated to comply with the NSA’s TTA.
  - The TCPIP package has been updated to use the local machine for connecting to the peripheral simulators rather than SPRE’s machine, which is no longer available.
  - Redundant “with” and “use type” clauses identified by the compiler have been removed.

4 Building the TIS Components

Appendix A lists the compilers that have been used to build and test the Tokeneer software for this release.

There are four components that must be compiled: the core software, the peripheral simulator, and the demo GUI.

For each component, a GNAT/GCC project file has been produced to automate compilation using the "gnatmake" tool. The name of the project file matches the name of the main subprogram of each component, with the extension ".gpr".

This release only supports IA32/Windows at this time. Ports to other platforms should be straightforward.

4.1 The Core Software

The core software is built with the commands

```
cd code\core
gnatmake -Ptis
```

This results in the binary tis.exe.

4.2 The Peripheral Simulator

The simulator is built with the commands

```
cd code\simulators
gnatmake -Psim
```

This results in the binary sim.exe.

4.3 The demo GUI

The project file for the GUI is called tis_main.gpr. This contains a reference to the directory where the GtkAda bindings have been installed. This might have to be updated according to your local environment.

The GUI is then built with the commands

```
cd code\guidemo\src
```

```
gnatmake –Ptis_main
```

This results in the binary tis_main.exe.

4.4 The makecard utility

The makecard utility is built with the commands

```
cd code\test
gnatmake –Pmakecard
```

This results in the binary makecard.exe.

5 Reproducing the Code Proofs

This section assumes the reader is familiar with the SPARK language and toolset.

In preparing the original release of Tokeneer, the SPARK analyses and proofs were recreated using the release 8.1 of the SPARK toolset. This led to a few minor changes and improvements, including:

- Changes in the SPARK language facilitated some improvement in completeness of the proofs (i.e. more proofs were automatically discharged). In particular, since release 7.2, SPARK has supported a notation for reasoning about variables that control the bound of a dynamic "for" loop. This meant that some loop-invariant assertions could be strengthened.

- Since release 7.4, SPARK has included an "accept" annotation that can be used to justify expected warnings and errors. These annotations have been added as appropriate.

- The VC Generator has been significantly improved when it comes to VCs associated with arithmetic overflow checks.

The Simplifier tool has also been improved in many ways since the original project was complete, so we would also expect this to yield an improvement in proof automation. As such, some of the manually-created Proof Checker scripts became redundant for this release, since the corresponding proofs are discharged automatically by the new toolset.

Subsequently, the analyses and proofs have been reproduced using the GPL 2011 Edition of the SPARK toolset.

5.1 Analysis using the SPARK tools on the command-line

The "Core" directory contains the SPARK support and project files necessary to analyse the software. In particular, a default switch file (spark.sw), warning control file (tis.wrn), index file (tis.idx), meta-file (tis.smf) and configuration file (config.adb) are all supplied.

Static semantic and flow analysis of any one unit can be achieved by simply running the Examiner on that unit body, such as:

```
spark configdata.adb
```

Analysis of the entire core software is achieved using the supplied meta-file:

```
spark @tis
```

With the GPL 2011 Edition of SPARK, this analysis should produce:

```
   No errors or warnings
  757 summarized warnings
   29 expected (justified) warnings
```

The theorem-prover is then run by the “sparksimp” program, for example:

```
sparksimp -a -l
```

The status of the proof can be summarized by the “POGS” tool. For example

```
pogs
```

Commands to re-generate and re-simplify all VCs are contained in the batch file “runall.bat” in the “core” directory. This results in a proof summary file “core.sum” that closes with the overall summary:

```
VC summary:
------------------------------------------------------------------------
Note: (User) denotes where the Simplifier has proved VCs using one or
      more user-defined proof rules.

Total VCs by type:
                        -----------Proved By Or Using-----------
                 Total Examnr    Simp (User) ViCToR Checkr Review  False Undisc
Assert/Post        935    599     317(   48)      0      0     19      0      0
Precondition        67      0      67(    5)      0      0      0      0      0
Check stmnt.        36      0      36(   22)      0      0      0      0      0
Runtime check     1127      0    1126(    2)      0      0      1      0      0
Refinem. VCs       212    182      26(   24)      0      0      4      0      0
Inherit. VCs         0      0       0             0      0      0      0      0
================================================================================
Totals:           2377    781    1572(  101)      0      0     24      0      0
%Totals:                  33%     66%(   4%)     0%     0%     1%     0%     0%
==================== End of Semantic Analysis Summary ========================
```

5.2 Analysis using the SPARK tools from GPS

The “Core” directory also contains a file tis_spark_tut.gpr which is a *Project File* for the AdaCore GPS IDE. This sets the toolset switches for GPS to the same values as in the spark.sw file described above. It contains:

```
with "../common.gpr";

project tis_spark_tut is

   for Source_Dirs use ("./**", "../support");
   for Main use ("tis");
   for Languages use ("Ada", "Index", "Listing", "Metafile", "Siv", "Vcg");

   package Compiler renames Common.Compiler;
```

```
   package Builder renames Common.Builder;

   package Ide is
      for Default_Switches ("examiner") use ("-index_file=tis", "-listing=ls_",
"-config=config.adb", "-warning_file=tis", "-noswitch", "-vcg", "-rules=lazy");
      for Default_Switches ("sparksimp") use ("-a", "-l", "-p=2");
   end Ide;

end tis_spark_tut;
```

For more details on the format of the GPR file, see the GPS and GNAT documentation.

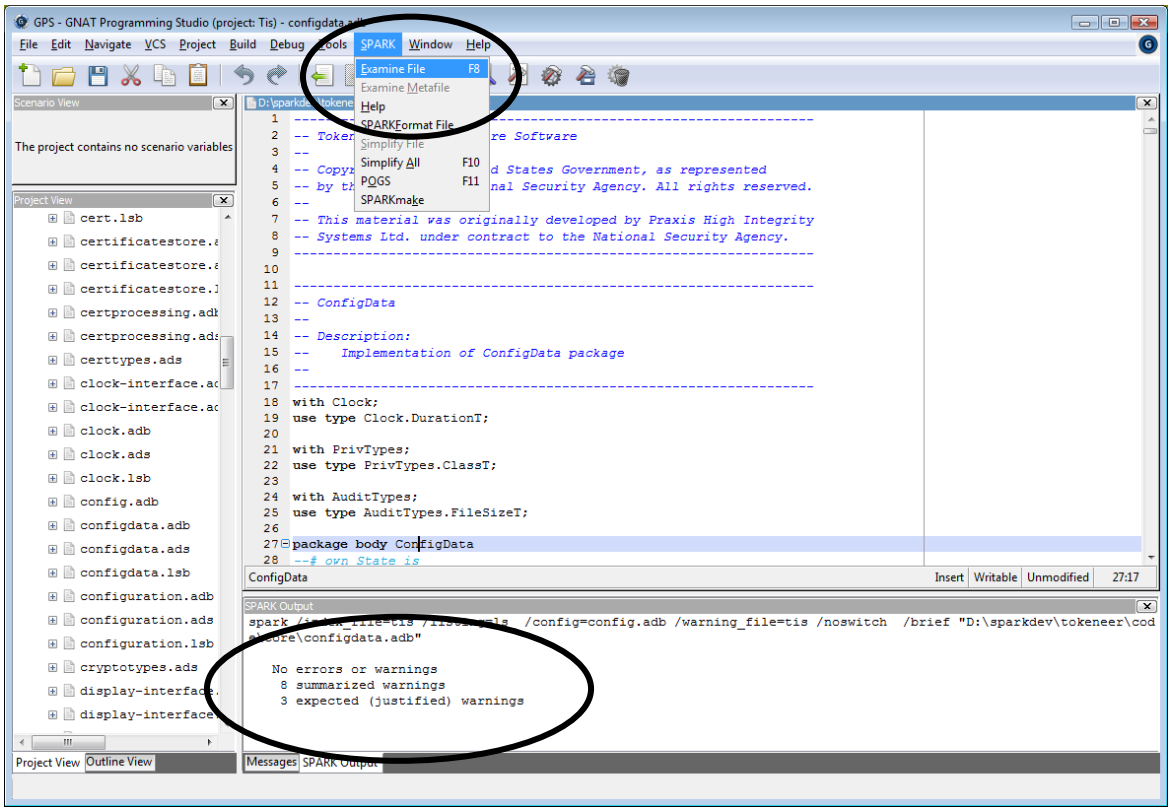
To analyse a single file, select that file in the GPS Editor Window, and do one of the following:

1 Select “Examine File...” from the SPARK Menu.

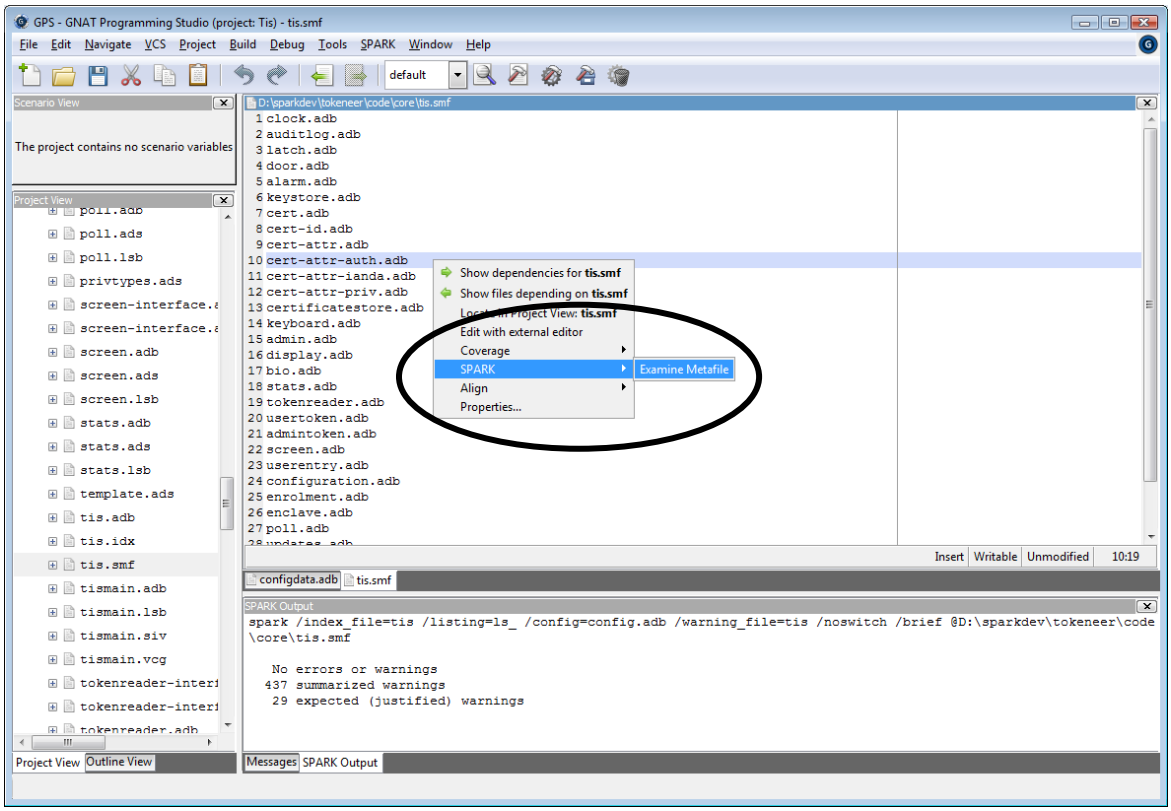
2 Select “Examine File...” from the contextual menu available from the right mouse button.

3 Press F8 on the keyboard.

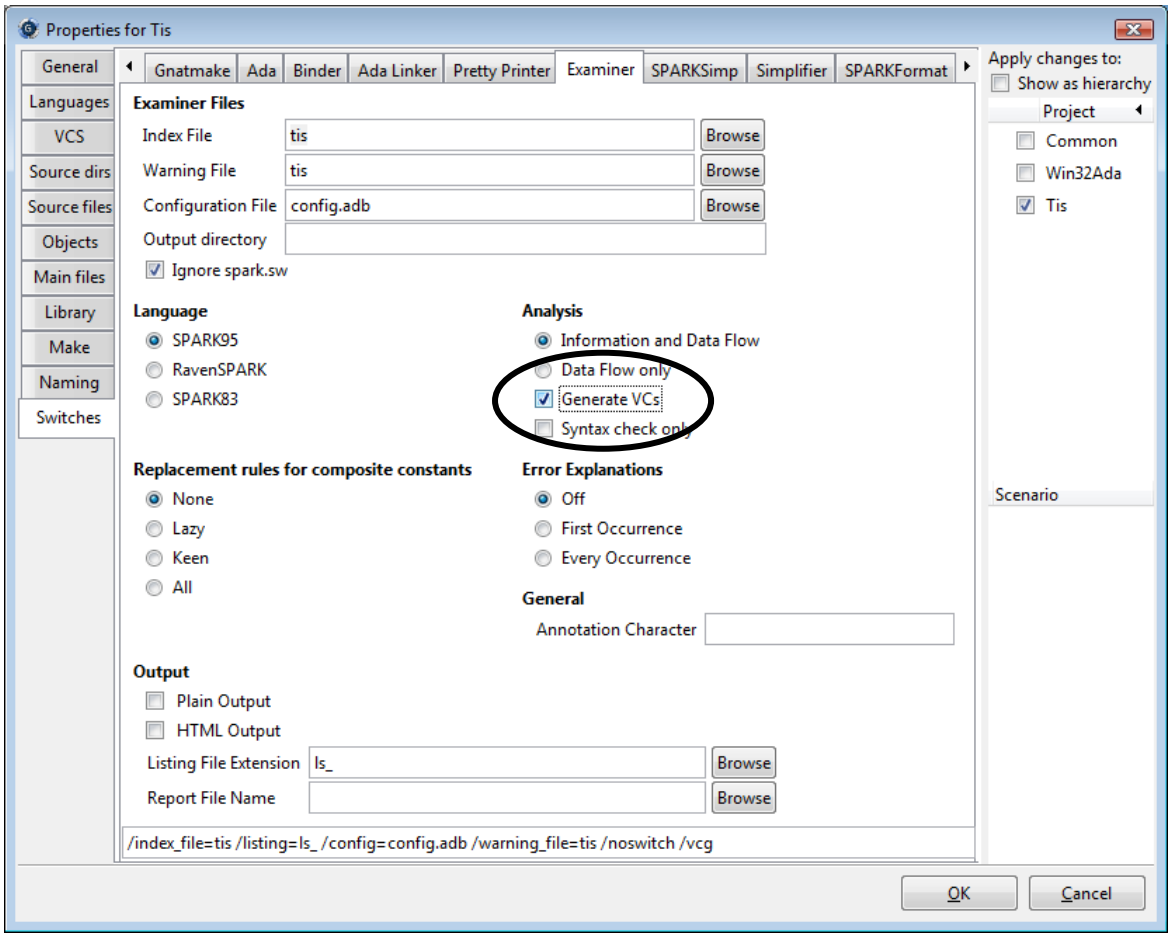
The following screen-shot illustrates the first of these options. The Examiner’s output summary can be seen in the “SPARK Output” window highlighted below:



To analyse the entire TIS core software, open the tis.smf meta-file in the GPS Editor, and select "Examine metafile…" from the menu, like this:

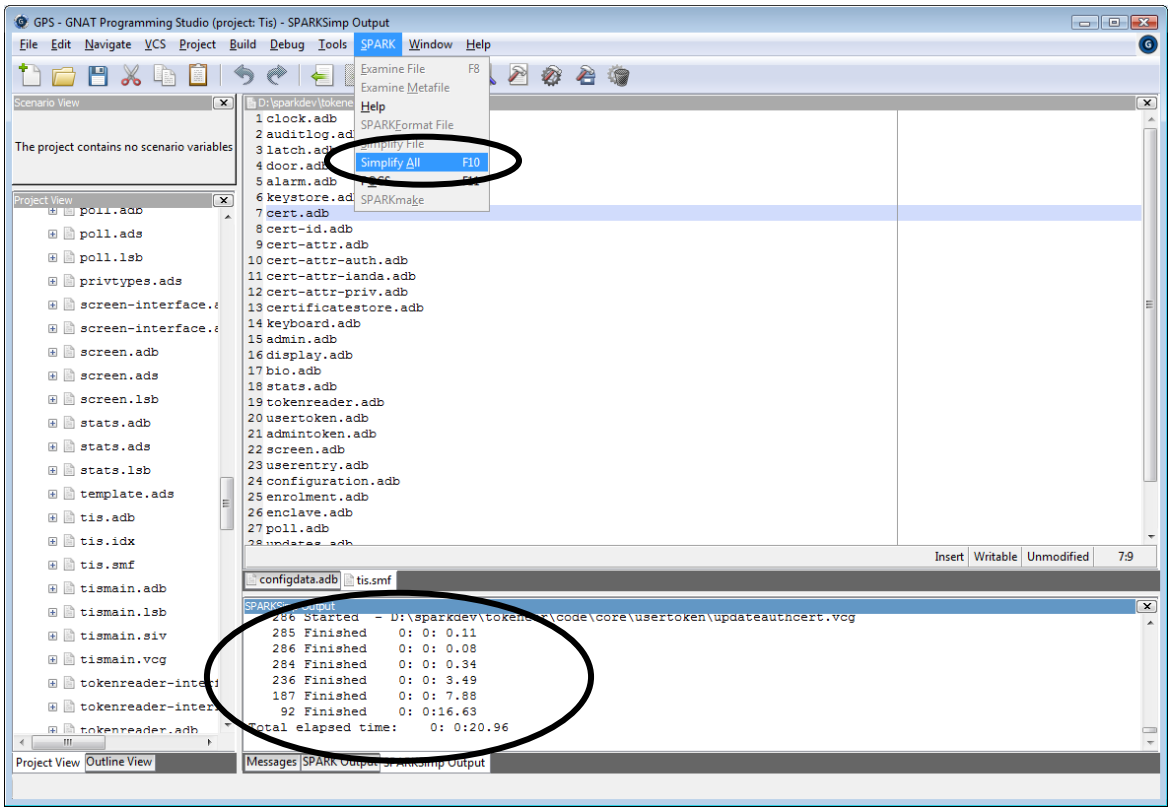


To generate VCs, add “Generate VCs” to the Examiner switches on the Edit Project Properties dialog (note: this is now default setting for Tokeneer release):



and save the project file using the Project/Save All menu. Then, reanalyse using the meta-file as before.

To simplify all VCs, select any source file in the editor window, and select “Simplify All” from the SPARK menu, or press F10. The progress of SPARKSimp can be seen in the “SPARKSimp Output” window:



Finally, POGS can be run from the SPARK menu. The resulting core.sum file can be opened using the File/Open dialog.

6 Defect Log

At the time the Tokeneer Project was made public in 2008, no defects had been found by the customer post delivery. Since the open-source release of the Tokeneer Project, its source code has been under the scrutiny of research teams and individual security experts, which has resulted in the discovery of a few defects. Currently four defects and a number of code quality issues have been found in the SPARK source code. Two of the defects could be found by later versions of the SPARK toolset than was used at the time of development, and two defects were found by directed review efforts. The four defects from the SPARK code are listed in the following subsections. Furthermore, 7 defects were found in the supporting Ada code. For a detailed description of all the issues, please refer to the ERTS 2010 paper on Tokeneer [4] or Woodcock's analysis in [5].

6.1 Possible Integer Overflow

The first defect was found by Rod Chapman when the Tokeneer code was reanalysed in preparation for the public release the POGS summary revealed a single undischarged VC. Further investigation showed this to be in the subprogram ConfigData.ValidateFile.ReadDuration.

The code in question concerns validation of an integer value that is read from a file, but is expected to be in the range 0 .. 200 seconds before it is converted into a number of tenths of seconds in the range 0 .. 2000.

The offending undischarged VC is essentially:

```
H1:    rawduration__1 >= - 2147483648 .
H2:    rawduration__1 <= 2147483647 .
 ->
C1:    success__1 -> rawduration__1 * 10 >= - 2147483648 and
                     rawduration__1 * 10 <= 2147483647 .
```

The code is from line 222 of configdata.adb:

```
if Success and then
  (RawDuration * 10 <= Integer(DurationT'Last) and
   RawDuration * 10 >= Integer(DurationT'First)) then
```

This VC clearly has a counter-example. For instance, when RawDuration = 10^9, H1 and H2 are True, but C1 is False. This reflects the possibility of an Integer overflow when multiplying by 10 *before* the range of RawDuration is checked.

The correction to the code is trivial. If replaced by:

```
if Success and then
  (RawDuration <= Integer(DurationT'Last) / 10 and
```

```
        RawDuration >= Integer(DurationT'First) / 10) then
```

then all VCs discharge successfully.

This change has been applied to the TIS Core software for this release, although the original code has been left "commented out" for reference alongside the corrected section.

6.1.1 Root cause analysis

Why was this defect not discovered and reported during the original development?

The original project used the SPARK Examiner's "rtc" switch to generate VCs – this generates VCs for partial correctness and run-time errors but *omits* those side-conditions relating to Ada's Overflow_Check. Previously, the SPARK toolset was limited in its capability to discharge these VCs, so these were omitted from the original project.

Subsequently, the SPARK toolset has become far more capable with regard to overflow conditions, through the use of the compiler-dependent configuration file, and the base-type assertion for integer types. Consequently, we can now generate VCs using the "vcg" switch which *does* include VCs for overflow checks.

It is interesting to note that this defect was not discovered by any testing during the original project, or any use or attempt to analyse the system since the initial delivery.

6.1.2 Security impact of this defect

Firstly, there is a potential denial-of-service attack resulting from this defect – a malicious user holding the "security officer" role can deliberately terminate the TIS core software by supplying a malformed configuration data file, rendering the system unusable.

More seriously, the software can be terminated in this fashion with the enclave door open.

6.2 AND instead of OR

The second defect found - the first after the open-source release of the Tokeneer Project - was a functional error. It was found by code review by Professor Diomidis Spinellis, who reported it in his blog in October 2008 [6].

In the following code from auditlog.adb, the value of AuditSystemFault is cleared if the deletion is successful, and not set if the deletion fails.

```
File.Delete (TheFile => TheFile,
             Success => OK);
AuditSystemFault := AuditSystemFault and not OK;
```

The problem is that a logical “and” operation was used instead of an “or” operation. This defect escaped basic static analysis and code review in the original project. The addition of a simple invariant on AuditSystemFault (i.e. once True, it should never go back to False) would have revealed this defect had this unit been subject to that level of analysis.

6.3 Wrong Variable Used

Running the GNAT compiler with warnings revealed that a condition in a test was known to be true (reported in [4]).

Indeed, the value of a variable RetValIni is tested twice instead of testing the value of another variable RetValDo the second time, which is typical of copy-paste errors.

```
if RetValIni = Interface.Ok then
   Interface.FindObjects
     (HandleCount   => HandleCount,
      ObjectHandles => Handles,
      ReturnValue   => RetValDo);
   if RetValIni = Interface.Ok then
      …
```

This warning was not present in the version of the compiler used during development of the Tokeneer Project. Note that it is also detected by the SPARK Dead Path Analyzer and AdaCore’s CodePeer tool.

6.4 Unused formal parameter

Again, a recent version of GNAT revealed this defect with all warnings enabled. This defect concerns the function NameOfType in the AuditLog package. This function uses a “slice” assignment, which is not permitted by SPARK, so it was hidden from the SPARK toolset (essentially, it is Ada, not SPARK), so was not subject to analysis with the SPARK toolset.

This function has a formal parameter called “E” which is not referenced in the body of the function. Instead, where the parameter E should appear, there is a reference to a global variable ElementID.

Further investigation reveals a single call to this function, with the actual parameter set to ElementID – meaning that the defect has no observable or functional effect and the behaviour of the code is actually correct for this one call.

Nevertheless, this is a bug “waiting to happen”, and would be fixed given the chance. We therefore consider it a defect in this context.

7 Running the TIS Demo GUI

The Tokeneer system can be run on a single machine using the TIS core software, peripheral simulator, and demo GUI all running on the same machine.

The software is currently written to expect all files to appear in a directory "C:\tokeneer\data". This can be changed in code\guidemo\src\enclave_pkg.ads if you want to install the system somewhere else.

This directory needs to contain the following files:

- Four "png" files supplied in code\guidemo*.png
- Eight token data files, supplied in code\testkeys\Admin*.dat (2 files) and code\testkeys\User*.dat (6 files)
- The binaries of the three main TIS applications – tis.exe, sim.exe, and tis_main.exe – plus the makecard.exe utility.
- 17 DLL files required by the GtkAda and Gtk GUI components. These are obtained from the "bin" directory of the GtkAda package that accompanies the GNAT compiler, as described in appendix A.

This totals 33 files. On our test machine, "dir /oe" of c:\tokeneer\data results in:

```
Directory of C:\tokeneer\data

22/08/2008  11:55             2,683 AdminLogin2_p06.dat
22/08/2008  11:57             2,683 AdminLogout1_p07.dat
22/08/2008  12:01             2,684 UserEntry5_p04.dat
22/08/2008  12:01             2,683 UserEntry4_p03.dat
22/08/2008  11:59             2,684 UserEntry2_p02.dat
22/08/2008  11:58             2,684 UserEntry1_p01.dat
22/08/2008  11:58             2,692 UserEntry13_p07.dat
22/08/2008  12:02             2,684 UserEntry6_p05.dat
18/08/2007  08:51           605,333 libgdk-win32-2.0-0.dll
18/08/2007  08:51           166,177 libgdk_pixbuf-2.0-0.dll
17/08/2007  18:07           642,115 libglib-2.0-0.dll
17/08/2007  18:07            28,853 libgmodule-2.0-0.dll
17/08/2007  18:07           223,026 libgobject-2.0-0.dll
18/08/2007  08:52         3,170,609 libgtk-win32-2.0-0.dll
08/08/2008  16:32         4,868,618 libgtkada-2.10.dll
07/04/2004  11:47            44,100 libintl-1.dll
17/08/2007  18:29           522,940 libcairo-2.dll
17/08/2007  18:36           262,784 libpango-1.0-0.dll
17/08/2007  18:36            62,334 libpangocairo-1.0-0.dll
17/08/2007  18:37            88,626 libpangowin32-1.0-0.dll
07/10/2001  01:52           171,008 libpng-3.dll
07/04/2004  11:46            58,077 libz.dll
07/04/2004  11:47           843,776 iconv.dll
17/08/2007  18:22           142,762 libatk-1.0-0.dll
16/01/2007  12:27           131,784 libjpeg6b.dll
22/08/2008  18:33         1,125,993 sim.exe
22/08/2008  18:33         2,118,000 tis.exe
22/08/2008  18:33         1,554,257 tis_main.exe
22/08/2008  10:24           810,959 makecard.exe
22/08/2008  15:36               159 light_green.png
22/08/2008  15:34               215 dark_red.png
```

```
22/08/2008  15:35               159 dark_green.png
22/08/2008  15:36               161 light_red.png
              33 File(s)     17,664,302 bytes
```

7.1 Preparing enrolment data

The original system assumed that the machine has a floppy disk for loading enrolment and configuration data. Luckily, the system will recognize a USB FLASH drive as a removable disk and work just as well. To start the system, enrolment data must be supplied on a FLASH drive or a floppy disk. The distribution includes three enrolment data files, in the directory code\testkeys.

Depending on the test case to be run, you will need one of three files installed on a single drive, which must be otherwise empty.

The files enrol1.dat and enrol2.dat are needed only for the test cases Enrol1 and Enrol2 respectively. The enrolment file enrol3.dat can be used for the test case Enrol3 or to start the system up for general use.

7.2 Starting the GUI

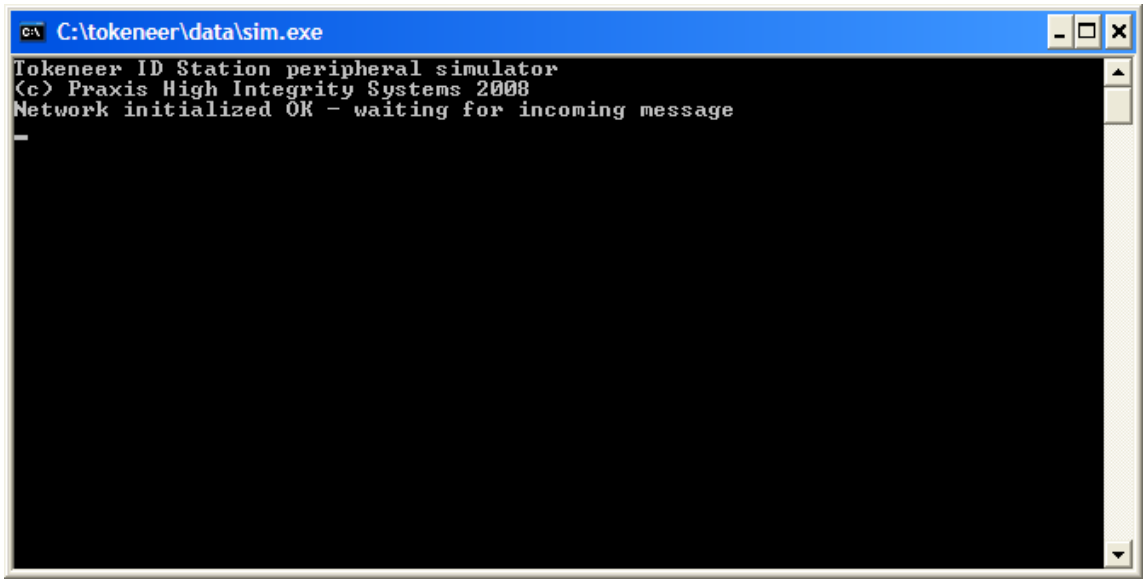
Insert a USB FLASH drive or a floppy disk containing the enrol3.dat file. This should be the first (i.e. closest to 'A') removable drive.

The GUI is started first by running the tis_main application from the directory above. In a Windows Shell:

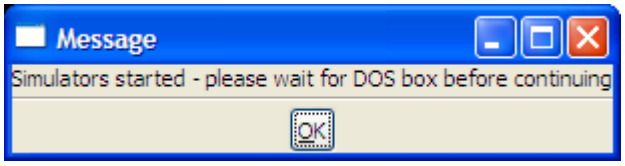
```
C:
cd \tokeneer\data
.\tis_main
```

This results in a small window with four buttons:

Press the "Start Simulators" button. This should result in a new Shell Window opening running the peripheral simulator application. You should see:

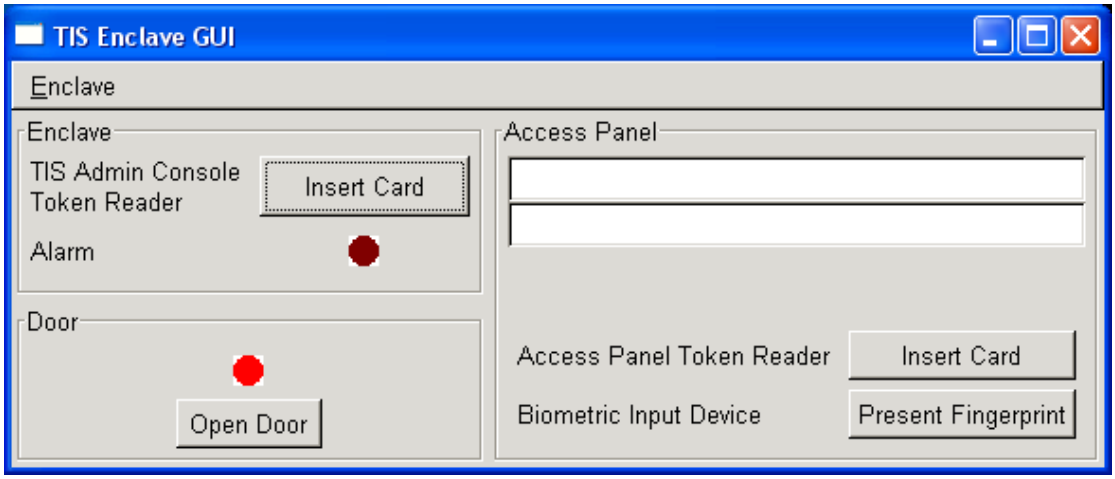


and a small dialog asking you click “OK” when the window above has appeared:

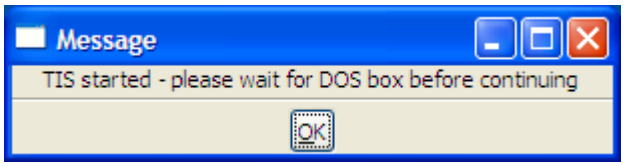


Click “OK”.

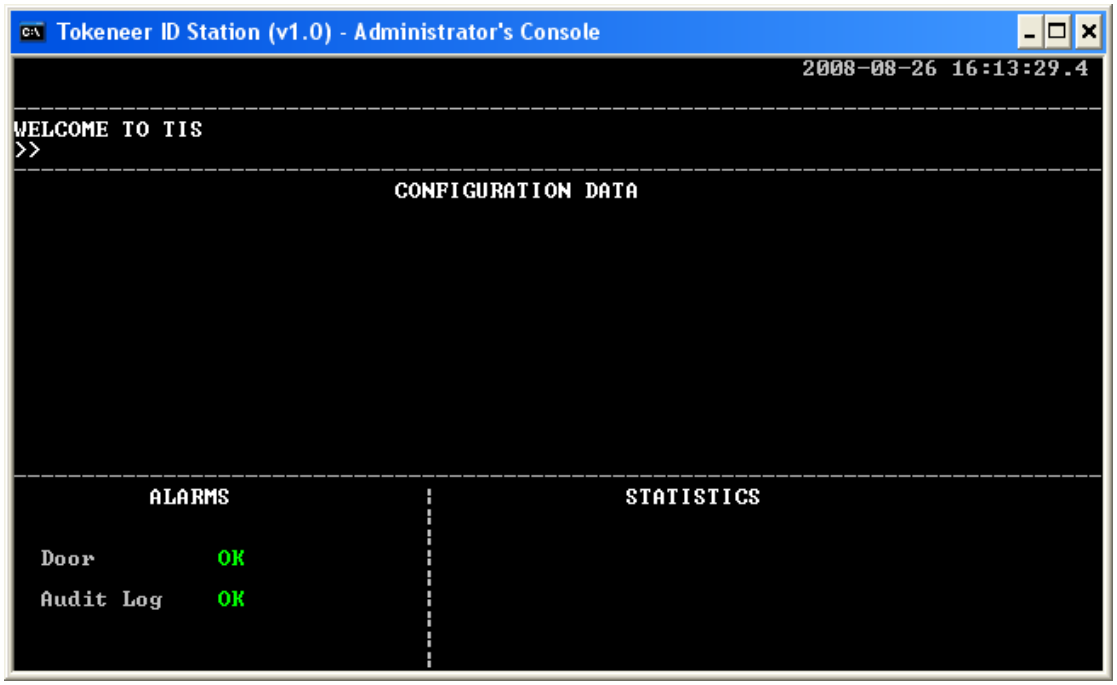
Now click on “Connect to Simulator.” This should open the Demo GUI’s main dialog:



Now click on “Start TIS”. You should see a final acknowledgement box:



and the main TIS core software running in another Shell Window:



If all is well, you should see the message “WELCOME TO TIS” and both the Door and Audit Log Alarms showing “OK”.

If the message “PLEASE INSERT ENROLMENT DATA FLOPPY” persists, then the system has not been able to find a floppy or other removable drive containing a suitable enrolment data file.

7.3 Stopping the System

For reasons unknown, the “Exit” button on the GUI is not effective. To stop the system:

- Type “Control-C” in the TIS Core Software Window. It should disappear.
- Type “Control-C” in the Simulator Windows. It too should disappear.
- Type “Control-C” in the Shell Window where you started tis_main. The GUI window should disappear and terminate at that point.

7.4 Resetting the System Enrolment data, Keystore and Log

The system keeps a cache of known enrolment data, keystore information and log data in two subdirectories “System” and “Log”. To reset the system to its initial state, simply remove these two directories and their contents.

8 Challenges with TIS

The Tokeneer project had a well-defined scope and budget that leave several topics open to further work and investigation. This section suggests a list of "challenges" that others might choose to pursue. In no particular order:

8.1 Remove dependence on floppy disks

In 2003, it seemed reasonable to expect machines to have a floppy disk. This is no longer the case.

Challenge: Re-engineer the core and support software to remove the need for a floppy disk for enrolment and configuration data.

8.2 Remove dependence on Windows

Several parts of the software explicitly depend on the Win32 API.

Challenge: Remove all dependence on Win32 and produce a "port" of the System to other systems such as GNU/Linux or OS X.

8.3 Remove "Proof by Review"

The "Proved by Review" (PRV) files are a weak point in the verification.

Challenge: Remove these and produce Checker proof scripts for all VCs.

8.4 Alternative theorem prover support

The GPL 2011 edition of the SPARK tools manages to prove 2252 out of 2377 Verification Conditions automatically, without the use of user-defined proof rules.

Challenge: Demonstrate automated proof of all VCs.

8.5 Proof of Security Properties

Within the scope of budget of the original project, we were only able to produce a formal proof of Security Property 3, and a proof of one aspect of Security Property 1, as described in Section 3.1 of the Code Verification Summary report.

Challenge: Complete proofs of all remaining security properties, at the level of the function specification (Z), formal design (Z), or code (SPARK), or possibly all of the above. Can these proofs also be automated? Can a formal refinement between all 3 levels also be established?

8.6 Hardware

The use of the simulated peripherals was a necessity in the original project, owing to the lack of access to the real Tokeneer hardware.

Challenge: produce a hardware mock-up of the Tokeneer system (a door, latch, alarm, token reader, fingerprint reader and so on) suitable for university teaching and research. Interface this hardware to some suitable low-cost single-board embedded computer, and re-write the Tokeneer interfacing and support software (in SPARK) to run on this hardware.

8.7 Re-implement using alternative languages and technologies

Designers and users of other specification languages, programming languages and verification tools could choose to re-implement the entire system, changing one or more of the notations and tools used.

A Tools Inventory

This section lists the various tools that are required to build and analyse the TIS software.

In this section the name "<tisroot>" refers to the directory where you have installed the TIS distribution.

A.1 Documents

The majority of documents have been prepared using Microsoft Word 2000 (Build 9.0.6926 SP3). Other versions of Word may well suffice.

Three documents (The formal specification, formal design, and security properties) contain the Z notation. These documents require the LaTeX text formatting system, the Praxis Z styles, the FUZZ typechecker, and the Praxis Z tools.

A.1.1 LaTeX

In preparing this release of the Tokeneer material, MikTex 2.7 for Windows has been used, obtained from http://miktex.org/2.7/Setup.aspx.

We installed MikTex in c:/Program Files/MikTex/ - this is referred to as "<texroot>" in subsequent sections.

A.1.2 Praxis Z styles

These are contained in the TIS distribution in the directory "<tisroot>/tools".

Create a directory <Texroot>/tex/latex/praxis, then copy *.cls and *.sty from <tisroot>/tools to <Texroot>/tex/latex/praxis.

Open the MikTex 2.6 "Settings" Utility (from the Windows Start Menu), and click the "Refresh FNDB" button. Click "OK" to exit the Settings Utility.

A.1.3 FUZZ

FUZZ is a type-checker for Z. We used the 2007-09-11 release of FUZZ, obtained from http://spivey.oriel.ox.ac.uk/mike/fuzz/

FUZZ is supplied in source form so you'll need a C compiler (I used GNAT Pro 6.1.2), plus up-to-date builds of the make, flex, bison, gawk, and cpp tools. On Windows, we use the Cygwin versions of these tools from www.cygwin.com.

Once you have compiled the FUZZ binary, the FUZZ fonts and styles must also be installed, as follows:

Create 2 more directories: <Texroot>/fonts/source/local and <Texroot>/tex/latex/fuzz

Find the "tex" subdirectory within the FUZZ installation. From this directory, copy *.mf to <Texroot>/fonts/source/local and *.sty to <Texroot>/tex/latex/fuzz.

Refresh the MikTex FNDB once again, as described in section A.1.2 above.

Set an environment variable called "FUZZLIB" to be the full path-name of the "fuzzlib" file within the FUZZ "src" directory. For example, in our environment, FUZZLIB is set to "d:\sparkdev\tis\fuzz\fuzz-2007-09-11\src\fuzzlib".

A.1.4 Praxis Z tools

Praxis uses two tools to produce cross-reference and index information for Z/LaTeX documents. These are called "zlat2" and "zmakeindex" respectively, both of which are written in PERL.

PERL 5.10.0 has been successfully used in reproducing the TIS material.

These tools are supplied in the directory <tisroot>/tools.

The "makefiles" for each of the Z documents sets an environment variable "PXTOOLDIR" to point at the directory where these tools reside. You will need to alter these makefiles as appropriate.

A.1.5 Reproducing the Z documents

The directories that contain each of the 3 Z documents contain a makefile with two targets "typecheck" and "typeset".

The makefiles assume that fuzz, latex, and perl are all on your "PATH". They also use "cp" and "rm" commands. On Windows, we use the Cygwin versions of these tools from www.cygwin.com.

To typecheck a document, do "make typecheck".

To typeset, do "make typeset". This results in a postscript file. These have been passed through Adobe's Acrobat Distiller to produce the PDF versions of the documents.

A.2 The SPARK Toolset

The TIS Core software has been analysed with SPARK GPL Edition 2011 and SPARK Pro 10.0.0 and also with the most recent "wavefront" release of the toolset at the time of writing. See www.adacore.com/sparkpro.

Praxis is committed to supplying and supporting the SPARK toolset for those using the TIS material for teaching and/or research. Please go to libre.adacore.com for details of how to obtain the SPARK toolset.

A.3 Compilers

The TIS software was designed to run on IA32/Windows.

In preparing the TIS release, we have compiled and tested the TIS software using the following Ada compilers:

GNAT GPL Edition 2011 on Windows XP SP2 – available from libre.adacore.com

GNAT Pro 6.4.2 on Windows XP SP2 – as above.

You'll also need the Win32Ada bindings package, and (to rebuild the Demo GUI) the GtkAda package. Both are available from AdaCore. Either the GAP, GPL, or Pro editions of these packages should be used to match the compiler.

GNAT GPL 2011 was also used to build the FUZZ tool.

Document Control and References

Changes history

Issue 0.1 (25th August 2008):	First draft issue for review.
Issue 0.2 (4th September 2008):	Second draft after review.
Issue 0.3 (10th September 2008):	Third draft following review within SPARK Team.
Issue 0.4 (11th September 2008):	Updated references for compilers.
Issue 1.0 (16th September 2008):	Definitive issue following review.
Issue 1.1 (23rd April 2009):	Updates for SPARK Toolset 8.1.x.
Issue 1.2 (24th April 2009):	Updated following review S.P0468.7.143.
Issue 1.3 (18th May 2009):	Updates for Tokeneer Discovery.
Issue 1.4 (20th May 2009):	Definitive issue following review.
Issue 1.5 (19th May 2010):	Updates for ERTS, defect log.
Issue 1.6 (9th September 2011):	Updates for GPL 2011 release of Tokeneer.
Issue 1.7 (4th October 2011):	Updates following review.

Changes forecast

None.

Document references

1 *Correctness by Construction—Developing a Commercial Secure System*. Roderick Chapman and Anthony Hall. IEEE Software Jan/Feb 2002.

2 *Engineering the Tokeneer Enclave Protection Software*. Janet Barnes, Rod Chapman, Praxis High Integrity Systems. Randy Johnson, National Security Agency. David Cooper, River River Limited. Bill Everett: SPRE Inc. Proceedings of the IEEE International Symposium on Secure Software Engineering (ISSSE) 2006.

3 See http://www.fmnet.info/gc6/

4 *Tokeneer: Beyond Formal Program Verification.* Yannick Moy, AdaCore. Angela Wallenburg, Altran Praxis. Presented at Embedded Real Time Software and Systems (ERTS) 2010, http://www.open-do.org/wp-content/uploads/2010/04/ERTS2010_final.pdf

5 *The Tokeneer Experiments.* Jim Woodcock, Emine Aydal, and Rod Chapman. in *Reflections on the Work of C. A. R. Hoare.* Springer 2010. ISBN 978-1-84882-911-4.

6 http://www.spinellis.gr/blog/20081018/index.html