NISTIR 7771

Conformance Test Architecture for Biometric Data Interchange Formats - Version Beta 2.0

Fernando L. Podio
Dylan Yaga
Mark Jerde

NIST

**National Institute of
Standards and Technology**

U.S. Department of Commerce

Conformance Test Architecture for Biometric Data Interchange Formats - Version Beta 2.0

Fernando L. Podio
Dylan Yaga
Mark Jerde

February 2011

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analysis to advance the development and productive use of information technology. ITL's responsibilities include the development of technical, physical, administrative, and management standards and guidelines for the cost-effective security and privacy of sensitive unclassified information in Federal computer systems. This Interagency Report discusses ITL's research, guidance, and outreach efforts in computer security, and its collaborative activities with industry, government, and academic organizations.

Abstract

The success of biometric applications is particularly dependent on the interoperability of biometric systems. Deploying these systems requires a comprehensive portfolio of biometric standards developed in support of interoperability and data interchange. A number of these domestic and international standards have been published and others are under development. The existence of these standards alone is not enough to demonstrate that products meet the technical requirements specified in the standards. Conformance testing captures the technical description of a specification and measures whether an implementation faithfully implements the specification. The Computer Security Division of NIST/ITL supports conformity assessment efforts through active technical participation in the development of conformance testing methodology standards and the development of associated conformance test architectures (CTA) and test suites (CTS). This NIST IR discusses the technological characteristics of the recently released CTA Beta 2.0. This architecture supports CTSs such as those designed to test implementations of biometric data interchange data formats. The information provided includes CTA module communication methods, key CTA features and high-level sequence diagrams such as testing and decoding operations. It also addresses an introduction to testing binary data, structure testing by groups of fields and a discussion on test cases. Ongoing work on related tools development is also presented.

Table of Contents

List of Figures

List of Tables

1 Introduction to Biometric Standards and Conformance Testing

1.1 Current Biometric Standard Development Activities in Support of Interoperability and Data Interchange

One of the critical issues related to secured Information Technology (IT) systems and applications is the verification of the users' identities. The relationship between a biometric characteristic (e.g., something that you are) and the users of a system or application provides a binding that is stronger than the binding that can be achieved between a user and other technologies that are currently in use for "personal authentication" such as passwords (e.g., something that you know) and tokens (e.g., something that you have). For decades, biometric technologies were used primarily in law enforcement applications. However, over the past several years the marketplace for biometrics solutions has significantly widened. Currently, they are increasingly being required in public and private sector applications worldwide to authenticate a person's identity, secure national borders and restrict access to secure sites including buildings and computer networks.

The success of biometric applications is particularly dependent on the interoperability of biometric systems. Deploying these systems requires a comprehensive portfolio of biometric standards developed in support of interoperability and data interchange. In the U.S., the InterNational Committee for Information Technology Standards (INCITS) Technical Committee M1 – *Biometrics* was established with the purpose to ensure a high priority, focused, and comprehensive approach in the United States for the rapid development and approval of formal national and international biometric standards [1]. A number of American National Standards developed by INCITS M1 have been published [2]. They include biometric technical interface standards, data interchange formats, conformance and performance testing methodologies and biometric application profiles.

NIST/ITL is responsible for the development of the ANSI/NIST ITL standard "Data Format for the Interchange of Fingerprint, Facial & Other Biometric Information". NIST/ITL (and its predecessor organizations) has been accredited by ANSI as a standards developer since October 5, 1984. NIST/ITL is accredited to develop voluntary consensus standards as a sponsor using the ANSI Canvass Method. This standard is used by law enforcement, intelligence, military, and homeland security organizations throughout the world. The current version of the ANSI/NIST-ITL standard is: Part 1 – (ANSI/NIST-ITL 1-2007) in Traditional Format and Part 2 – (ANSI/NIST-ITL 2-2008) in NIEM[1]-conformant XML format [3].

ISO/IEC JTC 1/SC 37 - *Biometrics* was established in June 2002 and is responsible for the development of a large portfolio of international biometric standards in support of interoperability and data interchange. Currently the subcommittee's membership includes twenty-eight national bodies, ten observer countries and a number of liaison organizations [4]. At the present time, the subcommittee is developing the "second generation" of

---

[1] NIEM, the National Information Exchange Model, is a partnership of the U.S. Department of Justice, the U.S. Department of Homeland Security, and the U.S. Department of Health and Human Services. It is designed to develop, disseminate and support enterprise-wide information exchange standards and processes that can enable jurisdictions to effectively share critical information in emergency situations, as well as support the day-to-day operations of agencies throughout the nation.

biometric standards with the goal of improving the existing standards, adding functionality to the published standards and reflecting technology innovations and new customers' needs. At the time of publication, forty-four international standards and six technical reports developed by the subcommittee have been published [5]. In addition to the type of standards developed by INCITS M1, ISO/IEC JTC 1/SC 37 is responsible for the development of biometric sample quality standards and standards and technical reports in support of cross jurisdictional issues related to the utilization of biometric technologies in commercial applications. The subcommittee is also developing a harmonized biometric vocabulary to serve the standards community as well as other customers.

1.2   Overview of Conformance Testing

The existence of base biometric standards alone (e.g., biometric data interchange standards, technical interface standards) is not enough to demonstrate that products meet the technical requirements specified in the standards. Conformance testing captures the technical description of a specification and measures whether an implementation faithfully implements the specification. INCITS M1 and JTC 1/SC 37 are responsible for the development of conformance testing methodology standards. At the present time, a number of these standards have been published and others are under development. They specify conformance testing methodology standards for biometric technical interfaces (e.g., BioAPI specification developed in INCITS M1 and JTC 1/SC 37) and biometric data interchange standard formats (e.g., 1st and 2nd generation of international biometric data interchange formats published or under development in JTC 1/SC 37).

The Computer Security Division (CSD) of NIST/ITL supports the development of biometric conformance testing methodology standards and other conformity assessment efforts through active technical participation in the development of these standards and the development of associated conformance test architectures and test suites. NIST/ITL CSD develops these conformance test tools to support users that require conformance to selected biometric standards and to also support product developers interested in conforming to biometric standards by using the same testing tools available to users. These test tools support the possible establishment of conformity assessment programs to validate conformance to biometric standards. Conformance testing provides developers, users, and purchasers, with increased levels of confidence in product quality and increases the probability of successful interoperability.

1.3   Previous and Current Development Work

1.3.1   BioAPI Conformance Test Suite

In 2006, NIST/ITL CSD released a BioAPI Conformance Test Suite (CTS) developed to test implementations of ANSI INCITS 358-2002, the BioAPI specification. This software tool was developed to help users verify the conformance of Biometric Service Providers to the BioAPI Specification 1.1 [6]. NIST/ITL CSD also co-sponsored with other members of INCITS M1 the development of a conformance testing methodology standard for BioAPI. This standard was published in 2008 as ANSI INCITS 429-2008, Information technology - Conformance Testing Methodology for ANSI INCITS 358-2002 BioAPI Specification. The BioAPI CTS implementation was developed using concepts and principles specified in the conformance testing methodology standard. This CTS was thoroughly tested with a number

of commercially available vendor biometric subsystems for different modalities (e.g., face, iris and fingerprint recognition) claiming conformance to the BioAPI standard. The test results were successfully cross-validated with another similar CTS independently developed by DoD's Biometric Task Force. The National Science and Technology Council Subcommittee on Biometrics and Identity Management listed the BioAPI CTS developments as one of the *"Technology Successes"* of 2006.

1.3.2 Conformance Test Architecture for Biometric Information Records - Beta 1.1

In August 2008 NIST/ITL CSD released a conformance test architecture for Biometric Information Records and a Conformance Test Suite (CTS) for Patron Format A data structures specified in ANSI INCITS 398-2008, Information technology - Common Biometric Exchange Formats Framework [7]. The CTS for Patron Format A supported by this conformance testing architecture was developed to help users determine whether binary file implementations of Biometric Information Records (BIRs) based on this Patron Format conform to the standard. NIST/ITL CSD sponsored in INCITS M1 development of a conformance testing methodology standard for CBEFF (Common Biometric Exchange Format Framework) data structures specified in ANSI INCITS 398-2008 and submitted to INCITS M1 the test assertions and related test cases developed for the Patron Format A Conformance Test Suite as well as test assertions and test cases for other Patron Formats specified in the ANSI INCITS 398-2008 standard.

1.3.3 Advanced CTA for Biometric Data Interchange Formats and BIRs - Beta 2.0

CTA Beta 2.0, recently developed, supports CTSs designed to test implementations of biometric data interchange formats and the three components of BIRs conforming to CBEFF standards. As detailed below, CTA Beta 2.0 incorporates features designed to improve the confidence and reliability of test results and the usability of the test tools. Software development testing approaches incorporated in this CTA version allow for the potential of cleaner, more trustworthy code. In August 2010, this CTA was released with CTSs designed to test implementations of finger image and finger minutiae biometric data interchange formats specified in four American National Standards:

- ANSI INCITS 378, American National Standard for Information technology – Finger Minutiae Format for Data Interchange (2004 and 2009 versions)
- ANSI INCITS 381, American National Standard for Information technology - Finger
- Image Based Interchange Format (2004 and 2009 versions)

Downloads of CTA Beta 2.0 and the CTS, associated documentation, and sample data can be found in the download page [8] of the NIST Biometric Resource Center which is maintained by NIST/ITL Computer Security Division [9]. This NIST IR provides information on the technical characteristics of CTA Beta 2.0, including Graphical user Interface/Controller communication methods, key CTA features and high-level sequence diagrams. This NIST IR also addresses an introduction to testing binary data, structure testing by groups of fields and a discussion on test cases.

1.3.4　Ongoing work

Upgrades to the current version of the CTA are being made to improve usability and performance. Development of CTSs to test implementations of selected international standards (finger, face and iris image data formats) is underway. NIST/ITL CSD is also developing a CTS to test implementations of selected Record Types of ANSI/NIST ITL 1-2007, Data Format for the Interchange of Fingerprint, Facial, & Other Biometric Information – Part 1" and started analysis of the existing draft for the 2011 version of the ANSI/NIST ITL standard [3].

2 Technical Characteristics of NIST/ITL CSD CTA Beta 2.0

2.1 Introduction

NIST/ITL CSD Beta 2.0 CTA supports CTSs designed to test implementations of biometric data interchange formats and any of the three components of Biometric Information Records (see Annex D). These CTSs determine whether data conforms to one or more implementations of standards or specifications. Typically, the user selects the data to be tested as well as the CTS designed to test implementations of the particular standard or specification, and then begins the testing process. Figure 2-1 depicts a conceptual representation of the CTA. The output of the selected CTS includes Graphical user Interface (GUI) indicators as well as user-selectable Logs in XML format and Reports developed in HTML format.
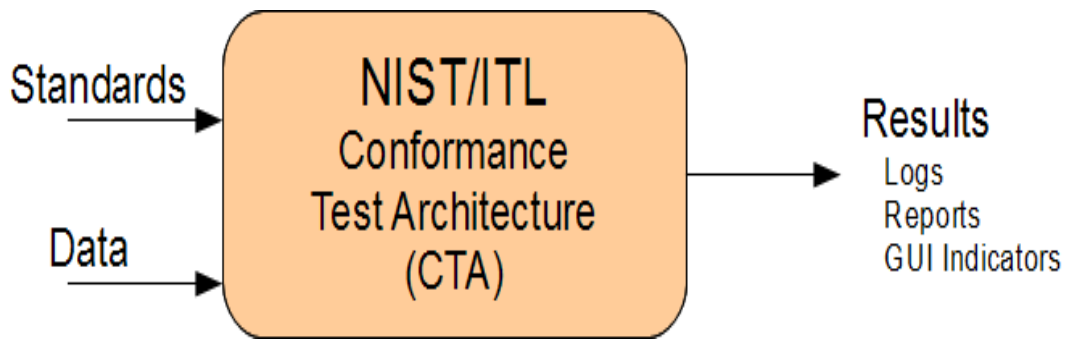
Figure 2-1: NIST/ITL CTA

2.2 High Level Architecture - GUI, Controller and CTS Modules

As shown in Figure 2-2, the CTA architecture has three main components: the GUI, the Controller and the CTS Modules.

The GUI is the component that interacts with the user to perform conformance testing. The Controller is the component between the GUI and the Testing Modules (CTSs) that accepts testing requests from the GUI, calls the specified modules with the required data, and returns the consolidated results from the Modules to the GUI. The Controller is more than a traffic cop between the GUI and the Modules. It performs high-level data testing such as: "Error: 512 bytes expected but only 100 passed." It also ensures all Field sub-arrays are initialized so that Modules are freed from null testing.

CTSs are the components that test implementations of a standard or specification for conformance to that standard or specification (e.g., ANSI INCITS 378-2009 - Information Technology - Finger Minutiae Format for Data Interchange). As stated above, the CTA currently supports CTS Modules designed to test implementations of biometric data interchange formats as well as Modules to test implementations of each component of a Biometric Information Record (i.e., Standard Biometric Header (SBH), Biometric Data Block (BDB), and Security Block (SB)).

5
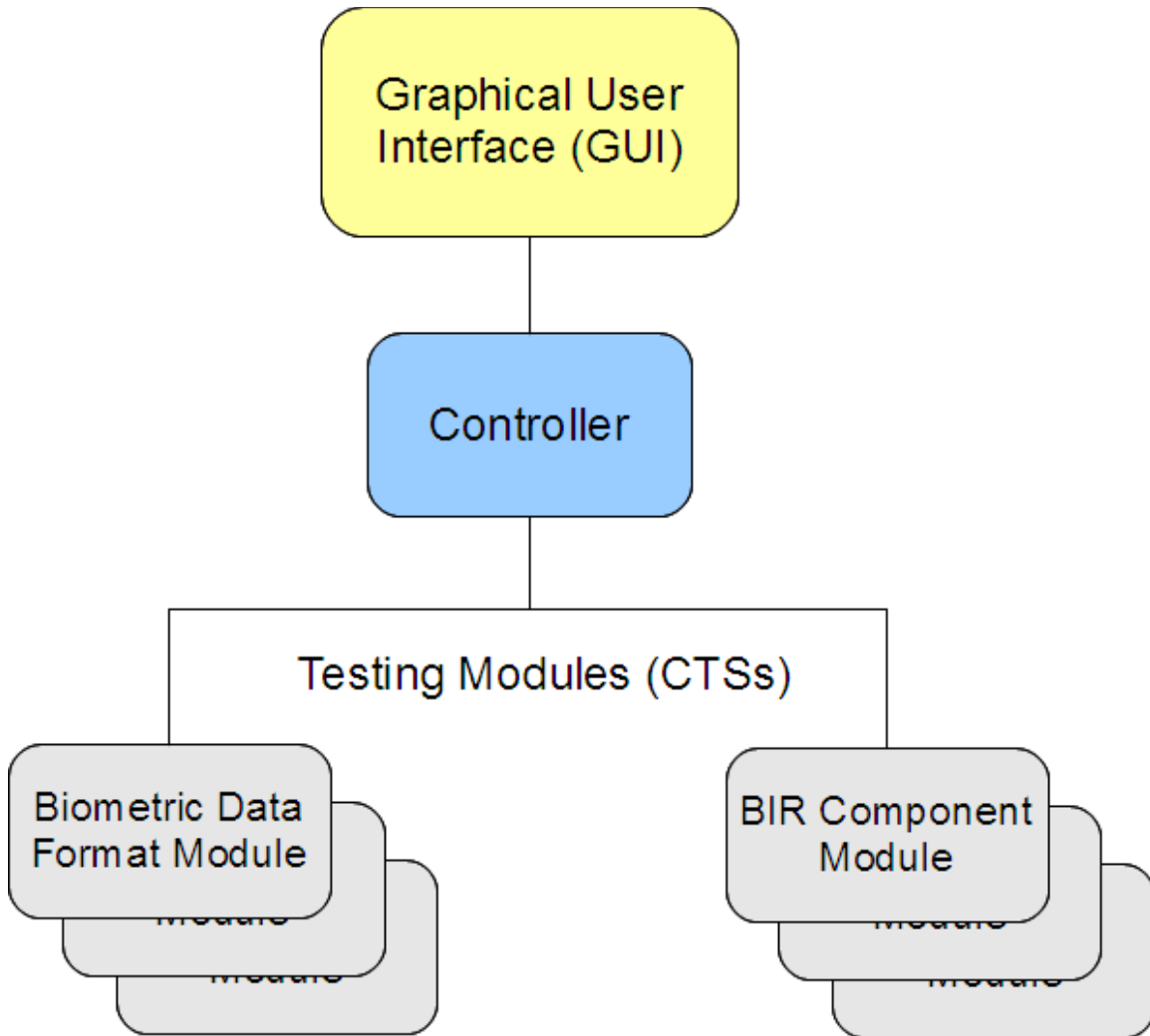
Figure 2-2: CTA High-Level Architecture

The architecture design supports conformance testing performed against any single Module or any combination of Modules to test BIR components. As indicated by the Figure above there can be multiple biometric data interchange format Modules as well as BIR component Modules. The Controller discovers and loads the Modules at runtime using an "Add-in" or "Plug-in" architecture similar to many other programs. The primary benefits of the dynamically-loaded Module architecture are simplified development and distribution:

− Simplified Development: The Modules are developed separately from the GUI and Controller. Significantly, no changes need to be made to the GUI or Controller to create a new Module or modify an existing Module.
− Simplified Distribution: The GUI and Controller can be installed independently of any Modules, which means installing new or updated Modules can be as simple as copying the Module to a folder.

The only requirement of a Module is that it be a .DLL that implements the IModule interface in the NIST_ITL.Modules namespace. The Modules are installed in the "Modules" subfolder.

2.3   GUI / Controller Communication Methods

The GUI and the Controller support two interfaces known as the *Local Controller* and the *Web Service Controller*. CTS Modules can be used on a local computer or they can be accessed on the Internet or an Intranet via Web Services. Advantages and disadvantages of the Web Service architecture versus the operation via the local controller are discussed in section 4. The recently released CTA Beta 2.0 supports only the local controller. The Web Services controller is under testing.

2.4   CTA Beta 2.0 Key Features

As discussed above, the CTA is a set of individual components that work together to perform conformance testing. Each component can be independently developed and tested. Users benefit from this approach since it allows for faster upgrades. Key features incorporated in the design of the architecture are discussed below.

2.4.1   Dynamically-Loaded Modules

The testing Modules are automatically loaded at runtime; they do not have to be "compiled into" or "linked into" the rest of the software. They can be developed and deployed without changing any other CTA software. A CTA user can simply download and install a new or upgraded testing Module without changing any other CTA software component.

2.4.2   Binary Data is in Context

When the CTA processes binary data, the first step is to place the data into an array of Field data structures. The primary benefit is that the users never have to interpret what the 27th (or 10,027th) byte is; the CTA shows which Field the byte belongs to. A Module's complexity is greatly reduced by the pre-parsed array of Fields. Fields are one of the keys that enable "Unit Testing for Data Formats Testing." Section 2.4.9 introduces the reader to a brief discussion on Unit Testing.

2.4.3   Fields Belong to Groups

The CTS Modules implement testing of groups of fields. Annex A- Structure Testing by Field Groups discusses this approach in detail and provides examples.

These are three typical rules for the Record Header group of fields:

− The Record Header must occur first.
− The Record Header can occur only once.
− The Record Header consists of these fields, in this order:
− Format Identifier, Version, Record Length, ...

Processing of fields by groups enhances CTS Module confidence and reliability by reducing software complexity. These are the primary reasons:

− The CTS Modules call the same software library for group processing. The library has been thoroughly tested. No CTS Module has to create its own software for processing fields by groups.

- Groups are defined in XML, not by hard-coded values typed by hand into a CTS Module. The XML format is far superior to hard-coded values for editing, proofreading, and transformation of the group information into other useful forms.
- The part of the CTS Module that determines if the set of groups forms a valid data structure is very small and clear.
- Once the structure has been determined valid, many level 2 assertions[2] become trivial to write and test.

Additional information on this approach is discussed in Annex A.

2.4.4   Additional Data

The CTA supports optional auxiliary binary information called "Auxiliary Data". The purpose of Additional Data is to allow a field to pass that would normally fail under certain conditions. For example, some Modules are designed to test the content of the CBEFF Product Identifier field. Some standards state that CBEFF Product Identifier codes can be registered with the International Biometric Industry Association (IBIA) at www.ibia.org. The CTA Modules are released with a version of the codes from IBIA.   During testing, if the CBEFF Product Identifier field has a code that is not on the list, the test fails. However, if the value matches exactly the value in Additional Data, the test passes. This allows tests to pass where the CBEFF Product Identifier code has not yet been registered, was registered after the release of the Module, or is strictly used for internal purposes and will not be registered. Additional Data is specified at the GUI level but acted upon at the Module level. The Module developer implements and documents the exact meaning of Additional Data for each field in which it is used. In existing Modules Additional Data is ignored for most fields and does not affect the processing at all.

2.4.5   XML Manifests

A Manifest defines the format to which a binary file conforms. It is saved in XML format. In testing, any number of binary files can be tested against a specific Manifest. When the format of a binary implementation of a standard is known in detail, the user provides the CTS that will be used to test the binary file(s) with information on the format via a Manifest. The CTA accepts Manifests in XML and provides the means for developing them. The Fields with all their information (Bytes, Lengths, Comments, etc.) can be saved in a Manifest. In addition to the Fields, the Manifest has additional information such as the Manifest's Name, a Description, the Module(s) used, etc. The Manifests can be easily shared and displayed. Since a Manifest defines a format, binary data is more easily analyzed and understood. Table 2-1 depicts binary data displayed in a hex editor.

---

[2] In Level 1 testing, an implementation is checked field-by-field, for conformance with the specification of the standard, both in terms of ranges, character types, encoding and cardinality. In Level 2 testing, an implementation is checked to determine if it is internally consistent. This is achieved by relating values from one or more fields or information items within an implementation to other values within the same implementation. Level 2 tests require more complex testing than Level 1, usually after the entire implementation has been parsed.

Table 2-1: Binary data displayed in a hex editor

```
Address   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
00000000  46 4d 52 00 20 32 30 00 01 4c 00 42 00 11 00 b5  FMR. 20..L.B...µ
00000010  02 00 02 00 00 c5 00 c5 02 00 07 00 5a 1b 40 64  .....Å.Å....Z.@d
00000020  00 0e 50 5a 40 a4 00 11 3c 50 80 37 00 12 10 5a  ..PZ@¤..<P€7...Z
00000030  80 4a 00 16 36 3c 40 70 00 16 40 50 80 2a 00 1f  €J..6<@p..@P€*..
00000040  1f 5a 80 93 00 23 24 5a 40 58 00 26 75 28 80 2b  .Z€".#$Z@X.&u(€+
00000050  00 2a 03 50 40 38 00 30 17 46 40 84 00 31 33 5a  .*.P@8.0.F@„.13Z
00000060  80 47 00 32 2f 50 00 5f 00 33 3a 5a 40 70 00 35  €G.2/P._.3:Z@p.5
00000070  5e 32 80 87 00 3a 17 50 00 29 00 3c 2a 46 80 43  ^2€‡.:.P.).<*F€C
00000080  00 3e 67 5a 40 5b 00 3f 5e 50 40 70 00 41 17 3c  .>gZ@[.?^P@p.A.<
00000090  40 35 00 47 20 5a 80 68 00 4a 09 50 40 4b 00 4f  @5.G Z€h.J.P@K.O
000000a0  0f 5a 80 30 00 50 41 5a 40 82 00 59 20 50 80 3f  .Z€0.PAZ@‚.Y P€?
000000b0  00 5f 5a 50 40 2f 00 6c 75 5a 80 7e 00 73 7a 1e  ._ZP@/.luZ€~.sz.
000000c0  00 00 02 00 46 16 40 28 00 5d 00 5a 80 74 00 64  ....F.@(.].Z€t.d
000000d0  00 50 40 52 00 5f 09 46 80 8c 00 71 0b 46 40 7a  .P@R._.F€Œ.q.F@z
000000e0  00 87 0d 50 80 37 00 48 0f 32 40 5e 00 4a 11 3c  .‡.P€7.H.2@^.J.<
000000f0  40 9b 00 3e 1e 50 80 2a 00 40 27 46 40 9b 00 55  @›.>.P€*.@'F@›.U
```

Table 2-2 shows a screenshot of the same data, saved in a Manifest and displayed in Altova's XMLSpy®. The binary data, shown in the last column, is associated with a Field Name, shown in the "= Name" column.

Table 2-2: Binary data displayed in Altova's XMLSpy®

	= Sequence	= Name	= DefinedLength	= ActualLength	= BytesHex
1	1	01_FormatIdentifier	4	4	46 4d 52 00
2	2	02_Version	4	4	20 32 30 00
3	3	03_RecordLength	2	2	01 4c
4	4	04_CbeffPidOwner	2	2	00 42
5	5	05_CbeffPidType	2	2	00 11
6	6	06_07_CaptureEqmtCompAndID	2	2	00 b5
7	7	08_ImageSizeX	2	2	02 00
8	8	09_ImageSizeY	2	2	02 00
9	9	10_ResolutionX	2	2	00 c5
10	10	11_ResolutionY	2	2	00 c5
11	11	12_NumFingerViews	1	1	02
12	12	13_Reserved	1	1	00

Table 2-3 shows additional data shown in the XMLSpy® screenshot, this additional information included in the Manifest is useful too. Note that the exact versions of the Modules that created the Manifest are saved in the Manifest.

Table 2-3: Additional data included in the Manifest

Manifest	
= Name	INCITS 378-2004 Sample No X Data
= Description	See Annex B, No 2nd Private Data
= SBH	(None)
= SBHversion	
= BDB	INCITS 378-2004 Finger Minutiae
= BDBversion	1.0.0.41440
= SB	(None)
= SBversion	

2.4.6   Decode

When the structure of a binary implementation of a standard is not known, a Manifest to describe the binary input to the CTS cannot be developed. In this case, Manifest-Based testing cannot be performed. The CTS develops the related Manifest based on decoding the input binary file. This is called Decode-Based binary testing. The structure of the data is determined as the input bytes are processed; there is no external template, schema or Manifest that describes the structure.

2.4.7   Test Cases

A CTA Test Case encapsulates both the data for and the expected results of a testing operation. When saved the Test Case is stored in XML format. A Test Case can test, for example, "This list of 10 fields should have these 4 specific errors." When the Test Case is tested, if 4 errors are found, and they exactly match the 4 expected errors, the GUI indicates the Test Case succeeded.

"I am expecting to find these 4 errors. These same 4 errors were found during testing. Since I found what I was looking for, proof that the Module finds the errors it should, the Test Case passes."

Well designed Test Cases are a critical tool to test the reliability of a CTS. They are primarily used by CTS Module developers. A CTA/CTS user may never need to use a Test Case. Test Cases are very similar to Manifests; the chief difference between them is their purpose. Generally speaking, CTA/CTS users use Manifests to perform data testing, whereas CTS Module developers use Test Cases to ensure the correct operation of their Modules. Manifests and Test Cases contain the format specification of binary data in a convenient form for storage, sharing and analysis. For testing, their data is converted by the GUI into the array of Fields and supporting information that is passed to the Controller. When testing is complete the results are stored in the Manifest or Test Case. Depending on the specific GUI operation, the updated Manifest or Test Case can optionally be saved.

To reduce CTS software complexity, Manifests and Test Cases have an identical format. In fact, a Manifest is just a "Complete" Test Case that expects to "Pass." For the convenience of CTA/CTS users, the GUI defaults to working with Manifests and Test Cases are different at the GUI and documentation level, but they have the same XML structure. A more detailed discussion on Test Cases can be found in Annex B.

2.4.8   Multiple Errors

To minimize users' time and work in correcting problems, the CTA supports Modules returning multiple errors from a single test.

2.4.9   Approaches to Increase Confidence and Reliability

To trust a software program, it is not enough to know that the software returns good results when presented with good data. The Module developer and the user also need to know that the software behaves as expected when it is presented with erroneous data. Perhaps the single greatest improvement in software reliability in the past 30 years has come from the use of Unit Testing. These tests verify the correct operation of the product under both good and bad conditions. When a bug is discovered it is common practice to first write a unit test that

duplicates the error, then fix the error, and then add the updated unit test to the product's suite of tests to ensure the bug never occurs again. By using both good software development methods and the Beta 2.0 CTA's Test Case capability, the software developers have a great deal of confidence that the Modules they create will respond appropriately to both good and bad data. The Beta 2.0 CTA brings Unit Testing to data formats testing.

3   Introduction to Testing Binary Data

As stated in section 2, testing binary data can be performed using a Manifest that describes the format of the binary data or by decoding the binary data.

3.1   Testing Binary Data – Manifest-Based

In this common GUI operation a user elects to test one or more binary files with one Manifest. The online help tool or the Users' Guide that can be downloaded from the NIST/ITL CTA Beta 2.0 release web page [8] provides detailed information on how to perform binary testing. The process within the GUI is as follows:

1. The Manifest is loaded in memory.
2. The binary file is loaded into the Manifest as illustrated in section 3.3.
3. The Manifest is passed to the Controller for Testing. See the High Level Sequence Diagrams in Annex C.
4. When the results are received back from the Controller, the GUI is updated. If selected by the user, the results are added to the Log and/or Report.
5. If there are more binary files, the process continues with step 2. If there are no more binary files the operation is complete.

3.2   Testing Binary Data - Decode-Based

This common GUI operation is very similar to the Manifest-Based Binary Data Testing process discussed above. The difference is that each binary file creates its own Manifest. These are the steps:

1. The binary file is loaded into memory and passed to the Controller for Decoding. See the High Level Sequence diagrams in Annex C.
2. The Controller returns the array of Field data structures created by the CTS Module "Decode" operation.
3. The array of fields is passed back to the Controller for Testing.
4. When the test results are received back from the Controller the GUI is updated. If selected by the user, the results are added to the Log and/or Report.
5. If there are more binary files, the process continues with step 1. If there are no more binary files, the operation is complete.

3.3   Loading Binary Data into a Manifest

Loading binary data into a Manifest happens frequently in GUI operations. These are the steps:

1. The Manifest is loaded into memory. For simplicity, the sample Manifest shown in Table 3-1 below has three fields and a total of six expected bytes. The Manifest initially looks like this.

Table 3-1: Initial Content of the Sample Manifest

Field	Length		Data (Hex)
	Expected	Actual	
Field_01	2	2	FF 07
Field_02	1	1	03
Field_03	3	3	C0 00 15

2. Next, the Manifest's data is erased. The cells highlighted in Table 3-2 (Actual Length) indicate length errors that would be identified by Testing.

Table 3-2: Sample Manifest Showing Data Erased

Field	Length		Data (Hex)
	Expected	Actual	
Field_01	2	0	
Field_02	1	0	
Field_03	3	0	

3. Next, as shown in Table 3-3, the input binary data is applied to the memory Manifest, filling each field completely with bytes before going to the next field. If there are too few bytes, one or more fields are not filled to capacity.

Table 3-3: Input Binary Data Applied to the Sample Manifest

Input: 01 02 03 04 (4 Bytes Total)

Field	Length		Data (Hex)
	Expected	Actual	
Field_01	2	2	01 02
Field_02	1	1	03
Field_03	3	1	04

As shown in Table 3-4, if the binary input is longer than expected, the excess data is in the last field and the actual field length is highlighted in yellow.

Table 3-4: Input Binary Data Longer Than Expected

Input: 01 02 03 04 05 06 07 08 (8 Bytes Total)

Field	Length		Data (Hex)
	Expected	Actual	
Field_01	2	2	01 02
Field_02	1	1	03
Field_03	3	5	04 05 06 07 08

When the length of the binary input data exactly matches the Manifest's total expected length, each field has the expected number of bytes as shown in Table 3-5 below.

Table 3-5: Input Binary Data Length Matches the Manifest's Total expected Length

Input: 01 02 03 04 05 06 (6 Bytes Total)

Field	Length		Data (Hex)
	Expected	Actual	
Field_01	2	2	01 02
Field_02	1	1	03
Field_03	3	3	04 05 06

The memory-Manifest is loaded with new binary data, ready for the next part of the GUI operation.

3.4   Field-Based vs. Binary-Input-Only Testing

The Fields data structure and its persistence as an XML Manifest bring a high level of understanding and documentation to binary data. This organization of binary data means the CTA operates somewhat differently from a binary-data-only testing approach where data input is just an array of bytes.

3.4.1 Separate "Decode" Operation

In binary-input-only testing, the structure of the data is determined as the input bytes are processed; there is no external template, schema or Manifest that describes the structure. In the CTA, with its Fields data structure architecture, determining the structure is called Decoding and it is a separate operation from Testing. (Decoding is discussed in section 2.4.6.) Most of the time when Decode is called in user testing, it is immediately followed by a Testing call.

3.4.2 Field-Length Testing

During binary-input-only testing, if the first field is two bytes long, the first two bytes of the input are the first field's data. There is no other interpretation.

In Field-Based testing, the first field is a Field data structure. It has Length and Data components. In the CTA, each Field's Length is verified. There is no analogous concept in binary-input-only testing. This is not as inconvenient as it may sound because of the intelligence built into the GUI. For example, users cannot change the length of a fixed-length field.

4 Characteristics of NIST/ITL CSD CTA Local and Web Services Controllers

4.1 Local Controller

Figure 4-1 depicts a graph of the Local Controller interface. In this configuration the GUI, Controller and Modules are all on the same computer and communicate in the simplest, fastest method.
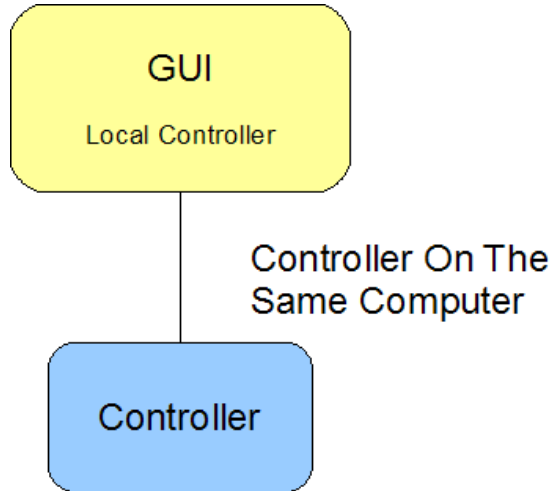
Figure 4-1: CTA's Local Controller

4.2 Web Services Controller

The recently released CTA Beta 2.0 supports only the local controller. The Web Services controller is under testing. Figure 4-2 shows the Web Services Controller configuration. In this configuration, the GUI and Controller use the Simple Object Access Protocol (SOAP) standard to exchange data. This has more overhead than Local Controller communication, but little difference in speed was noted during initial testing when the Web Services Controller was installed on the same PC as the GUI.
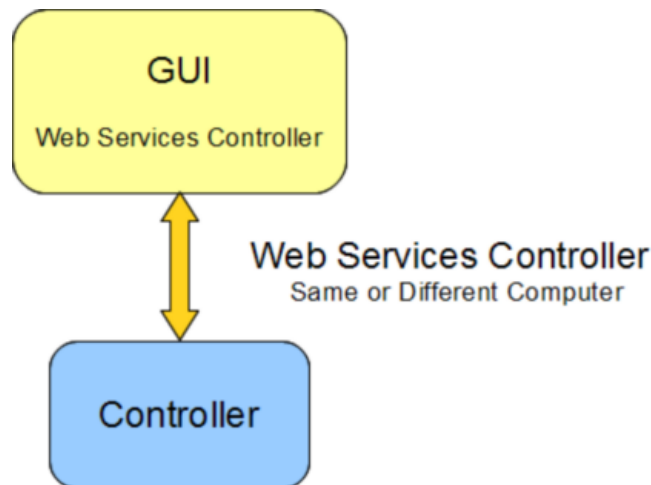
Figure 4-2: CTA's Web Services Controller

Not all operating systems support self-hosting of web services. For example, a Web Services Controller can be installed and run on Microsoft® Windows XP Professional, but not on Microsoft® Windows XP Home.

4.3 Local and Web Services Controllers

As shown in Figure 4-3, the GUI can be configured to support both communication methods, but only one can be active at a time.
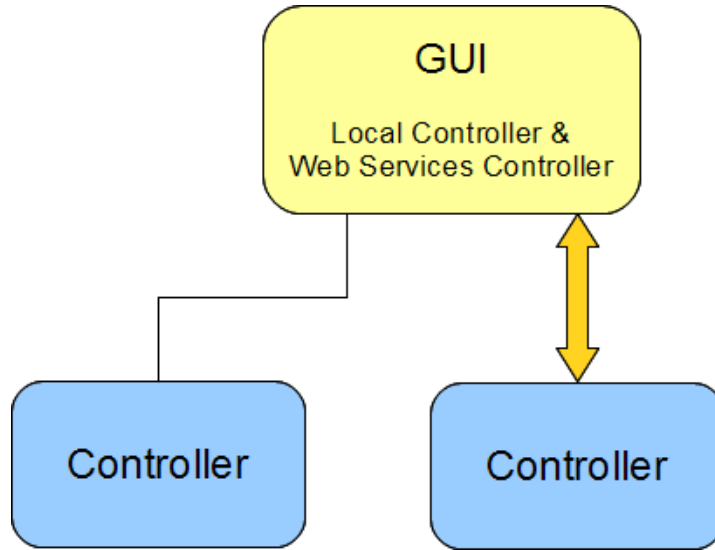
Figure 4-3: CTA's Local and Web Services Controller Configuration

4.4 Multiple Web Services Controllers

Finally, since Web Services are specified by a Uniform Resource Identifier (URI), similar to a URL, the GUI that supports the Web Services Controller can connect to many different Web Services Controllers, one at a time.
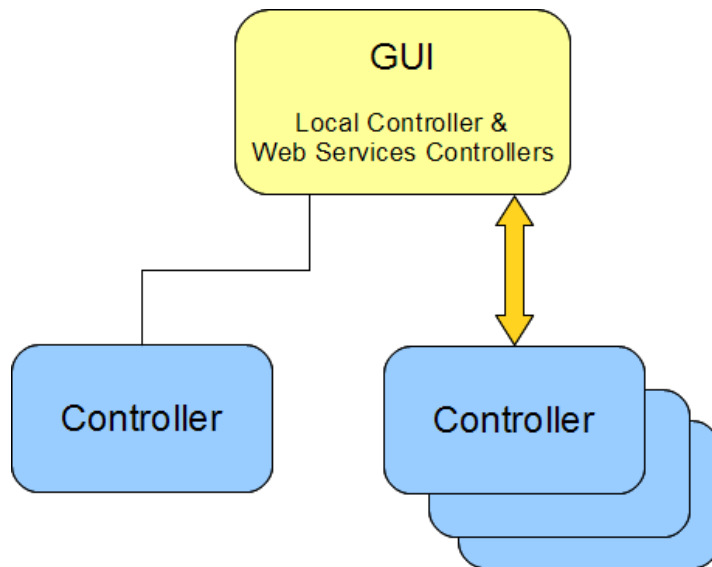
Figure 4-4: CTA's Local and Web Services Controller Configuration

4.5 Advantages and Disadvantages of the Web Services Architecture

4.5.1 Advantages:

There are three key advantages to implementing and using a Web Services architecture instead of using a local controller:

- **Simple PC Configuration and Maintenance**: Only the GUI needs to be installed on a user's PC. An organization can be responsible for maintaining the server(s) which have the Controller and Modules. In this architecture, upgrading every user to a new version of a specific Module is just a matter of updating the server, not each individual PC.

- **Testing Flexibility**: Since the GUI can connect with many different Web Services Controllers, specific Web Services Controllers can be established for different purposes. For example, Module developers may establish several Web Services Controllers that demonstrate the capabilities of their Module at various points in its development.

- **Enhanced Intellectual Property (IP) Protection**: If an organization develops a Module that contains IP, they may not wish to release the Module for installation on PCs, as there is little or no protection against a sufficiently-motivated individual disassembling their Module. If they make the Module available only from a Web Services Controller installed on their own secure server(s), the IP is much better protected.

4.5.2 Disadvantages

- **Connectivity:** If the Web Services Controller is located on a separate machine from the GUI, a network connection is required between the GUI computer and the Web Services Controller computer.

- **Potentially Slower Testing**: In NIST/ITL CSD's initial testing there was a negligible speed difference between the Local Controller configuration and the Web Services Controller configuration when the Web Service Controller was installed on the same computer as the GUI. Network conditions and Internet speed will be factors in execution speed.

Annex A- Structure Testing by Field Groups

A.1 Testing of Groups of Fields

The CTS Modules implement testing of groups of fields. Examples of these groups in biometric data interchange formats include a Record Header and a Finger View Header or a General Header and a Finger Minutiae Header (Finger Header). Processing of fields by groups reduces complexity. The main reasons for using this approach were detailed in CTA Beta 2.0 Key Features in Subsection 2.4.3. To better illustrate this concept, a simple finger minutiae standard and its field groups is described. Though the format is fictional, it highlights key points of actual standards and testing of their implementations with CTS Modules.

A.1.1 Simple Finger Minutiae Format

A simple finger minutiae standards' structure is a Record Header followed by one or more Finger Records. A Finger Record consists of a Finger Header followed by zero or more sets of (X, Y) Minutiae Points that make up a Minutiae Data Group. Table A-1 shows the field names, their abbreviations, and the group to which they belong.

Table A-1: Sample Manifest Showing Data Erased

Field Name	Abbrev.	Group
Format Identifier	FmtID	Record Header
Version	Ver	"
Record Length	RecLen	"
Number of Fingers	# Fng	"
Image Size X	SizeX	Finger Header
Image Size Y	SizeY	"
Number of Minutiae	# Min	"
Minutiae Position X	MinX	Minutiae Data
Minutiae Position Y	MinY	"

As shown in Table A-2, this is a sample record of three fingers with three, one and two Minutiae Point pairs each. Fields that are not important for this discussion are blank.

Table A-2: Sample Record Containing Three Finger Representations

Record Header			
FmtID	Ver	RecLen	# Fng
	2		3

Finger Header 1		
SizeX	SizeY	# Min
		3

MinX	MinY	
		(Minutiae Data Group 1)
		(Minutiae Data Group 2)
		(Minutiae Data Group 3)

Finger Header 2		
SizeX	SizeY	# Min
		1

MinX	MinY	
		(Minutiae Data Group 1)

Finger Header 3		
SizeX	SizeY	# Min
		2

MinX	MinY	
		(Minutiae Data Group 1)
		(Minutiae Data Group 2)
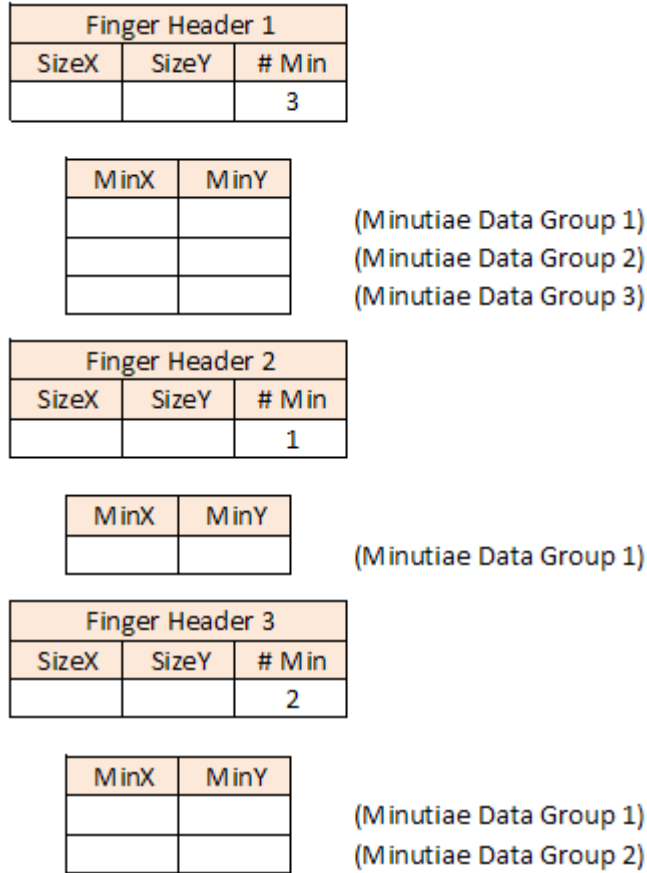
The higher level of abstraction of groups vs. fields has the potential for cleaner, more trustworthy code. Using table-driven logic, the software to test the record should begin with the Record Header. This is much simpler than field-by-field testing:

The first field should be the Format Identifier.
The next field should be the Version.
The next field should be …

This is especially true when the number of fields in a group, say the Minutiae Data group, varies based on a value in a header field. Some groups could have just (X, Y) sets but others may also have angle and quality information. In real-world Modules, some groups may have two fields and others five in different parts of the record.

The pseudo code below is for testing the Sample format record using Field Groups.

```
Check RecordHeader, Exit if bad
Read NumberOfFingers
ForEach NumberOfFingers
{
    Check FingerHeader, Exit if bad
    Read NumberOfMinutiae
    ForEach NumberOfMinutiae
        Check MinutiaeDataGroup, Exit if bad
}
```

Flag any remaining fields as Unexpected.

The same structure testing can be done using just the fields, but many more lines of code are needed.

A.1.2 Post Valid-Structure Processing

An exciting consequence of separate structure testing is that many other tests become easy to do as a separate pass once the structure is known to be good. Consider testing that every MinX is in range of its SizeX value. If the record structure is known to be good then there are no "orphan MinX" fields; each MinX field "belongs" to a Finger Header that has the SizeX field that applies to the MinX field. The algorithm for SizeX/MinX testing then becomes very simple as illustrated by this pseudo code.

```
ForEach Field
{
        If Field is SizeX
                Read ValueSizeX
        If Field is MinX
        {
                Read ValueMinX
                If ValueMinX >= ValueSizeX
                        Indicate Error
        }
}
```

In the following example a MinX value of 125 occurs twice. The first occurrence is an error because it is larger than its corresponding SizeX value. The second occurrence is valid.

Table A-3: Sample Record Showing an Error During MinX Values

Record Header			
FmtID	Ver	RecLen	# Fng
	2		2

Finger Header 1		
SizeX	SizeY	# Min
100		3

MinX	MinY	
125		Error
50		
75		

Finger Header 2		
SizeX	SizeY	# Min
200		3

MinX	MinY
125	
50	
75	

This is a very simple, made-up format with limited example potential. The principle is valid. In real-world Modules, there are tests that are much easier to perform if the record's structure is known to be good.

Annex B. More About Test Cases

B.1 Background

CTA Beta 2.0 introduces new, more powerful Test Case features that allow more testing with less effort. This Annex provides information primarily for Module Developers, but it is included in this document because some CTA users may find that the information helps them understand the CTA and associated Conformance Test Suites better.

Perhaps a made-up scenario will make this section as short and clear as possible. Suppose a Module is being written for a standard that has this text:

> *The name of the second field shall be "Foo". It shall be two bytes long and shall have the value 3 (0x 00 03).*

Ideally, all possible values should be tested, but realistically that cannot be done for non-trivial test Modules. One quickly encounters the "More tests are needed than electrons in the universe" combinatorial condition where the required testing time is measured in geologic eras.

Choosing test values is therefore somewhat of an art in which choosing "values likely to fail" is very important. The following are the required Foo value tests, the minimal set of tests necessary to gain confidence that the Module is very likely to handle Foo values properly:

1. Three (3). The expected value must pass.

2. Two (2) and four (4). It is important to test values just outside of the range of valid values to make sure they fail.

3. No-bits and all-bits (0x 00 00 and 0x FF FF). Boundaries, the edge conditions that overflow, underflow and signed-vs-unsigned considerations potentially raise havoc. Boundary-bits values should fail.

The 5 test values (3, 2, 4, 0x 00 00 and 0x FF FF) are sufficient to test this feature, provided that a competent programmer wrote software that is *intended* to work. Additional strategies are needed to protect against malicious code.

Therefore, a minimum of five values should be tested in this example: 3, 2, 4, 0x 00 00 and 0x FF FF. Only one value (3) should pass and the rest should fail. Prior to CTA Beta 2.0, these five test values required five different, complete Test Cases. They could not be combined in a single Test Case because they required a complete record. Over time the following problems were noted.

1. Developing Test Cases was a slow process. Each Test Case had to pass all the Module's tests with the exception of the specific error desired (e.g., having all fields passed but Format ID would be a failing Test Case for Format ID). Therefore, each Test Case was essentially a complete record, with at least the minimum number of fields.

2. Test Case maintenance was an issue. Each Test Case had to pass all Module tests or all-but-one test if it was testing for failure. Since the Module and its Test Cases were developed side by side as the project progressed, it was fairly common for a new test in the Module to break a number of existing Test Cases.

3. Test Cases were obscure. Since the one field in error was just another field in a list of 20, 30 or 50 fields, it did not stand out in any way. The purpose of the Test Case could not be determined by superficial examination.

4. At one point there were so many Test Cases developed for a Module that they had to be put in separate folders because the standard Windows® "Open File" dialog box, in Microsoft® Windows® XP Professional, could not open all of them at the same time.

5. There were so many Test Cases that there was a "Cannot see the forest for the trees" problem. It was hard to look at the list of Test Cases and determine "Test Coverage", that is, whether or not the existing Test Cases tested every desired test scenario.

B.2 Characteristics of the Test Cases Implemented in CTA Beta 2.0

The Modules supported by the previous CTA version were "internally modular." That is, the code that tested that Foo=3 was separate from the code that checked that Foo was the 2nd field, which was itself separate from code that made sure the Foo field only occurred once. Different parts of the code tested different characteristics of the data. Test Cases had always run all tests in the module, but changes could be made to allow the Module to run only some of its tests, and to allow the passing of a "flag value" to the Module to determine which code paths to run and which tests to perform.

In this CTA version, each module has associated Test Cases that "prove" that they work with a wide variety of data. The Test Cases allowed the developers to, if you will, "hammer loud and long" on the code, thoroughly chasing away all bugs. If and when more bugs were found, one or more Test Cases were created to show they have been fixed. These errors were not expected to reappear since all Test Cases were run before a Module was released.

Continuing the Foo scenario, it is suggested that Module developers support the "Data Only" type of Test Case testing, and implement Modules in such as way as to only call the value-testing parts of the code. In "Data Only" testing the code that makes sure Foo is the second field or the code that makes sure Foo occurs but once are not addressed in the Test Case (this tests data only without testing the relationship between fields and the structure). These are tested under testing types such as "Complete", "Structure", or "Partial". Just fields' data values are tested and all the structural considerations are ignored. As shown in Table B-1, doing this allows a *single* Test Case to test all five values discussed above.

Table B-1 – Test Case Results

Test Number	Field Name	Value	Expected Test Result
1	Foo	3	Pass
2	Foo	2	Fail
3	Foo	4	Fail

Test Number	Field Name	Value	Expected Test Result
4	Foo	0x 00 00	Fail
5	Foo	0x FF FF	Fail

When the Module returns these results, the developer can save this Test Case so that it "Passes" whenever it is tested and the Module returns these exact errors. The Test Case can be run anytime during the Module's lifetime to ensure Foo's value is handled correctly by the Module.

A real worth of Test Case Types is in limiting the Errors returned to those of interest.

It is suggested that Module developers adopt a Test Case naming convention that conveys maximum understanding by simply looking at the Test Case names. Below is a partial list of the Test Cases developed for a NIST/ITL CTS Module. Though there are fewer than 100 Test Cases, confidence is high that the Module performs as expected.

```
L1-02-PassFail-Format Identifier.xml³
L1-02-PassFail-Version.xml
L1-03-L2-PassFail-Record Length.xml
L1-04-PassFail-CBEFF PID Owner.xml
L1-05-Pass-CBEFF PID Type.xml
L1-06-07-Pass-CaptureEquipment.xml
L1-08-Pass-ImageSizeX.xml
L1-09-Pass-ImageSizeY.xml
...
L1-42-PassFail-Delta Angle 1.xml
L1-43-PassFail-Delta Angle 2.xml
L1-44-PassFail-Delta Angle 3.xml

L2-CoreDelta-01 Fail No XData Area.xml
L2-CoreDelta-02 Fail Not XDAType.xml
L2-CoreDelta-03 Fail  XDAType Not CoreDelta.xml
L2-CoreDelta-04 Fail Not CoreInfType_Num.xml
...
L2-RidgeCount-01 Fail No XData Area.xml
L2-RidgeCount-02 Fail Not XDAType.xml
L2-RidgeCount-03 Fail Not RidgeCount.xml
L2-RidgeCount-04 Fail Missing XDAL.xml
...
L2-XDLen 04 - Orphan XDAL.xml
L2-XDLen 05 - Len Err.xml
L2-XDLen 06 - Pass Multiple XDALs.xml
L2-XDLen 07 - Zero len XDAL Passes.xml
L2-XDLen 08 - Zero len XDAL Orphan.xml
```

---

[3] PassFail" represents a mixture of passes and fails.

Level 1 tests are named, "L1 - FieldNumber(s) – Results – FieldName".
Level 2 tests are named, "L2 - FieldsGroup – Counter Other-Information."
The Test Case "L1-03-L2-PassFail-Record Length.xml" is an exception. It is both possible and desirable for the field Record Length to perform both Level 1 testing ("Is the value in the allowable range?") and Level 2 testing ("Does the value match the number of bytes read?") at the same time in the same block of code. The Test Case name, therefore, reflects both L1 and L2 testing.

In summary, CTA Beta 2.0's Test Cases provide an excellent mechanism for thoroughly testing CTS Modules.

Annex C - High Level Sequence Diagrams

Three operations involve all the CTA components. They are initialization, testing and decoding.

C.1 Initialization

The diagram in Figure C-1 illustrates the initialization process.
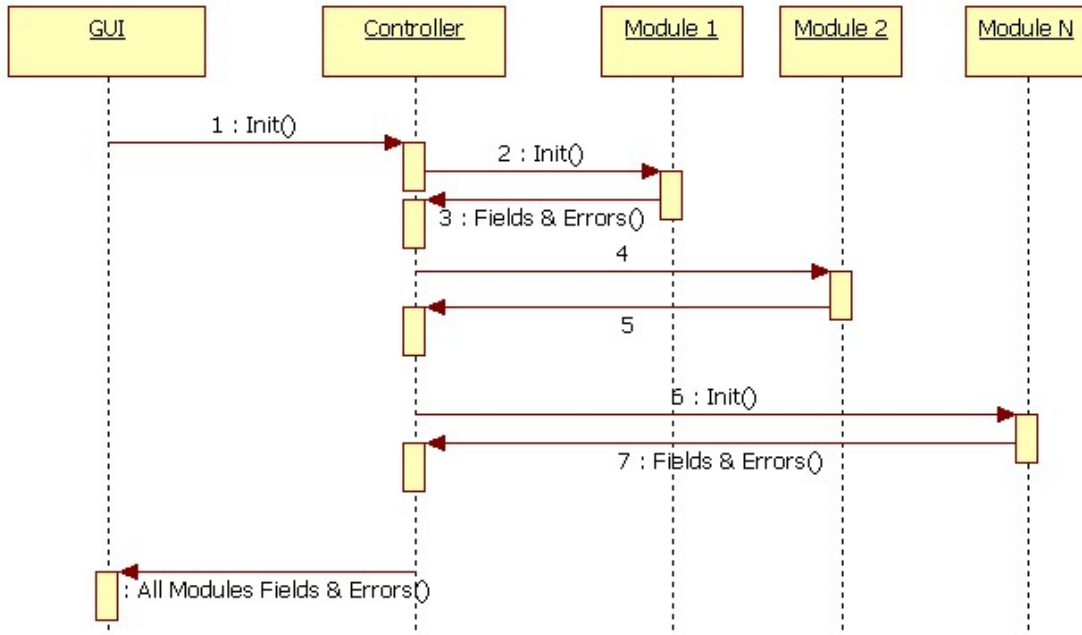

Figure C-1: Initialization Process

Note: "4" and "5" are the same as "2" and "3".

The GUI initializes the Controller, expecting to receive information about all available Modules. The controller loads each Module it finds, receiving from each Module the Module's type, its fields and the list of errors it can find as part of its testing. The Controller consolidates this information, returning to the GUI a list of all loaded Modules, their fields and their errors.

C.2 Testing

The diagram in Figure C-2 illustrates the testing process. This is the "behind the scenes" process called by a number of separate GUI operations: testing binary files with a Manifest, testing a Manifest or Test Case individually on the "Builder Tab", and testing a number of Test Cases at the same time.

The Controller gets the request from the GUI and calls up to three modules (i.e., SBH, BDB SB). See Annex D, More on Biometric Information Records. The Controller gets the list of

errors from each module, consolidates the errors into a single Error List and returns it to the GUI. When the GUI receives the results, it interprets the Error List and informs the user. For example, an Error List with four items would indicate failure of a Manifest + Binary File test, but it could be a success if the test was a Test Case and the four errors matched the expected errors.
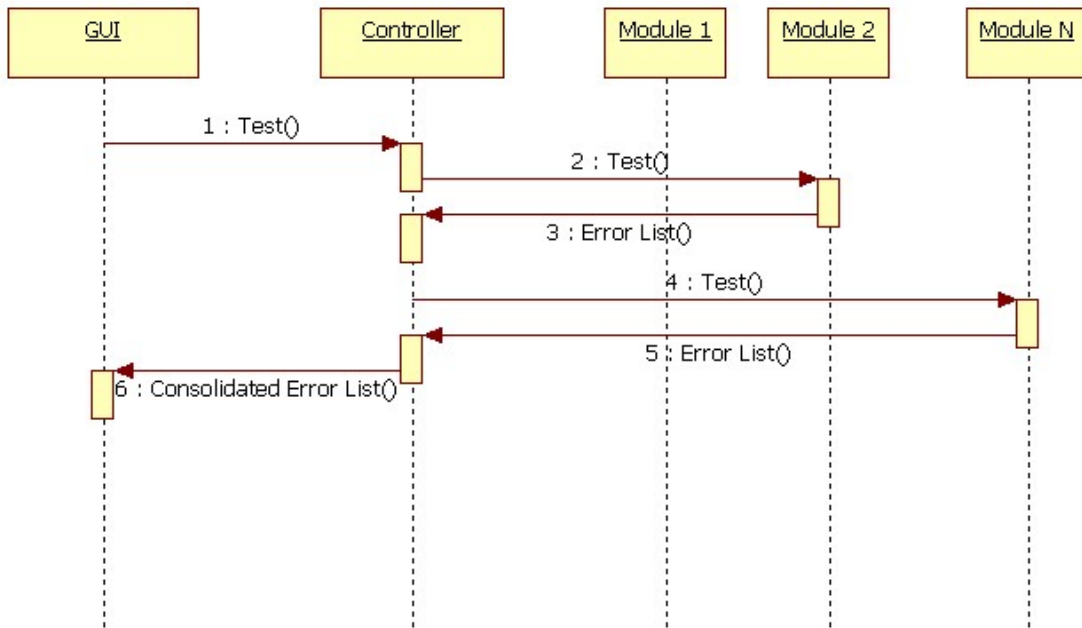

Figure C-2: The Testing Process

C.3 Decoding

As stated above, Decoding refers to an analysis operation performed by a Module. The input is binary (an array of bytes) and the output is an array of Fields that correspond to the binary input.

In most GUI operations the Decode operation is immediately followed by a Test operation, without any further user intervention. Put another way, to Decode a binary file is to develop or discover its structure.

The diagram in Figure C-3 illustrates the decoding process. The GUI sends the Controller an array of bytes. The Controller calls the proper Module, passing it the array of bytes. (CTA Beta 2.0 calls one Module as part of a Decode operation.) The Module passes back to the controller the array of Fields as determined by specific values in the byte array. The Controller returns the array of Fields to the GUI. For most operations in the GUI, the GUI immediately calls the Controller to perform a Test operation on that same array of Fields.
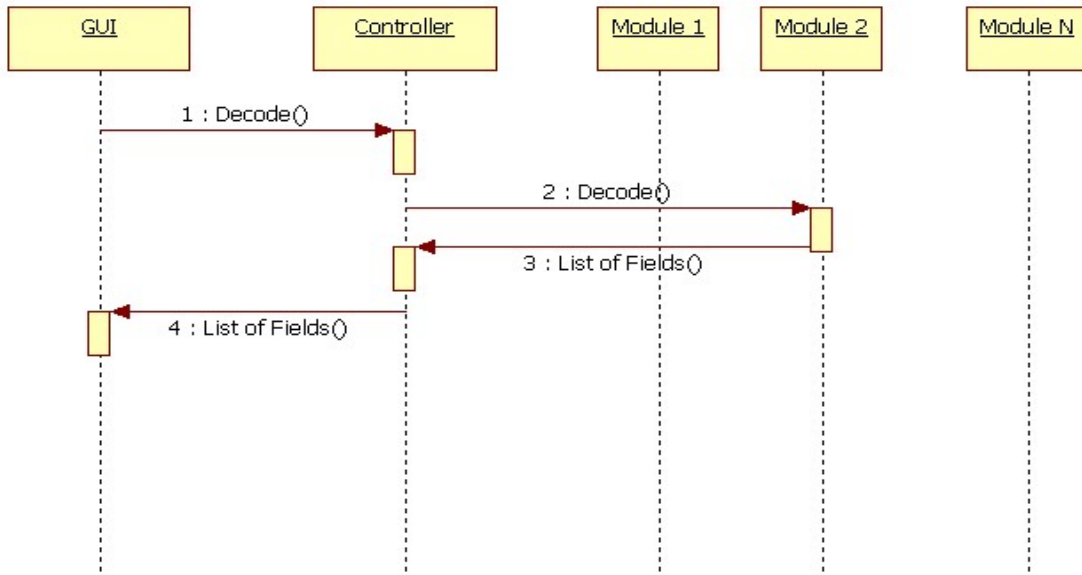
Figure C-3: The Decoding Process

Annex D - More on Biometric Information Records

The Common Biometric Exchange Formats Framework (CBEFF) standards describe a set of data elements and values to support biometric data records in a common way. These data elements can be encoded in a record header that facilitates exchange of biometric data between different system components or between systems. The CBEFF standards provide forward compatibility for technology improvements and allow creation of new patron formats (instantiations of CBEFF). The BIR structure defined by CBEFF supports security (digital signatures and data encryption), processing information and descriptive attributes of the biometric data.

The CBEFF record defines three components in a single structure:

1. The SBH (Standard Biometric Header) includes the data elements (required and optional) that describe the biometric data's structure and selected attributes (metadata) of its content. These include the appropriate standardized data elements and values defined by CBEFF, and may also include data elements and values defined by a patron format specification to satisfy unique requirements.

2. The BDB (Biometric Data Block) contains the biometric data. It can contain processed or unprocessed data. A BDB typically conforms to a standardized data interchange format such as the formats developed by INCITS M1 [2] or JTC 1/SC 37 [5] but could – alternatively – contain a proprietary data structure. Fields in the SBH specifically identify the BDB format and other metadata so the biometric application can pass the BIR or the BDB content to the correct biometric processing module.

3. The SB (Security Block) can contain a digital signature, message authentication code (MAC), or algorithm identifier information and parameters required for applying integrity and encryption processes to the BDB. CBEFF requires that the SBH not be encrypted, although it can be covered by integrity algorithms. As with the BDB format, the SB format is defined by an SB "owner" and can be adapted to the specific requirements of an application or system. Fields in the SBH identify the SB format to guide integrity checking of the entire BIR and to support appropriate decryption of the BDB.

As discussed in the introduction, CTA Beta 2.0 supports CTSs to test implementations of biometric data interchange data formats as well as implementations of the three components of CBEFF Biometric Information Records (BIRs).

Annex E- References

[1] INCITS M1 – *Biometrics* Public Homepage -
http://standards.incits.org/a/public/group/m1

[2] Published Biometric American National Standards Developed by INCITS M1 as of
September 2010 -
http://www.incits.org/tc_home/National_Standards_Published_as_of_09_08_2010.pdf

[3] ANSI/NIST-ITL Standards Homepage –

http://www.nist.gov/itl/iad/ig/ansi_standard.cfm

[4] ISO/IEC JTC1/SC 37 – *Biometrics* Public Homepage -
http://www.iso.org/iso/jtc1_sc37_home

[5] Published Biometric International Standards Developed by ISO/IEC JTC 1/SC 37 –
Biometrics as of September 2010 -
http://www.incits.org/tc_home/International_Standards_Published_as_of_09_08_2010.p
df

[6] BioAPI Conformance Test Suite and associated documentation download -
http://www.itl.nist.gov/div893/biometrics/BioAPI_CTS/index.htm.

[7] Conformance Test Architecture for Biometric Information Records and CBEFF Patron
Format A Conformance Test Suite -
http://www.itl.nist.gov/div893/biometrics/CBEFF_PFA_CTS/index.htm.

[8] NIST/ITL Conformance Test Architecture (CTA) and Test Suites (CTSs) for Biometric
Data Interchange Formats and Biometric Information Records Beta Implementation
V2.0 Homepage - http://www.nist.gov/itl/csd/biometrics/biocta_download.cfm

[9] NIST's Biometric Resource Center: http://www.nist.gov/itl/csd/biometrics/