

TULP2G – An Open Source Forensic Software Framework for Acquiring and Decoding Data Stored in Electronic Devices

Jeroen van den Bos and Ronald van der Knijff
Netherlands Forensic Institute

Abstract

TULP2G is a forensic software framework for acquiring and decoding data stored in electronic devices. The framework consists of a layered architecture with communication, protocol, conversion, and export plug-ins to acquire, decode, and report evidence in customizable layouts. All acquired data is stored in an XML formatted evidence file along with information for auditing purposes. XML files can also be used to customize the framework with different user interface languages. A profile mechanism is built in to save and load framework configuration settings for common investigations. Conversion and export plug-ins can also be used to decode data acquired with other data acquisition methods. TULP2G is implemented in C# using .NET1.1 and released under a BSD license. All software, including source code is available at <http://tulp2g.sourceforge.net/>. Currently available plug-ins are mainly targeted towards GSM phone examinations, but the applied open source strategy tries to stimulate other parties in developing more examination functionality.

1 Introduction

Preparing evidentiary images of mobile device media is different than making a forensically sound copy of a hard drive. Mobile device media are generally not meant to be exchangeable¹, but instead literally glued to the device's printed circuit boards². There is no general open interface for directly accessing mobile device media other than physical removal of the chips and reading them with so called *device programmers*³. These kinds of investigations can only be done by technical engineers in laboratory environments and are therefore only carried out in exceptional cases.

Because mobile devices might contain a lot of potential evidence, other investigation techniques are used to extract data from these devices [1]. The most basic investigation method is to use the normal user interface to manually extract as much user related

¹ Recent mobile phones are increasingly using exchangeable multi-media cards, but a lot of user related data is still stored in hard-soldered solid state memories.

² Current mobile phones store non-volatile data in Flash memories packaged in so called *micro ball grid arrays*, which means that all physical connections are on the bottom side of the chip, not accessible without first removing the entire chip from the printed circuit board.

³ A device programmer physically connects to a device and is used by the industry to program devices during manufacturing.

information as possible⁴. This works for the older types of mobile phones, but gets cumbersome with modern devices containing thousands of items in a lot of different formats. For this reason the NFI⁵ developed some tools at the end of the last century to help investigators with their manual investigations⁶. There were a number of reasons to replace these tools with a general framework for the examination of electronic devices:

1. Existing tools are based on ASCII text output format. The growing usage of Unicode⁷ and the emerging popularity of multi-media data in mobile communication demand a new storage and output format.
2. Forensic embedded systems specialists want to concentrate on data extraction and data decoding and not on integrating different methods into a user friendly software product. Most tools stay in an "only for laboratory use" stage and cannot be used by novice users. A framework can relieve forensic specialists from complex software development issues.
3. Embedded specialists at the NFI are busy with complex laboratory examinations and do not have time to implement all requested methods. With a framework more people are able to add solutions.

This paper introduces *TULP2G*⁸, a forensic software framework for acquiring and decoding data stored in electronic devices. Section 2 describes the framework architecture and the main functionality it offers. Section 3 illustrates typical framework usage with a tutorial on how to read pictures from a particular type of mobile phone. Advanced usage of TULP2G data decoding functionality is demonstrated in Section 4. Section 5 enumerates all currently available examination tools for the TULP2G framework and describes the desired update and support mechanism.

2 TULP2G Software Framework

2.1 Purpose

TULP2G is a forensic software framework to assist forensic investigators with their examinations of electronic devices. It is not a "push one button" tool automating the complete forensic analysis process. It assumes trained examiners who know how to investigate a device, but are in need of some assistance to speed up the forensic investigation process and to minimize human errors.

⁴ For mobile phone investigations this method can partly be standardized using a phone's menu structure as guideline.

⁵ Netherlands Forensic Institute.

⁶ *Cards4Labs* for reading SIM chip cards and *TULP* for reading data from mobile phones.

⁷ A set of standards aimed to provide a universal way of encoding characters of any language, regardless of the computer system, or platform, being used.

⁸ The name TULP2G refers to the former TULP program. TULP is Dutch for *tulip* and used as an acronym for *Telefoon UiTlees Programma*, which means "program to read phones".

The framework itself does not contain specific functionality for data extraction, decoding, or reporting. It only defines a general examination workflow for investigators and offers an abstract design to developers of so called *plug-ins*. These plug-ins contain the actual investigation methods. From a user's perspective the advantage of using a framework concept is the "learn once apply everywhere" principle. A drawback of this general purpose concept is the initial effort needed to get used to the workflow. The most important reason for introducing a framework concept is to make it easy for software developers to add specific device functionality without troubling them with GUI⁹ aspects and repeating common programming tasks. By creating this we hope to stimulate developers and increase the number of automated forensic extraction methods for electronic devices.

2.2 Framework architecture and plug-in concept

Figure 1 depicts the general framework architecture.

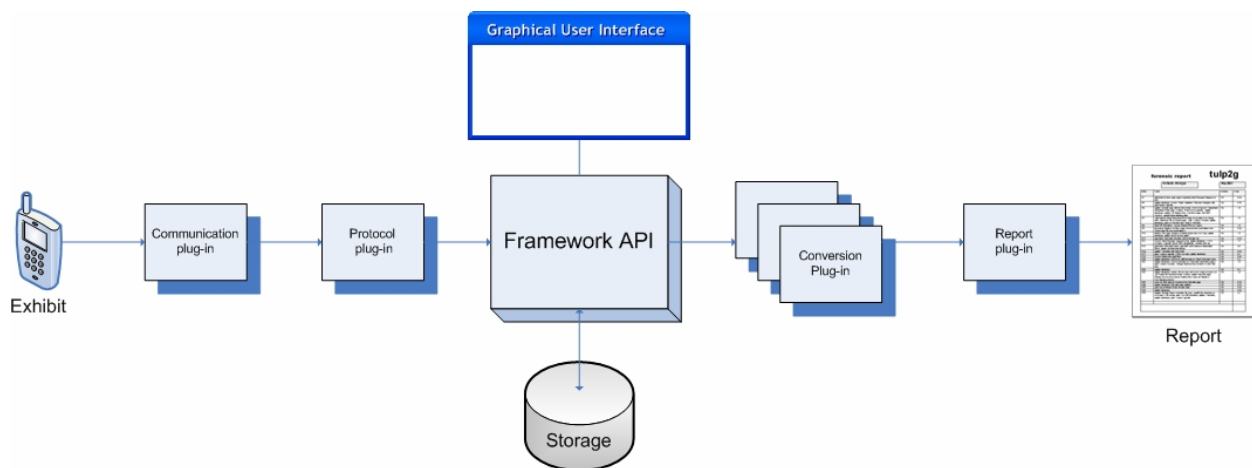


Figure 1: TULP2G Framework Architecture.

Each investigation starts with an exhibit containing data and ends with a report containing information originating from the exhibit data and/or related to the investigation process. The investigation process is modeled with four different *plug-in* categories: two for data acquisition and two for data conversion and export. A fifth *Tool* plug-in category has been defined for case-related tasks not directly linked to an exhibit or tasks which need a dedicated user interface. Investigations can be grouped into Cases. All data related to a case is stored in one XML [2] formatted file (see section 2.3 for details).

⁹ Graphical User Interface.

2.2.1 *Communication plug-ins*

These plug-ins handle the lowest level of communication with devices¹⁰: setting up and closing connections and sending and receiving raw data. Because of this they directly interface with the external device through a driver or other low-level piece of software. Examples of communication plug-ins are the serial communication plug-in for accessing devices via RS232, Infrared or Bluetooth, and the PC/SC communication plug-in for chip card access via the standardized PC/SC interface [3]. Communication plug-ins can call functions from the framework API¹¹ for logging and user interface purposes. Because communication plug-ins are specific to a certain type of communication and not to a device, they are not used on their own to read information from a device. Another plug-in that uses the services of a communication plug-in is required to supply the actual data that needs to be exchanged with the device to retrieve useful information from it.

2.2.2 *Protocol plug-ins*

Protocol plug-ins are the counterparts of communication plug-ins. They do not have the ability to actually connect to a device, but instead they implement a protocol and use a communication plug-in¹² to actually send and receive information from a device, according to the protocol that they implement. Examples of protocol plug-ins are the AT_ETSI protocol plug-in [0] for extracting data from mobile phones and the SIM protocol plug-in [5] for extracting data from SIM chip cards. Protocol plug-ins can call functions from the framework API for logging, user interface, and storing data from exhibits purposes.

2.2.3 *Conversion plug-ins*

A conversion plug-in receives data stored by the framework, originating from exhibits, and is able to convert specific data types. Conversion plug-ins can be chained in order to support sequential decoding (onion peeling). This mechanism is supported to maximize code re-use and avoid complex software maintenance procedures. Examples of conversion plug-ins are the SIM conversion plug-in for converting low level SIM data into report-ready XML data (except for low level SMS TPDU¹³, they are only “peeled” out of their “SIM skin”) and the SMS conversion plug-in for converting SMS PDUs not only originating from SIM cards, but from mobile phones. Conversion plug-ins can call functions from the framework API for logging and user interface purposes and are controlled through a selected *export plug-in*.

2.2.4 *Export plug-ins*

Export plug-ins send all data stored by the framework, originating from exhibits, and selected by the user, through the activated conversion plug-ins. If data is converted by a

¹⁰ This means the lowest level from the TULP2G Framework perspective. It does not necessarily mean the lowest possible level seen from the computer’s operating system.

¹¹ Application Programming Interface: a set of methods for building software applications.

¹² One protocol plug-in might be able to use different communication plug-ins for connecting to an exhibit. For this to work the developers of the communication plug-ins need to use the same data format in the implementation of the framework interface between the communication and protocol plug-ins.

¹³ Short Message Service Transfer Protocol Data Unit: a message of a given protocol comprising SMS payload and protocol-specific control information.

particular conversion plug-in, the export plug-in replaces the original data item with the converted one. Once all data conversions have taken place, the export plug-in converts the resulting data to a plug-in specific output format, such as XML, HTML¹⁴, PDF¹⁵, or DOC¹⁶. Most Export plug-ins will be using a template-based conversion method to allow for small changes to be made to the output format without having to recompile or rewrite an entire export plug-in. An example of an export plug-in is the XML export plug-in for generating XML and/or HTML using XSL¹⁷ stylesheets. Conversion plug-ins can call functions from the framework API for logging and user interface purposes (e.g. specifying a template file or an output folder for extracted multimedia data).

2.2.5 Tool plug-ins

The framework's tool functionality makes it possible to create additional plug-ins to perform investigation-related tasks using TULP2G's rich plug-in interface. Tool plug-ins are basically free-form; there is no strict interface they should adhere to besides the mechanism used by TULP2G to load plug-ins. This allows tool plug-ins to perform all kinds of different tasks. An example of a tool plug-in is the IMEI¹⁸ decoder plug-in for getting brand and type information from mobile phones.

2.3 Evidence file format

TULP2G heavily uses XML for data storage and retrieval. XML is not only used for saving and loading investigation data, but for storing output, configuration, and language data, as well as for internal data structures during conversion and export. Figure 2 outlines the format used by TULP2G for storing case related data.

¹⁴ Hyper Text Markup Language, the authoring language used to create documents on the World Wide Web.

¹⁵ Portable Document Format, a file format developed by Adobe Systems.

¹⁶ File format used by the Microsoft Word software.

¹⁷ Extensible Stylesheet Language, a specification for separating style from content when creating HTML or XML pages.

¹⁸ International Mobile Equipment Identity, a unique number given to every single mobile phone, typically found behind the battery.

```

<Case Name="..." Creator="..." DateCreated="..." DateModified="..." MD5="..." SHA1="...">
  <Notes>...</Notes>
  <Item Name="..." DateCreated="..." DataType="..." StorageType="..." ItemType="..." MD5="..." SHA1="...">
    ...
  </Item>
  <Investigation Name="..." Creator="..." DateCreated="..." MD5="..." SHA1="...">
    <Notes>...</Notes>
    <Item Name="..." DateCreated="..." DataType="..." StorageType="..." ItemType="..." MD5="..." SHA1="...">
      ...
    </Item>
    <Item Name="..." DateCreated="..." DataType="..." StorageType="..." ItemType="..." MD5="..." SHA1="...">
      ...
    </Item>
  </Investigation>
  <Investigation Name="..." Creator="..." DateCreated="..." MD5="..." SHA1="...">
    ...
  </Investigation>
  <Item Name="..." DateCreated="..." DataType="..." StorageType="..." ItemType="..." MD5="..." SHA1="...">
    ...
  </Item>
</Case>

```

Figure 2: TULP2G XML Evidence File Format.

Each TULP2G evidence file has only one *Case* element with three possible child types:

1. *Notes*: at most one element for storing case related notes.
2. *Item*: zero or more elements for storing logging data not belonging to an investigation (e.g. related to export and conversion actions).
3. *Investigation*: top level element for data belonging to a particular investigation. *Investigation* elements have two possible child types: at most one *Notes* element for storing investigation related notes, and zero or more *Item* elements for storing data belonging to the investigation.

DateCreated and *DateModified* are timestamps in FILETIME format¹⁹ equal to the local time of the computer running TULP2G.

MD5 and *SHA1* are hash values²⁰ calculated in the following way²¹:

$$\text{Hash}(\text{Item}) = \text{Hash}(\text{Name} + \text{DateCreated} + \text{DataType} + \text{StorageType} + \text{ItemType} + \text{StringContent})$$

$$\text{Hash}(\text{Investigation}) = \text{Hash}\left(\text{Name} + \text{Creator} + \text{DateCreated} + \text{Notes} + \sum_{i=1}^{\#\text{Items}} \text{Hash}(\text{Item}[i])\right)$$

¹⁹ The FILETIME data structure is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601.

²⁰ Hash values are added to ensure data integrity of evidence files.

²¹ "+" means: string concatenation, timestamps are first converted to UTC (Coordinated Universal Time).

$$\text{Hash}(\text{Case}) = \text{Hash}(\text{Name} + \text{Creator} + \text{DateCreated} + \text{DateModified} + \text{Notes} + \sum_{i=1}^{\#\text{ItemChildrenOfCase}} \text{Hash}(\text{Item}[i]) + \sum_{j=1}^{\#\text{Investigations}} \text{Hash}(\text{Investigation}[j]))$$

DataType is used by the conversion plug-ins to decide if conversion of specific item data is possible. *DataTypes* are defined by plug-in developers which need to assure that the same *DataType* is not used for different data formats. Example *DataTypes* are *ETSI_11.11_FILE* used by the SIM protocol plug-in and *ETSIAT* used by the AT_ETSI protocol plug-in.

StorageType is *String* for string data and *Binary* for base64²² encoded binary data items.

ItemType is *Error* for error logging data, *Plugin-Info* for plug-in related logging data, *Receive* for data received from exhibits, and *Send* for data sent to exhibits.

ItemTypes with value *Error* have two additional attributes:

ErrorClass with possible values: *Communication*, *Forensic*, *Reporting*, *System*, or *Unknown*

ErrorSeverity with possible values: *Warning*, *NonFatal*, *Fatal*, or *Unknown*

2.4 Auditability (logging)

For forensic purposes all actions taken by the investigator related to exhibits need to be stored for reviewing. The TULP2G framework logs case and investigation related user actions to the evidence file using the value *PluginInfo* of the *ItemType* attribute. For this reason loaded evidence files might need to be saved even if no new investigation is added. If no evidence file is open logging data is stored in a global log file specified with the *systemlogfile* element in the TULP2G framework configuration file *config.xml* (default value is *systemlog.xml*). *ItemTypes* with value *Error* are also displayed in the *Details* section on the TULP2G progress form (Figure 3).

²² Base64 is an encoding for binary data using 64 encoding characters (A-Z, a-z, 0-9, +, /).

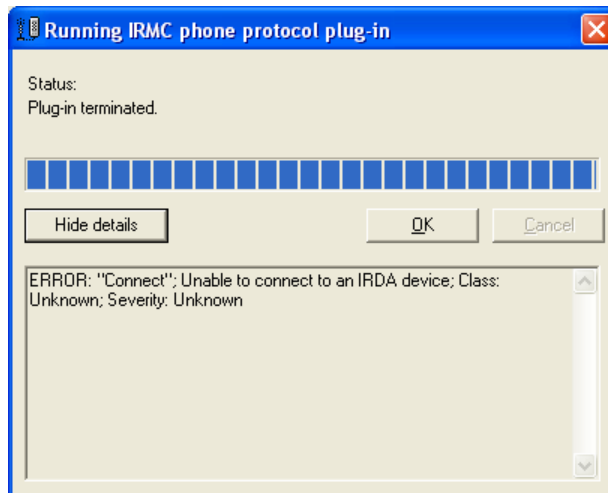


Figure 3: Logged Errors Shown on the Progress Form.

Plug-in developers are advised to create logging items with the framework API at least for all communication with exhibits and after each encountered error or abnormal event. Logged audit items can be used by export and/or conversion plug-ins for encapsulation into reports. If an extensive review is requested, a dedicated conversion plug-in can be created which extracts all low-level communication with an exhibit from the evidence file and converts it into an audit report.

To illustrate logging, assume a SIM card investigation with TULP2G using the SIM protocol plug-in and the PC/SC communication plug-in. The SIM protocol plug-in uses the framework API to store data from the SIM file system into an evidence file. The protocol plug-in also logs possible SIM-errors and relevant user actions like the PIN and PUK attempts. The PC/SC communication plug-in is controlled by the SIM protocol plug-in, but independently logs items to the evidence file like connection attempts, low level data transfer with the SIM, and any anomalies on the connection level. This log data is normally not used for reporting, but is stored in the evidence file for audit purposes. After completing the investigation, a report can be generated with the XML export plug-in and the SIM and SMS conversion plug-ins. During the export the framework creates a logging item describing the used export and conversion plug-ins. The used export and conversion plug-ins use the framework API to log possible errors to the evidence file and are also able to put specific items into the report. For example, the SIM conversion plug-in writes a table to the report with all PIN and PUK actions executed during the examination (Figure 4).

Card Holder Verification (CHV)

	PIN	PUK	PIN2	PUK2
Status	3 (enabled)	10	3	10
Verification	0000			
Status	3 (enabled)	10	3	10

Figure 4: Part of a SIM Report Generated from Data Logged by the SIM Protocol Plug-In.

2.5 User documentation

The framework supports modular context-sensitive help. Each plug-in can have its own compiled help file [8], possibly in different language versions (see section 2.6). The used naming convention is *TULP2G.<Plug-inType>.<Plug-inName>.<LanguageCode>.chm*. To link a specific help item in a compiled help file to a GUI element the frameworks assumes the following bookmark naming convention: *<ControlTag><ControlCaption>*, with:

<ControlTag> specifying the GUI control type with possible values *Text, Info, Combo, Button* and *Check*.

<ControlCaption> equal to the caption text of the control with all spaces replaced with underscores (“_”) and all other non-alphanumeric characters removed.

Example: pressing the [F1] key from the *Port* combo-box in the configuration form of the *Serial* plug-in will cause TULP2G to open the file *TULP2G.Communication.Serial.en-GB.chm* and display the item bookmarked with *ComboPort*.²³

Documentation related to report items is preferably added to the report in order to make a separation between people who do an investigation and people who have to interpret the investigation results. Current report templates embed this documentation in such a way that the reader can choose a level of documentation to display (each item is preceded by an explanation, only explanations are shown or an explanation is shown when the mouse is dragged over the item content).

2.6 Language configuration

The TULP2G framework is designed to support multiple user interface languages. Language dependent items are *tagged* and can be adapted to a specific language with external XML files. Adapting the TULP2G user interface to another language is a six step process:

²³ The framework expects all compiled help files in the path relative to the TULP2G executable folder, further specified with the *<onlinehelppath>* element in the global configuration file. The framework also adds a language extension based on the value of the *<language>* element in the global configuration file.

1. Install TULP2G onto an administrator's machine and configure it with the preferred plug-ins.
2. Quit TULP2G and open the *config.xml* file to change the *<language>* element to the target language using the ISO-639 language code [6] and the ISO-3166 two-letter country code [7] (for example nl-NL for Dutch as it is spoken in The Netherlands). Change the *<runmode>* element to *localize*.
3. Start the TULP2G framework. No user interface will be visible, the framework only generates template XML files for localization with names such as *TULP2G.<Plug-inType>.<Plug-inName>.<LanguageCode>.xml* (for example *TULP2G.Protocol.AT_SIEMENS.nl-NL.xml*) . Edit these template files and translate all English text element values into the target language. Also translate the file *GUI.<LanguageCode>.xml*.
4. Change the *<runmode>* to *GUI* and the framework will use the translated element values of the selected language. These translated language files can now be distributed onto target machines.
5. Translate all report templates.
6. Translate all online compiled help files (see section 2.5) using the same naming convention as in the localization files.

Besides user interface elements the framework supports translation of other language dependent data (such as error messages and status information). Plug-in developers will need to define label names when using language dependent data and these labels need to be added to the localization files manually. If needed, future framework versions could automate this process in the same way as with the generation of user interface translation templates.

2.7 Profiles

Configuring TULP2G for a specific investigation can be a time consuming task. First, the right plug-ins need to be installed. Then each data acquisition plug-in has to be configured before the investigation can be executed. After the data acquisition process is finished, but before a report can be generated, an export plug-in needs to be configured with a suitable template. Also the right conversion plug-ins must be selected, arranged, and possibly configured.

To speed up standard investigations the framework supports a profile mechanism that can be used to save and load particular investigation set-ups. After configuring and testing a complete investigation and reporting flow, the current configuration can be saved to an XML file with the *Save Profile* function. The next time a similar investigation needs to be performed, the saved profile can be loaded with the *Load Profile* function. This function first tries to configure installed plug-ins as specified in the saved profile. If a specific plug-in version is not installed, the framework looks for an installed plug-in with the same name and most recent version number. Then the framework asks the user for *Case*, *Investigator*, and *Investigation* names. Following the user's input the framework is configured as specified in the loaded profile and is ready to execute the investigation followed by generating an accompanying report.

3 Tutorial

To give an impression of typical framework usage, this section presents a tutorial for a specific investigation using TULP2G. This tutorial assumes the latest installed version of TULP2G (see section 5 for details). For this tutorial we use a *SonyEricsson* mobile phone, model *K700i* and we want to extract the pictures made with the built-in camera of the phone. Besides the GSM radio interface and the user interface, a *K700i* has three interfaces for data communication: cable, infrared and Bluetooth. In this tutorial the infrared interface is used to connect the phone to a personal computer with an external infrared adapter. For extracting the pictures the OBEX²⁴ protocol plug-in is used. The OBEX protocol plug-in is designed to use the Socket communication plug-in. A *socket* is one of the most fundamental technologies of computer networking. Sockets allow applications to communicate using standard mechanisms built into network hardware and operating systems.

After starting the TULP2G framework check that the Socket and OBEX plug-ins are loaded via the “List...” button on the “Tulp2g” tab. Otherwise load the files *TULP2G.Communication.Socket.dll* and *TULP2G.Protocol.OBEX.dll* from the TULP2G *Plugins* folder.

Go to the “Case” tab, type a “Case Name”, optionally some “Notes” and select the “Create” button. This creates the top level Case element and an Investigation element in the evidence file (see section 2.3).

²⁴ Object Exchange Protocol, a binary session protocol optimized for ad-hoc wireless links [9].

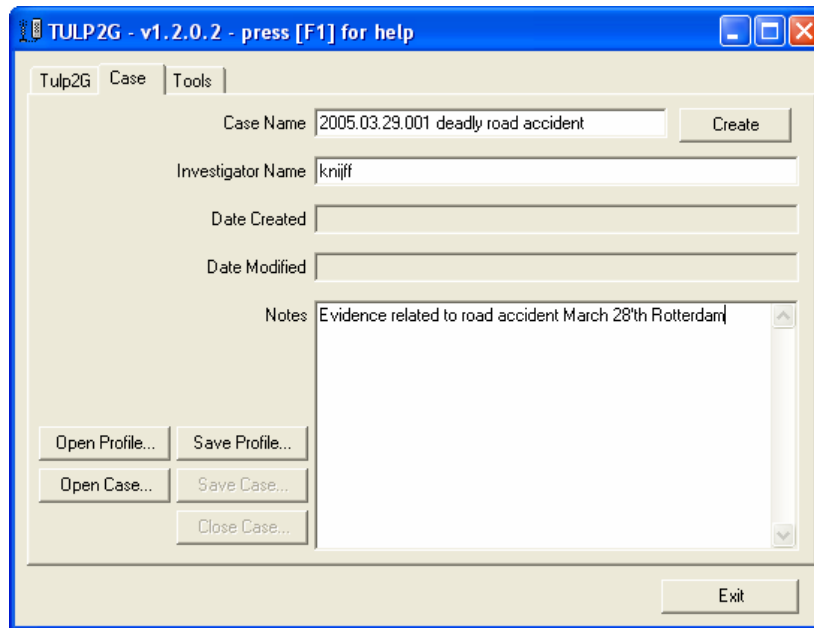


Figure 5: The Case Tab Before Selecting the "Create" Button.

Now go to the "Investigation" tab, type an "Investigation name," and optionally some "Notes". Select the "Socket communication" communication plug-in from the "Communication plug-in" combo-box, and select the "Configure..." button to the right. Now place the phone with the infrared sensor in front of the PC's infrared transceiver and make sure infrared transfer is enabled in the phone²⁵. The "Socket Type" should be "IRDA" [9] and after selecting the test button the "Status" field should change from "Unknown" to "OK" indicating the connection is working on this protocol level. Select the "OK" button to keep these settings, select the "OBEX phone protocol plug-in" and select the "Configure..." button to the right. Change the "OBEXFTP" value of the "Service" combo-box to "OBEXOBJECTPUSH" and select the "Test" button. The "Status" field should change from "Unknown" to "OK" indicating the connection is working on this protocol level. Select the "OK" button; both communication and protocol plug-ins are now configured.

²⁵ Switch the "Infrared port" to "On" via the phones "Connectivity" menu. If enabled the PC now displays an icon in the taskbar indicating infrared activity.

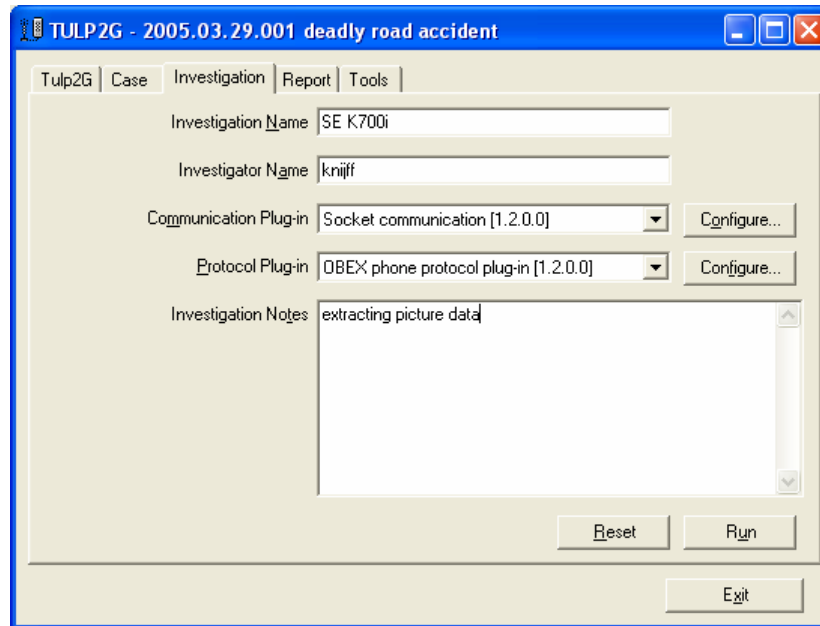


Figure 6: The Investigation Tab After Configuring of the Communication and Protocol Plug-Ins.

Select the “Run” button to start the data acquisition. The “Progress” form will appear showing the acquisition status. After the acquisition has been completed²⁶, error and warning messages can be displayed by selecting the “Show details” button on the “Progress” form. Select the “OK” button to complete the investigation. To store the evidence file go to the “Case” tab, select “Save Case...” and choose a file name and location with the file selection dialog. All evidence data is now saved for later use.

To make a report of the acquired data, the XML export plug-in needs to be installed along with the OBEX conversion plug-in. Go to the “Report” tab, select the XML/HTML export plug-in and select the “Configure...” button on the right. With the “Select stylesheet file...” button select the stylesheet “ReportOBEX.xsl”, and select the “OK” button. Now select the “Select...” button on the right of the “Selected conversion plug-in(s)” list-box and add the “OBEX phone data conversion” plug-in to the list of “Use plug-ins”. With the “Configure...” button a folder can be selected for all exported data items. The report can now be generated by selection of the “Run” button. The framework asks for a file name for the report and starts the reporting process. After all conversions are finished the report is loaded into the default installed browser with all pictures that are normally accessible via the phone’s user interface. All multi-media data (pictures, sounds, videos) are extracted to the selected export folder. The report file only contains links to these files to enable “click to open” functionality on the computer where the framework is running.

²⁶ This might take a long time (hours) if a lot of multi-media data is present in the phone. It is not exactly clear why it takes such a long time for a relatively small amount of data.

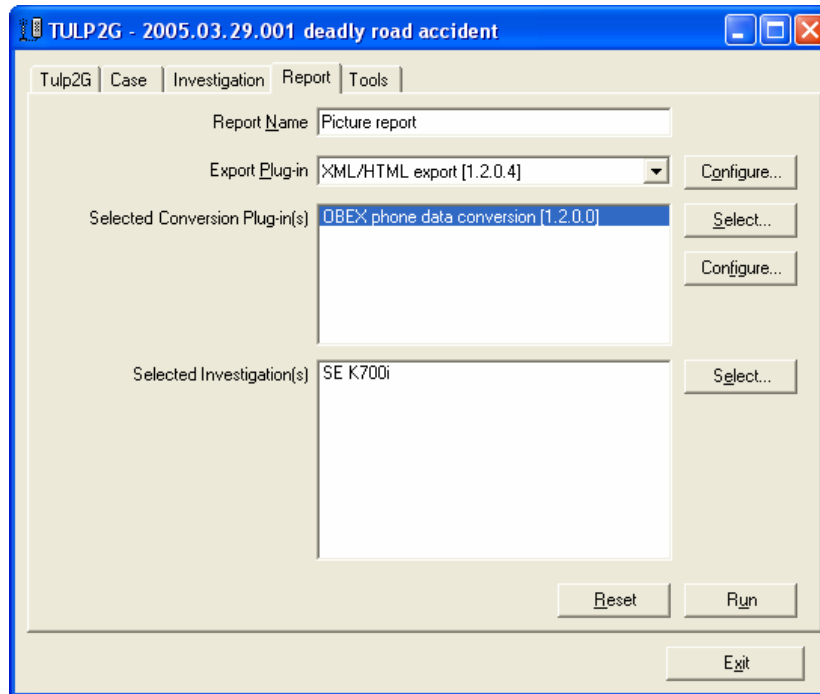


Figure 7: The "Report" Tab Just Before Selecting the "Run" Button.

For auditing purposes, the case needs to be saved before closing TULP2G or starting a new case to include the reporting process in the evidence file. This example only concentrates on extracting pictures from the exhibit. Other plug-ins might be able to get other data from this phone²⁷.

4 Advanced Usage of Conversion Plug-ins

This section will show how conversion plug-ins can be used for decoding data that is not acquired with protocol plug-ins, but imported from other sources. To do this the target data needs to be encapsulated into a valid evidence file. As an example, take the SMS TPDU (see section 2.2.3) showed in Figure 8. These TPDU might originate from a variety of sources, such as a wire tap, a hard drive, or low level image reading of a mobile phone's memory chip.

²⁷ For example the IRMC protocol plug-in for data from the phone book, text messages and calendar data, and the SIM protocol plug-in to extract data from the SIM chip card attached to the phone.

Figure 10.

Nr.	Ext. Type	Int. Type	Part	Timestamp
3	recovered	PictureMessage Type	All/3	30-5-2005 3:50:50 GMT+10.00
Service Centre Address			Originating Address	
+31655555000			+31612356781	
Content				
生存还是毁灭, 那是问题。快的棕色狐狸跳过了懒惰狗 				

Figure 10: Resulting Report of the Decoding Process, A Multi-Part SMS Picture Message.

5 Availability and Current Version

TULP2G has been implemented in C# using .NET 1.1. The software is open source and released under a BSD license. The BSD license is a very unrestrictive license, allowing various uses of the software, as well as (modified) redistribution in both source and/or binary form. Distribution under this license is an attempt by the NFI to stimulate the development of forensic software tools. For distribution and release management the open source software development website *SourceForge* is used. TULP2G can be downloaded from <http://tulp2g.sourceforge.net/>.

The NFI maintains a binary distribution of TULP2G which can be used for forensic investigations. To ascertain the integrity of this distribution each module is protected with a private key only available to the NFI. The .NET environment verifies each module before it is loaded using the NFI's public key. The 8-byte hash of the NFI's public key is equal to: 34 80 A3 62 4A C4 8F 93, and printed in all generated reports.

The current, full distribution package of TULP2G on Sourceforge contains the following plug-ins:

Communication

- *Serial Port* for communication over serial channels (e.g. RS232, Infrared, BlueTooth);
- *PCSC* for communication via PC/SC compatible chip card readers;
- *Socket* for socket communication.

Protocol

- *AT-ETSI* for mobile equipment which implements the AT command set for GSM Mobile Equipment;
- *AT-SIEMENS* for mobile equipment which implements AT commands specific for Siemens phones;

- *AT-SAMSUNG* for mobile equipment which implements AT commands specific for Samsung phones;
- *SIM* for GSM SIM cards;
- *OBEX* for OBEX compatible mobile phones via OBEX push or FTP service;
- *IRMC* for IRMC compatible mobile phones.

Conversion

- *AT-ETSI* for conversion of data extracted with the AT-ETSI protocol plug-in;
- *AT-SIEMENS* for conversion of data extracted with the AT-SIEMENS protocol plug-in;
- *AT-SAMSUNG* for conversion of data extracted with the AT-SAMSUNG protocol plug-in;
- *SIM* for conversion of data extracted with the SIM protocol plug-in;
- *SMS* for conversion of SMS TPDU's;
- *OBEX* for conversion of data extracted with the OBEX protocol plug-in;
- *IRMC* for conversion of data extracted with the IRMC protocol plug-in;
- *HexDump* for hex viewer like displaying.

Export

- *XML/HTML* for saving case data in XML and/or HTML format.

Tools

- *IMEI* for decoding serial numbers of mobile GSM equipment.

There is also a separate ZIP file available on Sourceforge with plug-ins to prepare a writable SIM card for examining a mobile phone without having the last inserted SIM and without changing important user data data in the examined phone.

6 Related Work

In recent years a number of forensic tools have appeared specifically targeted towards acquisition, analysis and reporting of data stored in mobile devices. For Personal Digital Assistants (PDAs) [10] gives an overview of available software and an understanding of their capabilities and limitations. A similar overview related to mobile phones is published by NIST in [11].

7 Conclusions and Future Work

This paper presented a forensic software framework for acquiring and decoding data stored in electronic devices. The current version of the framework along with the available plug-ins can already be used to assist investigators with their manual investigation tasks. To widen the usefulness, new functionality is needed to support more devices and enlarge the completeness of the data extraction and decoding process for supported devices. Publishing TULP2G as an open source package is an

attempt by the NFI to stimulate other parties in developing TULP2G plug-ins. To make it easier for developers to start with a plug-in, a dedicated developer paper will be published together with some tutorial plug-ins. Additional work is required in the area of testing and evaluation. The core framework functionality needs extensive testing and auditing and for testing specific plug-in setups a general test protocol needs to be defined.

© Copyright 2005 International Journal of Digital Evidence

About the Authors

Jeroen van den Bos (jeroen@holmes.nl) works at the Digital Technology department of the Netherlands Forensic Institute (<http://www.forensischinstituut.nl/NFI/en>) as a software architect in the software engineering group. He designs and implements software tools that automate and aid in doing forensic investigations.

Ronald van der Knijff (knijff@holmes.nl) works at the Digital Technology department of the Netherlands Forensic Institute (<http://www.forensischinstituut.nl/NFI/en>) as a scientific investigator and is responsible for the embedded systems group. He also lectures on “Smart Cards and Biometrics” at the Tias Business School’s Masters Program “Information Technology” and on “Cards & IT” at the Dutch Police Academy.

References

1. R.M. van der Knijff. Embedded Systems Analysis , chapter 11 of Handbook of Computer Crime Investigations - Forensic Tools and Technology, edited by Eoghan Casey, Academic press, 2002.
2. Extensible Markup Language (XML). <http://www.w3.org/XML/>
3. PC/SC Workgroup. <http://www.pcscworkgroup.com/>.
4. Digital cellular telecommunications system (Phase 2) (GSM); AT command set for GSM Mobile Equipment (ME) (GSM 07.07). <http://www.etsi.org/>.
5. Digital cellular telecommunications system (Phase 2+) (GSM); Specification of the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface (GSM 11.11). <http://www.etsi.org/>
6. ISO 639 2-letter language codes. <http://www.w3.org/WAI/ER/IG/ert/iso639.htm>
7. ISO 3166 country codes. <http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html>

8. Microsoft HTML Help. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/htmlhelp/html/vsconHH1Start.asp>
9. The Infrared Data Association (IrDA), specifications for infrared wireless communication. <http://www.irda.org/>.
10. R Ayers, W Jansen. PDA Forensic Tools: An overview and Analysis. National Institute of Standards and Technology (NIST). April 2004. <http://csrc.nist.gov/publications/nistir/nistir-7100-PDAForensics.pdf>
11. R Ayers, W Jansen., N Cilleros, R Daniellou. Cell Phone Forensic Tools: An overview and Analysis. National Institute of Standards and Technology (NIST). October 2005. <http://csrc.nist.gov/publications/nistir/nistir-7250.pdf>