NTNU
Innovation and Creativity

Forensic analysis of an unknown embedded device

Jarle Eide
Jan Ove Skogheim Olsen

Master of Science in Computer Science
Submission date: June 2006
Supervisor: Stig Frode Mjølsnes, ITEM
Co-supervisor: Svein Y. Willassen, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Sometimes investigators must extract digital evidence from an embedded device with unknown specifications. The manufacturer of the device could be unknown, or the manufacturer may choose to keep information about the device's inner workings confidential. In order to be able to interpret digital evidence on such devices, an investigator must carefully analyze a similar device by inserting known data and observing the changes in the contents of the device.

The task is to treat a mobile phone with the Windows Mobile Operating System as an unknown embedded device. Find a methodologically sound approach for analyzing the device. Interpret the contents in a such a way that the result can be applied in forensic analysis of real evidence.

Describe and evaluate the analysis methodology. Is the methodology applicable for forensic analysis of other unknown embedded devices?

Assignment given: 16. January 2006
Supervisor: Stig Frode Mjølsnes, ITEM

Abstract

Every year thousands of new digital consumer device models come on the market. These devices include video cameras, photo cameras, computers, mobile phones and a multitude of different combinations. Most of these devices have the ability to store information in one form or another. This is a problem for law enforcement agencies as they need access to all these new kinds of devices and the information on them in investigations. Forensic analysis of electronic and digital equipment has become much more complex lately because of the sheer number of new devices and their increasing internal technological sophistication. This thesis tries to help the situation by reverse engineering a Qtek S110 device. More specifically we analyze how the storage system of this device, called the object store, is implemented on the device's operating system, Windows Mobile. We hope to figure out how the device stores user data and what happens to this data when it is "deleted". We further try to define a generalized methodology for such forensic analysis of unknown digital devices. The methodology takes into account that such analysis will have to be performed by teams of reverse-engineers more than single individuals. Based on prior external research we constructed and tested the methodology successfully. We were able to figure our more or less entirely the object store's internal workings and constructed a software tool called BlobExtractor that can extract data, including "deleted", from the device without using the operating system API. The main reverse engineering strategies utilized was black box testing and disassembly. We believe our results can be the basis for future advanced recovery tools for Windows Mobile devices and that our generalized reverse engineering methodology can be utilized on many kinds of unknown digital devices.

Contents

List of Figures

List of Tables

Listings

Abbrevations

API Application Programming Interface

ARM Acorn RISC Machine

CEDB CE DataBase

CIM Computation Independant Model

DLL Dynamic Link Library

FAT File Allocation Table

HW Hex Workshop

IDA (PRO) Interactive Disassembler (Professional)

LSB Least Significant Bit

MDA Model-Driven Architecture

MDRE Model-Driven Reverse Engineering

MSB Most Significant Bit

MSDN Microsoft Developers Network

ODP Open Distributed Processing

OID Object Identifier

OMG Object Management Group

OS Operating System

PIM Platform Independant Model

PSM Platform Specific Model

RAM Random Access Memory

RAPI Remote Application Programming Interface

RM-ODP The Reference Model for Open Distributed Processing

SIM Subscriber Identity Module, smart card for mobile phones.

UML Unified Modeling Language

VFAT (see FAT)

Chapter 1

Introduction

1.1 Background

The digital world is all around us. Every year thousands of new digital consumer device models come on the market. These devices include video cameras, photo cameras, computers, mobile phones and a multitude of different combinations. Most of these devices have the ability to store information in one form or another. Cameras store video or pictures, computers store the users work and mobile phones store call logs and text messages.

Among the fastest growing segments of new digital consumer devices are so-called smartphones. These are hybrids between a small computer and a mobile phone. They have the phone capabilities of a mobile phone and at the same time they can play music, record video and run programs like any other computer. One of the biggest operating systems suppliers for such devices is Microsoft with their Windows Mobile platform. According to Gartner[1] the smartphone market is expected to double every year, reaching 200 million by 2008. Currently Windows Mobile runs on around 10 percent of these, but their market share is rising every year. Microsoft distributes parts of the internal functions in its operating system freely through its shared source initiative[2], but several of the key components are not included. Unfortunately this includes all code pertaining to how the device stores user data at a low level, which would be very helpful for forensic analysts. This fact combined with the future market share of Windows Mobile makes it a very attractive target for analysis, as it could be helpful in many investigations in years to come.

The explosion of new digital devices has led to an explosion of both amounts and types of available information. This is a problem for law enforcement agencies as they need access to all these new kinds of devices and the information on them in investigations[3]. Forensic analysis of electronic and digital equipment has become much more complex lately because of the sheer number of new devices and their internal technological sophistication. One would expect that manufacturers of these devices would help out in this endeavor, but the fact is that the manufacturers often can not or will not reveal the internal workings of their devices. They can not because they have simply bought the underlying technology from a third party or they will not because they view the technical implementation details as business secrets. The fact that manufacturers can be located anywhere in the world and can be difficult to get in touch with on tight investigation schedules hardly helps either. There are also legal problems with letting just anybody handle evidence, which these devices are in an investigation. This leaves it up to the law enforcement agencies themselves or third party specialists to analyze them. Earlier this could be done by lone wolfs with high technology skills, but with the sophistication level and high volume of new devices this is becoming increasingly difficult. Cooperation among several individuals is essential to keep up and they need to be able to exchange their findings. What is needed is a methodology and model for doing and documenting forensic analysis of unknown devices, based on the principles of reverse engineering.

1.2 Reverse Engineering

"The process of analyzing a subject system to identify the system's components and their interrelationships and to create representations of the system in another form or at a higher level of abstraction." (IEEE 1990)

Reverse engineering is the process of figuring out the inner technological workings of a device or system without having access to its architectural and design details. A device is to be understood as any device, be it mechanical, electrical, software or anything else. The process usually consists of, by some kind of means, deconstructing the device. How this is done, of course varies wildly according to what kind of device one has. Mechanical components are usually just taken apart physically to understand what the different parts are and how they interact. Electrical components are analyzed with advanced equipment like oscilloscopes and logical analyzers, while software can be tested with different input/output combinations or a reverse-engineer can analyze the raw machine instructions that make up the program with the help of a disassembler.

The purpose of this process can be several things, depending on the type of system one is reverser engineering. For software systems the two main purposes are *redocumentation* and *design recovery* according to [4]. They describe them like this:

Redocumentation ".. is the creation or revision of a semantically equivalent representation within the same relative abstraction level. The resulting form of representation are usually considered alternative views (for example, dataflow, data structure, and flow control) intended for a human audience.

Redocumentation is the simplest and oldest form of reverse engineering, and many consider it to be an unintrusive, weak form of restructuring. The "re-" prefix implies that the intent is to recover documentation about the subject system that existed or should have existed.

Some common tools used to perform redocumentation are pretty printers (which display a code listing in an improved form), diagram generators(which create diagrams directly from code, reflecting control flow or code structure), and cross-reference listing generators. A key goal of these tools is to provide easier ways to visualize relationships among program components so you can recognize and follow paths clearly."[4]

Design discovery ".. is a subset of reverse engineering in which domain knowledge, external information, and deduction or fuzzy reasoning are added to the observations of the subject system to identify meaningful higher level abstractions beyond those obtained directly by examining the system itself. Design recovery is distinguished by the sources and span of information it should handle."[4]

According to Biggerstaff design recovery also "recreates design abstractions from a combination of code, existing design documentation (if available), personal experience, and general knowledge about program and application domains ... Design recovery must reproduce all of the information required for a person to fully understand what a program does, how it does it, why it does it, and so forth. Thus, it deals with a far wider range of information than found in conventional software-engineering representations or code."[5]

1.3 Objective

The following presents the project description, as defined by the university:

Forensic analysis of an unknown embedded device

Sometimes investigators must extract digital evidence from an embedded device with unknown specifications. The manufacturer of the device could be unknown, or the manufacturer may choose to keep information about the device's inner workings confidential. In order to be able to interpret digital evidence on such devices, an investigator must carefully analyze a similar device by inserting known data and observing the changes in the contents of the device.

The task is to treat a mobile phone with the Windows Mobile Operating System as an unknown embedded device. Find a methodologically sound approach for analyzing the device. Interpret the contents in such a way that the result can be applied in forensic analysis of real evidence.

Describe and evaluate the analysis methodology. Is the methodology applicable for forensic analysis of other unknown embedded devices?

1.4 Focus

We now define more precisely what our focus will be in this thesis.

Based on Windows Mobile devices rising popularity and the thesis description we are going to treat one of these mobile phones as an unknown device and try to analyze it. Analysis of a phone can include many things, but we have chosen to focus on how the phone stores user created data.

User created data is of course interesting because it can contain a lot of information about the device's user. In criminal cases call logs that keep tracks of who the user has been talking to can help the investigation. Text messages not only shows who the user has been talking to, but also what has been said. Other user created files of interest are pictures, videos and just any file with data that can help an investigation.

How call logs, text messages and user files are physically formatted on the "disk" of the phone is unknown. Some Windows Mobile Smartphones do not actually have non-volatile storage. Instead, parts of the devices RAM is used as storage. This means the user can actually lose all his data if the device runs out of power. The parts of RAM allocated for storage is called the object store. In this store the device stores all non-OS data on the phone. The layout and implementation of this store is not publicly available outside Microsoft. The biggest focus in our task has been to recover this layout. With this information we can construct forensic utilities to extract data from the phone without relying on the API exposed by the operating system itself. This is very important for forensic work because we do not know what side effects using the API might lead to. Forensic analysis must be free of side effects that change the data in any way.The integrity of the data must always be maintained so that it can be used as possible evidence in legal matters.

The capture of the raw memory contents of the phone is a daunting task in itself, but it is not within our scope. Our task starts when such a dump is available. The exact way this is done, is not important for us. Both hardware and software ways of doing this can be constructed. As far as we know no one has done this with a pure hardware approach at present time, but both [6] and [7] have made available software tools capable of this. We will use these to get our memory dumps. Our device also has a compact flash drive for additional storage space. We have defined this to be outside our scope as it is not considered part of the device's object store.

The second part of the task has been to develop a model and methodology for doing our analysis. How should we start analysis of a completely unknown device? Where do we begin, what steps should we follow, which tools should we use and how do we document our progress? We have to find out and make a system out of it. This is primarily done before attacking the object store so that we have a model to work from. Our model will be based on prior models combined with our own ideas. After testing the model on the Microsoft Mobile device we will discuss whether the model is general enough to work with other types of unknown devices as well.

1.5 Document Outline

Reverse engineering a system such as the object store requires us as the reverse-engineers to dive into detailed low-level concepts in order to be able to analyze the system. When presenting the work from this process it makes no sense not to go into these details, even though they may require extra attention from the reader. We are fully aware of this dilemma, and have therefore taken some precautions in order to make the presentation as readable as possible.

Chapter 1, ″Introduction″, starts with an introduction to the domain of forensic analysis, and presents the objective and focus of the project.

Before moving on to present our work, chapter 2, ″Work Method″, gives a brief insight into our process throughout the project. This includes how we moved from planning and brainstorming sessions, towards defining a methodology and finally the actual reverse engineering. This chapter will also note some of the tools and programming languages we learned in order to carry out the project.

In chapter 3, ″A Forensic Reverse Engineering Methodology″, we start presenting our work on a methodology. The chapter starts with a discussion of why such a methodology could prove beneficial. It then continues by introducing the existing methodologies we decided to build our work on. Then, towards the end of the chapter, a new methodology is defined, which is to be used in our own reverse engineering process.

Chapter 4, ″Reverse Engineering the Object Store″, is where we present the detailed analysis of the object store. In order to make this chapter as readable as possible, we try to present the work by continuously referring to the methodology from the previous chapter in order to make it easier for the reader to follow the process. We have also made figures and screenshots a high priority in situations where we found them useful to explain details. We believe that this has resulted in a readable presentation of these low-level details.

Chapter 5, ″Discussion″, discuss both of the two objectives of the project. It starts by discussing the results from the reverse engineering process, and then moves on to discuss our experience with the methodology. Some thoughts on future work are presented for both topics.

In chapter 6, ″Conclusion″, we sum up the project with some final conclusions.

Chapter 2

Work Method

This chapter gives an overview of how we have arranged our work.

Our task was to define and evaluate a methodology for forensic analysis of an unknown device. We started of with brainstorming sessions on how we should attack our task. We soon found that we wanted to define a stepwise approach built around general concepts. This approach should be developed to help assist us in performing and documenting forensic analysis of any unknown device.

Seeing no need to re-invent the wheel, we did a literature study on methodologies for forensic analysis and general reverse engineering. Based on this study we constructed what we felt was a good methodology for approaching an unknown device for analysis. This methodology and the basis for constructing it is found in chapter 3.

After having constructed our methodology, we tested it quite thoroughly on our unknown device in chapter 4. This testing involved black box testing and disassembling of the device. The black box testing was performed by giving known input and dumping the phones memory content. This dump was then analyzed with the help of a hex editor and its built-in structure definition library. Several small test programs were written in C++/C for Windows Mobile. We also developed a memory dump validation and analysis tool called BlobExtractor. For this we used Visual Studio .NET 2005 and C#. The disassembling phase also gave us considerable challenges. First we had to learn how to use the advanced Interactive DisAssembler (IDA). In order to synchronize the discoveries found with IDA with each other we had to modify a third party utility to work with our version of IDA. This utility, called ida_sync, is written in Python and C++. Next, in order to use IDA on the dll files from the unknown device and understand its output we also had to learn the ARM assembly language and the executable file format used on Windows Mobile. We also had to familiarize ourselves with Windows Mobile (Windows CE)'s internal architecture and memory layout in order to interpret what the ARM assembly instructions were actually doing to the phone. In the disassembly phase we also further enhanced BlobExtractor to make use of the new knowledge gained here.

The results from the reverse engineering phase and possible further enhancements to the model or its utilization was then documented in chapter 5.

Figure 2.1 shows the large picture steps in our work method. We only used our method for one main cycle as we only analyzed only one unknown device.

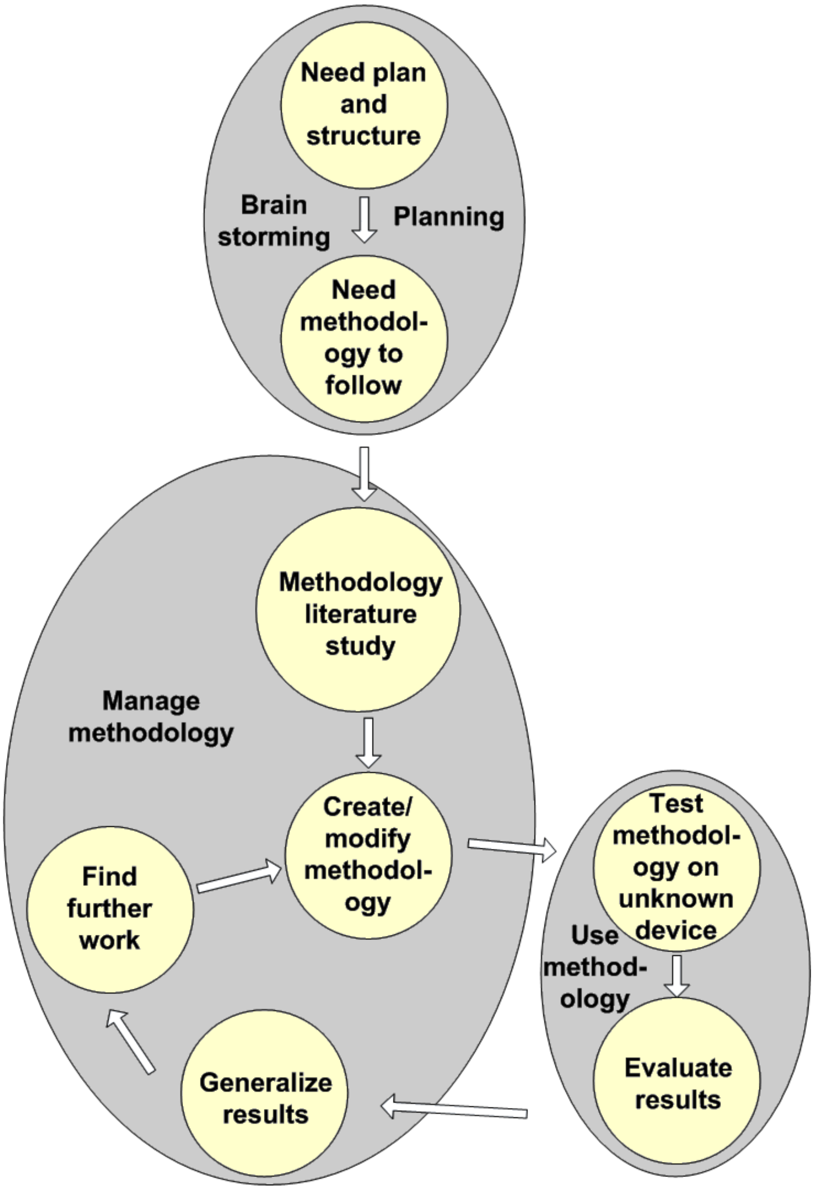


Figure 2.1: Work method.

Chapter 3

A Forensic Reverse Engineering Methodology

3.1 Benefits from Using a Methodology

Before going into the details of a reverse engineering methodology, we need to discuss why such a methodology could be beneficial. We believe that there are many aspects supporting the use of a defined methodology. The resources spent on developing the methodology itself are clearly justified by long-term benefits.

The new methodology should first of all build on techniques from existing related work. This could help avoid common pitfalls, and would take advantage of knowledge gained during years of research. This implies the use of well-defined modeling techniques, which in turn would bring several possible benefits to the table.

A standard indicating what to model would encourage the reverse-engineer to document all information found, and arrange it in a way that improves readability. Section 3.3.1 argues that the representation and arrangement of information is of great importance in a reverse engineering process.

Using well-defined modeling techniques also improves co-operation when several reverse-engineers work together on the same project. This could prove useful when time is a concern. It also gives the reverse-engineer easier access to the knowledge from other reverse engineering projects. When the same methodology is used over time, knowledge from previous projects become more available to upcoming projects. This is of particular interest to the forensic analysts, as the target devices often share characteristics.

Another benefit that could have great impact on both the quality and efficiency of the reverse engineering process is that using a standard methodology would lay the foundation for developing a more involving tool to be used throughout the process. The tool could feature many useful functions based on the methodology, including support for co-operation, automatic model creation, structured knowledge base arrangement, inconsistency checks, import of information from previous projects, resource gathering, report generation, and many more.

In general, considerable improvement in two vital areas could be the result of a defined methodology: quality and efficiency. Quality, because the reverse-engineer can follow a step by step procedure almost as a checklist, which guides the reverse-engineer using a quality-tested process. Also because the work of the reverse-engineer could more easily be quality-controlled afterwards. Efficiency, because the reverse-engineer can get into what he knows best, the actual reverse engineering, almost immediately. A substantial load from the planning phase of the project is removed, since this wheel has already been invented. The above discussion also mentioned several other aspects affecting the efficiency, including co-operation, using knowledge from other projects, and enabling advanced tools to be used.

From the forensic perspective, quality and efficiency is of great importance. The forensic analyst would need to document both the process and his findings, and a well-defined methodology could be of great assistance when defending the quality of the work. In addition, during an investigation, time is always an issue. If the forensic evidence could be discovered at an earlier stage, this could have a great impact on the investigation.

3.2 Exploring Previous Work

As discussed in the previous section, we wanted to build our new methodology on existing related work by taking advantage of well-defined standards and concepts. Our job was to gather only what we found most suitable for the methodology, and use these references as a guide.

3.2.1 Model-Driven Reverse Engineering (MDRE)

Model-Driven Reverse Engineering[8] is an ongoing research field, designed to overcome the difficulties of predicting the time consumption of a reverse engineering project, and evaluating the quality of the reverse engineering. As the name suggests, MDRE is based on using models to guide the reverse engineering process. The models are divided into different abstraction levels, and the key element of the process is to make connections between models at different abstraction levels.

MDRE is based on the Object Management Groups (OMG) Model-Driven Architecture (MDA). MDA is an approach to using models in software development. It is based on separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform. As MDA is designed for forward engineering, the idea of MDRE is to reverse processes described in MDA, but still use the same ideas and models. The following will include a short description of MDA concepts, taken from the MDA Guide Version 1.0.1 [9].

The Basic Concepts

System MDA concepts are presented in terms of some existing or planned system. That system may include anything, like: a program, a single computer system, some combination of parts of different systems, a federation of systems, each under separate control, people, an enterprise or a federation of enterprises.

Model A model of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in natural language.

Model-Driven MDA is an approach to system development, which increases the power of models in that work. It is model-driven because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification.

Architecture The architecture of a system is a specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors.

The Model-Driven Architecture prescribes certain kinds of models to be used, how those models may be prepared and the relationships of the different kinds of models.

Viewpoint A viewpoint on a system is a technique for abstraction using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within that system. Here abstraction is used to mean the process of suppressing selected detail to establish a simplified model. The concepts and rules may be considered to form a viewpoint language. The Model-Driven Architecture specifies three viewpoints on a system: a computation independent viewpoint, a platform independent viewpoint and a platform specific viewpoint.

View A viewpoint model or view of a system is a representation of that system from the perspective of a chosen viewpoint.

Platform A platform is a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented.

Application The term application is used to refer to a functionality being developed. A system is described in terms of one or more applications supported by one or more platforms.

Platform Independence Platform independence is a quality, which a model may exhibit. This is the quality that the model is independent of the features of a platform of any particular type. Like most qualities, platform independence is a matter of degree.

Computation Independent Viewpoint The computation independent viewpoint focuses on the on the environment of the system, and the requirements for the system; the details of the structure and processing of the system are hidden or as yet undetermined.

Platform Independent Viewpoint The platform independent viewpoint focuses on the operation of a system while hiding the details necessary for a particular platform. A platform independent view shows that part of the complete specification that does not change from one platform to another. A platform independent view may use a general purpose modeling language, or a language specific to the area in which the system will be used.

Platform Specific Viewpoint The platform specific viewpoint combines the platform independent viewpoint with an additional focus on the detail of the use of a specific platform by a system.

Computation Independent Model (CIM) A computation independent model is a view of a system from the computation independent viewpoint. A CIM does not show details of the structure of systems. A CIM is sometimes called a domain model and a vocabulary that is familiar to the practitioners of the domain in question is used in its specification. It is assumed that the primary user of the CIM, the domain practitioner, is not knowledgeable about the models or artifacts used to realize the functionality for which the requirements are articulated in the CIM. The CIM plays an important role in bridging the gap between those that are experts about the domain and its requirements on the one hand, and those that are experts of the design and construction of the artifacts that together satisfy the domain requirements, on the other.

Platform Independent Model (PIM) A platform independent model is a view of a system from the platform independent viewpoint. A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type.

A very common technique for achieving platform independence is to target a system model for a technology-neutral virtual machine. A virtual machine is defined as a set of parts and services (communications, scheduling, naming, etc.), which are defined independently of any specific platform and which are realized in platform-specific ways on different platforms. A virtual machine is a platform, and such a model is specific to that platform. But that model is platform independent with respect to the class of different platforms on which that virtual machine has been implemented. This is because such models are unaffected by the underlying platform and, hence, fully conform to the criterion of platform independence.

Platform Specific Model (PSM) A platform specific model is a view of a system from the platform specific viewpoint. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.

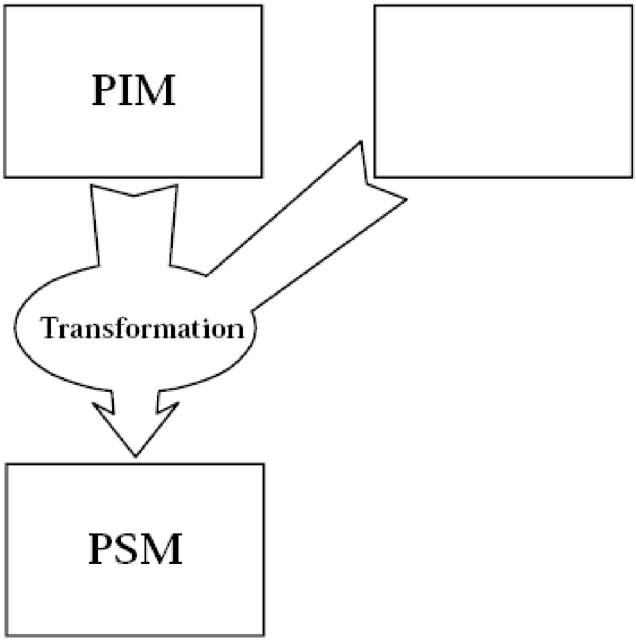


Figure 3.1: MDA transformation.

Model Transformation

Model transformation is the process of converting one model to another model of the same system. Figure 3.1 illustrates the MDA pattern, by which a PIM is transformed to a PSM.
The drawing is intended to be suggestive. The platform independent model and other information are combined by the transformation to produce a platform specific model.

The drawing is also intended to be generic. There are many ways in which such a transformation may be done. However it is done, it produces, from a platform independent model, a model specific to a particular platform.

3.2.2 UML and RM-ODP Viewpoints

UML

The Unified Modeling Language (UML)[10] is OMGs most-used specification. It is used to model not only application structure, behavior, and architecture, but also business process and data structure. The latest version of the specification, UML 2.0, defines thirteen types of diagrams, divided into three categories:

Structure Diagrams include the Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram, and Deployment Diagram.

Behavior Diagrams include the Use Case Diagram (used by some methodologies during requirements gathering), Activity Diagram, and State Machine Diagram.

Interaction Diagrams, all derived from the more general Behavior Diagram, include the Sequence Diagram, Communication Diagram, Timing Diagram, and Interaction Overview Diagram.

UML form a foundation for MDA, and can be used for CIM, PIM, and PSM.

RM-ODP Viewpoints

The Reference Model For Open Distributed Processing (RM-ODP)[11] was a joint effort by the international standards bodies ISO and ITU-T to develop a coordinating framework for the standardization of open distributed processing (ODP). In a world of interconnected computer systems, heterogeneity in interaction models prevents interworking between systems. RM-ODP targets this dilemma with an architecture that supports distribution, interworking, interoperability, and portability.

The RM-ODP framework defines ODP concerns using five viewpoints: enterprise, information, computational, engineering, and technology.

Enterprise view Three keywords describe the enterprise viewpoint: *purpose*, *scope*, and *policies*. It focuses on the environment and general organization of the system, concerned with *objects*, *communities*, and the *roles* of the objects within the communities.

Information view The information viewpoint focuses on the semantics of information and information processing, divided into three schemas. A *static schema* captures the state and structure of an object, typically represented by an object diagram. An *invariant schema* restricts the state and structure of an object, e.g. by a class diagram with associations and other constraints. A *dynamic schema* defines the permitted change in the state and structure of an object in a behavior specification.

Computational view The computational viewpoint is an object-based, modular view. A computational specification defines the objects within an ODP system, the activities within those objects, and the interactions that occur among objects.

Engineering view The engineering viewpoint focuses on the mechanisms and functions required to support distributed interactions between object in the system.

Technology view The technology viewpoint focuses on the choice of technology in the system.

MDA uses these ODP viewpoints as a guide for the models at different abstraction levels. A CIM of a system may include several models, based on the enterprise and information viewpoints. A PIM uses models based on the enterprise, information, and computational viewpoints. The more specific viewpoints, engineering and technology, are left for the PSM.

3.2.3 Other Reverse Engineering Methodologies

In "A Reverse Engineering Mehodology For Data Processing Applications" [12] K. Spencer and S. Rugaber from the Software Research Center at Georgia Institute of Technology defines a reverse engineering methodology based on four phases: a documentation review, an analysis of system input/output structure, an analysis of the structure of the input and output files, and a detailed analysis of the source code using a technique called Synchronized Refinement. It should be noted that the methodology was tested on a system targeted for redesign, with reasonably accurate documentation available, and also access to all source code, which the article claims was cleanly structured.

They start with a review of existing documents. The purpose of this phase is to establish an overview and a functional description of the system, ignoring the implementation details.

They continue with an analysis of the systems input/output behavior. The key concept here is data flow diagrams. The top level diagram is called a Context Diagram, which only include one activity, the system itself, and all external files as repositories with the direction of the arc denoting whether the file is used as input or output. The context diagram is then verified by examining source code. The data flow diagrams can be nested, to describe different levels in the system. That is, a process node at one level can be expanded into an entire diagram at a lower level. The nesting of the diagrams proceeds until all system input/output behavior has been described.

The next phase consists of an analysis of the structure of the files used in the system. The analysis is expressed in terms of Jackson Data Structure diagrams[13], which describes the file as a tree-structured collection of boxes.

The final phase uses a technique called Synchronized Refinement. Synchronized Refinement is a code reading technique developed by Rugaber et al. that simultaneously examines and abstracts source code while elaborating an application description. It is used to obtain a detailed description of a specific function. Captured understanding is expressed in terms of how identified code constructs realize specific application domain concepts. The process is driven by the detection of design decisions in the source code. The key concepts are *recognition* and *abstraction*. When a design decision is recognized and annotated, the code segment representing the design decision is replaced by a description of what it does. This way, the source code gets shorter, while the description of the system grows.

The process begins with a high-level description obtained from the documentation review. This description leads to some expectations about the system. A dynamic list of expectations is kept, while exploring expectations by examining the source code. This may lead to some expectations being discarded, while new expectations emerge. This process is continued until enough connections have been made between the high-level and low-level descriptions to cover the system being analyzed.

3.3 Defining a Methodology

This section will introduce our new methodology for forensic reverse engineering. As mentioned, it is based on concepts from other methodologies and standards presented in the previous section.

3.3.1 Guidelines for a New Methodology

As part of our preparations for the development of this methodology, we made some guidelines to lead us in our development. The guidelines are based on a combination of experiences from similar projects studied and our own thoughts on important aspects of such a methodology.

First of all, the methodology needs to be specific enough to be used as a guide through a reverse engineering process, while at the same time be general enough to be used on entirely different systems.

”Program Comprehension For Reverse Engineering” [14] discuss some concerns related to program comprehension, and states that reverse engineering is difficult because of the need to bridge different worlds. Of particular importance, they mention five gaps:

- The gap between a problem from some application domain and a solution in some programming language.
- The gap between the concrete world of physical machines and computer programs and the abstract world of high level descriptions.
- The gap between the desired coherent and highly structured description of the system and the actual system whose structure may have disintegrated over time.
- The gap between the hierachical world of programs and the associational nature of human cognition.
- The gap between the bottom-up analysis of the source code and the top-down synthesis of the description of the application.

[14] states that these difficulties manifest themselves in three ways: lack of a systematic methodology, lack of an appropriate representation for the information discovered during reverse engineering, and lack of powerful tools to facilitate the reverse engineering process.

The lack of a systematic methodology is the main objective of this project, and we will target this problem directly.

We also want to target the problem with lack of appropriate representation. We believe that in a reverse engineering project, having a clear and organized overview of what is known at all times may be the difference between success and failure. New information needs to be connected to what we already know. To do this, the representation of information will be important to decrease the chances of overlooking possibly important new information. We will not, however, go into the details of any requirements or languages used in the representation of information, but acknowledge its importance, encourage the reverse-engineer to see its importance, and build a methodology that supports this idea and future work on a specific representation.

Finally, we want to target the problem related to a lack of supporting tools indirectly by developing a methodology suitable to lay the foundation for such tools to be developed based on the it.

3.3.2 Sketching a Forensic Reverse Engineering Methodology

It is time to start presenting the new methodology. First of all, the methodology is based on the ideas from Model-Driven Reverse Engineering and the Model-Driven Architecture, presented in section 3.2.1. We want the main focus of our methodology to be aimed at the Computation Independent Model, the Platform Independent Model, the Platform Specific Model, and the mappings between these models. These models are general enough to be adjusted to any reverse engineering problem, but still specific enough to lay the foundation for the reverse engineering process. One of our main thoughts behind the idea of building the methodology around such models is to encourage the reverse-engineer to document all stages of the process. This is particularly important in a forensic setting.

In most cases, the reverse-engineer will have some level of knowledge about the domain in question. That is, if the target device was a digital camera, depending on experience, the reverse-engineer would immediately expect to find some sort of storage chip, a file system, some algorithms and parameters related to interpolation, compression, etc. This knowledge is part of the foundation for building the CIM. The other part is gained from a documentation review. As with the methodology presented in 3.2.3, developed by K. Spencer and S. Rugaber, we want a documentation review to be one of the initial stages of the process. This should support the building of the CIM, and also construct an initial knowledge base that at any stage contains all the information known about the system so far. But, as opposed to Spencer and Rugaber's methodology, we believe that the choice of strategy and techniques used in the reverse engineering process should be chosen after the documentation review. While they defined their strategy at the very beginning of the project, we believe that the reverse-engineer can come to a better conclusion about his strategy after having gained more knowledge about the system. In addition, we don't want the strategy to be static, but instead use a dynamic approach where strategies may be swapped as the process reveals new information about the system.

When a CIM of the system has been built, the fundamental idea is to pick strategies and techniques for reverse engineering and gradually build a PSM of the system. The CIM works as a high-level guide in the reverse engineering, guiding the choice of attack-angles, and giving a better understanding of the revealed information. From the general ideas of the CIM to the specific information represented in the PSM, the goal is to abstract the PSM to create a PIM. This is done by mappings from the PSM to the PIM, based on recognizing the intended purpose of PSM concepts. The process can be seen as the opposite of how MDA uses a PIM to transform it into a PSM. To validate the result of the reverse engineering, we want to use the idea of "adequate reverse engineering" from "Model-Driven Reverse Engineering" [8]. The article compares adequate reverse engineering with how the term adequacy is used in software testing. Testers use various adequacy criteria to ensure that requirements have been met. These criteria derive their benefit from being deterministic and measurable.

The article further states that "if adequacy criteria existed for reverse engineering, then software engineers could start collecting experience reports and building databases of project statistics to help predict reverse engineering time and effort." Two characteristics are introduced to form the basis for adequacy: *thoroughness* and *lucidity*. *Thoroughness* is "the extent to which the reverse engineering covers the entire system being examined." *Lucidity* is "the extent to which the reverse engineering sheds light on the purpose of the system and how that purpose is accomplished by the code." The intension is to use the models to help measure the thoroughness and

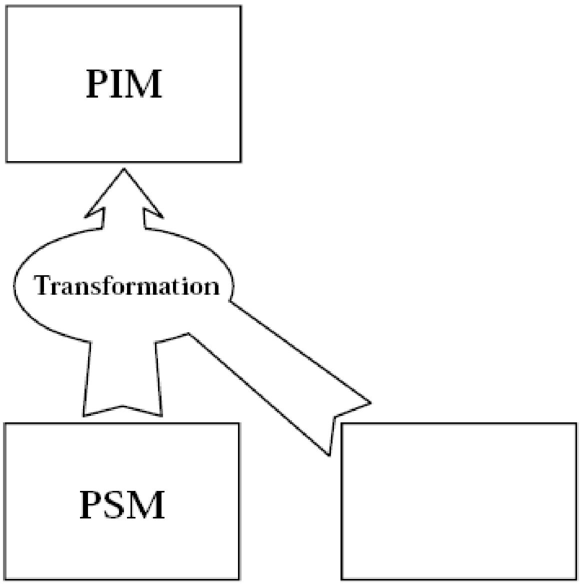


Figure 3.2: MDRE transformation (modified version of figure 3.1, taken from [9]).

lucidity of the system. To do this, they introduce the term ″reverse reverse engineering.″ Reverse reverse engineering is used for validation, by using the resulting models from the reverse engineering to build a new implementation of the system, hence reversing the reverse engineering. The idea is to compare this new version with the original, to determine if the match is close enough. The article discusses reversing a software program, using a code generation tool for the reverse reverse engineering task, generating the code automatically from a standard representation of the models and comparing the results. In our situation, the system may not be a software program, but the general idea of reverse reverse-engineering still hold.

Our methodology is presented with the flow chart shown in figure 3.3, and a step by step explanation is given below.

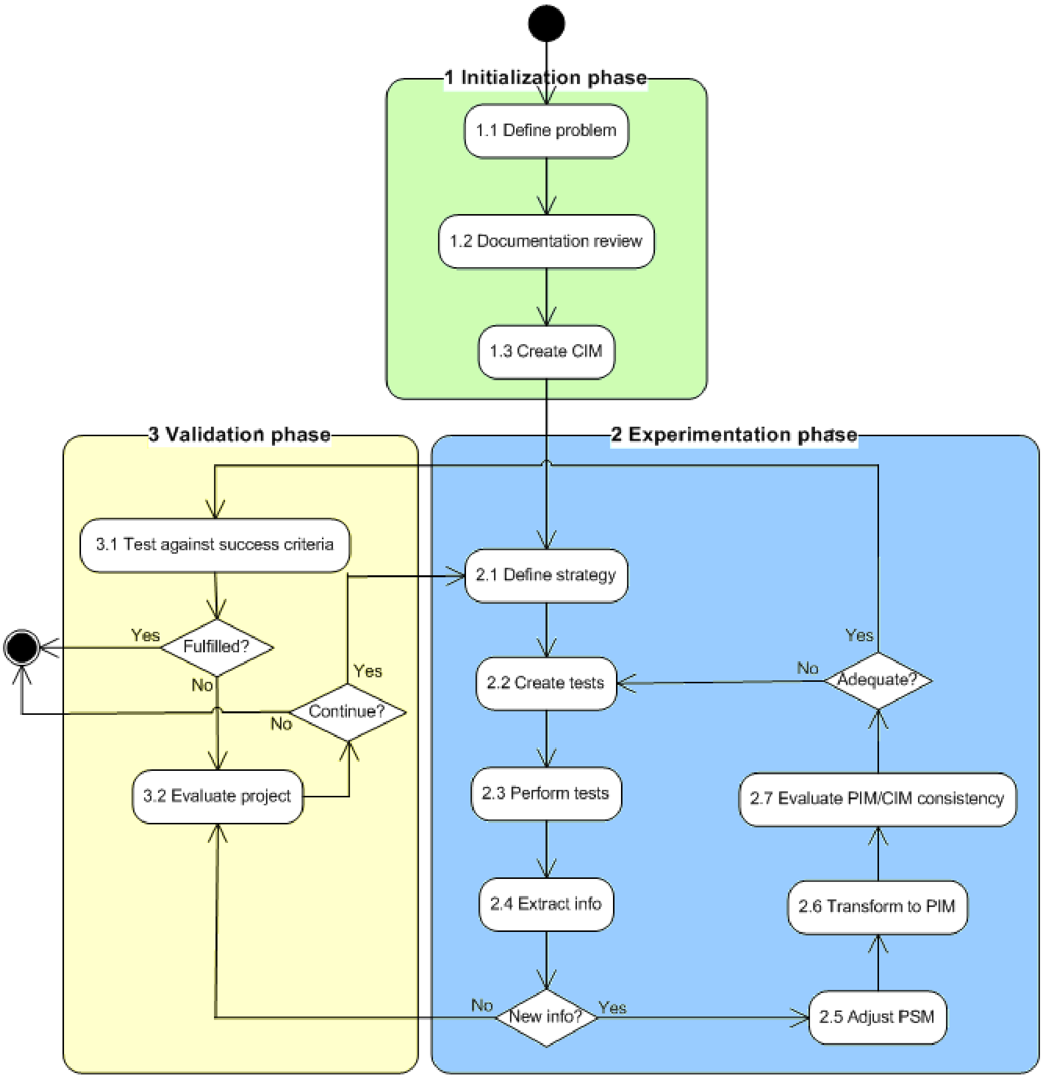


Figure 3.3: Flow chart of our methodology.

1. Initialization phase This phase includes the initial preparations before moving on to analyze the system itself. At the end of this phase, the reverse-engineer should have a general idea of the functioning of the target system.

1.1 Define problem

The first step of the methodology is to define the problem, in order to get a clear understanding of what to target. The important part in this step is to limit the problem to only include what is necessary. This step should include a definition of the success criteria.

1.2 Documentation review

The documentation review was discussed above. The goal of this step is to get an initial understanding of the problem and its domain. The knowledge gained during the documentation review will be the basis for three important aspects of the methodology:

1. A knowledge base is formed based on the information found. This knowledge base should contain all known information at any time, and represent the reverse-engineers current understanding of the system.
2. A CIM is created based on a general idea of the domain of the system.
3. The first strategy chosen in the *Experimentation phase* is based on knowledge from the documentation review.

1.3 Create CIM

A CIM should be created based on the domain knowledge gained during the documentation review. The CIM will be used throughout the process to guide what part of the system to address at any time and to validate the models created for consistency. The CIM can change during the process, in cases where the actual domain differs from the expected.

2. Experimentation phase
This is the phase where all tests and experiments on the system are performed. The idea is to increase the knowledge about the system incrementally by continuously making new connections between new information

and the current state of the knowledge base.

2.1 Define strategy

When the reverse-engineer has gained a general impression of the system, it is time to start analyzing it. The choice of strategy will vary depending on the system and the information found during the documentation review. A chosen strategy is kept as long as it produces new information, but a strategy could be re-used at some later stage if new information from other strategies causes this strategy to be valuable again. Examples of strategies, which will be seen in action in our case study, are black-box testing and disassembling.

2.2 Create tests

At this stage, tests are created according to the chosen strategy. The tests should be targeted at a particular part of the CIM they intend to explore.

2.3 Perform tests

Tests are performed according to the plan created in the previous step.

2.4 Extract info

Information is extracted from the results of the tests.

2.5 Adjust PSM

If new information was found in the previous step, the information is incorporated into the PSM.

2.6 Transform to PIM

When new information is incorporated into the PSM, we try to abstract the PSM into a PIM. The CIM helps us understand the concepts explored in the PSM.

2.7 Evaluate PIM/CIM consistency

As the PIM grows, we evaluate the consistency between the PIM and the CIM to see if we have covered the entire system. This corresponds to the discussion of *adequate reverse engineering* above, and the characteristics *thoroughness* and *lucidity*. If we believe to have covered what we need, we move on to test against our success criteria. If we have not yet covered everything, we go back to create a new set of tests to test new aspects of the system.

3. Validation phase

There are two ways to enter this phase. Either you believe to have covered the entire system in question, and need to test against the success criteria. Or you have performed a set of tests that gave no new information.

3.1 Test against success criteria

When the reverse engineering is believed to be adequate, it is time to test against the success criteria. This is done by *reverse reverse engineering*, to see if our understanding of the system is accurate. This stage could either end with the conclusion that the project is successfully completed, or if the test fails, a valuation of the project is needed to determine if the project should be closed, or if we should continue analyzing the system.

3.2 Evaluate project

When this stage is reached, the question to answer is whether or not we still see possibilities of finding new information about the system. We could reach this stage either by failing to match the success criteria, or by failing to extract new information from a given set of tests. In the latter case, we would normally decide to continue, but with a different strategy, a new test set, or a different target for the test set. This is unless we have failed to extract information from several strategies in a row, and see no reason to continue the project. On the other hand, when the stage is reached from failing to meet the success criteria, we should perform a more detailed evaluation of the project to determine if and how we are likely to reach a more accurate understanding of the system.

Chapter 4

Reverse Engineering the Object Store

Now that a sketch of a methodology has been developed, we need a system to test the methodology on. The system chosen for this task was Windows Mobile's object store. This choice was made not only because the object store suits as a good candidate for testing the methodology, but also because knowing the inner workings of the object store could be an important forensic discovery in itself.

4.1 Initialization Phase

4.1.1 Defining the Problem

The system to be analyzed is Windows Mobile's object store. We are not concerned with Windows Mobile functionality that does not affect the object store. We seek to get enough information about the structure of the object store to be able to locate objects and their data using only our own program code. We will base our work on the acquisition techniques discussed in "Mobile Forensics" [6], which also showed examples of recognizable data in the object store. We wish to get a good understanding of the data related to the objects, in order to extract the important parts of information. We should also be able to distinguish between objects that are deleted and those that are not. More precisely, we want to be able to locate all unused areas of the object store, which are the areas where deleted information can be found.

We defined three success criteria for the analysis:

1. Be able to distinguish between deleted and non-deleted data.
2. Understand the format used to store objects, in order to extract data and attributes.
3. Make sure that we have covered the entire object store.

All tests will be performed on a Qtek S110[15] device (Appendix A).

4.1.2 Documentation Review

We started off with an in-depth documentation review. Our focus was on gathering as much relevant information as possible in order to establish a knowledge base for the upcoming process. This included:

"Mobile Forensics" The work presented by Jarle Eide in the report named "Mobile Forensics"[6] was available. The report gave an introduction to forensic analysis of Qtek S110.

Qtek S110 basics The basics of Qtek S110 (appendix A) were known from "Mobile Forensics".

Windows Mobile An overview of Windows Mobile was also presented in "Mobile Forensics". This included some basic knowledge of the object store. Microsoft entered the handheld market with the Windows CE operating system. Since then, many different platforms have been based on the core Windows CE functionality. Windows Mobile is the name of a subset of these platforms, including Pocket PC and Smartphone. The Pocket PC operating system evolved along with Windows CE from WinCE 2.0/PPC 2000 to WinCE 3.0/PPC 2002 to WinCE 4.2/PPC 2003. PPC 2003 was re-branded as Windows Mobile 2003, and with some additional functionality a new version was released as Windows Mobile 2003 Second Edition in March 2004.

Windows CE API The Windows CE API was available from the Microsoft Developers Network (MSDN) website [16].

ARM instruction set As Qtek S110 uses an ARM-processor, the ARM instruction set was available in case we needed to use a disassembler. See appendix C for a ARM instruction Set Quick Reference found at [17]. For a thorough explanation of the ARM processor architecture see [18].

Shared Source Initiative Some of the Windows CE source code are available through Microsoft's Shared Source Initiative[2].

In addition, we had several tools available:

Itsutils The Itsutils tools [7] were available, which provide useful functionality when working with Windows Mobile.

IDA PRO Disassembler The IDA PRO Disassembler (4.2.5) was made available to us from the university, which made it possible to disassemble system files if necessary.

Judas Forensic Tool The Judas Forensic Tool was available from ”Mobile Forensics”, in order to make complete bit-by-bit images of memory on the phone.

4.1.3 Creating CIM

To represent our expectations of the domain, we created the CIM shown in figure 4.1. The model was fairly simple, containing two main components: an object allocation table and the object as found in the object store.

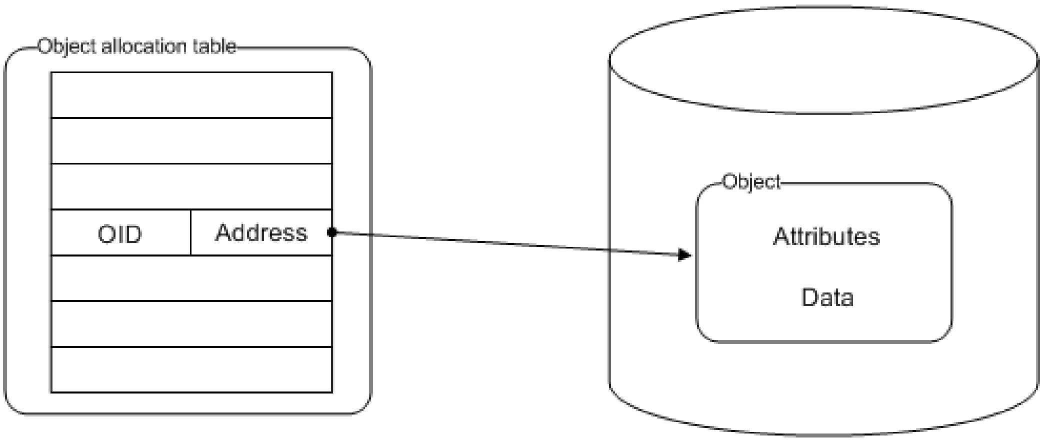


Figure 4.1: Computation Independent Model

From the documentation review we knew that object identifiers (OID), a unique 32-bit numerical value, were used to identify objects. We expected to find some sort of allocation table, mapping these identifiers to the locations of the objects.

Many examples of stored objects had been presented in [6]. We expected to find both object attributes and the data stored together according to some unknown format, where the data could possibly be compressed.

4.2 Exploration Phase

The initialization phase had given us the basic system understanding we needed to start the analysis. According to the methodology, the next step was to define a strategy. We will present our analysis by continuously referring to the steps defined in the methodology. However, steps such as *2.7 Evaluate PIM/CIM consistency* will not be included in the early stages where every such evaluation concluded that our knowledge was still inadequate. In addition, we will not show the results from *3.2 Evaluate project* every time no new information was found, since this step, though useful during the process, does not provide any information valuable for the presentation.

4.2.1 Defining a Strategy - First Loop

"Mobile Forensics" ([6]) had shown that memory dumps from Qtek S110 could provide a complete bit-by-bit copy of the object store, and also provided many examples on how investigation of such memory dumps could reveal information about the contents of the object store. It seemed like a good idea to continue this work. But while "Mobile Forensics" sought to determine if it was at all possible to re-discover previously deleted data, we were now interested in finding exactly how objects are stored, and determine what actually happens when objects are deleted.

We summarized our situation with the following:

- have an "unknown" device, with unknown behavior
- have the ability to input data to the device
- have the ability to delete data from the device
- have the ability to reset the device to its initial state
- have the ability to make a complete bit-by-bit copy of data on the device

From this we decided that our first strategy should be to run *black box tests*, treating the phone as the black box, and analyze the changes seen in memory dumps as input is introduced.

4.2.2 About Black Box Testing

Black box testing[19] is a well known testing methodology. It gets its name from the fact that it treats the system you test as a black box you can not look inside. The inner workings of the system are not known by the people doing the test. The testing consists of giving the system certain controlled inputs and comparing the resulting outputs to the functional specification. Because of its dependency on the functional requirements, black box testing is also often called functional, behavioral, opaque box and closed box testing. Because testing every possible permutation of inputs to a device is extremely time-consuming and not practical in any realistic scenario, there are some techniques available to reduce the input testing space.

Equivalence sets Also called *equivalence partitioning*. This technique tries to partition the possible input set into subsets that are expected to test the system in the same way. You then make sure you select at least 1 input from all the subsets. It's optimal to design the sets such that all values belong to one set and one set alone. The difficulty with this technique is choosing the correct strategy for set partitioning without knowing anything about the inner workings of the component.

Limit testing Also called boundary value analysis. This technique tests the limits of the input set. These limits are set by the input type and the input domain. If you have a black box device that it supposed to give you the square root of its input you would check the limits -1, 0, 1 and the max value of the input type. The downside of this technique is that few tests are generated and you can miss essential parts of the black box inner workings.

Black box testing has a natural counterpart in white box testing, where one uses information about the structure of the program to check how it functions. White box testing is better suited for times when one has access to the systems detailed plans and/or source code. With an unknown device this is not very likely. The difference between black box and white box testing is illustrated in figure 4.2.

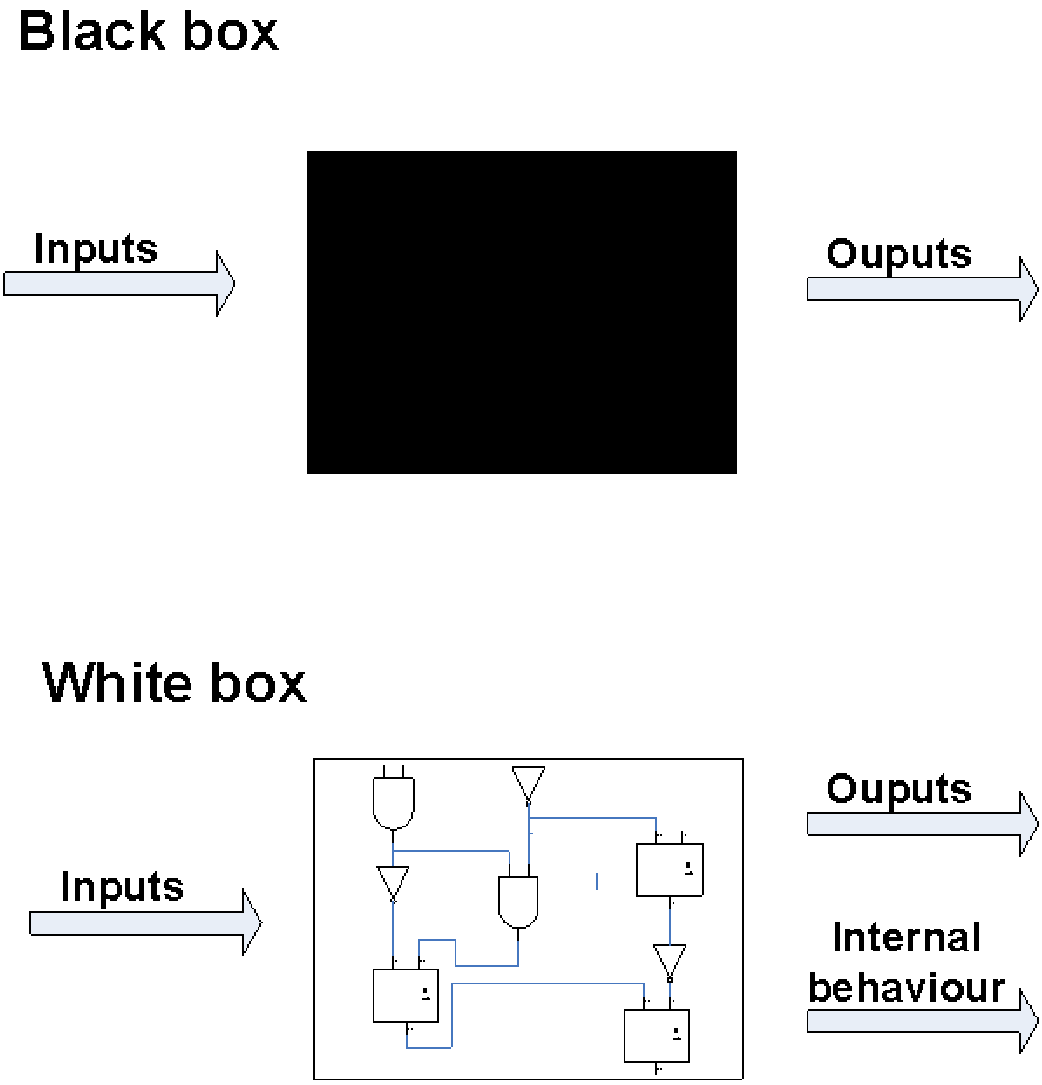


Figure 4.2: Black box and white box testing.

Hex Workshop (HW)

Hex Workshop is a hex editor. It displays data from files in both their raw hexadecimal numeric representation and the visual symbols associated with these representations through standards like ASCII. It gives the user the ability to view, find and edit the data. Hex Workshop also has advanced features like its own structure language, bookmarks and comparison between different files.

Concept clarifications

To help the readability of the rest of the thesis we define a couple of key concepts that appear in the text:

0x All numerical values prefixed with *0x* are in base 16, hexadecimal.

byte An unsigned byte, 8 bits wide.

word An unsigned word, 16 bits wide.

dword An unsigned double word, 32 bits wide.

quad An unsigned quadruple word, 64 bits wide.

little-endian/big-endian This concerns how data is stored in a computer. *Little-endian* means the LSB is stored first and *big-endian* means the MSB is stored first. Given the value *0x4A3B2C1D*, it is stored as *0x1D2C3B4A* in *little-endian* and as *0x4A3B2C1D* in *big-endian.*

id, oid, OID, CEOID, identifer All these are meant as the 32 bits wide numerical identifer of an object in the object store.

4.2.3 Testing

The ultimate goal of our testing was to figure out the physical and logical format our device use in order to store objects in the object store. The ability to do this is important because one is no longer limited by the official

APIs, but can access all the raw data and might extract more information than the API makes available.

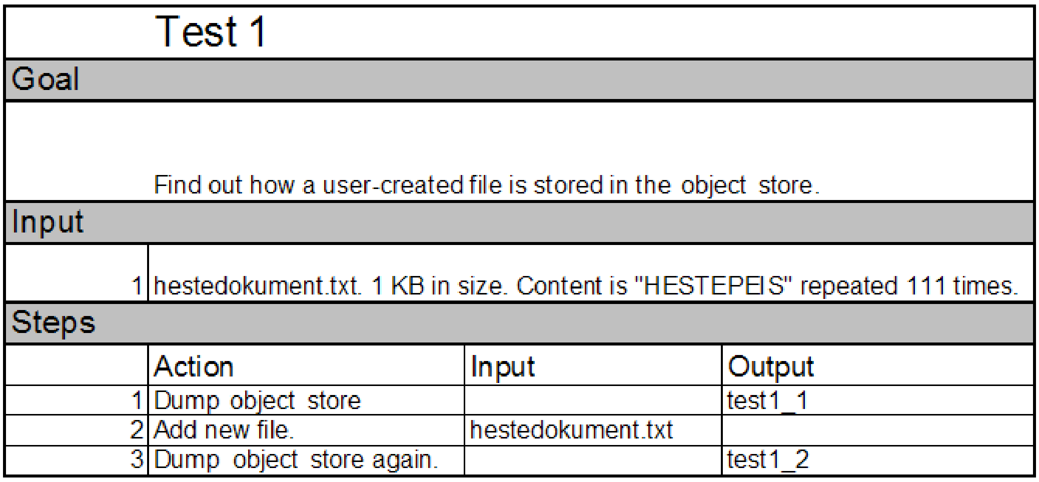
Referring to the methodology as shown in figure 3.3, the following will include the tests executed during this first choice of strategy.

4.2.3.1 Test 1

Create test

We want to find out how a user-created file is stored in the object store. We assume the data bytes in the file are stored in sequence and that metadata about that file is also somehow saved. We also assume that there exists some way to link the data and the metadata to each other.

Figure 4.3 contains the first test.

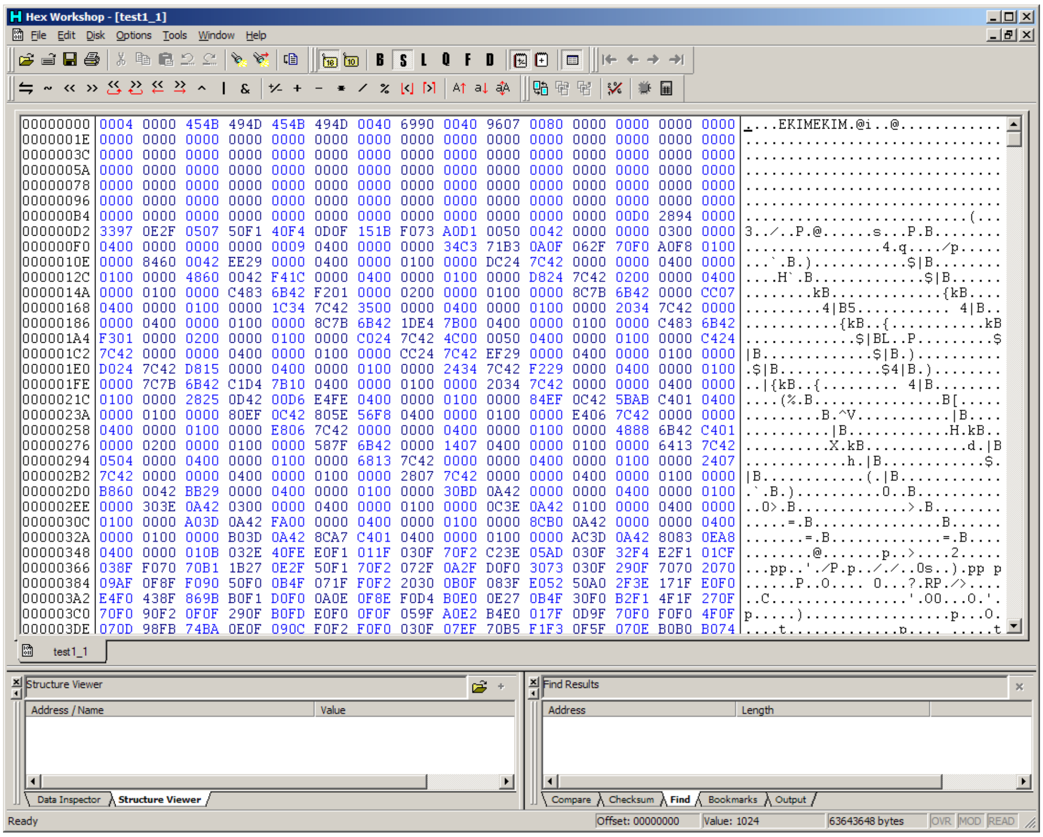


	Test 1		
Goal			
	Find out how a user-created file is stored in the object store.		
Input			
1	hestedokument.txt. 1 KB in size. Content is "HESTEPEIS" repeated 111 times.		
Steps			
	Action	Input	Output
1	Dump object store		test1_1
2	Add new file.	hestedokument.txt	
3	Dump object store again.		test1_2

Figure 4.3: Test 1 - User-created file.

Extract info/Adjust PSM

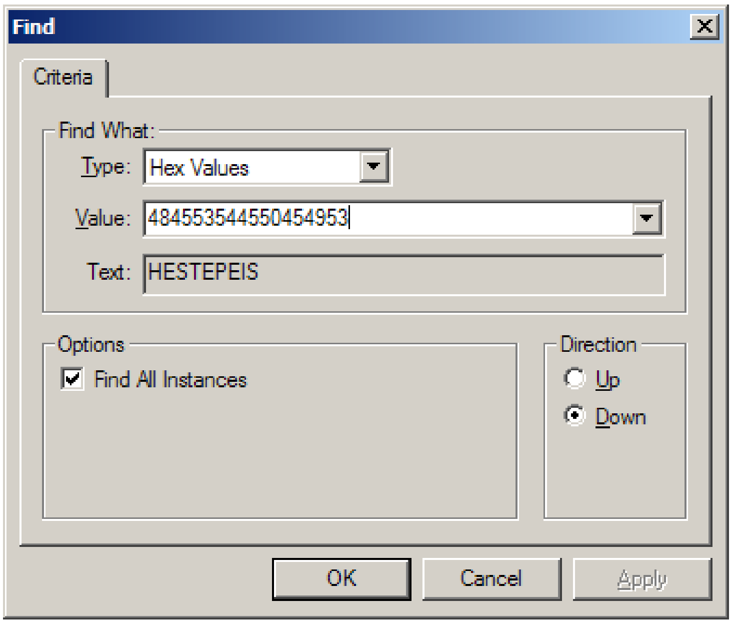
Our tool of choice was Hex Workshop. Opening *test1_1.bin* in it we got the default view in figure 4.4. The first thing we notice is the hex value at offset 0x4; *0x454B494D454B494D* or the text version *"EKIMEKIM"*. These bytes are always present first in the object store as a "magic number" identifier. This is not officially documented by Microsoft, but was found to al-



```
00000000 0004 0000 454B 494D 454B 494D 0040 6990 0040 9607 0080 0000 0000 0000 0000 ....EKIMEKIM.@i..@...........
0000001E 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .............................
0000003C 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .............................
0000005A 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .............................
00000078 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .............................
00000096 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .............................
000000B4 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 00D0 2894 0000 .........................(...
000000D2 3397 0E2F 0507 50F1 40F4 0D0F 151B F073 A0D1 0050 0042 0000 0000 0300 0000 3../..P.@......s...P.B........
000000F0 0400 0000 0000 0000 0009 0400 0000 0000 34C3 71B3 0A0F 062F 70F0 A0F8 0100 ................4.q..../p.....
0000010E 0000 8460 0042 EE29 0000 0400 0000 0100 0000 DC24 7C42 0000 0000 0400 0000 ...`.B.)...........$|B........
0000012C 0100 0000 4860 0042 F41C 0000 0400 0000 0100 0000 D824 7C42 0200 0000 0400 ....H`.B............$|B......
0000014A 0000 0100 0000 C483 6B42 F201 0000 0200 0000 0100 0000 8C7B 6B42 0000 CC07 ........kB.............{kB....
00000168 0400 0000 0100 0000 1C34 7C42 3500 0000 0400 0000 0100 0000 2034 7C42 0000 .........4|B5........... 4|B..
00000186 0000 0400 0000 0100 0000 8C7B 6B42 1DE4 7B00 0400 0000 0100 0000 C483 6B42 ...........{kB..{..........kB
000001A4 F301 0000 0200 0000 0100 0000 C024 7C42 4C00 0050 0400 0000 0100 0000 C424 .............$|BL..P.........$
000001C2 7C42 0000 0000 0400 0000 0100 0000 CC24 7C42 EF29 0000 0400 0000 0100 0000 |B.............$|B.)..........
000001E0 D024 7C42 D815 0000 0400 0000 0100 0000 2434 7C42 F229 0000 0400 0000 0100 .$|B............$4|B.)........
000001FE 0000 7C7B 6B42 C1D4 7B10 0400 0000 0100 0000 2034 7C42 0000 0000 0400 0000 ..|{kB..{......... 4|B........
0000021C 0100 0000 2825 0D42 00D6 E4FE 0400 0000 0100 0000 84EF 0C42 5BAB C401 0400 ....(%.B...............B[.....
0000023A 0000 0100 0000 80EF 0C42 805E 56F8 0400 0000 0100 0000 E406 7C42 0000 0000 .........B.^V...........|B....
00000258 0400 0000 0100 0000 E806 7C42 0000 0000 0400 0000 0100 0000 4888 6B42 C401 ..........|B............H.kB..
00000276 0000 0200 0000 0100 0000 587F 6B42 0000 1407 0400 0000 0100 0000 6413 7C42 ..........X.kB............d.|B
00000294 0504 0000 0400 0000 0100 0000 6813 7C42 0000 0000 0400 0000 0100 0000 2407 ............h.|B............$.
000002B2 7C42 0000 0000 0400 0000 0100 0000 2807 7C42 0000 0000 0400 0000 0100 0000 |B............(.|B............
000002D0 B860 0042 BB29 0000 0400 0000 0100 0000 30BD 0A42 0000 0000 0400 0000 0100 .`.B.)..........0..B..........
000002EE 0000 303E 0A42 0300 0000 0400 0000 0100 0000 0C3E 0A42 0100 0000 0400 0000 ..0>.B.............>.B........
0000030C 0100 0000 A03D 0A42 FA00 0000 0400 0000 0100 0000 8CB0 0A42 0000 0000 0400 .....=.B...............B......
0000032A 0000 0100 0000 B03D 0A42 8CA7 C401 0400 0000 0100 0000 AC3D 0A42 8083 0EA8 .......=.B.............=.B....
00000348 0400 0000 010B 032E 40FE E0F1 011F 030F 70F2 C23E 05AD 030F 32F4 E2F1 01CF ........@.......p..>....2.....
00000366 038F F070 70B1 1B27 0E2F 50F1 70F2 072F 0A2F D0F0 3073 030F 290F 7070 2070 ...pp..'./P.p../../..0s..).pp p
00000384 09AF 0F8F F090 50F0 0B4F 071F F0F2 2030 0B0F 083F E052 50A0 2F3E 171F E0F0 ......P..0.... 0...?.RP./>....
000003A2 E4F0 438F 869B B0F1 D0F0 0A0E 0F8E F0D4 B0E0 0E27 0B4F 30F0 B2F1 4F1F 270F ..C................'.00...O.'.
000003C0 70F0 90F2 0F0F 290F B0FD E0F0 0F0F 059F A0E2 B4E0 017F 0D9F 70F0 F0F0 4F0F p.....)..................p...O.
000003DE 070D 98FB 74BA 0E0F 090C F0F2 F0F0 030F 07EF 70B5 F1F3 0F5F 070E B0B0 B074 ....t..............p.... .....t
```

Ready Offset: 00000000 Value: 1024 63643648 bytes

Figure 4.4: Test 1 - Hex Workshop default view.



Find What:

Type: Hex Values

Value: 484553544550454953

Text: HESTEPEIS

Options

☑ Find All Instances

Direction

○ Up

● Down

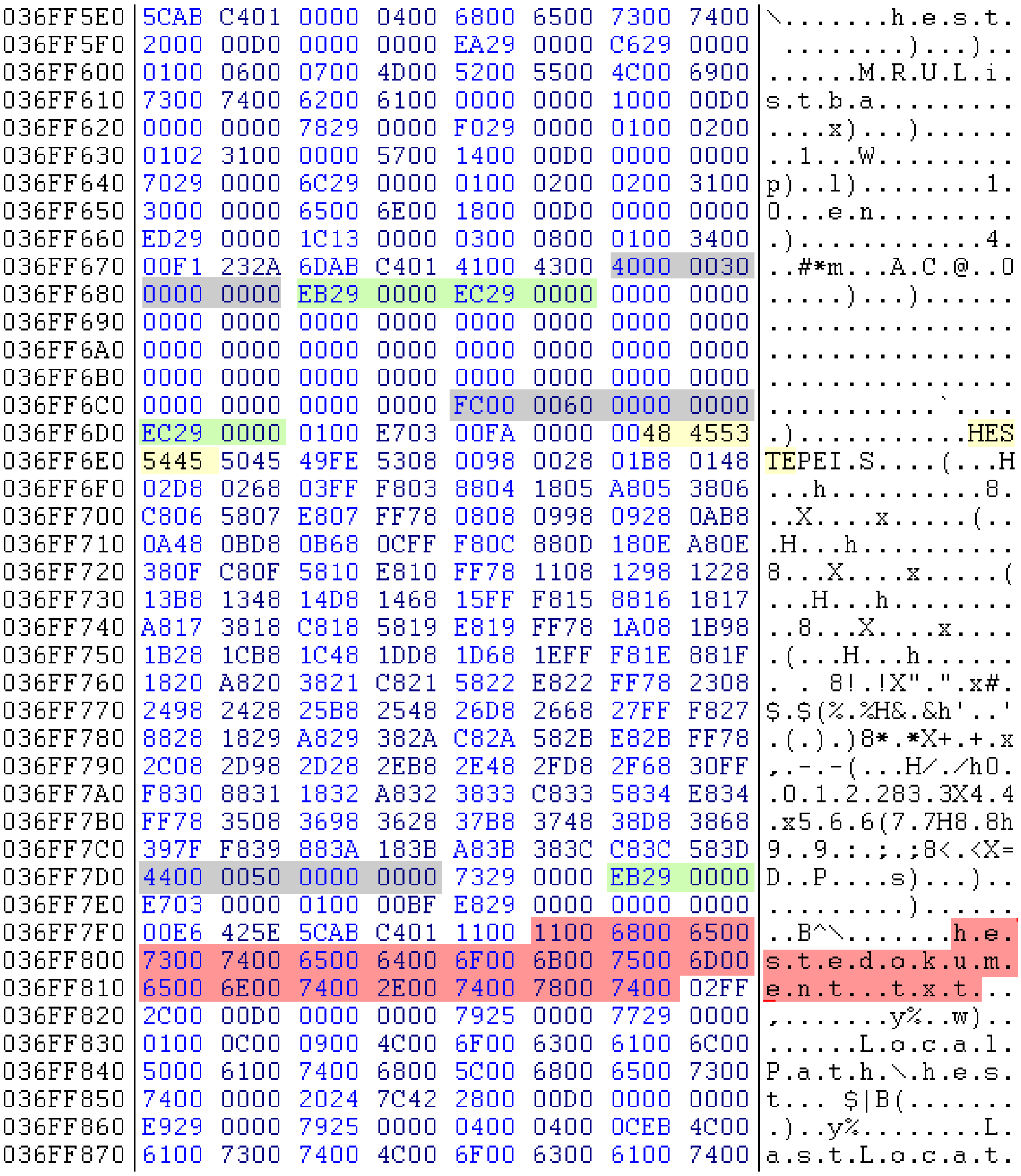
Figure 4.5: Test 1 - Hex Workshop search box.

ways be present by Jarle Eide in his "Mobile Forensics"[6]. We will use the presence of this value as a strong indicator that we have a memory dump from the correct location.

We also notice that there is a strong repeating pattern of the data in the area of memory from offset 0xE0 to offset 0x34C. Exactly what this data is is unknown at this point but its noted as an interesting area to explore later. The interest in this area is further enhanced by the knowledge that many file systems (like VFAT) store their lookup tables for file indexing at start of the disk. Utilizing the search-functionality of HW we conducted a binary search(figure 4.5) for the content of our test file, *hestedokument1.txt*. The content of our file was the text *"HESTEPEIS"* repeated 111 times. This means that the phrase *"HESTEPEIS"* and its hex equivalent *0x484553544550454953* should be present at least 111 times unless the object store utilized some kind of data compression. In its documentation[20] Microsoft do however mention that the object store may internally use compression, but this should be completely transparent for anybody utilizing the official APIs. We are not, and have to investigate whether or not compression is used. The search resulted in no instances found, which immediately implies that compression is indeed used (given that our memory dump is correct, something we assume at this point). Refining the search to *"HESTE"* and its hex equivalent *0x4845535445* yields one hit at offset 0x036FF6DD (figure 4.6 - yellow markings).

We see that the string *"HESTEPEI"* is followed by much "random" data until offset 0x036FF7FC. We have no knowledge of the format of this data, but we make the assumption that some or all of it contains the compressed data of our test file. The data from offset 0x036FF7FC is very interesting, however (marked in red in figure 4.6), as this is actually the name of our file, encoded in some kind of Unicode variant where each character occupies two bytes. The filename is not zero-terminated, but is prefixed with the value *0x1100*, which in decimal is 17. This also happens to be the number of characters in the filename. At this point this can be purely coincidental, but we make the assumption that every Unicode string in the object store is prefixed with its length. We will check if this holds true for other files in later tests. We also make the assumption that files in the object store is stored with the file's data first, in a compressed form, with the file's name afterwards.

Next we notice that right before our *"HESTEPEI"* string, at offset 0x036FF6D0, the hex value *0xEC290000* is stored. Also, before the filname, at offset 0x036FF7DA, we find the hex value *0xEB290000*. Now these two values look like a counter that is increased with one. Might this be some kind of id for our file? It could of course be only coincidence, but this seems less likely when we also notice that the same two values are stored at offset 0x036FF684, which is right before our *"HESTEPEI"* string. These values are marked in green. In fact, the whole byte patterns before each of the values seem to have things in common. Looking at the grey markings we see that before the value *0xEC290000* at offset 0x0x036FF7DA, the value *0xEB290000* at offset 0x036FF7DA and the value *0xEB290000* at offset 0x036FF684 we see that the bytes in front of them seem to follow this pattern: *XX00 00Y0 0000 0000* where *XX* is a number between 0 and 0xFF, and *Y* is a number greater than zero. The fact that this pattern is repeated before both the data and the name of the file leads us to suspect that it is part of some kind of header and that the data and the filename both individually have this header. This again leads us to suspect that they are actually threated as two separate objects in the object store.



```
036FF5E0 5CAB C401 0000 0400 6800 6500 7300 7400 \.......h.e.s.t.
036FF5F0 2000 00D0 0000 0000 EA29 0000 C629 0000  ........)...)..
036FF600 0100 0600 0700 4D00 5200 5500 4C00 6900 ......M.R.U.L.i.
036FF610 7300 7400 6200 6100 0000 0000 1000 00D0 s.t.b.a.........
036FF620 0000 0000 7829 0000 F029 0000 0100 0200 ....x)...)......
036FF630 0102 3100 0000 5700 1400 00D0 0000 0000 ..1...W.........
036FF640 7029 0000 6C29 0000 0100 0200 0200 3100 p)..l)........1.
036FF650 3000 0000 6500 6E00 1800 00D0 0000 0000 0...e.n.........
036FF660 ED29 0000 1C13 0000 0300 0800 0100 3400 .)............4.
036FF670 00F1 232A 6DAB C401 4100 4300 4000 0030 ..#*m...A.C.@..0
036FF680 0000 0000 EB29 0000 EC29 0000 0000 0000 .....)...)......
036FF690 0000 0000 0000 0000 0000 0000 0000 0000 ................
036FF6A0 0000 0000 0000 0000 0000 0000 0000 0000 ................
036FF6B0 0000 0000 0000 0000 0000 0000 0000 0000 ................
036FF6C0 0000 0000 0000 0000 FC00 0060 0000 0000 ...........`....
036FF6D0 EC29 0000 0100 E703 00FA 0000 0048 4553 .)...........HES
036FF6E0 5445 5045 49FE 5308 0098 0028 01B8 0148 TEPEI.S....(...H
036FF6F0 02D8 0268 03FF F803 8804 1805 A805 3806 ...h..........8.
036FF700 C806 5807 E807 FF78 0808 0998 0928 0AB8 ..X....x.....(..
036FF710 0A48 0BD8 0B68 0CFF F80C 880D 180E A80E .H...h..........
036FF720 380F C80F 5810 E810 FF78 1108 1298 1228 8...X....x.....(
036FF730 13B8 1348 14D8 1468 15FF F815 8816 1817 ...H...h........
036FF740 A817 3818 C818 5819 E819 FF78 1A08 1B98 ..8...X....x....
036FF750 1B28 1CB8 1C48 1DD8 1D68 1EFF F81E 881F .(...H...h......
036FF760 1820 A820 3821 C821 5822 E822 FF78 2308 . . 8!.!X".".x#.
036FF770 2498 2428 25B8 2548 26D8 2668 27FF F827 $.$(%.%H&.&h'..'
036FF780 8828 1829 A829 382A C82A 582B E82B FF78 .(.).)8*.*X+.+.x
036FF790 2C08 2D98 2D28 2EB8 2E48 2FD8 2F68 30FF ,.-.-(...H/./h0.
036FF7A0 F830 8831 1832 A832 3833 C833 5834 E834 .0.1.2.283.3X4.4
036FF7B0 FF78 3508 3698 3628 37B8 3748 38D8 3868 .x5.6.6(7.7H8.8h
036FF7C0 397F F839 883A 183B A83B 383C C83C 583D 9..9.:.;.;8<.<X=
036FF7D0 4400 0050 0000 0000 7329 0000 EB29 0000 D..P....s)...)..
036FF7E0 E703 0000 0100 00BF E829 0000 0000 0000 .........)......
036FF7F0 00E6 425E 5CAB C401 1100 1100 6800 6500 ..B^\.......h.e.
036FF800 7300 7400 6500 6400 6F00 6B00 7500 6D00 s.t.e.d.o.k.u.m.
036FF810 6500 6E00 7400 2E00 7400 7800 7400 02FF e.n.t...t.x.t...
036FF820 2C00 00D0 0000 0000 7925 0000 7729 0000 ,.......y%..w)..
036FF830 0100 0C00 0900 4C00 6F00 6300 6100 6C00 ......L.o.c.a.l.
036FF840 5000 6100 7400 6800 5C00 6800 6500 7300 P.a.t.h.\.h.e.s.
036FF850 7400 0000 2024 7C42 2800 00D0 0000 0000 t... $|B(.......
036FF860 E929 0000 7925 0000 0400 0400 0CEB 4C00 .)..y%........L.
036FF870 6100 7300 7400 4C00 6F00 6300 6100 7400 a.s.t.L.o.c.a.t.
```

Figure 4.6: Test 1 - Found ”HESTE” in memory dump.

Transform to PIM

Listing 4.1 shows the Hex Workshop structure we can construct based on this information. This structure can be super-imposed by the program upon the raw hex listing, letting us view the data as an instance of the structure.

Listing 4.1: Test 1 - Blob structures v1

```
struct ObjectHeader
{
        WORD unknown;  //XX00
        WORD unknown;  //00Y0
        DWORD zeroFiller; //0000 0000
        DWORD suspected_ID;
};
```

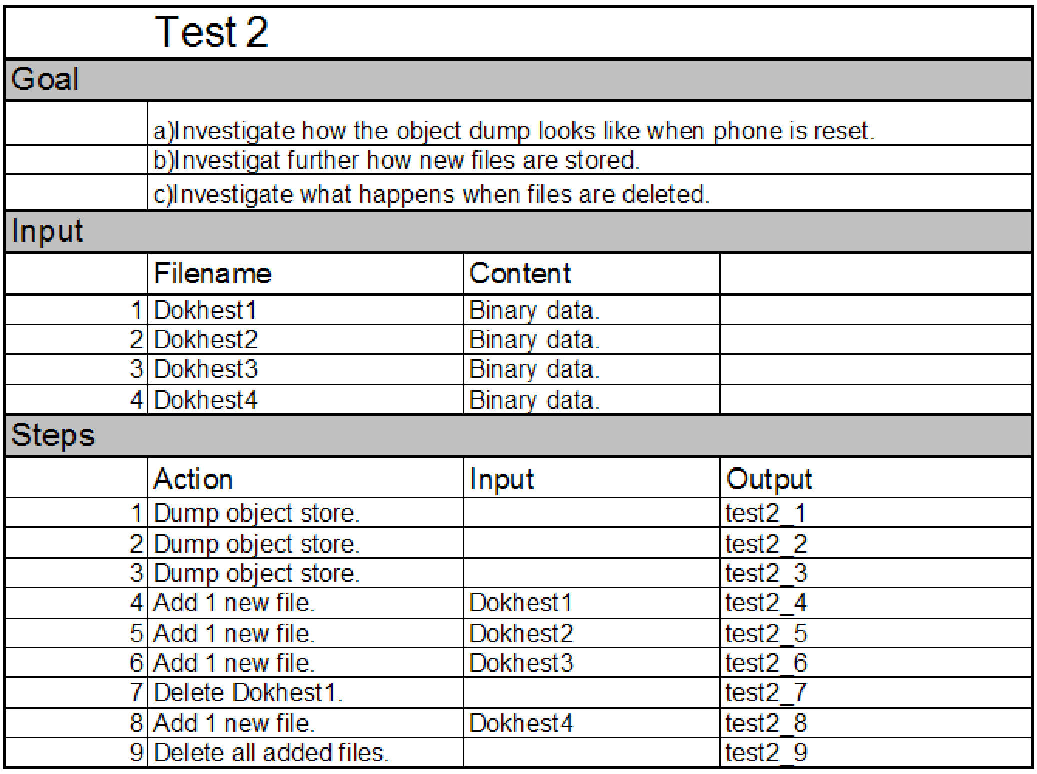
4.2.3.2 Test 2

Create test

We now want to further check our assumption on the existence of an id for each file, and how the data and metadata of a file is connected to each other. From the first test we made the assumption that they are treated separately and somehow linked together. An id would be practical for just this purpose. We now check these assumptions by testing if they hold when adding multiple files.

We also wanted to start looking at what happens when we delete a file from the object store. How does this affect the data in object store? Is it just marked as deleted, overwritten by new data or actually erased by filling it with zeros or any other deletion pattern?

When we compare two memory dumps we can see what differences there exist between them. These differences can be caused by our actions, but they can also be caused by other process on the telephone or the operating system itself. We have no way of getting single atomic access to the phone, so when analyzing changes we have to do it "best effort" and keep our eyes open for changes not connected to our actions. In order to prepare for this, we need to have a "base" dump of the phone. This means



	Test 2		
Goal			
	a)Investigate how the object dump looks like when phone is reset.		
	b)Investigat further how new files are stored.		
	c)Investigate what happens when files are deleted.		
Input			
	Filename	Content	
1	Dokhest1	Binary data.	
2	Dokhest2	Binary data.	
3	Dokhest3	Binary data.	
4	Dokhest4	Binary data.	
Steps			
	Action	Input	Output
1	Dump object store.		test2_1
2	Dump object store.		test2_2
3	Dump object store.		test2_3
4	Add 1 new file.	Dokhest1	test2_4
5	Add 1 new file.	Dokhest2	test2_5
6	Add 1 new file.	Dokhest3	test2_6
7	Delete Dokhest1.		test2_7
8	Add 1 new file.	Dokhest4	test2_8
9	Delete all added files.		test2_9

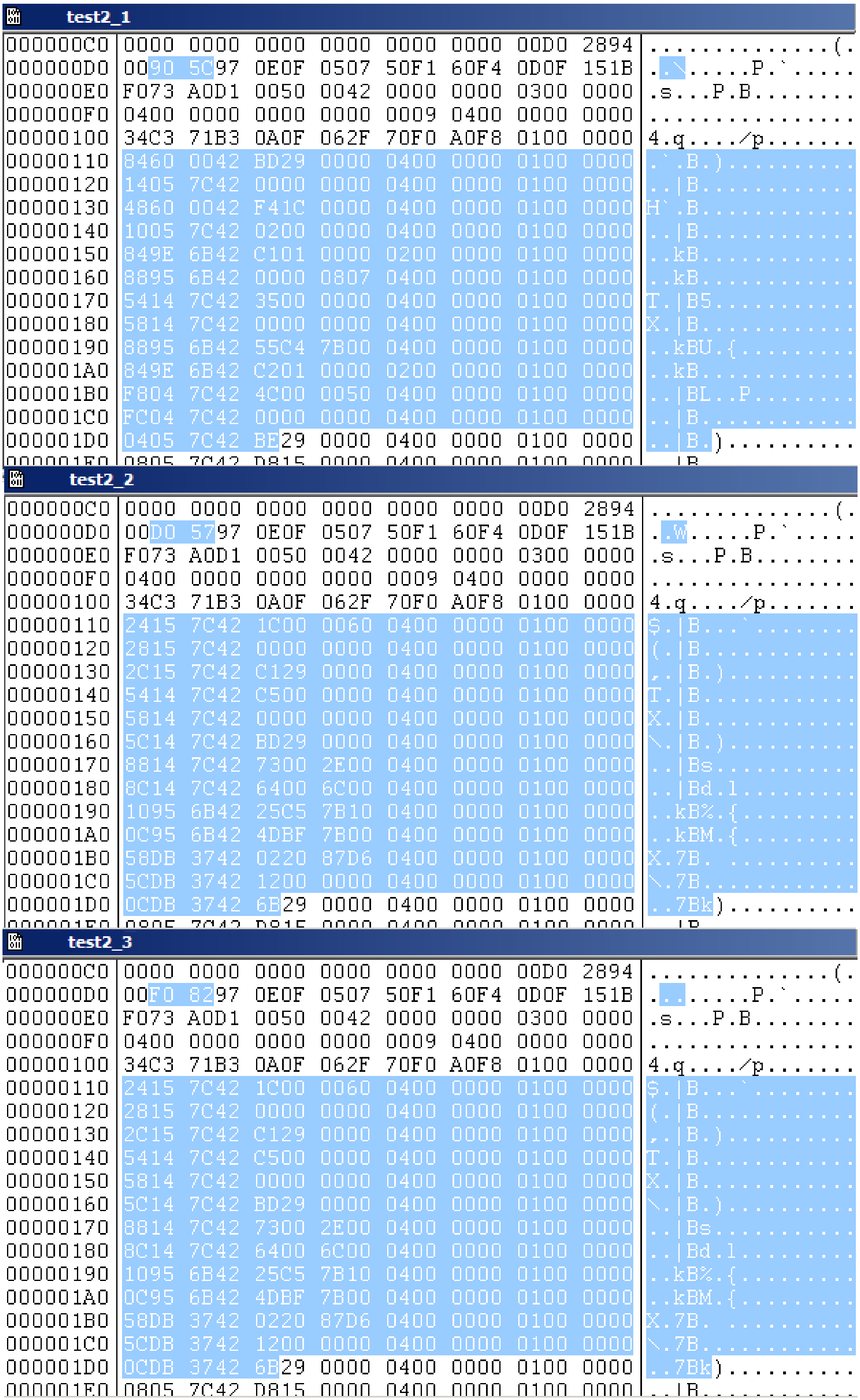
Figure 4.7: Test 2 - New file.

a dump of the phone without any user actions having had any influence on it. This is achieved by dumping the phones memory just after removing the battery and letting it drain itself completely for both primary and backup power. We dump three times to make sure the "base" is more or less stable.

Figure 4.7 contains the second test.

Extract info/Adjust PSM

From figure 4.8 we can see parts of the three "base" dumps. They are overall almost identical, except for the big block starting at offset 0x110 and ending at 0x1D4. These bytes are different (marked with blue), but they follow an overall pattern. From this we conclude that this area can change quite a lot even without user interference, and any changes here should not automatically be considered effects of user input. The structured layout and constant location gives us the distinct impression that this might be part of the file systems own data structures, not user data itself. It cannot be user data, as there has been no user data stored on the device after it was reset. At this point we hazard to guess that it is indeed part of the object stores object table, the structure that maps objects to the memory addresses that contain the object (much like the FAT table in the VFAT file system is). This is later proven not to be the case, but further tests were needed to come to that conclusion. Next we turned our attention to the new files that had been added to the phone, looking at the dump named *test2_6*. By now we know that the name of files can be found as Unicode clear text in the object store dumps, so we did a binary search and ended up with three hits. These are marked with yellow in figure 4.9. We see that our assumption that Unicode strings are prefixed with a length word holds true for all the filenames. *dokhest1* is indeed 8 characters long, as is *dokhest* and *dokhest3*. In test 1 we found some kind of header before both the file data and the file metadata (filename). Is this present in this object store dump as well? Yes, it is. In fact, we can see the pattern from test 1 (*XX00 00Y0 0000 0000*) ten times, marked with red in figure 4.9. Examining the Y-part of the pattern at the different locations gives us the following values of it: 3,6,5,3,6,5,D,3,6,5. The pattern with Y value D is disregarded for now, it only appears one time. Three files were added to the device before this dump. Three times three headers was added, with Y values 3, 6, 5 respectively. This does not seem like a coincidence! If we look closer we notice that the pattern appearing right before the metadata of the file (filename) the Y-value is always 5! The header right before the



test2_1

```
000000C0 0000 0000 0000 0000 0000 0000 00D0 2894 ..............(.
000000D0 0090 5C97 0E0F 0507 50F1 60F4 0D0F 151B ..\.....P.`.....
000000E0 F073 A0D1 0050 0042 0000 0000 0300 0000 .s...P.B........
000000F0 0400 0000 0000 0000 0009 0400 0000 0000 ................
00000100 34C3 71B3 0A0F 062F 70F0 A0F8 0100 0000 4.q..../p.......
00000110 8460 0042 BD29 0000 0400 0000 0100 0000 .`.B.)..........
00000120 1405 7C42 0000 0000 0400 0000 0100 0000 ..|B............
00000130 4860 0042 F41C 0000 0400 0000 0100 0000 H`.B............
00000140 1005 7C42 0200 0000 0400 0000 0100 0000 ..|B............
00000150 849E 6B42 C101 0000 0200 0000 0100 0000 ..kB............
00000160 8895 6B42 0000 0807 0400 0000 0100 0000 ..kB............
00000170 5414 7C42 3500 0000 0400 0000 0100 0000 T.|B5...........
00000180 5814 7C42 0000 0000 0400 0000 0100 0000 X.|B............
00000190 8895 6B42 55C4 7B00 0400 0000 0100 0000 ..kBU.{.........
000001A0 849E 6B42 C201 0000 0200 0000 0100 0000 ..kB............
000001B0 F804 7C42 4C00 0050 0400 0000 0100 0000 ..|BL..P........
000001C0 FC04 7C42 0000 0000 0400 0000 0100 0000 ..|B............
000001D0 0405 7C42 BE29 0000 0400 0000 0100 0000 ..|B.)..........
000001E0 0805 7C42 D815 0000 0400 0000 0100 0000 ..|B
```

test2_2

```
000000C0 0000 0000 0000 0000 0000 0000 00D0 2894 ..............(.
000000D0 00D0 5797 0E0F 0507 50F1 60F4 0D0F 151B ..W.....P.`.....
000000E0 F073 A0D1 0050 0042 0000 0000 0300 0000 .s...P.B........
000000F0 0400 0000 0000 0000 0009 0400 0000 0000 ................
00000100 34C3 71B3 0A0F 062F 70F0 A0F8 0100 0000 4.q..../p.......
00000110 2415 7C42 1C00 0060 0400 0000 0100 0000 $.|B...`........
00000120 2815 7C42 0000 0000 0400 0000 0100 0000 (.|B............
00000130 2C15 7C42 C129 0000 0400 0000 0100 0000 ,.|B.)..........
00000140 5414 7C42 C500 0000 0400 0000 0100 0000 T.|B............
00000150 5814 7C42 0000 0000 0400 0000 0100 0000 X.|B............
00000160 5C14 7C42 BD29 0000 0400 0000 0100 0000 \.|B.)..........
00000170 8814 7C42 7300 2E00 0400 0000 0100 0000 ..|Bs...........
00000180 8C14 7C42 6400 6C00 0400 0000 0100 0000 ..|Bd.l.........
00000190 1095 6B42 25C5 7B10 0400 0000 0100 0000 ..kB%.{.........
000001A0 0C95 6B42 4DBF 7B00 0400 0000 0100 0000 ..kBM.{.........
000001B0 58DB 3742 0220 87D6 0400 0000 0100 0000 X.7B. ..........
000001C0 5CDB 3742 1200 0000 0400 0000 0100 0000 \.7B............
000001D0 0CDB 3742 6B29 0000 0400 0000 0100 0000 ..7Bk)..........
000001E0 0805 7C42 D815 0000 0400 0000 0100 0000 ..|B
```

test2_3

```
000000C0 0000 0000 0000 0000 0000 0000 00D0 2894 ..............(.
000000D0 00F0 8297 0E0F 0507 50F1 60F4 0D0F 151B ........P.`.....
000000E0 F073 A0D1 0050 0042 0000 0000 0300 0000 .s...P.B........
000000F0 0400 0000 0000 0000 0009 0400 0000 0000 ................
00000100 34C3 71B3 0A0F 062F 70F0 A0F8 0100 0000 4.q..../p.......
00000110 2415 7C42 1C00 0060 0400 0000 0100 0000 $.|B...`........
00000120 2815 7C42 0000 0000 0400 0000 0100 0000 (.|B............
00000130 2C15 7C42 C129 0000 0400 0000 0100 0000 ,.|B.)..........
00000140 5414 7C42 C500 0000 0400 0000 0100 0000 T.|B............
00000150 5814 7C42 0000 0000 0400 0000 0100 0000 X.|B............
00000160 5C14 7C42 BD29 0000 0400 0000 0100 0000 \.|B.)..........
00000170 8814 7C42 7300 2E00 0400 0000 0100 0000 ..|Bs...........
00000180 8C14 7C42 6400 6C00 0400 0000 0100 0000 ..|Bd.l.........
00000190 1095 6B42 25C5 7B10 0400 0000 0100 0000 ..kB%.{.........
000001A0 0C95 6B42 4DBF 7B00 0400 0000 0100 0000 ..kBM.{.........
000001B0 58DB 3742 0220 87D6 0400 0000 0100 0000 X.7B. ..........
000001C0 5CDB 3742 1200 0000 0400 0000 0100 0000 \.7B............
000001D0 0CDB 3742 6B29 0000 0400 0000 0100 0000 ..7Bk)..........
000001E0 0805 7C42 D815 0000 0400 0000 0100 0000 ..|B............
```

Figure 4.8: Test 2 - Compare base dumps.

data of a file mysteriously always has 6 as Y-value. The same goes for the headers with Y-value 3, they always appear right before a header with Y-value 6. By now it is a pretty safe bet to assume that the second word of our pattern is a type indicator. The value *0x0050* means "this is metadata" and *0x0060* means "this is data". The meaning of *0x0030* is not yet clear, but as it seems to always appear next to file metadata and file data it's a good bet that it has something to do with the files!

Transform to PIM

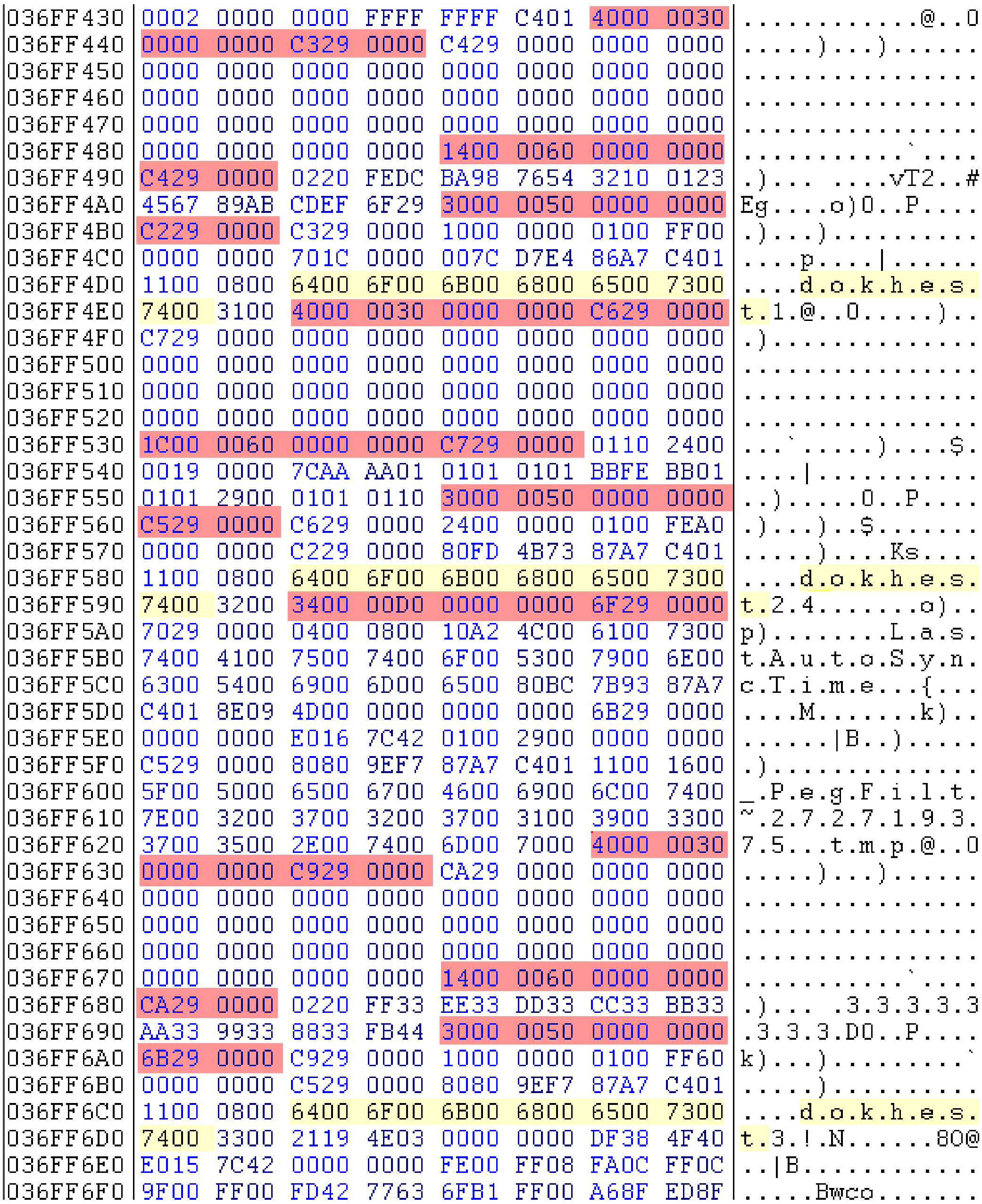
We can now safely say that metadata about a file and the data itself is treated as separate entities prefixed by the similar headers. We call these entities for blobs and update our earlier HW-structure to take into account our new discoveries (listing 4.2). The second word of the pattern is defined as a word enumeration, called *BLOBTYPE*. As we see in figure 4.10, this helps us to quickly assess what the data we are looking at means. When the cursor is placed at an offset, a BlobHeader structure is filled with data from there. The result is that we can view the data as a structure of data types instead of a simple hex dump of a byte stream.

Listing 4.2: Test 2 - Blob structures v2

```
typedef enum tagBLOBTYPE
{
        UNKNOWN_FILE_RELEATED= 12288, //0x3000
        FILEMETADATA = 20480, //0x5000
        FILEDATA = 24576, //0x6000
} BLOBTYPE;

struct BlobHeader
{
        WORD unknown; //XX00
        BLOBTYPE blobType; //00Y0
        DWORD zeroFiller; //0000 0000
        DWORD suspected_ID;
};
```

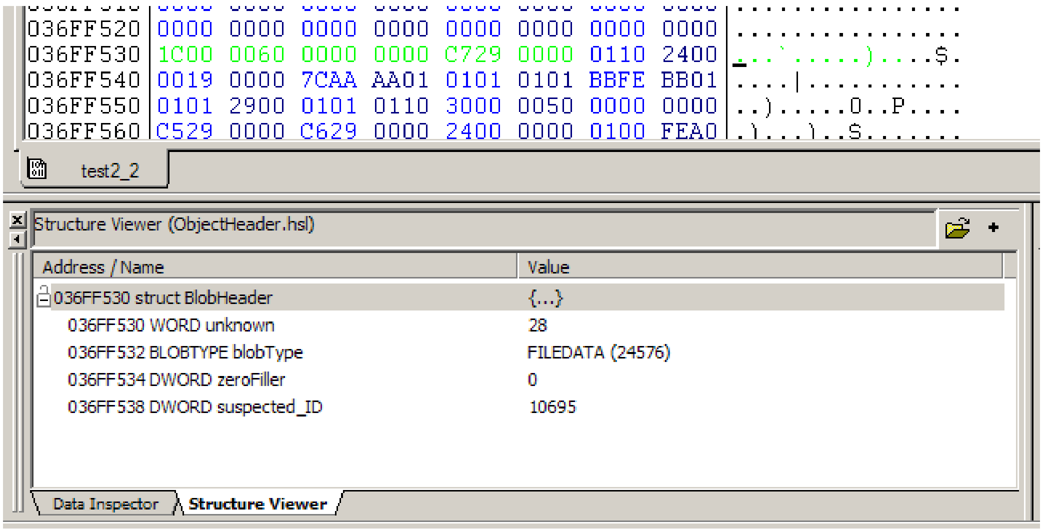
Extract info/Adjust PSM

We still have a number of unknowns in our BlobHeader, like the first word



```
036FF430 | 0002 0000 0000 FFFF FFFF C401 4000 0030 | ............@..0
036FF440 | 0000 0000 C329 0000 C429 0000 0000 0000 | .....)...)......
036FF450 | 0000 0000 0000 0000 0000 0000 0000 0000 | ................
036FF460 | 0000 0000 0000 0000 0000 0000 0000 0000 | ................
036FF470 | 0000 0000 0000 0000 0000 0000 0000 0000 | ................
036FF480 | 0000 0000 0000 0000 1400 0060 0000 0000 | ...........`....
036FF490 | C429 0000 0220 FEDC BA98 7654 3210 0123 | .)... ....vT2..#
036FF4A0 | 4567 89AB CDEF 6F29 3000 0050 0000 0000 | Eg....o)0..P....
036FF4B0 | C229 0000 C329 0000 1000 0000 0100 FF00 | .)...)..........
036FF4C0 | 0000 0000 701C 0000 007C D7E4 86A7 C401 | ....p....|......
036FF4D0 | 1100 0800 6400 6F00 6B00 6800 6500 7300 | ....d.o.k.h.e.s.
036FF4E0 | 7400 3100 4000 0030 0000 0000 C629 0000 | t.1.@..0.....)..
036FF4F0 | C729 0000 0000 0000 0000 0000 0000 0000 | .)..............
036FF500 | 0000 0000 0000 0000 0000 0000 0000 0000 | ................
036FF510 | 0000 0000 0000 0000 0000 0000 0000 0000 | ................
036FF520 | 0000 0000 0000 0000 0000 0000 0000 0000 | ................
036FF530 | 1C00 0060 0000 0000 C729 0000 0110 2400 | ...`.....)....$.
036FF540 | 0019 0000 7CAA AA01 0101 0101 BBFE BB01 | ....|...........
036FF550 | 0101 2900 0101 0110 3000 0050 0000 0000 | ..).....0..P....
036FF560 | C529 0000 C629 0000 2400 0000 0100 FEA0 | .)...)..$.......
036FF570 | 0000 0000 C229 0000 80FD 4B73 87A7 C401 | .....)....Ks....
036FF580 | 1100 0800 6400 6F00 6B00 6800 6500 7300 | ....d.o.k.h.e.s.
036FF590 | 7400 3200 3400 00D0 0000 0000 6F29 0000 | t.2.4.......o)..
036FF5A0 | 7029 0000 0400 0800 10A2 4C00 6100 7300 | p)........L.a.s.
036FF5B0 | 7400 4100 7500 7400 6F00 5300 7900 6E00 | t.A.u.t.o.S.y.n.
036FF5C0 | 6300 5400 6900 6D00 6500 80BC 7B93 87A7 | c.T.i.m.e...{...
036FF5D0 | C401 8E09 4D00 0000 0000 0000 6B29 0000 | ....M.......k)..
036FF5E0 | 0000 0000 E016 7C42 0100 2900 0000 0000 | ......|B..).....
036FF5F0 | C529 0000 8080 9EF7 87A7 C401 1100 1600 | .)..............
036FF600 | 5F00 5000 6500 6700 4600 6900 6C00 7400 | _.P.e.g.F.i.l.t.
036FF610 | 7E00 3200 3700 3200 3700 3100 3900 3300 | ~.2.7.2.7.1.9.3.
036FF620 | 3700 3500 2E00 7400 6D00 7000 4000 0030 | 7.5...t.m.p.@..0
036FF630 | 0000 0000 C929 0000 CA29 0000 0000 0000 | .....)...)......
036FF640 | 0000 0000 0000 0000 0000 0000 0000 0000 | ................
036FF650 | 0000 0000 0000 0000 0000 0000 0000 0000 | ................
036FF660 | 0000 0000 0000 0000 0000 0000 0000 0000 | ................
036FF670 | 0000 0000 0000 0000 1400 0060 0000 0000 | ...........`....
036FF680 | CA29 0000 0220 FF33 EE33 DD33 CC33 BB33 | .)... .3.3.3.3.3
036FF690 | AA33 9933 8833 FB44 3000 0050 0000 0000 | .3.3.3.D0..P....
036FF6A0 | 6B29 0000 C929 0000 1000 0000 0100 FF60 | k)...).........`
036FF6B0 | 0000 0000 C529 0000 8080 9EF7 87A7 C401 | .....)..........
036FF6C0 | 1100 0800 6400 6F00 6B00 6800 6500 7300 | ....d.o.k.h.e.s.
036FF6D0 | 7400 3300 2119 4E03 0000 0000 DF38 4F40 | t.3.!.N......8O@
036FF6E0 | E015 7C42 0000 0000 FE00 FF08 FA0C FF0C | ..|B............
036FF6F0 | 9F00 FF00 FD42 7763 6FB1 FF00 A68F ED8F | .....Bwco.......
```

Figure 4.9: Test 2 - Looking at new files.



```
036FF520 0000 0000 0000 0000 0000 0000 0000 0000 ................
036FF530 1C00 0060 0000 0000 C729 0000 0110 2400 ...`.....)....$.
036FF540 0019 0000 7CAA AA01 0101 0101 BBFE BB01 ....|...........
036FF550 0101 2900 0101 0110 3000 0050 0000 0000 ..).....0..P....
036FF560 C529 0000 C629 0000 2400 0000 0100 FEA0 .)...)..$.......
```

Address / Name	Value
036FF530 struct BlobHeader	{...}
036FF530 WORD unknown	28
036FF532 BLOBTYPE blobType	FILEDATA (24576)
036FF534 DWORD zeroFiller	0
036FF538 DWORD suspected_ID	10695

Figure 4.10: Test 2 - Applying the BlobHeader structure to a blob in Hex Workshop.

of data. Figure 4.10 shows us that its value is *28(0x1C00)*, which is a low value considering that an unsigned word (16 bits) can have values as high as *65535(0xFFFF)*. This in itself is not evidence of anything, but when we look at the first word of all the other headers in figure 4.9 we see that this word has the following values: *0x4000*, *0x1400*, *0x3000*, *0x4000*, *0x1C00*, *0x3000*, *0x3400*, *0x4000*, *0x1400*, *0x3000*. If we once again disregard the header with a blob type of *0x00D0*, we see a repeating pattern. In headers with blob type UNKNOWN_FILE_RELEATED(*0x0030*), the first word is always *0x4000*. Likewise, in headers of blobtype FILEMETADATA, the first word is always *0x3000*. This leads us to suspect the BLOBTYPE of maybe actually being a double word (dword) instead of a word, because the unknown first word seems to have a special value based on the value of the next word, the blob type. This suspicion was false, as headers with the blobtype value FILEDATA have different values of the unknown first word (*0x1C00* and *0x1400*). Keep in mind that our device stores information in little-endian format, so the value *0x1400* in the hex dump is the decimal value *20*.

So, we had no luck uncovering the meaning of the first word in our header. Starting from another angle we see that at offset 0x036FF678 we have a header starting with value *20(0x1400)*, and at offset 0x036FF698 we have a header starting with the value *48(0x3000)*. The length of each header is

12(0x0C) bytes. The space between the end of the first header and the start of the next header is 0x036FF698 - (0x036FF678 + 0x0C) = 20 decimal. This is the same value as the unknown word! The unknown word gives the distance from after the header until the next header, or put in another way: the size of the data between headers. Looking at the actual data between the two headers reveals *0xFF33EE33DD33CC33BB33AA3399338833*. This is the file content of our input file *dokhest3*. We know assume that the data immediately following a blob header is the data the user stored, and that a header and data together makes a complete object in the object store, which we call blob. The first word in a blob's header is the size of the blob's data.

Transform to PIM

Once again we update our HW-structure based on the new knowledge(listing 4.3). Applying it to the start of each header gives us further faith that the first word is size. All the blobs can be described by it. The end offset indicated by the size field of each of the blobs match perfectly with the start offset of the next blob, except for the blob with blob type *0x00D0*. An example is the blob at offset 0x036FF62C. According to our updated HW-structure it should last until the next blob, which starts at 0x036FF678. Using the formula: start offset + length (header) + data size = start of next header, we get 0x036FF62C + 0x0C + 0x40 = 0x036FF678. Success.

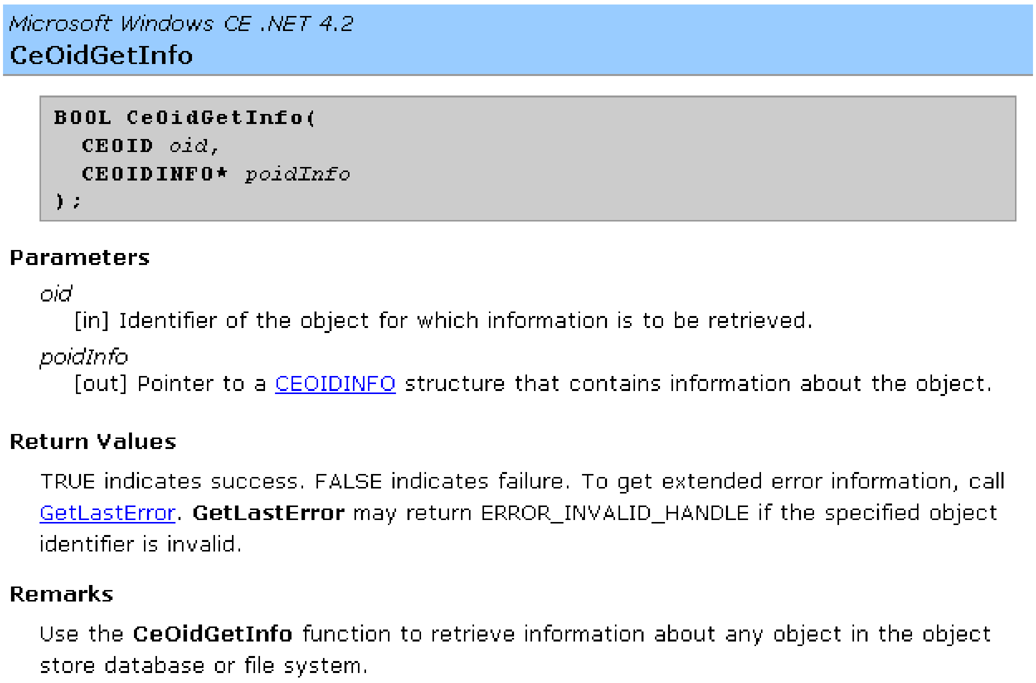
Listing 4.3: Test 2 - Blob structures v3

```
typedef enum tagBLOBTYPE
{
        UNKNOWN_FILE_RELEATED= 12288, //0x3000
        FILEMETADATA = 20480, //0x5000
        FILEDATA = 24576, //0x6000
} BLOBTYPE;

struct Blob
{
        WORD size;
        BLOBTYPE blobType; //00Y0
        DWORD zeroFiller; //0000 0000
        DWORD suspected_ID;
        BYTE Data[size];
};
```

Extract info/Adjust PSM

After test 1 we had a suspicion that the last dword of the header was some kind of id. This is pretty much confirmed in this test. Every header has a unique value at this location, and the values vary with only one or two from header to header (*0xC3290000, 0xC4290000, 0xC2290000 and so on*). At this point we wanted to verify completely that this was indeed an id because it would be important to know when analyzing further. In the Windows CE API documentation Microsoft mention that every object in the object store has its own unique id. So far, so good. But we need to make sure that what we believe to be the id in fact is this very same id as mentioned by Microsoft. In order to do this we decided to extend the C++ Forensic tool, called Judas, created by Jarle Eide[6]. The tool was modified to take an id as input and looking it up using the Windows CE API. If we feed this tool what we believe to be the id of a file from our analysis and the API call returns with information about the correct file, we can conclude that this is in fact a valid id. The API call utilized for this was *CeOidGetInfo*(figure 4.11). The id *0xC2290000(10690)* was reported by Windows CE as belonging to a file with the name *dokhest1*, which we already know from looking at the blob at offset 0x036FF4A8 in figure 4.9. It is confirmed, the last field of the blob header is definitely the id of a blob!

The size field is only 16 bits wide. This means a max value of *65535*(given that the value is unsigned). Does this mean the object store can't store objects larger than this? According to the API the maximum file size is 32MB. How can it support files larger than 65535 bytes when the object size is limited to this? The obvious solution is to utilize several objects for each file. One would have to construct some kind of mechanism to map a filename or id to several file data objects. This mechanism would have to be able to connect the id of a metadata blob with the id of several filedata blobs. If we look at offset 0x036FF564(figure 4.9)we see that the first dword in the data field of the blob starting here contains what looks suspiciously like an id (*0xC6290000*). In fact it's the id of the UNKNOWN_FILE_RELEATED blob located right above, at offset 0x036FF4E4! The metadata blob of a file has a "pointer" to a blob of type UNKNOWN_FILE_RELEATED. Looking closely at this UNKNOWN_FILE_RELEATED blob we see that the first dword of it's data contains yet another id(*0xC7290000*), which is the id of the FILEDATA blob starting at 0x036FF530. Now this is interesting! This means that we have a "pointer chain" for this file. The file's metadata blob contains a "pointer" to a UNKNOWN_FILE_RELEATED blob which again contains a "pointer" to a FILEDATA blob. We also notice that the



Microsoft Windows CE .NET 4.2

CeOidGetInfo

```
BOOL CeOidGetInfo(
  CEOID oid,
  CEOIDINFO* poidInfo
);
```

Parameters

oid
[in] Identifier of the object for which information is to be retrieved.

poidInfo
[out] Pointer to a CEOIDINFO structure that contains information about the object.

Return Values

TRUE indicates success. FALSE indicates failure. To get extended error information, call GetLastError. **GetLastError** may return ERROR_INVALID_HANDLE if the specified object identifier is invalid.

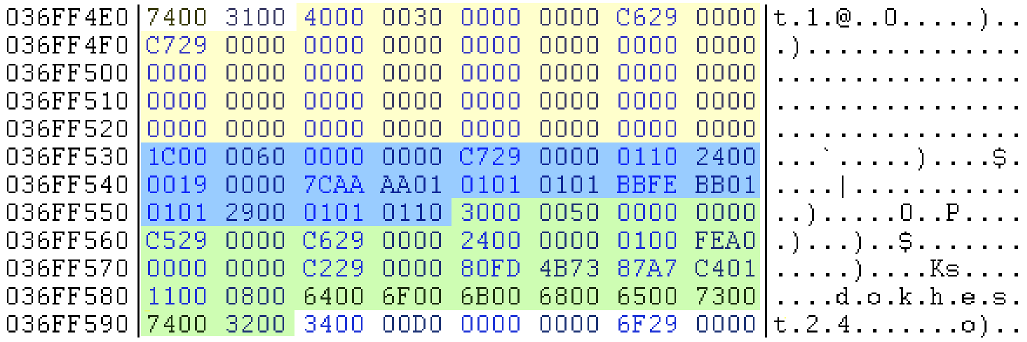
Remarks

Use the **CeOidGetInfo** function to retrieve information about any object in the object store database or file system.

Figure 4.11: Windows CE API - CeOidGetInfo

UNKNOWN_FILE_RELEATED blob has a data size of 0x40 bytes according to it's header, yet it only utilizes 8 of them. We think this is because it has reserved these bytes in the case that the file's content should grow beyond what a single FILEDATA can store. If this happens, it could just insert a pointer to yet another FILEDATA blob right after the first one. We investigate this further in later tests. UNKNOWN_FILE_RELEATED is no longer completely unknown; we assume it's a list of "pointers" to FILEDATA blobs. From now on we call it FILEDATALIST instead of UNKNOWN_FILE_RELEATED.

Armed with our new knowledge we follow the "pointer chain" from FILEMETADATA to FILEDATA via FILEDATALIST for all the input files in this test. *dokhest1* and *dokhest3* is easily proven correct as the FILEDATA blob for each of these contains the file content directly, only prefixed by the hex value *0x0220*. The data in the FILEDATA blob of *dokhest2* is different however. It is prefixed with another hex value, *0x110*, and the data is now stored as the raw file content. Might this be because the object store has compressed the contents? Anyway it is obvious that the first word of the



```
036FF4E0 7400 3100 4000 0030 0000 0000 C629 0000 t.1.@..0.....)..
036FF4F0 C729 0000 0000 0000 0000 0000 0000 0000 .)..............
036FF500 0000 0000 0000 0000 0000 0000 0000 0000 ................
036FF510 0000 0000 0000 0000 0000 0000 0000 0000 ................
036FF520 0000 0000 0000 0000 0000 0000 0000 0000 ................
036FF530 1C00 0060 0000 0000 C729 0000 0110 2400 ...`.....)....$.
036FF540 0019 0000 7CAA AA01 0101 0101 BBFE BB01 ....|...........
036FF550 0101 2900 0101 0110 3000 0050 0000 0000 ..).....0..P....
036FF560 C529 0000 C629 0000 2400 0000 0100 FEA0 .)...)..$.......
036FF570 0000 0000 C229 0000 80FD 4B73 87A7 C401 .....)....Ks....
036FF580 1100 0800 6400 6F00 6B00 6800 6500 7300 ....d.o.k.h.e.s.
036FF590 7400 3200 3400 00D0 0000 0000 6F29 0000 t.2.4.......o)..
```

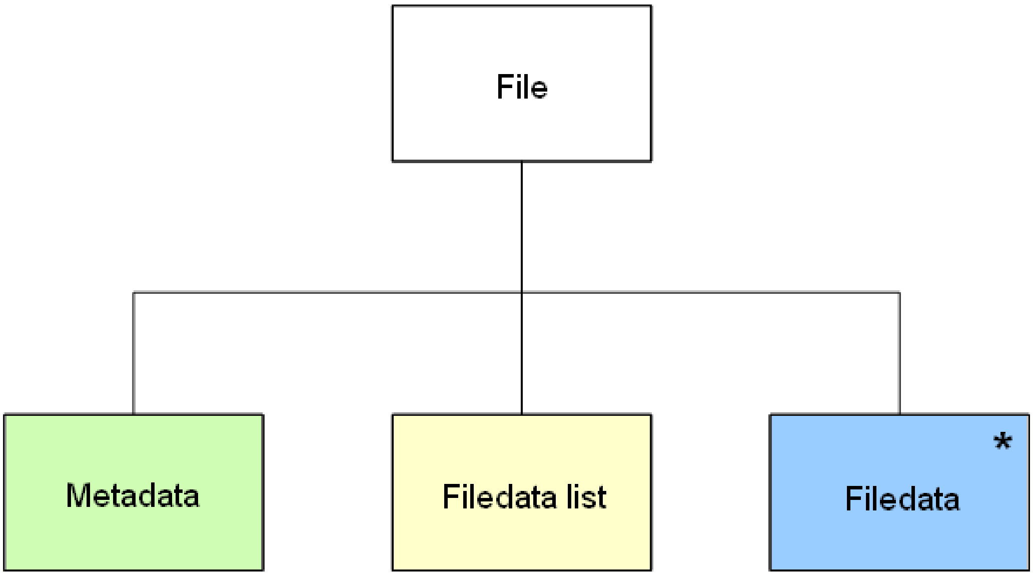


Figure 4.12: Test 2 - Jackson Data Structure diagram of a file.

data in a FILEDATA blob is some kind of type or flag field indicating the way the data is stored.

Transform to PIM

A Jackson Data Structure diagram of a file is illustrated in figure 4.12.

All our new findings make us able to refactor and update our HW-structures quite a lot. Listing 4.4 summarizes what we now know about the object store data structures, called blobs.

Listing 4.4: Test 2 - Blob structures v4

```
typedef enum tagBLOBTYPE
{
        FILEDATALIST= 12288, //0x3000
        FILEMETADATA = 20480, //0x5000
        FILEDATA = 24576, //0x6000
} BLOBTYPE;


typedef struct BlobHeader
{
        WORD size;
        BLOBTYPE blobType;
        DWORD zeroFiller;
        DWORD ID;
} HEADER;


struct GeneralBlob
{
        HEADER header;
        BYTE data[header.size];
};


struct FileDataListBlob
{
        HEADER header;
        DWORD fileDataID;
        BYTE unknown[header.size - 4]; //we suspect
            this might contain more fileDataIDs if the
            file is larger than 65535 bytes.
}


struct FileMetaDataBlob
{
        HEADER header;
        DWORD fileDataListID;
        BYTE unknown[26];
        WORD filenameLength;
        WORD filename[filenameLength];

}

struct FileDataBlob
{
        HEADER header;
        WORD suspected_storageType;
        BYTE fileData[header.size - 2];
}
```

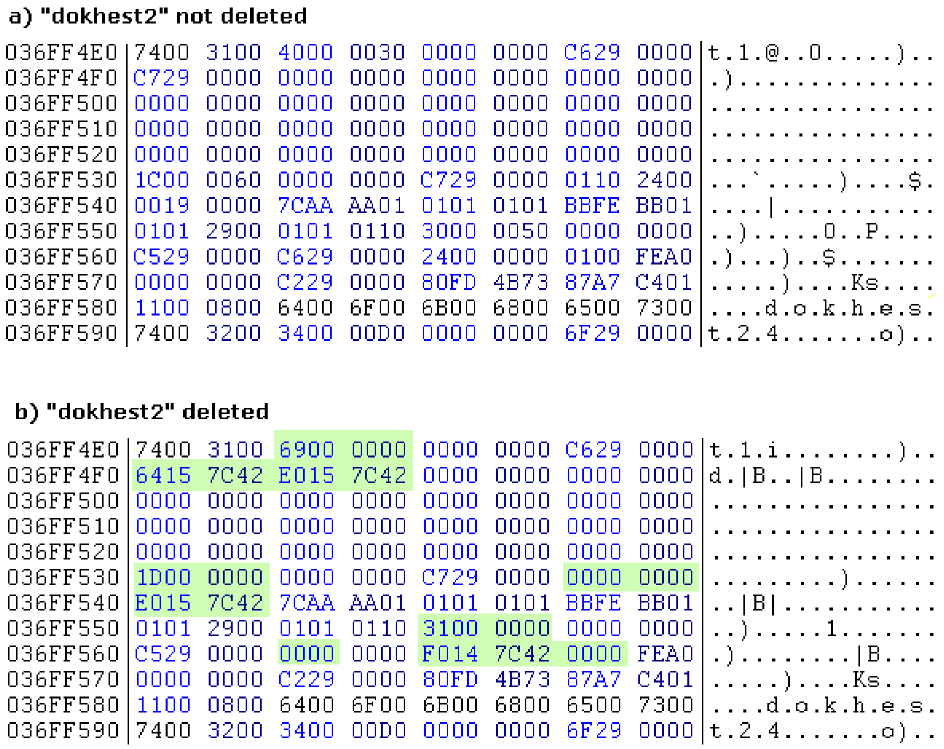
Extract info/Adjust PSM

The last thing we wanted to check in test 2 was what happens in the object store when you delete a file. Figure 4.13 give us some starting points. It shows the difference, marked with green, between the three blobs connected to *dokhest2*(FILEMETADATA, FILEDATALIST, FILEDATA) before and after the file is deleted.

The first thing we notice is that the length and type field in all the blobs have been altered. The type field is simply set to *0*. The length field is somewhat more complicated. For all the blobs it has at least the least significant bit set. This has never been observed in the length field of any normal blob, as these are always divisible by 2. Maybe this is quick way for OS to check whether a blob is deleted or not? For the FILEDATA and METADATA blobs at offsets 0x036FF530 and 0x036FF558 we see that the length offsets have simply had their least significant bit set. The FILE-

DATALIST blob at offset 0x036FF4E0 however has been given the value 0x69 as its length. Why this is, we don't know yet. Further testing is needed. An interesting observation is that 0x69 is pretty close to 0x40 + 0x1C, which is the combined length of FILEDATALIST and FILEDATA. We also see that the first 8 bytes of the blobs' data area have been overwritten with new data. Not much can be said about this at the time, but we do notice that the 8 bytes seem to consist of two dwords each having 42 as their most significant byte (remember that data is stored little-endian wise). According to Microsoft (figure 4.14), *0x42000000* is the start of the object store and memory mapped files in our mobile device. So a value with 42 as the most significant byte might be addresses in this memory area. Could these values be some kind of pointers? We have to test more to find out.

The last step in the test was to delete all the added files. No new results were drawn from this, as the blobs were altered the same way as *dokhest2*.



a) "dokhest2" not deleted

```
036FF4E0|7400 3100 4000 0030 0000 0000 C629 0000|t.1.@..0.....)..
036FF4F0|C729 0000 0000 0000 0000 0000 0000 0000|.)..............
036FF500|0000 0000 0000 0000 0000 0000 0000 0000|................
036FF510|0000 0000 0000 0000 0000 0000 0000 0000|................
036FF520|0000 0000 0000 0000 0000 0000 0000 0000|................
036FF530|1C00 0060 0000 0000 C729 0000 0110 2400|...`.....)....$.
036FF540|0019 0000 7CAA AA01 0101 0101 BBFE BB01|....|...........
036FF550|0101 2900 0101 0110 3000 0050 0000 0000|..)....0..P.....
036FF560|C529 0000 C629 0000 2400 0000 0100 FEA0|.)...)..$.......
036FF570|0000 0000 C229 0000 80FD 4B73 87A7 C401|.....)....Ks....
036FF580|1100 0800 6400 6F00 6B00 6800 6500 7300|....d.o.k.h.e.s.
036FF590|7400 3200 3400 00D0 0000 0000 6F29 0000|t.2.4.......o)..
```

b) "dokhest2" deleted

```
036FF4E0|7400 3100 6900 0000 0000 0000 C629 0000|t.1.i.........)..
036FF4F0|6415 7C42 E015 7C42 0000 0000 0000 0000|d.|B..|B........
036FF500|0000 0000 0000 0000 0000 0000 0000 0000|................
036FF510|0000 0000 0000 0000 0000 0000 0000 0000|................
036FF520|0000 0000 0000 0000 0000 0000 0000 0000|................
036FF530|1D00 0000 0000 0000 C729 0000 0000 0000|.........)......
036FF540|E015 7C42 7CAA AA01 0101 0101 BBFE BB01|..|B|...........
036FF550|0101 2900 0101 0110 3100 0000 0000 0000|..).....1.......
036FF560|C529 0000 0000 0000 F014 7C42 0000 FEA0|.)........|B....
036FF570|0000 0000 C229 0000 80FD 4B73 87A7 C401|.....)....Ks....
036FF580|1100 0800 6400 6F00 6B00 6800 6500 7300|....d.o.k.h.e.s.
036FF590|7400 3200 3400 00D0 0000 0000 6F29 0000|t.2.4.......o)..
```

Figure 4.13: Test 2 - What happens to the blobs of a deleted file.

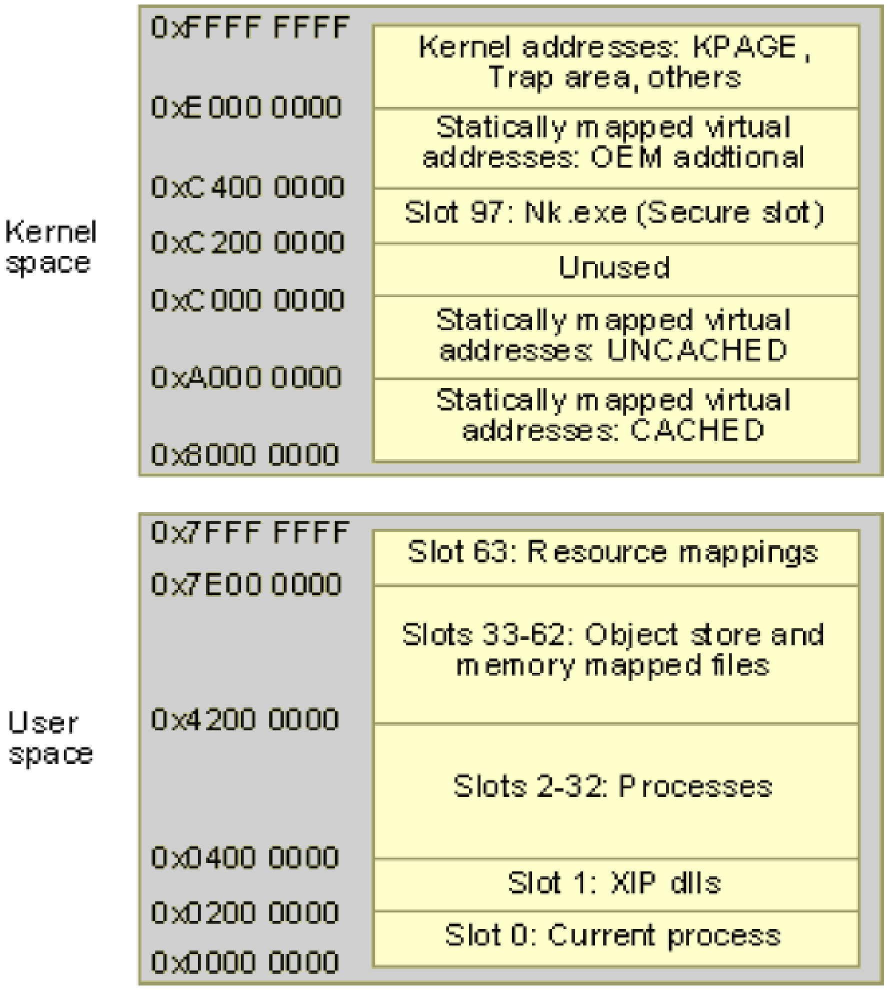
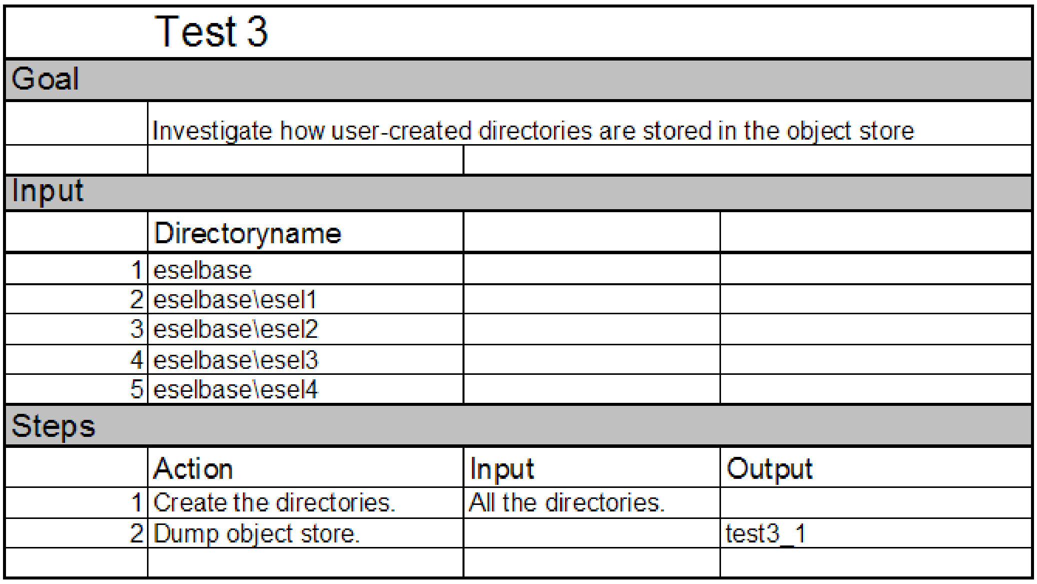

Kernel space
0xFFFF FFFF
Kernel addresses: KPAGE, Trap area, others
0xE000 0000
Statically mapped virtual addresses: OEM addtional
0xC400 0000
Slot 97: Nk.exe (Secure slot)
0xC200 0000
Unused
0xC000 0000
Statically mapped virtual addresses: UNCACHED
0xA000 0000
Statically mapped virtual addresses: CACHED
0x8000 0000
User space
0x7FFF FFFF
Slot 63: Resource mappings
0x7E00 0000
Slots 33-62: Object store and memory mapped files
0x4200 0000
Slots 2-32: Processes
0x0400 0000
Slot 1: XIP dlls
0x0200 0000
Slot 0: Current process
0x0000 0000

Figure 4.14: Windows CE API - Windows CE Memory Layout[16]



	Test 3		
Goal			
	Investigate how user-created directories are stored in the object store		
Input			
	Directoryname		
1	eselbase		
2	eselbase\esel1		
3	eselbase\esel2		
4	eselbase\esel3		
5	eselbase\esel4		
Steps			
	Action	Input	Output
1	Create the directories.	All the directories.	
2	Dump object store.		test3_1

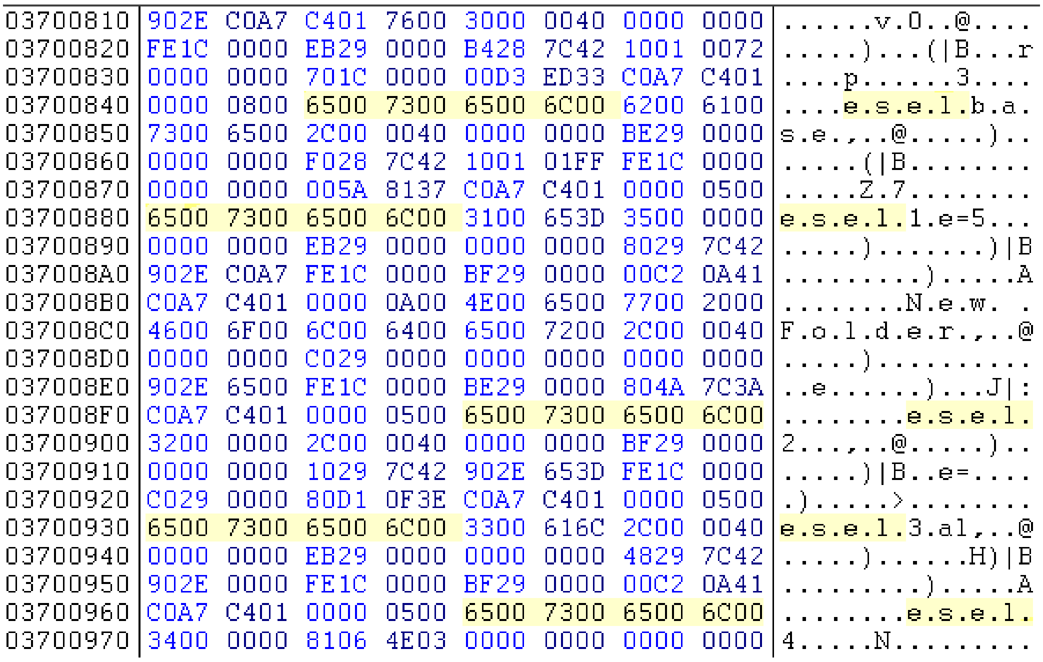
Figure 4.15: Test 3 - User-created directories

4.2.3.3 Test 3

Create test

We want to find out how user-created directories are stored in the object store. We assume that in addition to the name of the directory, other metadata like the creation date and some flags are stored. Most likely there must also exist some kind of mechanism to allow the OS to keep track of the hierarchies of directories. That is, to know which directories are sub directories of other directories. Making assumptions based on what we now know about files, it would not be surprising if each directory is stored as at least one blob. Most likely this blob will have the same header as the other blobs, but with a new value for the blob type field to indicate that this is a directory.

Figure 4.15 contains the third test.



```
03700810|902E C0A7 C401 7600 3000 0040 0000 0000|......v.0..@....
03700820|FE1C 0000 EB29 0000 B428 7C42 1001 0072|.....)...(|B...r
03700830|0000 0000 701C 0000 00D3 ED33 C0A7 C401|....p......3....
03700840|0000 0800 6500 7300 6500 6C00 6200 6100|....e.s.e.l.b.a.
03700850|7300 6500 2C00 0040 0000 0000 BE29 0000|s.e.,..@.....)..
03700860|0000 0000 F028 7C42 1001 01FF FE1C 0000|.....(|B........
03700870|0000 0000 005A 8137 C0A7 C401 0000 0500|.....Z.7........
03700880|6500 7300 6500 6C00 3100 653D 3500 0000|e.s.e.l.1.e=5...
03700890|0000 0000 EB29 0000 0000 0000 8029 7C42|.....).......)|B
037008A0|902E C0A7 FE1C 0000 BF29 0000 00C2 0A41|.........).....A
037008B0|C0A7 C401 0000 0A00 4E00 6500 7700 2000|........N.e.w. .
037008C0|4600 6F00 6C00 6400 6500 7200 2C00 0040|F.o.l.d.e.r.,..@
037008D0|0000 0000 C029 0000 0000 0000 0000 0000|.....)..........
037008E0|902E 6500 FE1C 0000 BE29 0000 804A 7C3A|..e......)...J|:
037008F0|C0A7 C401 0000 0500 6500 7300 6500 6C00|........e.s.e.l.
03700900|3200 0000 2C00 0040 0000 0000 BF29 0000|2...,..@.....)..
03700910|0000 0000 1029 7C42 902E 653D FE1C 0000|.....)|B..e=....
03700920|C029 0000 80D1 0F3E C0A7 C401 0000 0500|.).....>........
03700930|6500 7300 6500 6C00 3300 616C 2C00 0040|e.s.e.l.3.al,..@
03700940|0000 0000 EB29 0000 0000 0000 4829 7C42|.....)......H)|B
03700950|902E 0000 FE1C 0000 BF29 0000 00C2 0A41|.........).....A
03700960|C0A7 C401 0000 0500 6500 7300 6500 6C00|........e.s.e.l.
03700970|3400 0000 8106 4E03 0000 0000 0000 0000|4.....N.........
```

Figure 4.16: Test 3 - View of user-created directories

Extract info/Adjust PSM

A quick search for the string *"esel"* gives us 5 relevant hits. They all reside in the same area of memory shown in figure 4.16.

All of them have the blob header, and the blob type value for directories is *0x0040*. The last data in the data field is the Unicode encoded directory name. Looking closer at the data field of the directory blobs we see that the first dword often contains values of the pattern *0xXX290000*. So does the 4th and 5th dword. Values matching this pattern have earlier been shown to have a high likelihood of being an id of another blob. Might this be the case also? We make the analysis easier by showing the name and id together with the first, fourth and fifth dwords of the data of each blob in table 4.1.
Now we see straight away that *esel1-4* all have the same fourth dword, with the value *0xFE1C0000*. This is also the id of *eselbase*. We assume this is a "parent" field that points to the directory's immediate parent directory. *eselbase* has *0* as it's parent id, which makes sense considering that this directory was put straight in the root of the object store and therefore

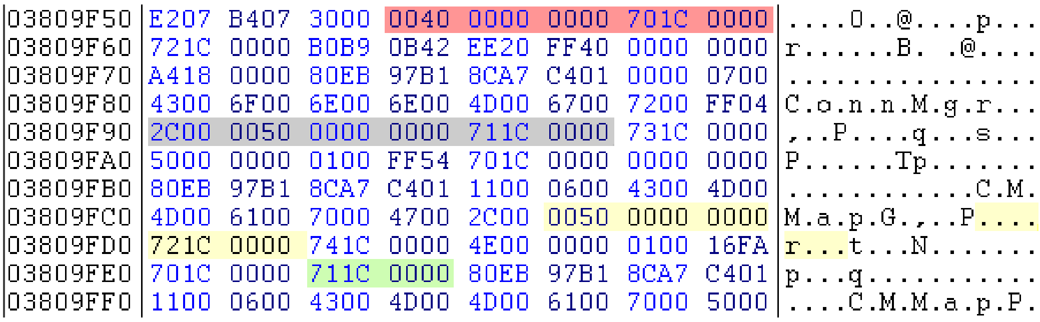
Directory	ID	dword 1	dword 4	dword 5
eselbase	0xFE1C0000	0xEB290000	0	0x701C0000
eselbase/esel1	0xBE290000	0	0xFE1C0000	0
eselbase/esel2	0xC0290000	0	0xFE1C0000	0xBE290000
eselbase/esel3	0xBF290000	0	0xFE1C0000	0xC0290000
eselbase/esel4	0xEB290000	0	0xFE1C0000	0xBF290000

Table 4.1: Test 3 - 5 directories in object store

has no actual parent directory. The fourth dword is a parent id.

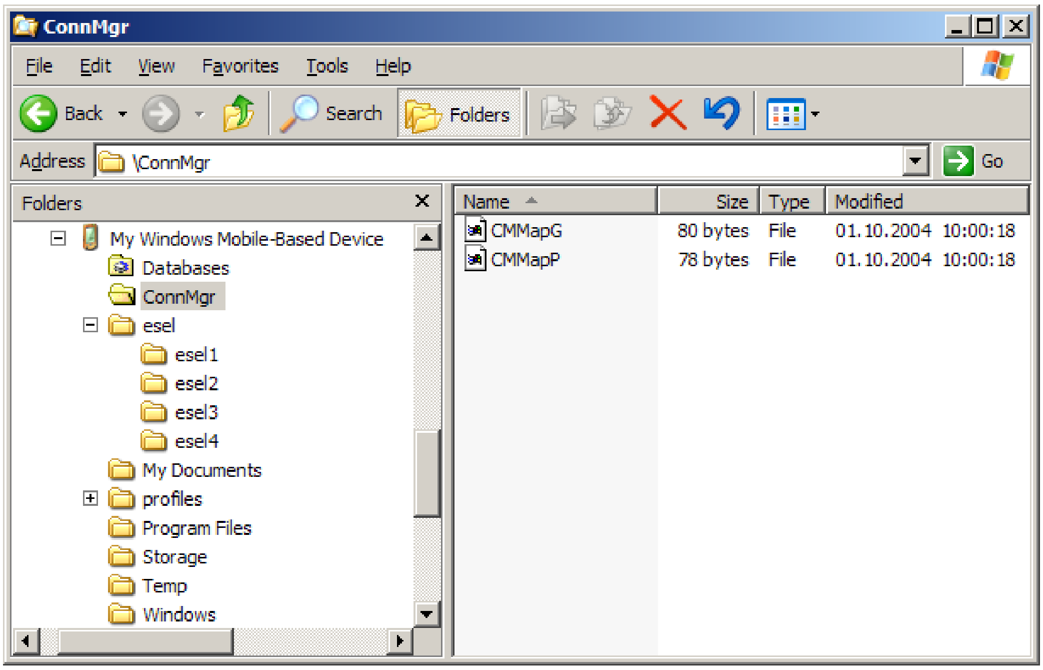
esel4's fifth word contains the id of *esel3*, which in turn has *esel2's* id as its 5th word. Not surprisingly, *esel2's* fifth word is *esel1's* id. We have a linked list of pointers from the "last" subdirectory to the "first" subdirectory. This fifth word is some kind of neighbor id. It makes perfect sense for a object store to have this, as this makes directory traversal very easy. Simply start with the "last" directory and follow it's neighbor id field recursively. Now how does the object store know which subdirectory is the "last"? The first dword in *eselbase* conveniently has the "last" directory's (*esel4*) id in it. This might be a child id field, with the id of the first or "last" child added. Directory traversal is now very simple. Just look up the id in the directory's child field and follow the neighbor ids recursively from there.

Further evidence that the fifth dword is a neighbor id was needed. *eselbase* has the hex value *0x701C0000* as it's neighbor id. A search for the hex string *"0040 0000 0000 701C 0000"*, which is the header of the blob with id *0x701C0000*, results in one hit. The hit is the directory blob for the directory called *"ConnMgr"*(see figure 4.17, red markings). Looking at the file listing of the phone (figure 4.18) we see that this is indeed a directory with the same parent as *eselbase*. The fifth dword is a neighbor id. In the exact same manner we see that the child id in *"ConnMgr's"* blob is 0x721C0000. The blob with this id is located at offset 0x03808FC8(figure 4.17, marked with yellow). It is a file called *"CMMapP"*. This file can be found in the *"ConnMgr"* directory on the phone. Notice also its fourth data dword has its parent directory id in it. The file points right back to the *"ConnMgr"* blob. The first dword is a child id.



```
03809F50 E207 B407 3000 0040 0000 0000 701C 0000 ....0..@....p...
03809F60 721C 0000 B0B9 0B42 EE20 FF40 0000 0000 r......B. .@....
03809F70 A418 0000 80EB 97B1 8CA7 C401 0000 0700 ................
03809F80 4300 6F00 6E00 6E00 4D00 6700 7200 FF04 C.o.n.n.M.g.r...
03809F90 2C00 0050 0000 0000 711C 0000 731C 0000 ,..P....q...s...
03809FA0 5000 0000 0100 FF54 701C 0000 0000 0000 P......Tp.......
03809FB0 80EB 97B1 8CA7 C401 1100 0600 4300 4D00 ............C.M.
03809FC0 4D00 6100 7000 4700 2C00 0050 0000 0000 M.a.p.G.,..P....
03809FD0 721C 0000 741C 0000 4E00 0000 0100 16FA r...t...N.......
03809FE0 701C 0000 711C 0000 80EB 97B1 8CA7 C401 p...q...........
03809FF0 1100 0600 4300 4D00 4D00 6100 7000 5000 ....C.M.M.a.p.P.
```

Figure 4.17: Test 3 - Neighbor and child id.



Name	Size	Type	Modified
CMMapG	80 bytes	File	01.10.2004 10:00:18
CMMapP	78 bytes	File	01.10.2004 10:00:18

Figure 4.18: Test 3 - Data files on the device.

Transform to PIM

Our assumptions about the nature of the first, fourth and fifth word of the directory blobs has yet to see any contradictional evidence. We celebrate this by creating a new HW-structure for directory blobs(listing 4.5).

Listing 4.5: Test 3 - Directory blob structure

```
typedef enum tagBLOBTYPE
{
        FILEDATALIST= 12288, //0x3000
        DRECTORYMETADATA= 16384, //0x4000
        FILEMETADATA = 20480, //0x5000
        FILEDATA = 24576, //0x6000
} BLOBTYPE;

struct DirectoryInfoStoreBlob
{
    HEADER header;
    DWORD childID;
    DWORD unknown[2];
    DWORD ParentID;
    DWORD neighbourID;
    DWORD unknown[2];
    WORD unknown;
    WORD DirectoryNameLength;
    WORD DirectoryName[DirectoryNameLength] ;
} ;

struct FileMetaDataBlob
{
        HEADER header;
        DWORD fileDataListID;
        DWORD unknown[2];
        DWORD parentID;
        DWORD unknown[3];
        WORD unknown;
        WORD filenameLength;
        WORD filename[filenameLength];
}
```

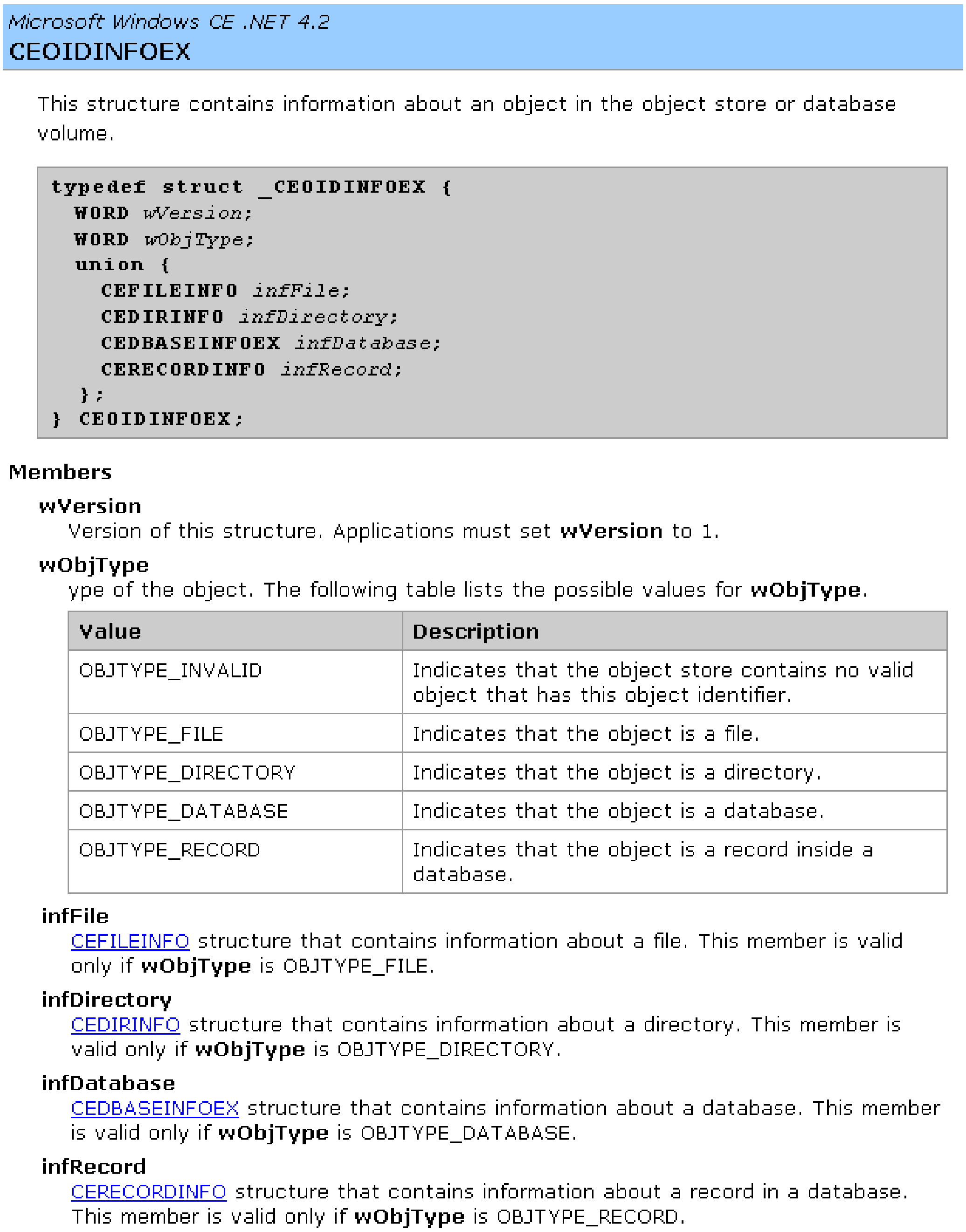
Extract info/Adjust PSM

Now, looking at DirectoryInfoStoreBlob and FileMetaDataBlob in listing 4.5 we see that they have a striking resemblance to each other. They have the same header of course. The next field they have is an id to some kind of child blob. Files have id of FILEDATALIST, while directories have the id of their first child. Next they both have two currently unknown dword, before both have a dword indicating their parent blob. Directories then have a neighbor field and two unknown dwords, while files have 3 unknowns. Seeing how they've matched each other thus far, how about we check if the first of these last 3 unknown dwords for files might possibly be a neighbor field too! Look at *CMMapP's* file blob starting at offset 0x03809FC0. Add 28 to get to the suspected neighbor field. The value here is *0x711C0000*(figure 4.17, green markings). This is also the id of *CMMapG*(the grey markings), it's neighbor.

Seeing as DirectoryInfoStoreBlob and FileMetaDataBlob has been more or less identical so far, we suspect the last unknown parts of each of them to also be similar. What other metadata is it reasonable for the OS to store about a file? A timestamp of last access or creation and some kind of system flags certainly are. The presence of a timestamp is supported by both the fact that you can view the last accessed time of a file on the device (figure 4.21), and the fact that a call to the *CeOidGetInfo* procedure(figure 4.11) mentioned in test two fills out a *CEOIDINFOEX* structure (figure 4.19) which again contains a *CEFILEINFO* structure (figure 4.20).

We see that this structure has a field of type *FILETIME* in it. This *FILETIME* has to come from somewhere, and since other metadata about a file is already stored in the FILEMETADATA blob, it makes sense to store it there. So let's see if some of the unknown bytes in FileMetaDataBlob(a blob with type FILEMETADATA) and DirectoryInfoStoreBlob(a blob with type DIRECTORYINFO) might be *FILETIME*. Looking at listing 4.5 we notice that the two possibilities are either between the childID/fileDataListID or right after neighborID.

Microsoft defines *FILETIME* in its API(figure 4.22). It consists of two dwords, which incidentally is the size of our two possibilities too. These two dwords together represent the number of 100 nanosecond intervals since 01.01.1601. The first dword represents the low bits, and the second the high bits. The high, or most significant, bits should not change for files or directories



Microsoft Windows CE .NET 4.2

CEOIDINFOEX

This structure contains information about an object in the object store or database volume.

```
typedef struct _CEOIDINFOEX {
  WORD wVersion;
  WORD wObjType;
  union {
    CEFILEINFO infFile;
    CEDIRINFO infDirectory;
    CEDBASEINFOEX infDatabase;
    CERECORDINFO infRecord;
  };
} CEOIDINFOEX;
```

Members

wVersion
Version of this structure. Applications must set **wVersion** to 1.

wObjType
ype of the object. The following table lists the possible values for **wObjType**.

Value	Description
OBJTYPE_INVALID	Indicates that the object store contains no valid object that has this object identifier.
OBJTYPE_FILE	Indicates that the object is a file.
OBJTYPE_DIRECTORY	Indicates that the object is a directory.
OBJTYPE_DATABASE	Indicates that the object is a database.
OBJTYPE_RECORD	Indicates that the object is a record inside a database.

infFile
CEFILEINFO structure that contains information about a file. This member is valid only if **wObjType** is OBJTYPE_FILE.

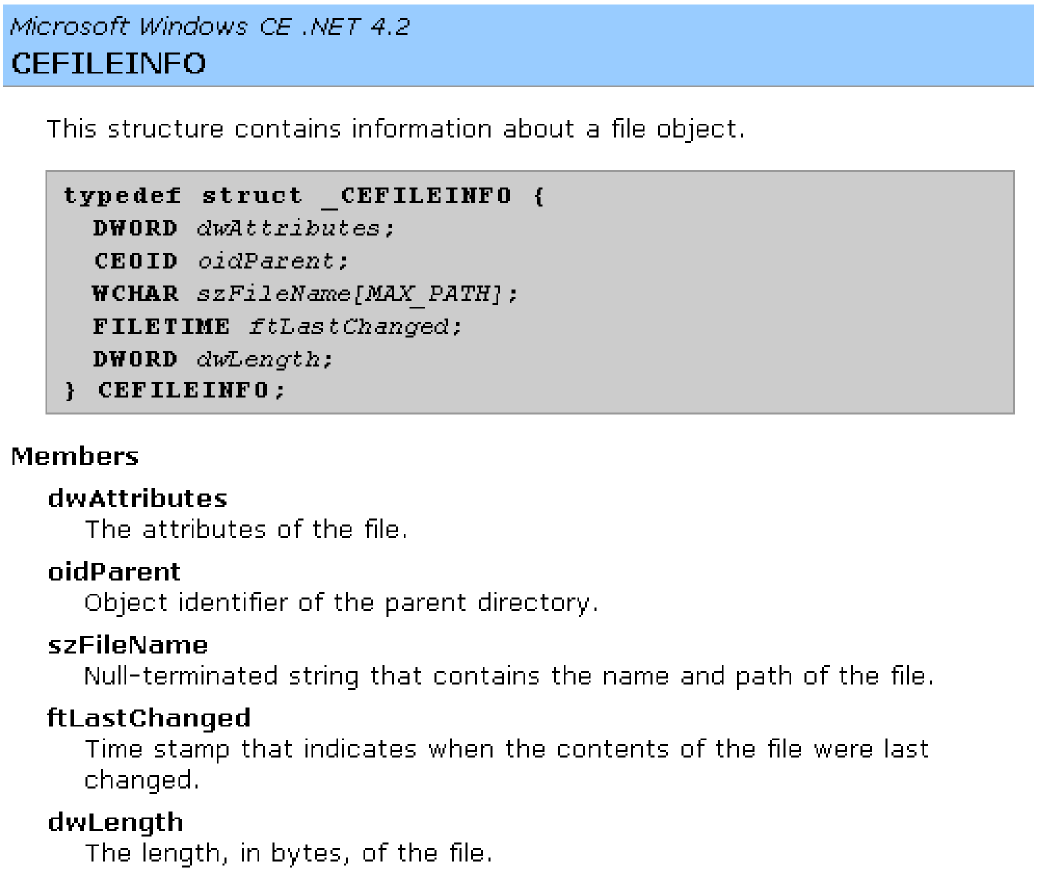
infDirectory
CEDIRINFO structure that contains information about a directory. This member is valid only if **wObjType** is OBJTYPE_DIRECTORY.

infDatabase
CEDBASEINFOEX structure that contains information about a database. This member is valid only if **wObjType** is OBJTYPE_DATABASE.

infRecord
CERECORDINFO structure that contains information about a record in a database. This member is valid only if **wObjType** is OBJTYPE_RECORD.

Figure 4.19: WINDOWS CE API - CEOIDINFOEX structure

stored at approximately the same time. Let us investigate the values of the two unknown dwords after neighborID in the input files from test 2(fig-



Microsoft Windows CE .NET 4.2

CEFILEINFO

This structure contains information about a file object.

```
typedef struct _CEFILEINFO {
  DWORD dwAttributes;
  CEOID oidParent;
  WCHAR szFileName[MAX_PATH];
  FILETIME ftLastChanged;
  DWORD dwLength;
} CEFILEINFO;
```

Members

dwAttributes
The attributes of the file.

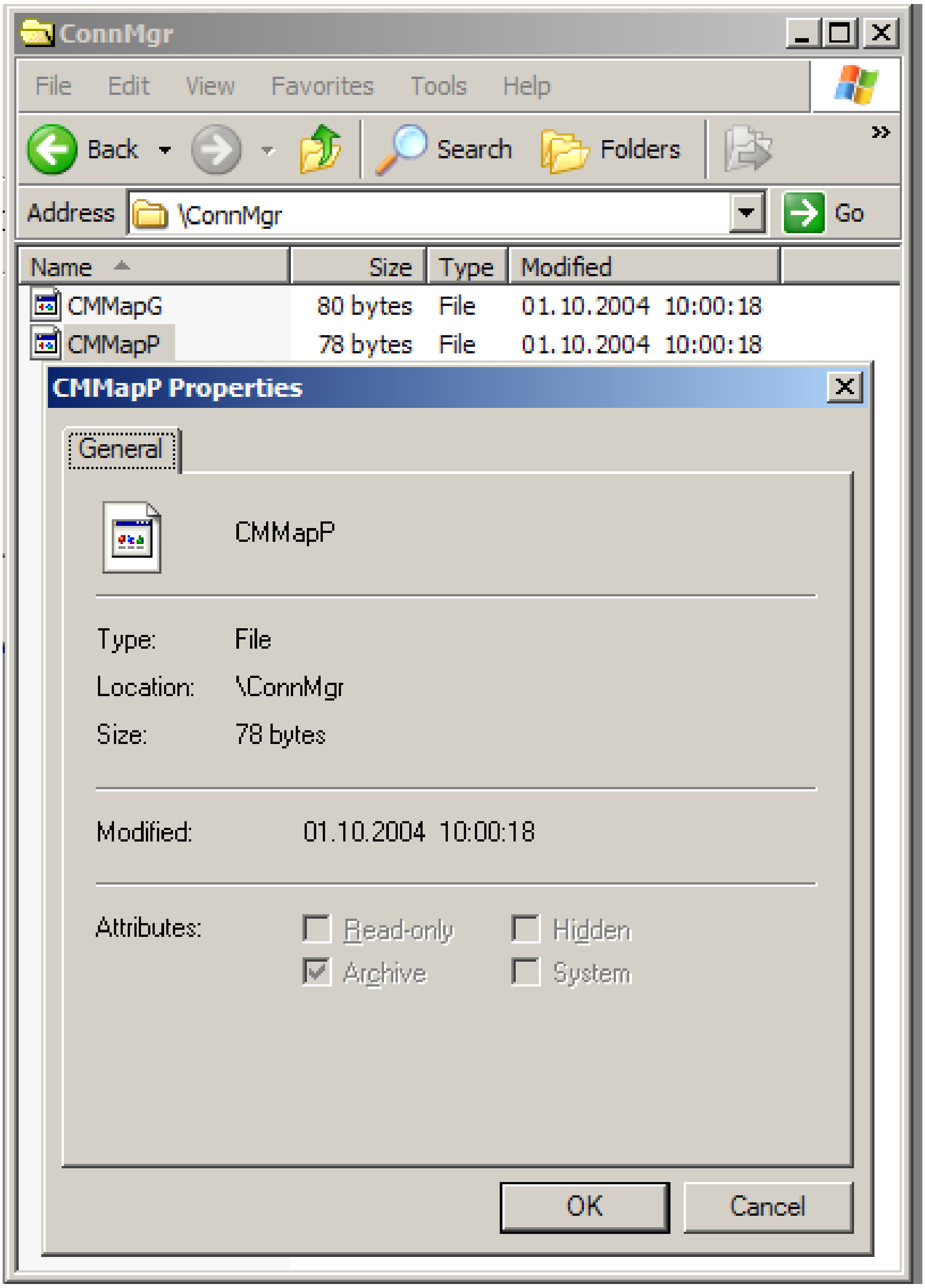
oidParent
Object identifier of the parent directory.

szFileName
Null-terminated string that contains the name and path of the file.

ftLastChanged
Time stamp that indicates when the contents of the file were last changed.

dwLength
The length, in bytes, of the file.

Figure 4.20: WINDOWS CE API - CEFILEINFO structure



Address: \ConnMgr

Name	Size	Type	Modified
CMMapG	80 bytes	File	01.10.2004 10:00:18
CMMapP	78 bytes	File	01.10.2004 10:00:18

CMMapP Properties

General

CMMapP

Type: File

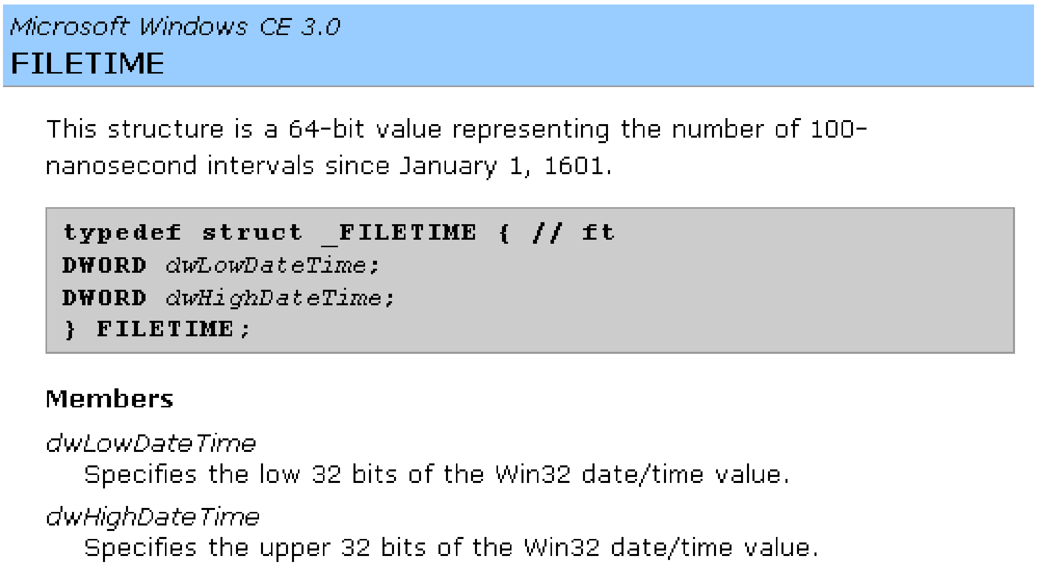
Location: \ConnMgr

Size: 78 bytes

Modified: 01.10.2004 10:00:18

Attributes: □ Read-only □ Hidden ☑ Archive □ System

Figure 4.21: Test 3 - Properties of a file.



Microsoft Windows CE 3.0

FILETIME

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

```
typedef struct _FILETIME { // ft
DWORD dwLowDateTime;
DWORD dwHighDateTime;
} FILETIME;
```

Members

dwLowDateTime
Specifies the low 32 bits of the Win32 date/time value.

dwHighDateTime
Specifies the upper 32 bits of the Win32 date/time value.

Figure 4.22: WINDOWS CE API - FILETIME structure

ure 4.9) and the directories from test 3(figure 4.16). Table 4.2 list these values. We quickly see that files that were stored at approximately the same time show few, if any changes in the 2. unknown dword. This is because it takes quite a while to count to 2^{32}, even if you increase your count every 100 nanosecond. We confirm that these two dwords are indeed filetime stamps by filling a *FILETIME* structure with them in a short test program in C. Converting the structure to its string representation with the API method FileTimeToSystemTime yields the same exact time as the properties of the file the timestamp was taken from.

The last word in DirectoryInfoStoreBlob and FileMetaDataBlob was suspected to be a property flag field. Our reasoning for this was that the value seemed to almost never change, and the observed values had the distinct look of being 1-bit flags logically or-ed together. In table 4.3 we list values of this last word for different blobs. Most of the blobs are user created files that we put on the phone. There are also some of the files included with the operating system included. Together with the value, shown in binary format, we also list the properties of the files as listed by our Judas Forensic Tool (4.1).

The combination of the raw data values and the OS-reported properties we

Name	Offset	unknown dword 1	unknown dword 2
dokhest1	0x036FF4C8	0x007CD7E4	0x86A7C401
dokhest2	0x036FF578	0x80FD4B73	0x87A7C401
dokhest3	0x036FF6B8	0x80809EF7	0x87A7C401
eselbase	0x03700810	0x00D3ED33	0xC0A7C401
eselbase/esel1	0x03700850	0x005A8137	0xC0A7C401
eselbase/esel2	0x037008CC	0x804A7C3A	0xC0A7C401
eselbase/esel3	0x03700904	0x80D10F3E	0xC0A7C401
eselbase/esel4	0x0370093C	0x00C20A41	0xC0A7C401

Table 4.2: Test 3 - File time dwords

Name	Suspected flag word(binary)	Reported properties
dokhest1	1000100000000	archive, compressed
dokhest2	1000100000000	archive, compressed
dokhest3	1000100000000	archive, compressed
desktop.ini	1011000000000	compressed, hidden, system
GCounterFile.mmf	1001100000000	archive, compressed, hidden
eselbase	0000000000000	directory
eselbase/esel1	0000000000000	directory
eselbase/esel2	0000000000000	directory
eselbase/esel3	0000000000000	directory
eselbase/esel4	0000000000000	directory

Table 4.3: Test 3 - Property flag word.

can conclude that this is in fact a property flag. We also deducted which bits that corresponded to a particular property. We see that all the blobs which have the "archive" property set has bit 8 set to 1, and all those who do not have this property has bit 8 set to 0. Bit 8 is the archive bit. This leaves bit 12 as the compressed bit, as this is the only other bit set in the *dokhest* files. Now that we know this, the only bit available as the hidden bit for *GCounterFile.mmf* is bit 9, as the two other bits are the compressed and archive bit. Finally we can deduct that bit 10 is the system bit because this is the only bit left set for *desktop.ini* given that bit 12 is the compressed bit and bit 9 is the hidden bit. The results are listed in table 4.4.

Bit mask	Property
compressed	1000000000000
system	0010000000000
hidden	0001000000000
archive	0000100000000

Table 4.4: Test 3 - Property flag bit masks

Transform to PIM

Listing 4.6 sums up what we know about the object store at the end of test 3.

Listing 4.6: Test 3 - Blob structures v5

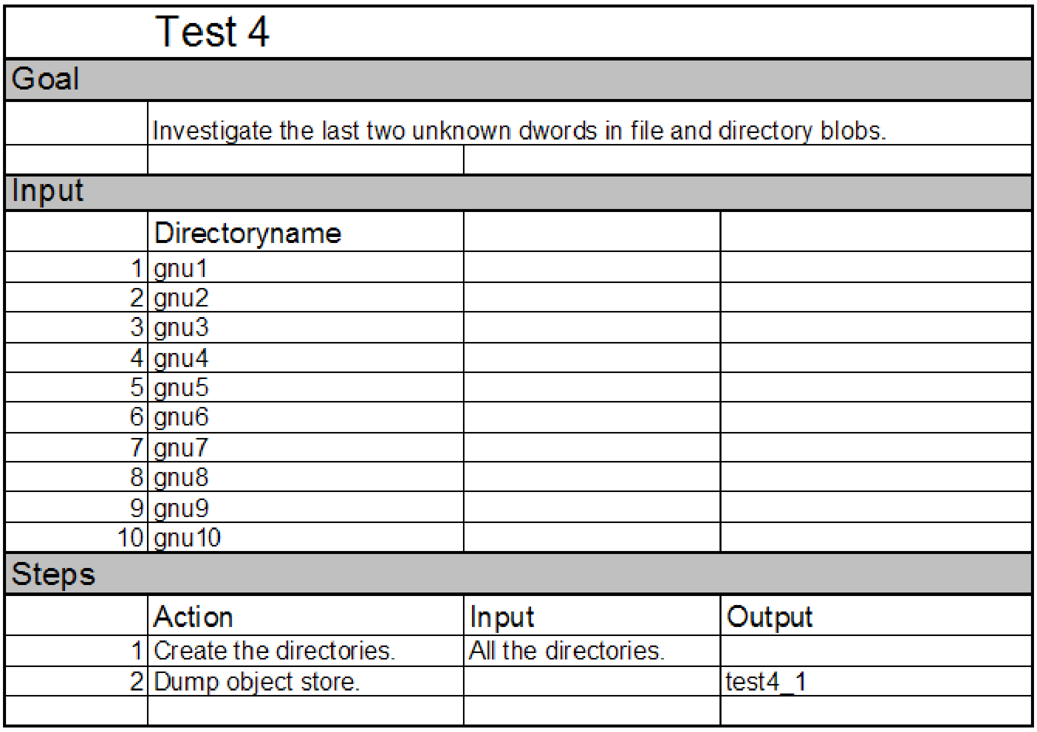
```
typedef enum tagBLOBTYPE
{
        FILEDATALIST= 12288, //0x3000
        DIRECTORYINFO = 16384, //0x4000
        FILEMETADATA = 20480, //0x5000
        FILEDATA = 24576, //0x6000
} BLOBTYPE;

typedef struct BlobHeader
{
        WORD size;
        BLOBTYPE blobType;
        DWORD zeroFiller;
        DWORD ID;
} HEADER;

struct GeneralBlob
{
        HEADER header;
        BYTE data[header.size];
};

struct FileDataListBlob
{
        HEADER header;
        DWORD fileDataID;
        BYTE unknown[header.size - 4]; //we suspect this might contain more fileDataIDs if the file is larger than 65535 bytes.
}

struct FileMetaDataBlob
{
        HEADER header;
        DWORD fileDataListID;
        DWORD unknown[2];
        DWORD parentID;
        DWORD neighbourID;
        DWORD lowOrderTime;
    DWORD highOrderTime;
        WORD propertyFlags;
        WORD filenameLength;
        WORD filename[filenameLength];

}

struct DirectoryInfoStoreBlob
{
    HEADER header;
    DWORD childID;
    DWORD unknown[2];
    DWORD parentID;
    DWORD neighbourID;
    DWORD lowOrderTime;
    DWORD highOrderTime;
    WORD propertyFlags;
    WORD DirectoryNameLength;
    WORD DirectoryName[DirectoryNameLength]  ;
}

struct FileDataBlob
{
        HEADER header;
        WORD suspected_storageType;
        BYTE fileData[header.size - 2];
}
```

4.2.3.4 Test 4

Create test

For file and directory blobs we only lack information about 2 dwords. We now want to find out what these dwords contains. As we don't have any clues about these fields we add 10 new files and see if we can draw some conclusions by looking at the differences between the unknown dwords in them .

Figure 4.23 contains the fourth test.



	Test 4		
Goal			
	Investigate the last two unknown dwords in file and directory blobs.		
Input			
	Directoryname		
1	gnu1		
2	gnu2		
3	gnu3		
4	gnu4		
5	gnu5		
6	gnu6		
7	gnu7		
8	gnu8		
9	gnu9		
10	gnu10		
Steps			
	Action	Input	Output
1	Create the directories.	All the directories.	
2	Dump object store.		test4_1

Figure 4.23: Test 4 - New user-created directories

Extract info/Adjust PSM

Other than concluding that the first dword seemed to remain constant, hex value *0x15000000*, we did not manage to draw any more knowledge from this test.

What these two dwords contain seems to remain unknown at this time, but what we do know is that what we have is more than enough information to extract files and directories directly from a memory dump of the device without them. The unknown dwords are some kind of metadata which is of little interest because of this. What we need to do now is to develop a tool to utilize and confirm our findings so far. We also need to investigate and conduct tests of other blob types. The device is known to have a database system, how is this stored? And how are text messages stored?

4.2.3.5 Test 5

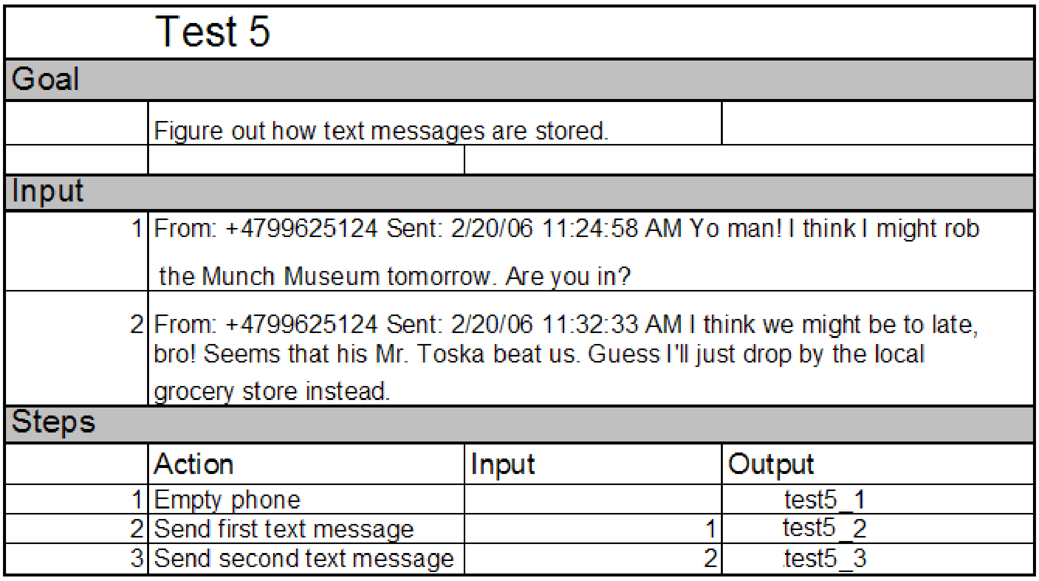
Create test

As our device is a mobile phone; text messages are of great interest. We want to figure out how they are stored. We know from [6] that they are stored using Windows CE's built in database functionality[20]. If this is the case, we need to find out how such databases are stored in the object store. Grattan and Brain[21] shows that database records are considered first class citizens in the Windows CE object store and each get a unique oid. This leads us to suspect that records might simply be yet another blob, with a new type value. We test this by sending two text messages to the phone.

Figure 4.24 contains the fifth test.

Extract info/Adjust PSM

A text search for the phrase *"rob the"* yields a hit at offset 0x037039EB. This again is clearly a part of the blob of length 0x108(decimal 264) starting at offset 0x03703954 (figure 4.25). As suspected, the message is stored as a new type of blob, with a blob type of *0x8000*. What kind of blob is this? There are a couple of possibilities. Either text messages have their very own blob type, or blobs of type *0x8000* are actually database records. Considering the fact that the specific operating system Windows Mobile, which our device runs, is based on the much more general Windows CE, we find it unlikely that text messages have their own blob type. Why would Windows CE be designed to treat text messages as something special when a high percentage of the devices that utilize it dont even have mobile phone abilities?



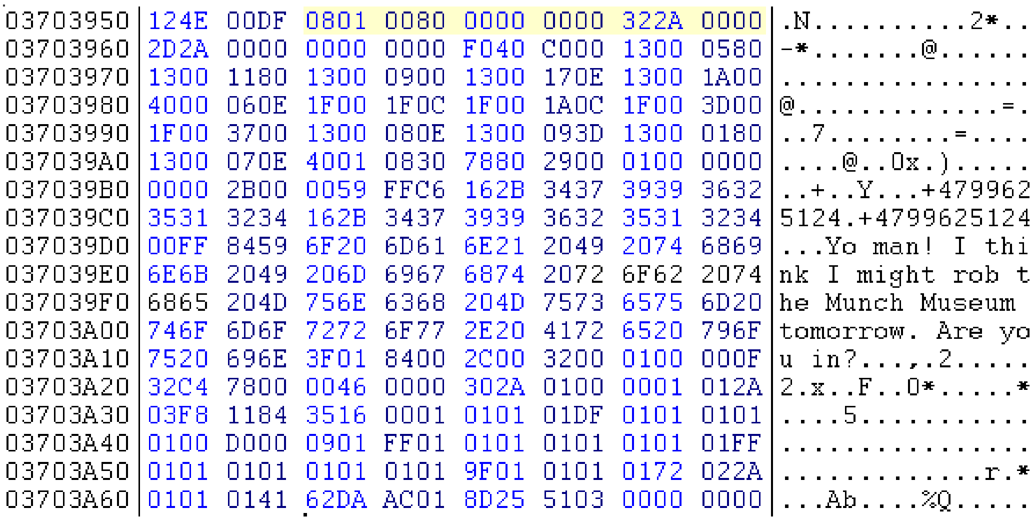
	Test 5		
Goal			
	Figure out how text messages are stored.		
Input			
1	From: +4799625124 Sent: 2/20/06 11:24:58 AM Yo man! I think I might rob the Munch Museum tomorrow. Are you in?		
2	From: +4799625124 Sent: 2/20/06 11:32:33 AM I think we might be to late, bro! Seems that his Mr. Toska beat us. Guess I'll just drop by the local grocery store instead.		
Steps			
	Action	Input	Output
1	Empty phone		test5_1
2	Send first text message	1	test5_2
3	Send second text message	2	test5_3

Figure 4.24: Test 5 - Text messages

We confirm that this indeed is a database record by looking at the first dword in its data field. This reads *0x2D2A0000* and from the other blobs we know that this field usually indicates some kind of parent id. Looking up this parent id we get a hit at offset 0x037026BC (figure 4.26). Here we find a blob with the type value of *0x7000* that contains the string *"fldr1000cb"*. This happens to be the name of one of the databases exported by the phone, available through the remote file explorer (figure 4.27). We conclude that blobs of type *0x8000* are database records and blobs of type *0x7000* are blobs that contains metadata about a database.

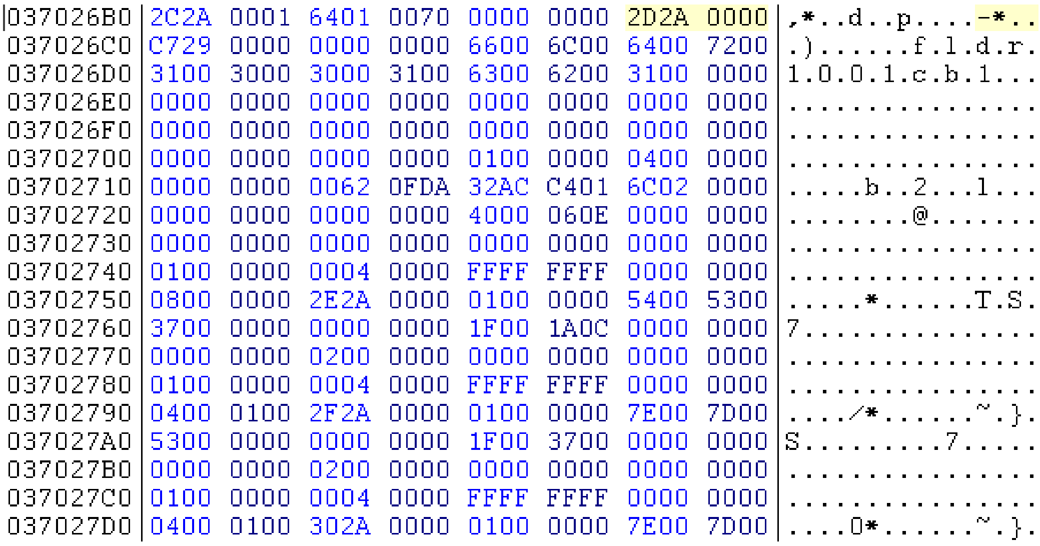
Looking at the data in figure 4.25 we recognize three strings immediately: *"+4799625124"*, *"+4799625124"* and *"Yo man! I think I might rob the Munch Museum tomorrow. Are you in?"*. The first two strings are phone numbers, most likely from and to given that this text message was sent and received with the same SIM card. The last string is of course the content in the first text message of this test. Each string is stored in pure ASCII(not Unicode) and seems to be prefixed with a byte that indicates the length of the string times two. *"+4799625124"* is 11 characters long, and is prefixed with the byte value 0x16(decimal 22, which is 11 * 2).

The rest of the blob was harder to interpret. We see that there are repeat-



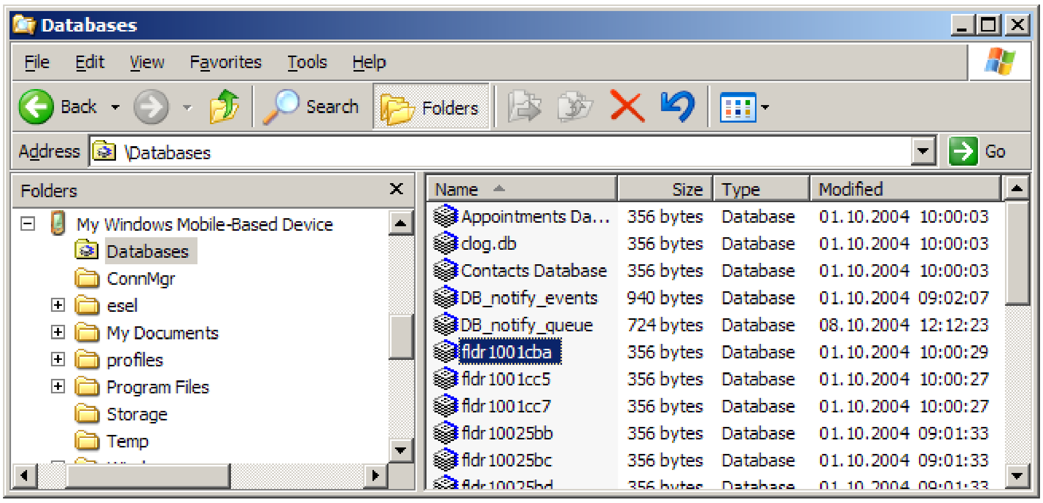
```
03703950|124E 00DF 0801 0080 0000 0000 322A 0000|.N..........2*..
03703960|2D2A 0000 0000 0000 F040 C000 1300 0580|-*.......@......
03703970|1300 1180 1300 0900 1300 170E 1300 1A00|................
03703980|4000 060E 1F00 1F0C 1F00 1A0C 1F00 3D00|@.............=.
03703990|1F00 3700 1300 080E 1300 093D 1300 0180|..7.......=.....
037039A0|1300 070E 4001 0830 7880 2900 0100 0000|....@..0x.).....
037039B0|0000 2B00 0059 FFC6 162B 3437 3939 3632|..+..Y...+479962
037039C0|3531 3234 162B 3437 3939 3632 3531 3234|5124.+4799625124
037039D0|00FF 8459 6F20 6D61 6E21 2049 2074 6869|...Yo man! I thi
037039E0|6E6B 2049 206D 6967 6874 2072 6F62 2074|nk I might rob t
037039F0|6865 204D 756E 6368 204D 7573 6575 6D20|he Munch Museum 
03703A00|746F 6D6F 7272 6F77 2E20 4172 6520 796F|tomorrow. Are yo
03703A10|7520 696E 3F01 8400 2C00 3200 0100 000F|u in?...,.2.....
03703A20|32C4 7800 0046 0000 302A 0100 0001 012A|2.x..F..0*.....*
03703A30|03F8 1184 3516 0001 0101 01DF 0101 0101|....5...........
03703A40|0100 D000 0901 FF01 0101 0101 0101 01FF|................
03703A50|0101 0101 0101 0101 9F01 0101 0172 022A|.............r.*
03703A60|0101 0141 62DA AC01 8D25 5103 0000 0000|...Ab....%Q.....
```

Figure 4.25: Test 5 - Rob the museum.



```
037026B0|2C2A 0001 6401 0070 0000 0000 2D2A 0000|,*..d..p....-*..
037026C0|C729 0000 0000 0000 6600 6C00 6400 7200|.)......f.l.d.r.
037026D0|3100 3000 3000 3100 6300 6200 3100 0000|1.0.0.1.c.b.1...
037026E0|0000 0000 0000 0000 0000 0000 0000 0000|................
037026F0|0000 0000 0000 0000 0000 0000 0000 0000|................
03702700|0000 0000 0000 0000 0100 0000 0400 0000|................
03702710|0000 0000 0062 0FDA 32AC C401 6C02 0000|.....b..2...l...
03702720|0000 0000 0000 0000 4000 060E 0000 0000|........@.......
03702730|0000 0000 0000 0000 0000 0000 0000 0000|................
03702740|0100 0000 0004 0000 FFFF FFFF 0000 0000|................
03702750|0800 0000 2E2A 0000 0100 0000 5400 5300|.....*......T.S.
03702760|3700 0000 0000 0000 1F00 1A0C 0000 0000|7...............
03702770|0000 0000 0200 0000 0000 0000 0000 0000|................
03702780|0100 0000 0004 0000 FFFF FFFF 0000 0000|................
03702790|0400 0100 2F2A 0000 0100 0000 7E00 7D00|..../*......~.}.
037027A0|5300 0000 0000 0000 1F00 3700 0000 0000|S.........7.....
037027B0|0000 0000 0200 0000 0000 0000 0000 0000|................
037027C0|0100 0000 0004 0000 FFFF FFFF 0000 0000|................
037027D0|0400 0100 302A 0000 0100 0000 7E00 7D00|....0*......~.}.
```

Figure 4.26: Test 5 - Rob the museum's parent.



Name	Size	Type	Modified
Appointments Da...	356 bytes	Database	01.10.2004 10:00:03
clog.db	356 bytes	Database	01.10.2004 10:00:03
Contacts Database	356 bytes	Database	01.10.2004 10:00:03
DB_notify_events	940 bytes	Database	01.10.2004 09:02:07
DB_notify_queue	724 bytes	Database	08.10.2004 12:12:23
fldr1001cba	356 bytes	Database	01.10.2004 10:00:29
fldr1001cc5	356 bytes	Database	01.10.2004 10:00:27
fldr1001cc7	356 bytes	Database	01.10.2004 10:00:27
fldr10025bb	356 bytes	Database	01.10.2004 09:01:33
fldr10025bc	356 bytes	Database	01.10.2004 09:01:33
fldr10025bd	356 bytes	Database	01.10.2004 09:01:33

Figure 4.27: Test 5 - Database found.

ing patterns of *"1300 XXXX"*, where *XXXX* varies, in the start of the blob. This indicates that something similar is repeated several times. What can be similar in a database record?

To answer this we have to look at the whole database model[21] in Windows CE. The databases are defined in a quite uncommon way. A database is just a collection of database records, kind of what we normally call a database table. In the traditional view, a database record would just be a row in this table. The table would also have a schema defining the data type and name of the different columns in it. In Windows CE a database has no schema. Instead each record defines a set of properties, which each consist of a type, an id and a value. All the records of a database can define their own set of properties, so two database records in the same database can actually be very different. The property types defined in Windows CE is given in listing 4.7

Listing 4.7: Test 5 - CEDB property types

```
#define CEVT_I2       0x02   // short
#define CEVT_UI2      0x12  // unsigned short
#define CEVT_I4       0x03   // int
#define CEVT_UI4      0x13  // unsigned int
#define CEVT_FILETIME 0x40  // FILETIME
#define CEVT_LPWSTR   0x1F  // LPWSTR
```

```
#define CEVT_BLOB       0x41   // BYTE*
#define CEVT_BOOL       0x0B   // BOOL
#define CEVT_R8         0x05    // double
```

Now we see a clearer picture forming in figure 4.25. Let's start at offset 0x0370396C. We see the mentioned *"1300 XXXX"* pattern. 0x13 is the data type for an unsigned 32-bit integer, a quite common data type to store integer values in. Might *"1300 XXXX"* actually mean "UINT32 ID"? This is likely when we look right after the *1300*-pattern stops. There we have *"4000 060E"*, followed by a new repeating pattern of *"1F00 XXXX"* before finishing up with a couple more *"1300 XXX"*. 0x40 is the data type for *FILETIME* and 0x1F is the data type of a string. This chunk of data is simply declaration of the type and id of the properties in the database record. If we count the number of property declarations we get 15. The value immediately prefixing the first property is 0xC0, which is a value below the data size of this blob, 0x108. Remembering the Windows CE developers fondness of prefixing length to their data structures, we think this might be the length of the property data. If one subtracts 0xC0 and 0x0C(bytes between start of data and the first property) from 0x108 you get 0x3C. 0x3C is 0x0F times 4. 0x0F is 15. You get the following formula: length (data) = 0x0C + numberOf (propertyDeclarations)*sizeOf (propertyDeclaration) + length (propertydata). The value right in front of the property declarations is the length of the property values. We checked this also for the other text message, it holds true.

The data after the property declarations is the values for the properties, in the same order as they were declared. Utilizing this knowledge, we can transform figure 4.25 into the much more readable listing 4.8. Notice however that while we now know what data types the text message database record consists of, we don't know what the individual fields actually are. This is because this information is never stored in the database. It is up to the applications that use the database records to interpret the id of the property to give it a meaningful context. We did not spend much more time on trying to figure out each field exactly, as we already know the most important things like the text of the message, the sender and the receiver. The time the message was sent is also most probably stored in the *FILETIME* field.

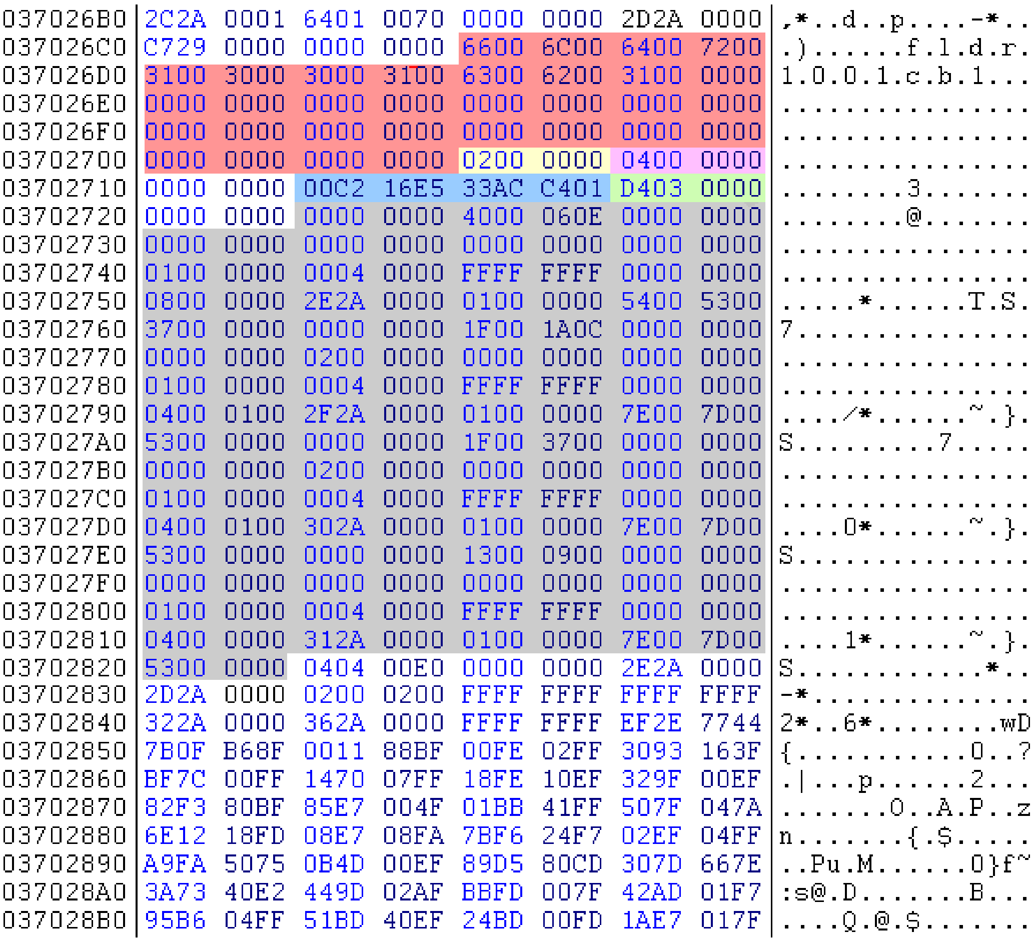
Listing 4.8: Test 5 - Text message

```
CEVT_UI4 id0580 = 0x0830
CEVT_UI4 id1180 = 0x7880
CEVT_UI4 id0900 = 0x2900
CEVT_UI4 id170E = 0x0100
CEVT_UI4 id1A00 = 0x0000
CEVT_FILETIME id060E = 0x00002B00 0059FFC6
CEVT_LPWSTR id1F0C = "+4799625124"
CEVT_LPWSTR id1A0C = "+4799625124"
CEVT_LPWSTR id3D00 = "Yo␣man!␣I␣think␣I␣might␣rob␣the␣
    Munch␣Museum␣tomorrow.␣Are␣you␣in?"
CEVT_LPWSTR id3700 = ""
CEVT_UI4 id080E = 0x2C00
CEVT_UI4 id093D = 0x3200
CEVT_UI4 id0180 = 0x0100
CEVT_UI4 id070E = 0x000F
```

We know that each record has an id to the blob of their parent database. Figure 4.28 shows the blob for the database containing the two text messages in this test. It starts at offset 0x037026B4.

The size of the blob is 0x164(356 decimal). This is quite large. Reverse engineering every one of those bytes will take a long time, and the data might not even be very interesting. The records are where the actual data is kept, and we already know enough to extract those. How can we cut down on the amount of work needed to understand the database blob? If we knew what we were looking for, it would help immensely. Thinking back to test 3, we tested some assumptions by writing small test programs in C that called the *CeOidGetInfo* procedure (4.19). It takes a object identifier as parameter and fills out an *CEOIDINFOEX* (figure 4.19). If the object identifier is the id of a database, the *CEOIDINFOEX* structure contains a *CEDBASEINFOEX* (figure 4.29) structure. The information in this structure must be stored by the operating system in the object store, so we can expect all of the items mentioned here to be present in the blob!

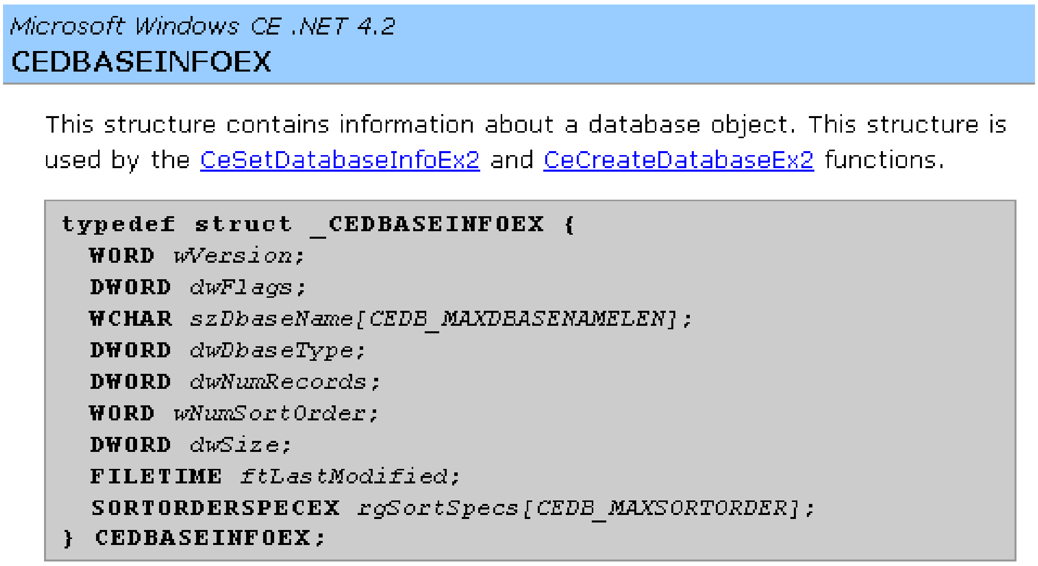
This fact reduces the job from guessing blind to mapping the elements in *CEDBASEINFOEX* to the blob in figure 4.28. Let's start with the name of the database. According to *CEDBASEINFOEX* this field should be *CEDB_MAXDBASENAMELEN* * 2 (wide char is 2 bytes) bytes long. Look-



```
037026B0 2C2A 0001 6401 0070 0000 0000 2D2A 0000 ,*..d..p....-*..
037026C0 C729 0000 0000 0000 6600 6C00 6400 7200 .)......f.l.d.r.
037026D0 3100 3000 3000 3100 6300 6200 3100 0000 1.0.0.1.c.b.1...
037026E0 0000 0000 0000 0000 0000 0000 0000 0000 ................
037026F0 0000 0000 0000 0000 0000 0000 0000 0000 ................
03702700 0000 0000 0000 0000 0200 0000 0400 0000 ................
03702710 0000 0000 00C2 16E5 33AC C401 D403 0000 ........3.......
03702720 0000 0000 0000 0000 4000 060E 0000 0000 ........@.......
03702730 0000 0000 0000 0000 0000 0000 0000 0000 ................
03702740 0100 0000 0004 0000 FFFF FFFF 0000 0000 ................
03702750 0800 0000 2E2A 0000 0100 0000 5400 5300 .....*......T.S.
03702760 3700 0000 0000 0000 1F00 1A0C 0000 0000 7...............
03702770 0000 0000 0200 0000 0000 0000 0000 0000 ................
03702780 0100 0000 0004 0000 FFFF FFFF 0000 0000 ................
03702790 0400 0100 2F2A 0000 0100 0000 7E00 7D00 ..../*......~.}.
037027A0 5300 0000 0000 0000 1F00 3700 0000 0000 S.........7.....
037027B0 0000 0000 0200 0000 0000 0000 0000 0000 ................
037027C0 0100 0000 0004 0000 FFFF FFFF 0000 0000 ................
037027D0 0400 0100 302A 0000 0100 0000 7E00 7D00 ....0*......~.}.
037027E0 5300 0000 0000 0000 1300 0900 0000 0000 S...............
037027F0 0000 0000 0000 0000 0000 0000 0000 0000 ................
03702800 0100 0000 0004 0000 FFFF FFFF 0000 0000 ................
03702810 0400 0000 312A 0000 0100 0000 7E00 7D00 ....1*......~.}.
03702820 5300 0000 0404 00E0 0000 0000 2E2A 0000 S.............*..
03702830 2D2A 0000 0200 0200 FFFF FFFF FFFF FFFF -*..............
03702840 322A 0000 362A 0000 FFFF FFFF EF2E 7744 2*..6*........wD
03702850 7B0F B68F 0011 88BF 00FE 02FF 3093 163F {...........0..?
03702860 BF7C 00FF 1470 07FF 18FE 10EF 329F 00EF .|...p......2...
03702870 82F3 80BF 85E7 004F 01BB 41FF 507F 047A .......O..A.P..z
03702880 6E12 18FD 08E7 08FA 7BF6 24F7 02EF 04FF n.......{.$.....
03702890 A9FA 5075 0B4D 00EF 89D5 80CD 307D 667E ..Pu.M......0}f~
037028A0 3A73 40E2 449D 02AF BBFD 007F 42AD 01F7 :s@.D.......B...
037028B0 95B6 04FF 51BD 40EF 24BD 00FD 1AE7 017F ....Q.@.$.......
```

Figure 4.28: Test 5 - Database blob.

ing at the dump we see that the name starts at 0x037026C0. *CEDB_MAXDBASENAMELEN* is defined as 32 in Windows CE, so the 64 next bytes are the name field (marked with red). Right after this field we find a dword with the value *2* (marked with yellow). This is the only place in the entire blob that we find the value 2. 2 is the number of records. This is probably the dwNumRecords field from *CEDBASEINFOEX*. Then we have the value *4* (marked with purple). The Windows CE API specifies that the wNumSortOrder field can be a maximum of 4. Let's assume this word is the sort order for now. Next we have a dword with value *0*. It is hard to tell what this is. After this, however, we have two dwords that we instantly recognize as the bytes of a *FILETIME* structure (marked with blue) The high word of *0xC401* is well known by now, from the other tests,



Microsoft Windows CE .NET 4.2

CEDBASEINFOEX

This structure contains information about a database object. This structure is used by the CeSetDatabaseInfoEx2 and CeCreateDatabaseEx2 functions.

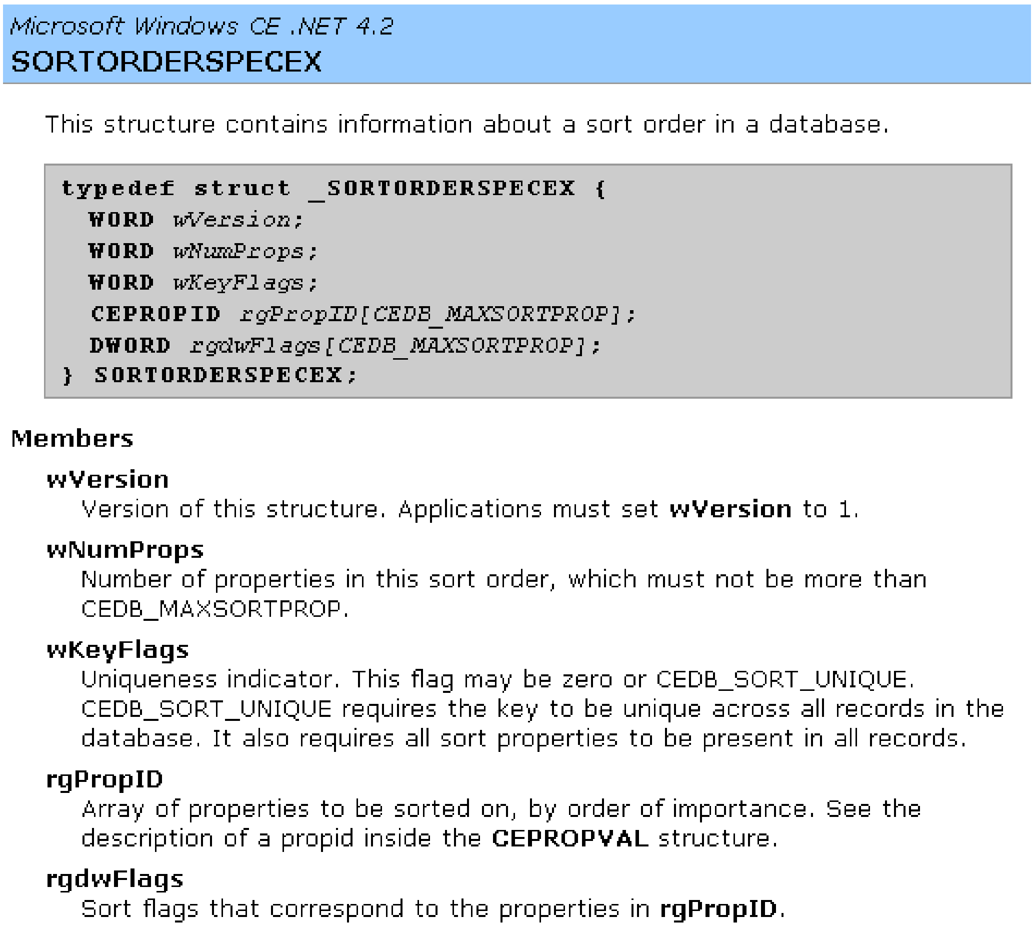
```
typedef struct _CEDBASEINFOEX {
  WORD wVersion;
  DWORD dwFlags;
  WCHAR szDbaseName[CEDB_MAXDBASENAMELEN];
  DWORD dwDbaseType;
  DWORD dwNumRecords;
  WORD wNumSortOrder;
  DWORD dwSize;
  FILETIME ftLastModified;
  SORTORDERSPECEX rgSortSpecs[CEDB_MAXSORTORDER];
} CEDBASEINFOEX;
```

Figure 4.29: WINDOWS CE API - CEDBASEINFOEX structure

as the most significant bytes of *FILETIME*. So this is the ftLastModified field.

Assuming that we've identified the wNumSortOrder field correctly there should now be 4 *SORTORDERSPECEX* (4.30) structures in the unknown bytes, a dwFlags dword, a dwSize and maybe a wVersion. Saying anything conclusive about the dwFlags and wVersion is difficult because these might very well be zero, and therefore can map to several places. dwSize can not, because we know there are at least two records in the database. The last bytes (marked in grey) in the blob consists of a pattern repeating 4 times. This fits well with wNumSortOrder being 4, as this would entail there being 4 *SORTORDERSPECEX* structures. Now that these last bytes are filled by *SORTORDERSPECEXs* the only non-zero dword left in the blob is at offset 0x0370271C (marked with green). This then has to be the dwSize field from the *CEDBASEINFOEX* structures.

All these findings were confirmed by applying our assumptions on other database blobs and finding them to be correct for them too.



Microsoft Windows CE .NET 4.2

SORTORDERSPECEX

This structure contains information about a sort order in a database.

```
typedef struct _SORTORDERSPECEX {
  WORD wVersion;
  WORD wNumProps;
  WORD wKeyFlags;
  CEPROPID rgPropID[CEDB_MAXSORTPROP];
  DWORD rgdwFlags[CEDB_MAXSORTPROP];
} SORTORDERSPECEX;
```

Members

wVersion
Version of this structure. Applications must set **wVersion** to 1.

wNumProps
Number of properties in this sort order, which must not be more than CEDB_MAXSORTPROP.

wKeyFlags
Uniqueness indicator. This flag may be zero or CEDB_SORT_UNIQUE. CEDB_SORT_UNIQUE requires the key to be unique across all records in the database. It also requires all sort properties to be present in all records.

rgPropID
Array of properties to be sorted on, by order of importance. See the description of a propid inside the **CEPROPVAL** structure.

rgdwFlags
Sort flags that correspond to the properties in **rgPropID**.

Figure 4.30: WINDOWS CE API - SORTORDERSPECEX structure

After the *SORTORDERSPECEXs* a new blob starts. It has type *0xE000*. The first dword of its data is the id to another blob, in our case the just analyzed database blob. We think this blob is some kind of index blob for the database. It contains the id of all the database records in the database, sorted in different ways. It also contains several *FILETIME* timestamps. We found this kind of blob to be pretty uninteresting for us, given our success criteria. We did not spend lots of time trying to decipher it exactly.

Transform to PIM

The results we got from a short overview for this blob and all the other results from this test were combined and gave the new HW-structures in listing 4.9.

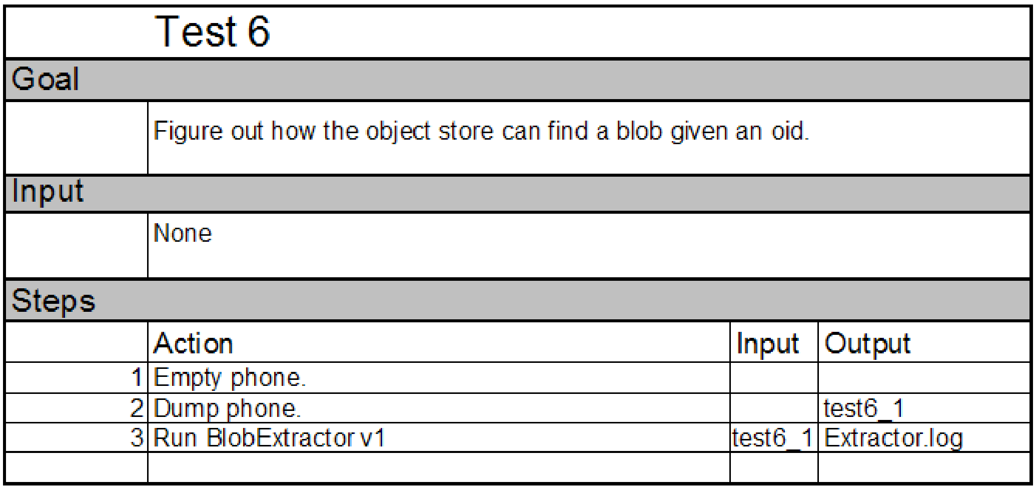
Listing 4.9: Test 5 - Database blob structures

```
typedef enum tagBLOBTYPE
{
        FILEDATALIST= 12288, //0x3000
        FILEMETADATA = 20480, //0x5000
        FILEDATA = 24576, //0x6000
        DATABASE = 28672, //0x7000
        DATABASERECORD = 32768, //0x8000
        DATABASEINDEX = 57344 //0xE000
} BLOBTYPE;


struct DatabaseBlob
{
    HEADER header;

    DWORD ParentID;
    DWORD databaseType;
    WORD databaseName[32];
    DWORD numRecords;
    DWORD numSortOrder;
    DWORD unknown;
    DWORD lastModifiedLowTime;
    DWORD lastModifiedLowTime;
    DWORD dataBaseSize;
    DWORD unknown;
    SortOrder sortOrders[4];
};

struct DatabaseRecordBlob
{
    HEADER header;

    DWORD ParentID;
    BYTE data[header.Size -4]
};

struct DatabaseIndexBlob
{
    HEADER header;

    DWORD parentID;
    WORD unknown;
    WORD numRecords;
    DWORD delimiter1[2]; //FFFFFFFF
    DWORD recordID[numRecords];
    DWORD delimiter2; //FFFFFFFF
    DWORD unknown[numRecords*10];
    UQUAD timeStamps[numRecords];
};
```

4.2.3.6 Test 6

Create test

We have a pretty good understanding of the internal structure of the blobs by now. One of the large remaining questions is how the blobs are combined into the large structure called the object store. Why are blobs stored at the offsets they are stored at, how does the operating system find a blob given an oid? Our initial guess was based on the information we had in the CIM. From it we know that most file systems/object stores have some kind of mechanism to look up an element given some kind of id, without searching through the entire file system. File systems/object stores must also be able to partition an element into as many subparts as needed by the underlying physical medium. For hard disks this is usually sectors.



	Test 6		
Goal			
	Figure out how the object store can find a blob given an oid.		
Input			
	None		
Steps			
	Action	Input	Output
1	Empty phone.		
2	Dump phone.		test6_1
3	Run BlobExtractor v1	test6_1	Extractor.log

Figure 4.31: Test 6 - VFAT?

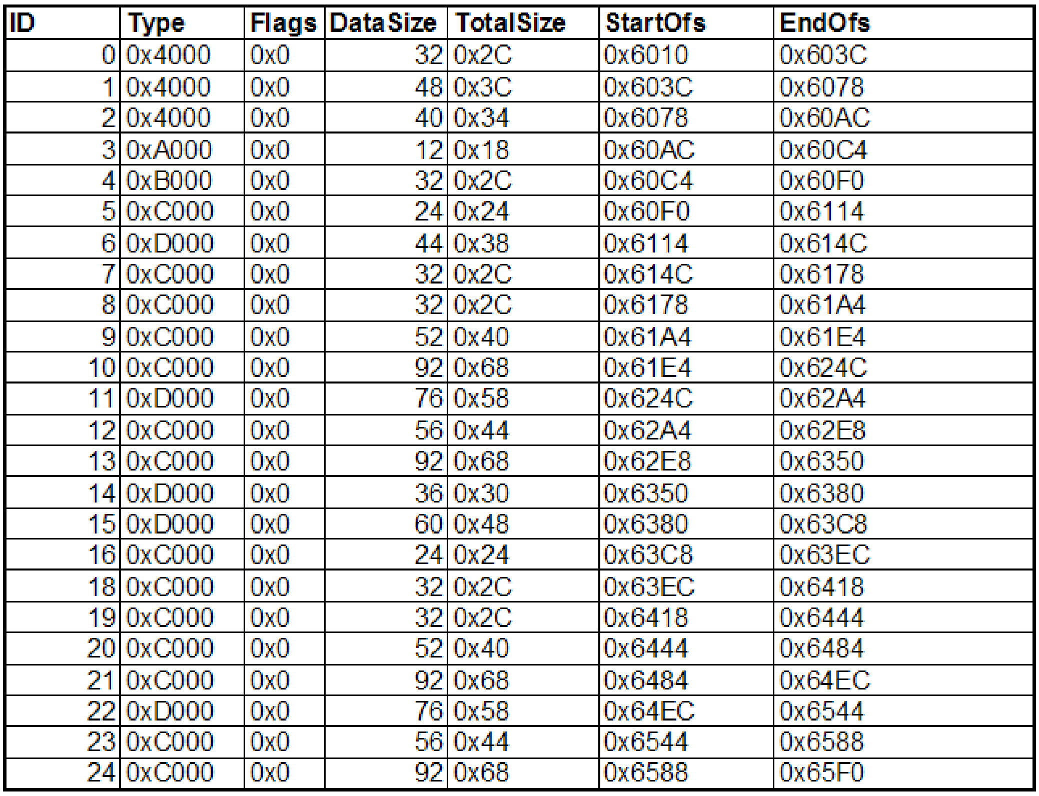
We've already seen this in the Windows CE object store. The content of files is split into several FILEDATA blobs which are connected through a FILEDATALIST blob.

With this test we basically want to check if we can uncover how the object store keeps track of which parts of the available storage has been assigned and which has not. Figuring out this will basically let us say that we know that we have access to all the data in the object store, not just a subset. Our test tool (the BlobExtractor) can, based on our findings so far, scan through a memory dump byte by byte using several heuristics to recognize blobs. While we know that it recognized most blobs this way, we can not be certain that our heuristics are wide enough to catch all cases or that they even are 100 percent correct. It would be better if we could figure out how the blobs are assigned their offsets in the object store in the first place and access them directly. We are hoping to find some kind of connection between a blob's oid and the actual offset of the blob.

Figure 4.31 contains the sixth test.

Extract info/Adjust PSM

The first thing we did was to run our BlobExtractor on the memory dump. After it was done it had recognized over 10000 individual blobs and writ-

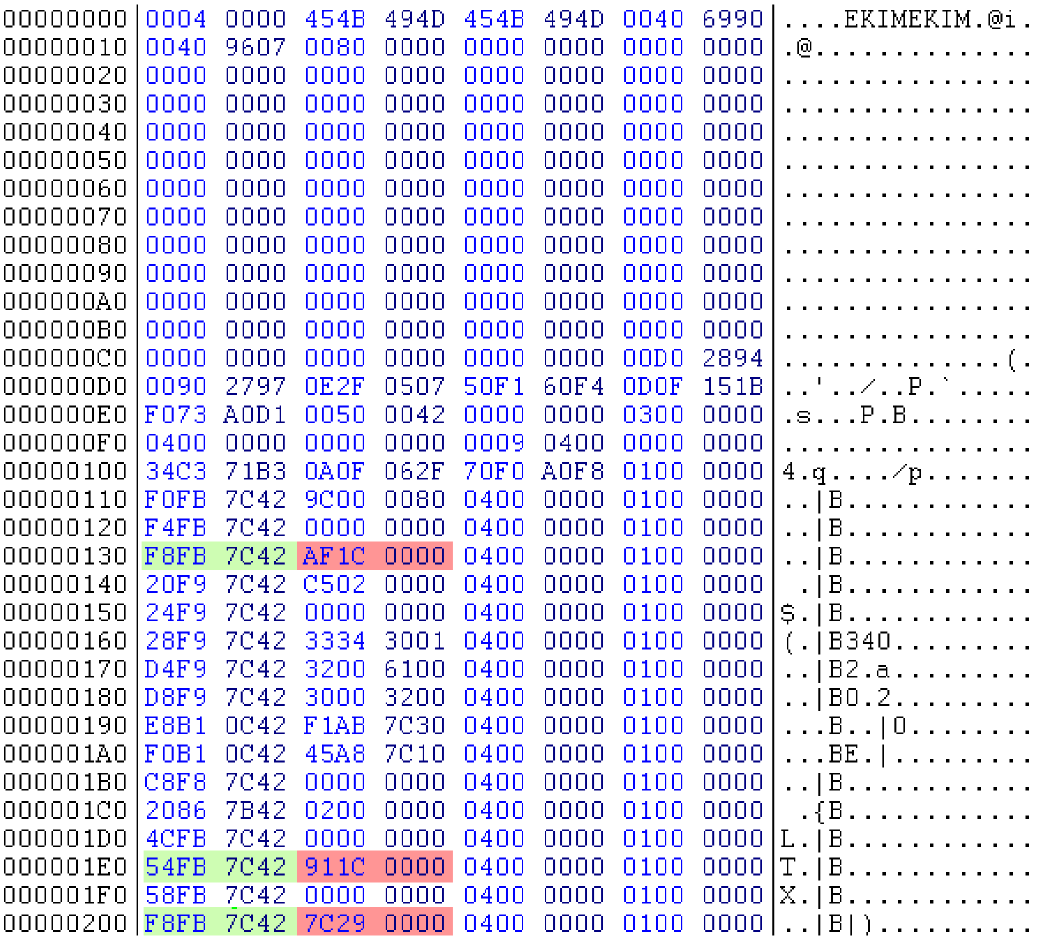


ID	Type	Flags	DataSize	TotalSize	StartOfs	EndOfs
0	0x4000	0x0	32	0x2C	0x6010	0x603C
1	0x4000	0x0	48	0x3C	0x603C	0x6078
2	0x4000	0x0	40	0x34	0x6078	0x60AC
3	0xA000	0x0	12	0x18	0x60AC	0x60C4
4	0xB000	0x0	32	0x2C	0x60C4	0x60F0
5	0xC000	0x0	24	0x24	0x60F0	0x6114
6	0xD000	0x0	44	0x38	0x6114	0x614C
7	0xC000	0x0	32	0x2C	0x614C	0x6178
8	0xC000	0x0	32	0x2C	0x6178	0x61A4
9	0xC000	0x0	52	0x40	0x61A4	0x61E4
10	0xC000	0x0	92	0x68	0x61E4	0x624C
11	0xD000	0x0	76	0x58	0x624C	0x62A4
12	0xC000	0x0	56	0x44	0x62A4	0x62E8
13	0xC000	0x0	92	0x68	0x62E8	0x6350
14	0xD000	0x0	36	0x30	0x6350	0x6380
15	0xD000	0x0	60	0x48	0x6380	0x63C8
16	0xC000	0x0	24	0x24	0x63C8	0x63EC
18	0xC000	0x0	32	0x2C	0x63EC	0x6418
19	0xC000	0x0	32	0x2C	0x6418	0x6444
20	0xC000	0x0	52	0x40	0x6444	0x6484
21	0xC000	0x0	92	0x68	0x6484	0x64EC
22	0xD000	0x0	76	0x58	0x64EC	0x6544
23	0xC000	0x0	56	0x44	0x6544	0x6588
24	0xC000	0x0	92	0x68	0x6588	0x65F0

Figure 4.32: Test 6 - BlobExtractor first edition output

ten them to a log file. Figure 4.32 shows the start of this file. As we see, the first blob was found around offset 0x6000 and the other blobs followed successively. This means that a lookup table for oids has to either be located before these 0x6000 bytes or it has to be intermingled with the rest of the blobs in the object store from offset 0x6000 and out.

Lets look at the data before offset 0x6000. From 0x16 to 0xD0 we have nothing, everything is zero-filled. From 0xD0 to until 0x3D0 there is clearly some kind of structure to the data. We also notice that there are what seems to be object identifiers mentioned here (figure 4.33 red markings). We also see that they appear right after what seems to be pointers (marked green) to places in the object store, given that the store is located at offset 0x42000000 in memory as figure 4.14 indicates. This is interesting, as it shows a connection between oids and pointers to addresses in the object store! But looking at the size of the data, this simply cannot be our lookup

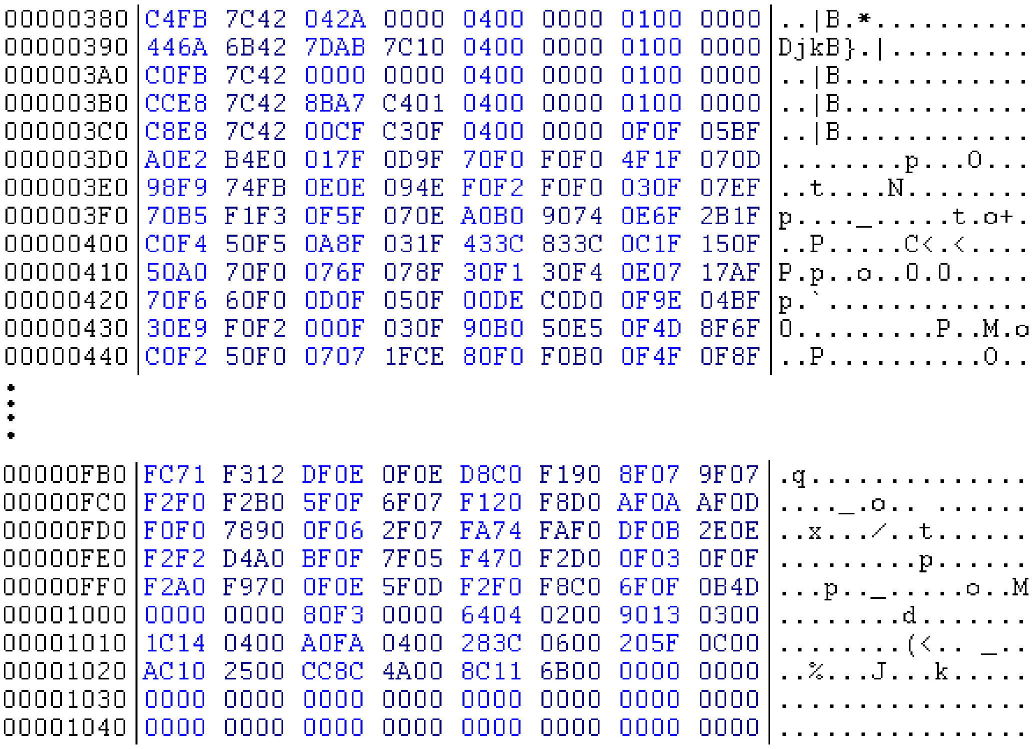


```
00000000 0004 0000 454B 494D 454B 494D 0040 6990 ....EKIMEKIM.@i.
00000010 0040 9607 0080 0000 0000 0000 0000 0000 .@..............
00000020 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000030 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000040 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000050 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000060 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000070 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000080 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000090 0000 0000 0000 0000 0000 0000 0000 0000 ................
000000A0 0000 0000 0000 0000 0000 0000 0000 0000 ................
000000B0 0000 0000 0000 0000 0000 0000 0000 0000 ................
000000C0 0000 0000 0000 0000 0000 0000 00D0 2894 ..............(.
000000D0 0090 2797 0E2F 0507 50F1 60F4 0D0F 151B ..'../..P.`.....
000000E0 F073 A0D1 0050 0042 0000 0000 0300 0000 .s...P.B........
000000F0 0400 0000 0000 0000 0009 0400 0000 0000 ................
00000100 34C3 71B3 0A0F 062F 70F0 A0F8 0100 0000 4.q..../p.......
00000110 F0FB 7C42 9C00 0080 0400 0000 0100 0000 ..|B............
00000120 F4FB 7C42 0000 0000 0400 0000 0100 0000 ..|B............
00000130 F8FB 7C42 AF1C 0000 0400 0000 0100 0000 ..|B............
00000140 20F9 7C42 C502 0000 0400 0000 0100 0000  .|B............
00000150 24F9 7C42 0000 0000 0400 0000 0100 0000 $.|B............
00000160 28F9 7C42 3334 3001 0400 0000 0100 0000 (.|B340.........
00000170 D4F9 7C42 3200 6100 0400 0000 0100 0000 ..|B2.a.........
00000180 D8F9 7C42 3000 3200 0400 0000 0100 0000 ..|B0.2.........
00000190 E8B1 0C42 F1AB 7C30 0400 0000 0100 0000 ...B..|0........
000001A0 F0B1 0C42 45A8 7C10 0400 0000 0100 0000 ...BE.|.........
000001B0 C8F8 7C42 0000 0000 0400 0000 0100 0000 ..|B............
000001C0 2086 7B42 0200 0000 0400 0000 0100 0000  .{B............
000001D0 4CFB 7C42 0000 0000 0400 0000 0100 0000 L.|B............
000001E0 54FB 7C42 911C 0000 0400 0000 0100 0000 T.|B............
000001F0 58FB 7C42 0000 0000 0400 0000 0100 0000 X.|B............
00000200 F8FB 7C42 7C29 0000 0400 0000 0100 0000 ..|B|)..........
```

Figure 4.33: Test 6 - Offset 0x0

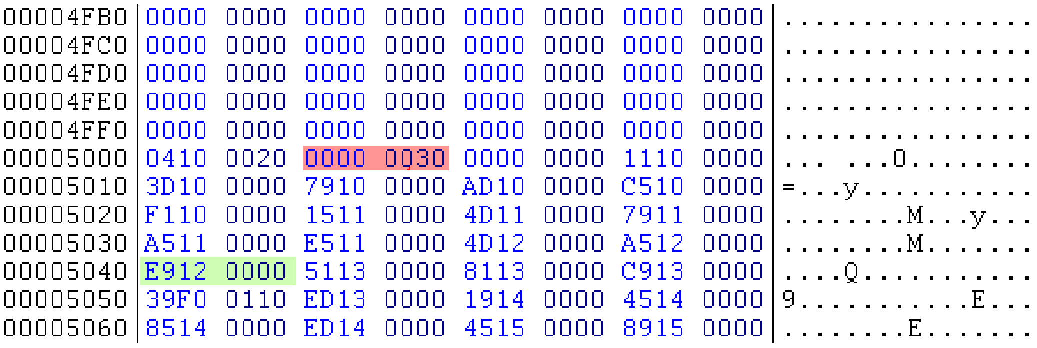
table. It would need room for over 10000 of these connections. 10000 * 4 * 2 = 80000, which is way more than the 0x300 available.

Next we have a section of similar data from offset 0x3D0 until 0x1000 (figure 4.34). The data looks uniform; the bytes have similar value ranges in the entire section. None of them give us any clue of their use but using the same reasoning as the last section, there simply isn't enough data here to hold a lookup table. From offset 0x1000 the data changes form. The 30 first bytes here definitely lie in a low value interval if we look at the data as dwords. None of them use their most significant byte. We also notice that the dwords rise in value from the first to the last. Interesting, but again there simply is not enough room for a lookup table for 10000



```
00000380 C4FB 7C42 042A 0000 0400 0000 0100 0000 ..|B.*..........
00000390 446A 6B42 7DAB 7C10 0400 0000 0100 0000 DjkB}.|.........
000003A0 C0FB 7C42 0000 0000 0400 0000 0100 0000 ..|B............
000003B0 CCE8 7C42 8BA7 C401 0400 0000 0100 0000 ..|B............
000003C0 C8E8 7C42 00CF C30F 0400 0000 0F0F 05BF ..|B............
000003D0 A0E2 B4E0 017F 0D9F 70F0 F0F0 4F1F 070D ........p...O...
000003E0 98F9 74FB 0E0E 094E F0F2 F0F0 030F 07EF ..t....N........
000003F0 70B5 F1F3 0F5F 070E A0B0 9074 0E6F 2B1F p...._.....t.o+.
00000400 C0F4 50F5 0A8F 031F 433C 833C 0C1F 150F ..P.....C<.<....
00000410 50A0 70F0 076F 078F 30F1 30F4 0E07 17AF P.p..o..0.0.....
00000420 70F6 60F0 0D0F 050F 00DE C0D0 0F9E 04BF p.`.............
00000430 30E9 F0F2 000F 030F 90B0 50E5 0F4D 8F6F 0.........P..M.o
00000440 C0F2 50F0 0707 1FCE 80F0 F0B0 0F4F 0F8F ..P..........O..
•
•
•
•
00000FB0 FC71 F312 DF0E 0F0E D8C0 F190 8F07 9F07 .q..............
00000FC0 F2F0 F2B0 5F0F 6F07 F120 F8D0 AF0A AF0D ...._.o.. ......
00000FD0 F0F0 7890 0F06 2F07 FA74 FAF0 DF0B 2E0E ..x.../..t......
00000FE0 F2F2 D4A0 BF0F 7F05 F470 F2D0 0F03 0F0F ..........p.....
00000FF0 F2A0 F970 0F0E 5F0D F2F0 F8C0 6F0F 0B4D ...p.._......o.M
00001000 0000 0000 80F3 0000 6404 0200 9013 0300 ........d.......
00001010 1C14 0400 A0FA 0400 283C 0600 205F 0C00 ........(<.. _..
00001020 AC10 2500 CC8C 4A00 8C11 6B00 0000 0000 ..%...J...k.....
00001030 0000 0000 0000 0000 0000 0000 0000 0000 ................
00001040 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

Figure 4.34: Test 6 - Offset 0x1000



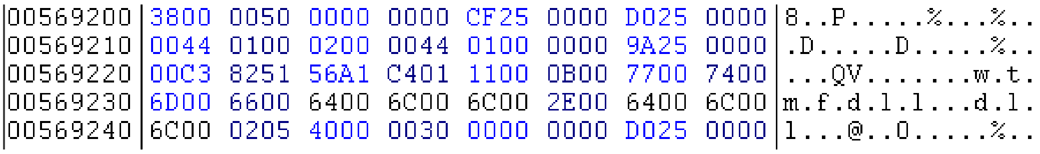
```
00004FB0 0000 0000 0000 0000 0000 0000 0000 0000 ................
00004FC0 0000 0000 0000 0000 0000 0000 0000 0000 ................
00004FD0 0000 0000 0000 0000 0000 0000 0000 0000 ................
00004FE0 0000 0000 0000 0000 0000 0000 0000 0000 ................
00004FF0 0000 0000 0000 0000 0000 0000 0000 0000 ................
00005000 0410 0020 0000 0030 0000 0000 1110 0000 ... ...0........
00005010 3D10 0000 7910 0000 AD10 0000 C510 0000 =...y...........
00005020 F110 0000 1511 0000 4D11 0000 7911 0000 ........M...y...
00005030 A511 0000 E511 0000 4D12 0000 A512 0000 ........M.......
00005040 E912 0000 5113 0000 8113 0000 C913 0000 ....Q...........
00005050 39F0 0110 ED13 0000 1914 0000 4514 0000 9...........E...
00005060 8514 0000 ED14 0000 4515 0000 8915 0000 ........E.......
```

Figure 4.35: Test 6 - Offset 0x5000

objects, as everything from 0x1030 to 0x5000 is filled with zeros. The data from 0x5000 to 0x6000 looks very interesting. First of all, we see that this is in fact a blob. Our heuristics missed this because it has a flag field with a non-zero value (figure 4.35, red markings), which we've never seen in any other blob. This blob has blob type of *0x2000*, which is also new. We see that the data in the blob follows a distinct pattern. Most likely these are dwords stored after each other, but only the bottom half of the dword is used. Might this be our table? The 2 lowest bytes from one of the dwords (green markings) have the value *0x12E9*. What if this is some kind of delta offset from the start of the data, 0x5000 or from the start of the entire object store? None of these theories seem to be correct, as we end up in what seems like "random" data, not at the start of a blob. This might be because we use the wrong base offset, or the dwords might not be pointers at all. All we know is that we can't conclude either way.

We also observe that the blob extractor recognizes several blobs of unknown type: *0xA000*, *0xB000*, *0xC000* and *0xD000* (figure 4.32). However, the size of these blobs is typically less than 100 and inspection of the blobs reveal that they are most likely connected with the registry part of Windows CE as registry keys and registry values of different data types. A lookup table will be quite large and while it could be split into several parts we see no reason to make these parts as small as 100 bytes.

Trying to attack the problem from another angle we chose a random blob in the dump; the file metadata blob of wtmfdll.dll at offset 0x00569200 (figure 4.36). If there is such a thing as a lookup table, it should contain the offset of the blob and/or the id of the blob. Let's try searching for the



```
00569200|3800 0050 0000 0000 CF25 0000 D025 0000|8..P.....%...%..
00569210|0044 0100 0200 0044 0100 0000 9A25 0000|.D.....D.....%..
00569220|00C3 8251 56A1 C401 1100 0B00 7700 7400|...QV.......w.t.
00569230|6D00 6600 6400 6C00 6C00 2E00 6400 6C00|m.f.d.l.l...d.l.
00569240|6C00 0205 4000 0030 0000 0000 D025 0000|l...@..0.....%..
```

Figure 4.36: Test 6 - Random file

id and the offset. The id is *0xCF250000*. This value is not used anywhere else in the dump, so a table with the id is out of the question. How about the offset? If the system can find the blob, and therefore the offset, given an id it has to store this offset somewhere. Alternatively it can calculate the offset from the id, but this is unlikely given that the ids are increasing by 1 for each new blob and each blob can be of a different size. Searching directly for the value of the offset 0x569200 yields nothing. The offset of the blob might very well be combination of a base offset and a delta offset. Lets make some educated guesses about possible base offsets. We search for *0x568200(base 0x1000)*, *0x564200(base 0x5000)* and *0x563200(base 0x6000)*. We had no luck, as we found nothing.

Clearly, a new information gathering strategy for finding out how exactly objects are given their place in the object store is needed. Our black box testing needs to be supported by something else.

Transform to PIM

The last thing we do before defining our new strategy is to update our HW-structure to include the new data types observed in this test (listing 4.10).

Listing 4.10: Test 6 - Blob structures v6

```
typedef enum tagBLOBTYPE
{
        UNKNOWN_POINTER_LIST? = 8192, //0x2000
        FILEDATALIST= 12288, //0x3000
        DIRECTORYINFO = 16384, //0x4000
        FILEMETADATA = 20480, //0x5000
        FILEDATA = 24576, //0x6000
        DATABASE = 28672, //0x7000
        DATABASERECORD = 32768, //0x8000
        REGISTER_UNKNOWN_A = 40960, //0xA000
        REGISTER_UNKNOWN_B = 45056, //0xB000
        REGISTER_KEY= 49152, //0xC000
        REGISTER_INT = 53248, //0xD000
        DATABASEINDEX = 57344 //0xE000
} BLOBTYPE;

typedef struct BlobHeader
{
        WORD size;
        BLOBTYPE blobType;
        DWORD zeroFiller;
        DWORD ID;
} HEADER;

struct GeneralBlob
{
        HEADER header;
        BYTE data[header.size];
};

struct FileDataListBlob
{
        HEADER header;
        DWORD fileDataID;
        BYTE unknown[header.size - 4];
}

struct FileMetaDataBlob
{
        HEADER header;
        DWORD fileDataListID;
        DWORD unknown[2];
        DWORD parentID;
        DWORD neighbourID;
        DWORD lowOrderTime;
    DWORD highOrderTime;
        WORD propertyFlags;
        WORD filenameLength;
        WORD filename[filenameLength];
}

```

```
struct DirectoryInfoStoreBlob
{
    HEADER header;
    DWORD childID;
    DWORD unknown[2];
    DWORD parentID;
    DWORD neighbourID;
    DWORD lowOrderTime;
    DWORD highOrderTime;
    WORD propertyFlags;
    WORD DirectoryNameLength;
    WORD DirectoryName[DirectoryNameLength] ;
}

struct FileDataBlob
{
        HEADER header;
        WORD suspected_storageType;
        BYTE fileData[header.size - 2];
}

struct DatabaseBlob
{
    HEADER header;

    DWORD ParentID;
    DWORD databaseType;
    WORD databaseName[32];
    DWORD numRecords;
    DWORD numSortOrder;
    DWORD unknown;
    DWORD lastModifiedLowTime;
    DWORD lastModifiedLowTime;
    DWORD dataBaseSize;
    DWORD unknown;
    SortOrder sortOrders[4];
} ;

struct DatabaseRecordBlob
{
    HEADER header;

    DWORD ParentID;
    BYTE data[header.Size-4]
};

struct DatabaseIndexBlob
{
    HEADER header;

    DWORD parentID;
    WORD unknown;
    WORD numRecords;
    DWORD delimiter1[2]; //FFFFFFFF
    DWORD recordID[numRecords];
    DWORD delimiter2; //FFFFFFFF
    DWORD unknown[numRecords*10];
    UQUAD timeStamps[numRecords];
};
```

4.2.4 Defining a Strategy - Second Loop

The black-box testing had given us a great deal of knowledge about the object store, but one important question remained unanswered: how does the object store keep track of all its files? As we had failed to locate some kind of file allocation table, we needed to re-think our strategy. Though we did not expect any direct answer, we tried to consult Microsoft. As expected, they would not answer our questions, but ended their response with *"FWIW it does not use file allocation tables."* It seemed like a waste of time to continue the search for such a table in memory dumps.

We needed to come up with a plan. After discussing our new situation, we defined the problem as follows:

Goal: *Find out how Windows Mobile keeps track of objects.*

Known: *Every object has an object identifier.*

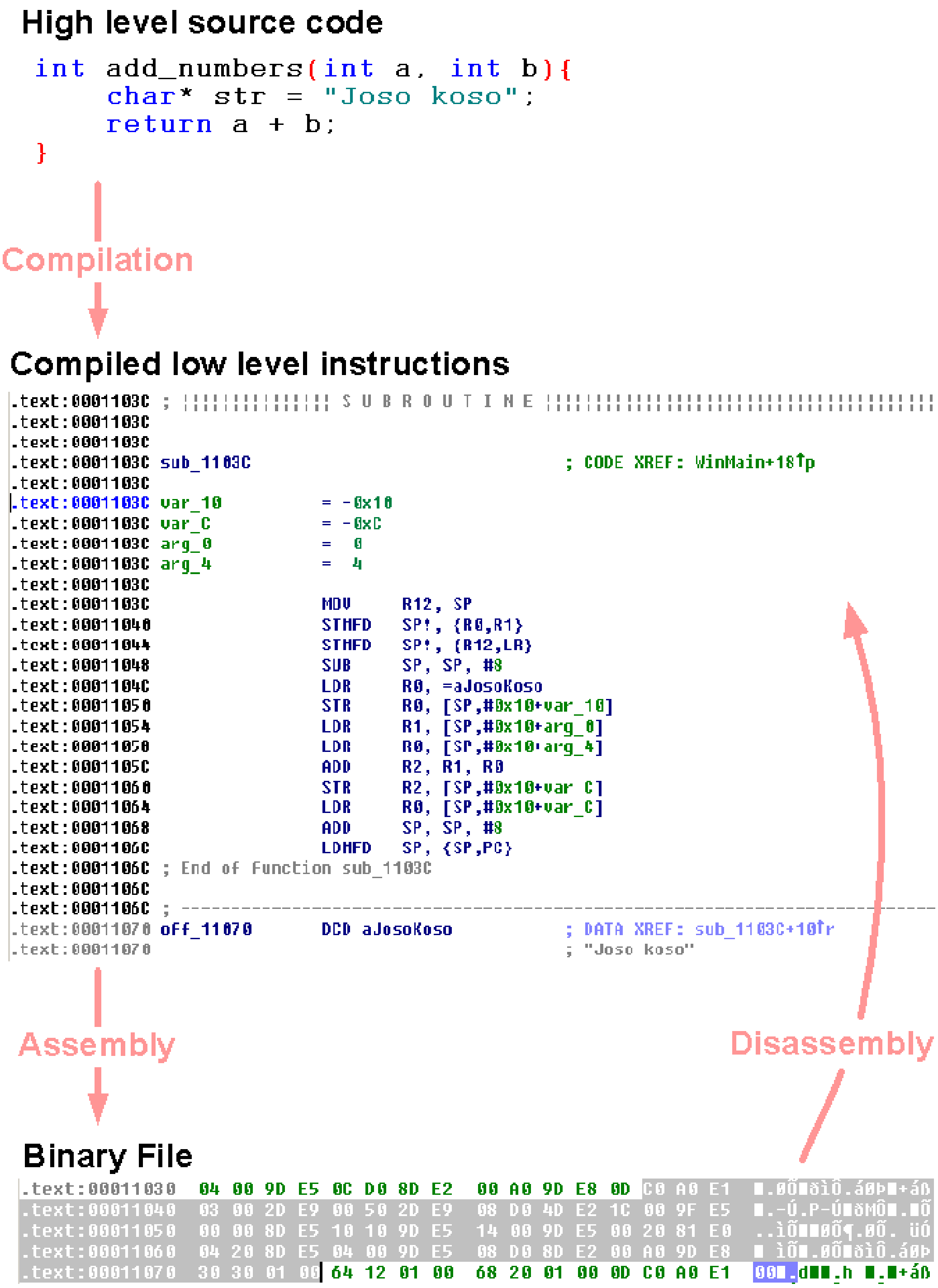
Strategy: *Find a procedure in the Windows CE API that can locate an object from an object identifier. Disassemble this procedure, to see how the object is found.*

4.2.5 About Disassembling

All programs start out as source code. Usually this is written by humans in a high level language like C or C++. High level language means that the functionality of the program can be written in a somewhat humanly readable form with much more abstract concepts than a computer can utilize. Computers support a given number of low level basic instructions that they can execute very quickly. Typical examples are instructions that fetch or store a value in memory and instructions that do simple arithmetic operations like addition and subtraction. Bridging that gap between a high level language source code and the instructions of a specific processor is the job of a compiler. It maps high level concepts to several low level instructions which can be performed by the processor. The source code writer usually never has to see this process. The low level instructions are then encoded in a binary format that is easy for computers to work with. This is called assembling a binary.

A disassembler does the opposite. Its primary task is to take a compiled and assembled binary file and produce a listing of the low level processor instructions and data contained in it. Figure 4.37 illustrates this.

In addition to just listing the low level processor instructions, a good disassembler utilizes several techniques to enhance the readability of this listing. Some of these techniques are:



```c
int add_numbers(int a, int b){
    char* str = "Joso koso";
    return a + b;
}
```

```
.text:0001103C ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:0001103C
.text:0001103C
.text:0001103C sub_1103C                               ; CODE XREF: WinMain+18↑p
.text:0001103C
.text:0001103C var_10          = -0x10
.text:0001103C var_C           = -0xC
.text:0001103C arg_0           =  0
.text:0001103C arg_4           =  4
.text:0001103C
.text:0001103C                 MOV     R12, SP
.text:00011040                 STMFD   SP!, {R0,R1}
.text:00011044                 STMFD   SP!, {R12,LR}
.text:00011048                 SUB     SP, SP, #8
.text:0001104C                 LDR     R0, =aJosoKoso
.text:00011050                 STR     R0, [SP,#0x10+var_10]
.text:00011054                 LDR     R1, [SP,#0x10+arg_0]
.text:00011058                 LDR     R0, [SP,#0x10+arg_4]
.text:0001105C                 ADD     R2, R1, R0
.text:00011060                 STR     R2, [SP,#0x10+var_C]
.text:00011064                 LDR     R0, [SP,#0x10+var_C]
.text:00011068                 ADD     SP, SP, #8
.text:0001106C                 LDMFD   SP, {SP,PC}
.text:0001106C ; End of Function sub_1103C
.text:0001106C
.text:0001106C ; -----------------------------------------------------------------------------
.text:00011070 off_11070       DCD aJosoKoso           ; DATA XREF: sub_1103C+10↑r
.text:00011070                                         ; "Joso koso"
```

```
.text:00011030  04 00 9D E5 0C D0 8D E2  00 A0 9D E8 0D C0 A0 E1  ■.0õ■ðìÔ.á0Þ■+áß
.text:00011040  03 00 2D E9 00 50 2D E9  08 D0 4D E2 1C 00 9F E5  ■.-Ú.P-Ú■ðMÔ■.■õ
.text:00011050  00 00 8D E5 10 10 9D E5  14 00 9D E5 00 20 81 E0  ..ìõ■■0õ¶.0õ. üó
.text:00011060  04 20 8D E5 04 00 9D E5  08 D0 8D E2 00 A0 9D E8  ■ ìõ■.0õ■ðìÔ.á0Þ
.text:00011070  30 30 01 00 64 12 01 00  68 20 01 00 0D C0 A0 E1  00■.d■■.h ■.■+áß
```

Figure 4.37: The compile, assemble, disassemble cycle.

Data/code differentiation Every disassembler must be able to discern the difference between data and code. This seems very basic, but can be very tricky as assemblers can generate an enormous amount of different code/data variations.

Procedure identification The ability to recognize procedures in the code, and when they are called. Might also be able to figure out the types of the procedure's parameters.

String recognition Recognition of string literals in the data. Strings are messages meant for humans and referencing strings can often help a lot in figuring out what some piece of code is actually doing.

System / library call identification A disassembler should be able to figure when the code is calling operating system functions and annotate these calls in a specific way. Better disassemblers can do the same thing for calls to well known library functions (like the C common runtime), and the best disassemblers can recognize any library function if the library is available at the time of the disassembly.

Data and code cross references A disassembler should annotate a recognized procedure with a cross reference to all the places in the code that calls the procedure. In the same manner it should annotate data addresses, or variables, with cross references to all the code that access them.

Data structure identification Good disassemblers can recognize known data structures from libraries and improve the readability of the code listing by incorporating this information.

Path analysis Analyze the code paths and branching instructions to enable enhanced code analysis coverage and enable the possibility of graph visualization.

There exists a multitude of disassemblers:

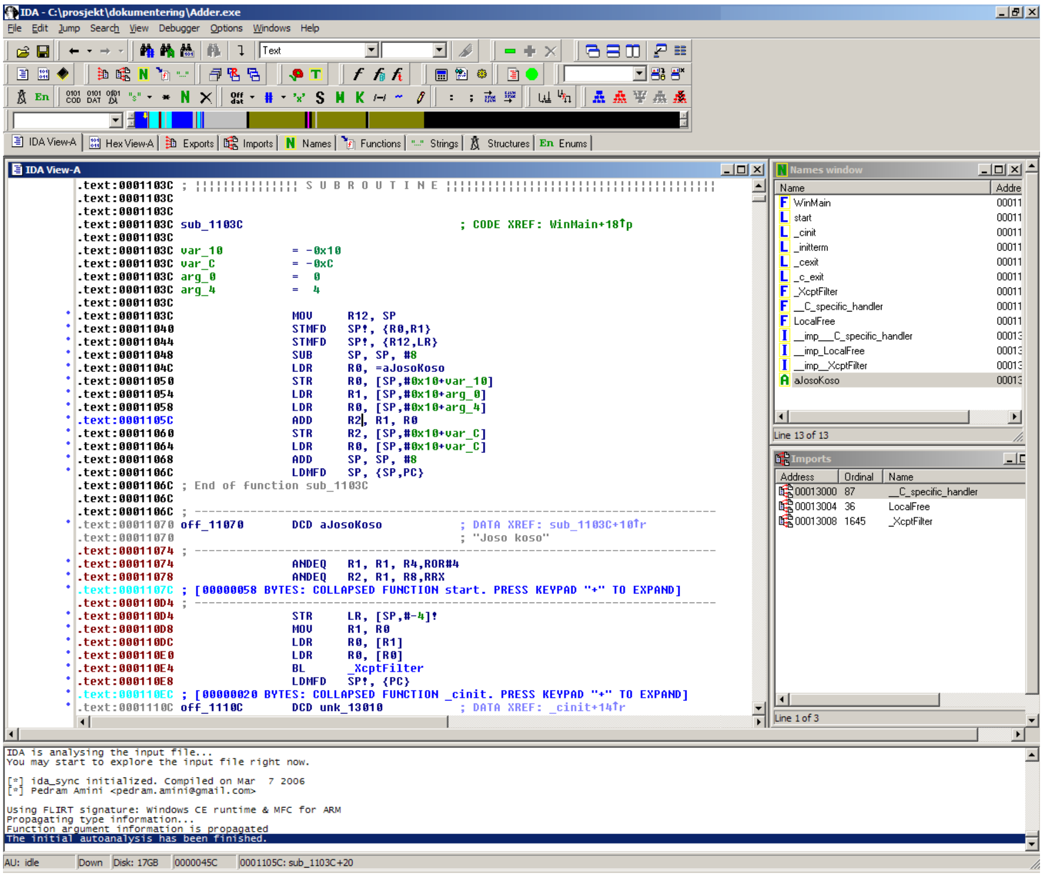
- Sourcer[22]
- BORG[23]

- BDASM[24]
- W32Dasm[25]
- PEDasm[26]
- Diss[27]
- Disassembler[28]
- ARMDis[29]

All of these disassembler were found to be inferior to IDA, described in the next section, because of one or more of the following reasons: does not support the ARM processor, lacking interactive nature(letting the user correct errors in the disassembly) or does not automatically identify API and system calls. IDA does this and much more, making it the best choice for any serious reverse engineering of binary executables.

Interactive DisAssembler (IDA)

IDA is a disassembler written by Ilfak Guilfanov and DataRescue[30]. It supports interactive disassembly of an impressive list of processor types and binary file formats, including the PE file format and ARM processor found in our "unknown" device. IDA has a efficient built-in support for automatically separating code from data, but at the same time the user can at anytime override all automatically made choices and have IDA update itself accordingly. This kind of interactivity is unique for IDA and is not found in other disassembling products. IDA does not produce just lists of low level instructions, but rather gives the user a complete environment for reverse engineering work. The interface can be seen in figure 4.38. Some of the functionality supported by IDA include:



```
.text:0001103C ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:0001103C
.text:0001103C
.text:0001103C sub_1103C                               ; CODE XREF: WinMain+18↑p
.text:0001103C
.text:0001103C var_10          = -0x10
.text:0001103C var_C           = -0xC
.text:0001103C arg_0           =  0
.text:0001103C arg_4           =  4
.text:0001103C
.text:0001103C                 MOV     R12, SP
.text:00011040                 STMFD   SP!, {R0,R1}
.text:00011044                 STMFD   SP!, {R12,LR}
.text:00011048                 SUB     SP, SP, #8
.text:0001104C                 LDR     R0, =aJosoKoso
.text:00011050                 STR     R0, [SP,#0x10+var_10]
.text:00011054                 LDR     R1, [SP,#0x10+arg_0]
.text:00011058                 LDR     R0, [SP,#0x10+arg_4]
.text:0001105C                 ADD     R2, R1, R0
.text:00011060                 STR     R2, [SP,#0x10+var_C]
.text:00011064                 LDR     R0, [SP,#0x10+var_C]
.text:00011068                 ADD     SP, SP, #8
.text:0001106C                 LDMFD   SP, {SP,PC}
.text:0001106C ; End of function sub_1103C
.text:0001106C
.text:0001106C ; ---------------------------------------------------------------------------
.text:00011070 off_11070       DCD aJosoKoso            ; DATA XREF: sub_1103C+10↑r
.text:00011070                                          ; "Joso koso"
.text:00011074 ; ---------------------------------------------------------------------------
.text:00011074                 ANDEQ   R1, R1, R4,ROR#4
.text:00011078                 ANDEQ   R2, R1, R8,RRX
.text:0001107C ; [00000058 BYTES: COLLAPSED FUNCTION start. PRESS KEYPAD "+" TO EXPAND]
.text:000110D4 ; ---------------------------------------------------------------------------
.text:000110D4                 STR     LR, [SP,#-4]!
.text:000110D8                 MOV     R1, R0
.text:000110DC                 LDR     R0, [R1]
.text:000110E0                 LDR     R0, [R0]
.text:000110E4                 BL      _XcptFilter
.text:000110E8                 LDMFD   SP!, {PC}
.text:000110EC ; [00000020 BYTES: COLLAPSED FUNCTION _cinit. PRESS KEYPAD "+" TO EXPAND]
.text:0001110C off_1110C       DCD unk_13010            ; DATA XREF: _cinit+14↑r
```

	Name	Addre
F	WinMain	00011
L	start	00011
L	_cinit	00011
L	_initterm	00011
L	_cexit	00011
L	_c_exit	00011
F	_XcptFilter	00011
F	__C_specific_handler	00011
F	LocalFree	00011
I	__imp___C_specific_handler	0001З
I	__imp_LocalFree	0001З
I	__imp__XcptFilter	0001З
A	aJosoKoso	0001З

Line 13 of 13

Address	Ordinal	Name
00013000	87	__C_specific_handler
00013004	36	LocalFree
00013008	1645	_XcptFilter

Line 1 of 3

```
IDA is analysing the input file...
You may start to explore the input file right now.

[*] ida_sync initialized. Compiled on Mar  7 2006
[*] Pedram Amini <pedram.amini@gmail.com>

Using FLIRT signature: Windows CE runtime & MFC for ARM
Propagating type information...
Function argument information is propagated
The initial autoanalysis has been finished.
```

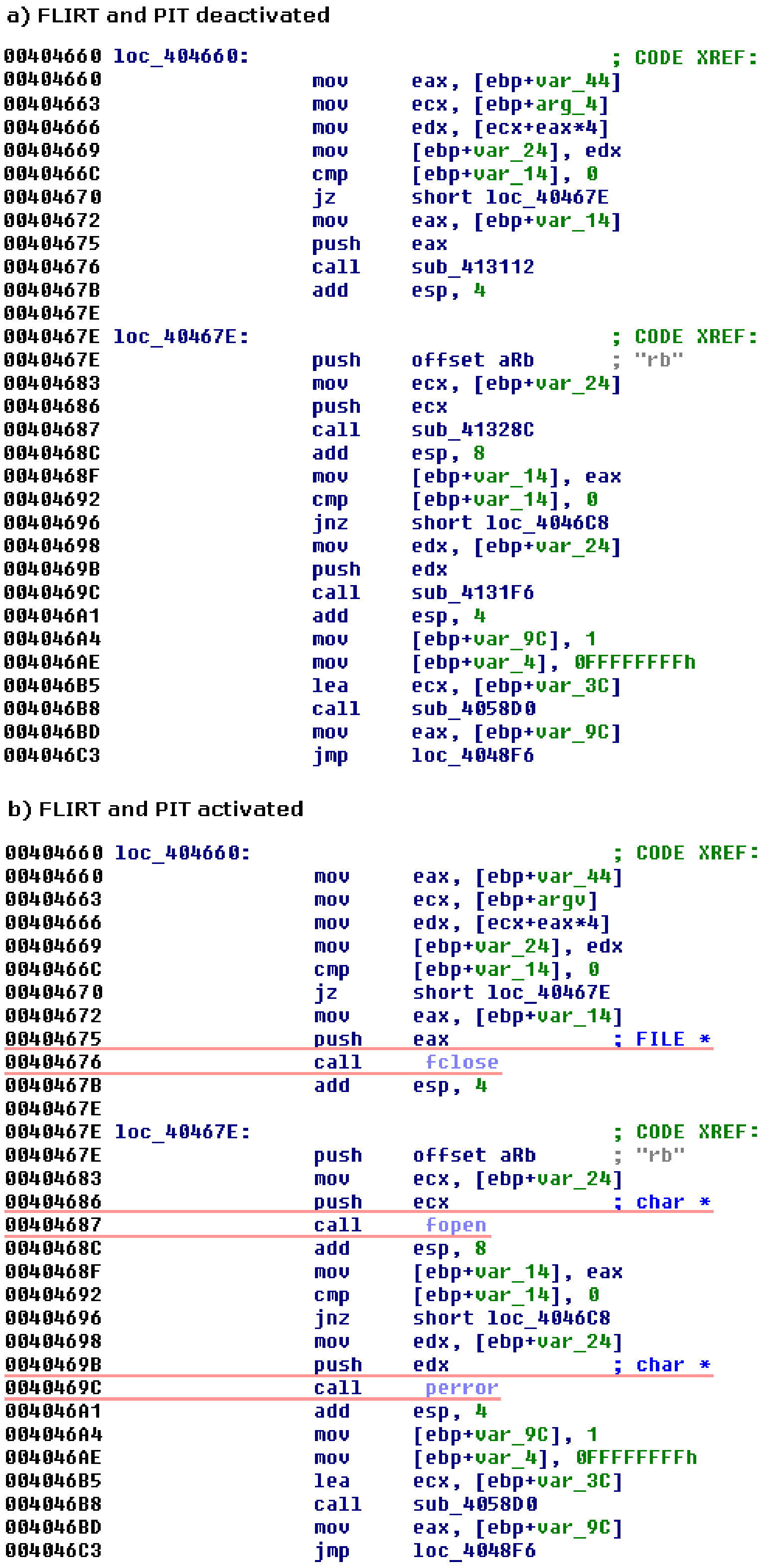
AU: idle | Down | Disk: 17GB | 0000045C | 0001105C: sub_1103C+20

Figure 4.38: IDAs user interaction interface.

FLIRT - Fast Library Identification and Recognition Technology. This feature identifies standard system calls and API function calls from several popular compilers. The result is that the user instantly can figure out that the call being made is to a known function without having to inspect the function at all. Figure 4.39 shows the difference between FLIRT being active and not. FLIRT can also be used on any custom library if the library file is available.

PIT - Parameter Identification and Tracking Tracks stack parameters as they are used. That means that if IDA sees that a certain memory address is given as an argument to a known system or library function it can automatically deduct the type of the variable stored there based on the function definition. This is propagated throughout the rest of the disassembly, helping with the overall readability. Figure 4.39 shows the difference PIT makes.

High level constructs. IDA lets one define and assign high level constructs like unions, structures and enumerations to any memory address, including stack based ones. This furthers the readability of disassembled code and narrows the gap between the low level machine instructions and the high level language they originated from. IDA automatically assigns well known structures to the parameters of subroutines found with its FLIRT technology. Figure 4.40 shows how the high level structure feature in IDA affects a disassembly.



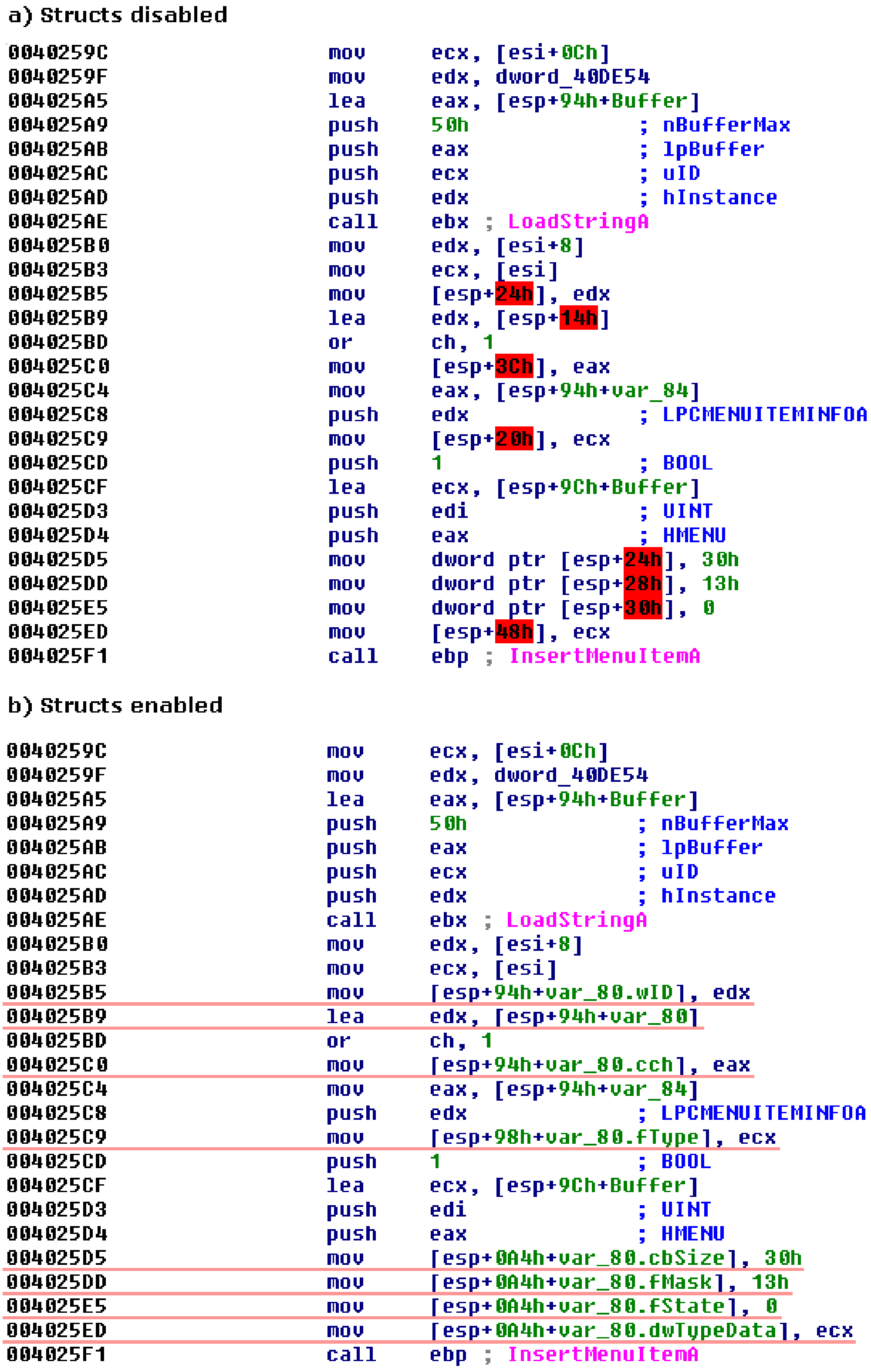
a) FLIRT and PIT deactivated

```
00404660 loc_404660:                               ; CODE XREF:
00404660                 mov     eax, [ebp+var_44]
00404663                 mov     ecx, [ebp+arg_4]
00404666                 mov     edx, [ecx+eax*4]
00404669                 mov     [ebp+var_24], edx
0040466C                 cmp     [ebp+var_14], 0
00404670                 jz      short loc_40467E
00404672                 mov     eax, [ebp+var_14]
00404675                 push    eax
00404676                 call    sub_413112
0040467B                 add     esp, 4
0040467E
0040467E loc_40467E:                               ; CODE XREF:
0040467E                 push    offset aRb        ; "rb"
00404683                 mov     ecx, [ebp+var_24]
00404686                 push    ecx
00404687                 call    sub_41328C
0040468C                 add     esp, 8
0040468F                 mov     [ebp+var_14], eax
00404692                 cmp     [ebp+var_14], 0
00404696                 jnz     short loc_4046C8
00404698                 mov     edx, [ebp+var_24]
0040469B                 push    edx
0040469C                 call    sub_4131F6
004046A1                 add     esp, 4
004046A4                 mov     [ebp+var_9C], 1
004046AE                 mov     [ebp+var_4], 0FFFFFFFFh
004046B5                 lea     ecx, [ebp+var_3C]
004046B8                 call    sub_4058D0
004046BD                 mov     eax, [ebp+var_9C]
004046C3                 jmp     loc_4048F6
```

b) FLIRT and PIT activated

```
00404660 loc_404660:                               ; CODE XREF:
00404660                 mov     eax, [ebp+var_44]
00404663                 mov     ecx, [ebp+argv]
00404666                 mov     edx, [ecx+eax*4]
00404669                 mov     [ebp+var_24], edx
0040466C                 cmp     [ebp+var_14], 0
00404670                 jz      short loc_40467E
00404672                 mov     eax, [ebp+var_14]
00404675                 push    eax               ; FILE *
00404676                 call     fclose
0040467B                 add     esp, 4
0040467E
0040467E loc_40467E:                               ; CODE XREF:
0040467E                 push    offset aRb        ; "rb"
00404683                 mov     ecx, [ebp+var_24]
00404686                 push    ecx               ; char *
00404687                 call     fopen
0040468C                 add     esp, 8
0040468F                 mov     [ebp+var_14], eax
00404692                 cmp     [ebp+var_14], 0
00404696                 jnz     short loc_4046C8
00404698                 mov     edx, [ebp+var_24]
0040469B                 push    edx               ; char *
0040469C                 call     perror
004046A1                 add     esp, 4
004046A4                 mov     [ebp+var_9C], 1
004046AE                 mov     [ebp+var_4], 0FFFFFFFFh
004046B5                 lea     ecx, [ebp+var_3C]
004046B8                 call    sub_4058D0
004046BD                 mov     eax, [ebp+var_9C]
004046C3                 jmp     loc_4048F6
```

Figure 4.39: Flirt and PIT- Fast Library Identification and Recognition Technology - Parameter Identification and Tracking.



a) Structs disabled

```
0040259C                 mov     ecx, [esi+0Ch]
0040259F                 mov     edx, dword_40DE54
004025A5                 lea     eax, [esp+94h+Buffer]
004025A9                 push    50h             ; nBufferMax
004025AB                 push    eax             ; lpBuffer
004025AC                 push    ecx             ; uID
004025AD                 push    edx             ; hInstance
004025AE                 call    ebx ; LoadStringA
004025B0                 mov     edx, [esi+8]
004025B3                 mov     ecx, [esi]
004025B5                 mov     [esp+24h], edx
004025B9                 lea     edx, [esp+14h]
004025BD                 or      ch, 1
004025C0                 mov     [esp+3Ch], eax
004025C4                 mov     eax, [esp+94h+var_84]
004025C8                 push    edx             ; LPCMENUITEMINFOA
004025C9                 mov     [esp+20h], ecx
004025CD                 push    1               ; BOOL
004025CF                 lea     ecx, [esp+9Ch+Buffer]
004025D3                 push    edi             ; UINT
004025D4                 push    eax             ; HMENU
004025D5                 mov     dword ptr [esp+24h], 30h
004025DD                 mov     dword ptr [esp+28h], 13h
004025E5                 mov     dword ptr [esp+30h], 0
004025ED                 mov     [esp+48h], ecx
004025F1                 call    ebp ; InsertMenuItemA
```

b) Structs enabled

```
0040259C                 mov     ecx, [esi+0Ch]
0040259F                 mov     edx, dword_40DE54
004025A5                 lea     eax, [esp+94h+Buffer]
004025A9                 push    50h             ; nBufferMax
004025AB                 push    eax             ; lpBuffer
004025AC                 push    ecx             ; uID
004025AD                 push    edx             ; hInstance
004025AE                 call    ebx ; LoadStringA
004025B0                 mov     edx, [esi+8]
004025B3                 mov     ecx, [esi]
004025B5                 mov     [esp+94h+var_80.wID], edx
004025B9                 lea     edx, [esp+94h+var_80]
004025BD                 or      ch, 1
004025C0                 mov     [esp+94h+var_80.cch], eax
004025C4                 mov     eax, [esp+94h+var_84]
004025C8                 push    edx             ; LPCMENUITEMINFOA
004025C9                 mov     [esp+98h+var_80.fType], ecx
004025CD                 push    1               ; BOOL
004025CF                 lea     ecx, [esp+9Ch+Buffer]
004025D3                 push    edi             ; UINT
004025D4                 push    eax             ; HMENU
004025D5                 mov     [esp+0A4h+var_80.cbSize], 30h
004025DD                 mov     [esp+0A4h+var_80.fMask], 13h
004025E5                 mov     [esp+0A4h+var_80.fState], 0
004025ED                 mov     [esp+0A4h+var_80.dwTypeData], ecx
004025F1                 call    ebp ; InsertMenuItemA
```

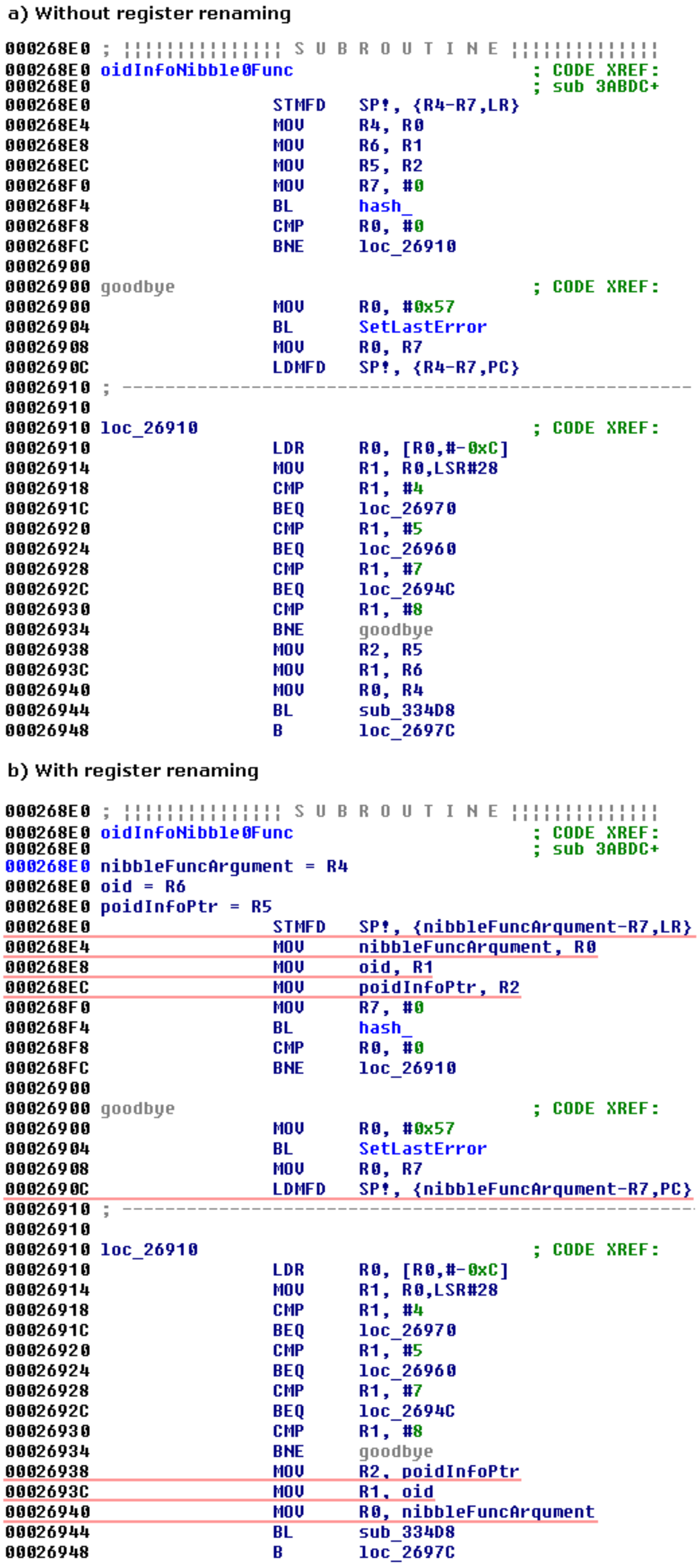
Figure 4.40: High level constructs.

Interactive Register renaming. A listing of low level machine code is greatly enhanced by the ability to rename stack and heap based variables to more meaningful names. RISC processors, like ARM, have a large number of general data registers. In order to work efficiently the processors try to keep as much data as possible in these registers. This means that it loads most of the ”variables” it is currently working here. For human readability this is bad, because a helpful variable name like ”counter” suddenly just becomes register R4. This problems only increases as more general registers become available. The user has to remember what is in each register at all times. IDA has come up with a technique for countering this problem. They allow you to rename a register on the fly, just like a normal variable. The user selects simply selects a register they want to rename, a start and end address the renaming should be for and IDA does the rest for you. Figure 4.41 show an example of this useful feature.

IDC One of the most powerful features in IDA is the built-in C-like script language that user can write scripts to enhance parts of the reverse engineering process. Examples of uses are scripts to unscramble packed and compressed code, or scripts that handles file formats not supported out of the box from DataRescue. Everything from the user interface to the disassembly process can be customized. Much of the built-in functionality is actually implemented in IDC.

Plug-in architecture. For the tasks where IDC is not powerful enough IDA has support for precompiled plug-ins. These can be written in C or Python and are normally used to support a completely new architectures or processors. Recent editions of IDA also have a built-in X86/ARM debugger which is implemented as a plug-in.

Graphing. IDA supports visualization of code control flow. This gives the user a way to quickly understand the overall flow of the program, something which is easy to lose track of in a disassembly. Figure 4.42 shows an example of IDAs graph support.



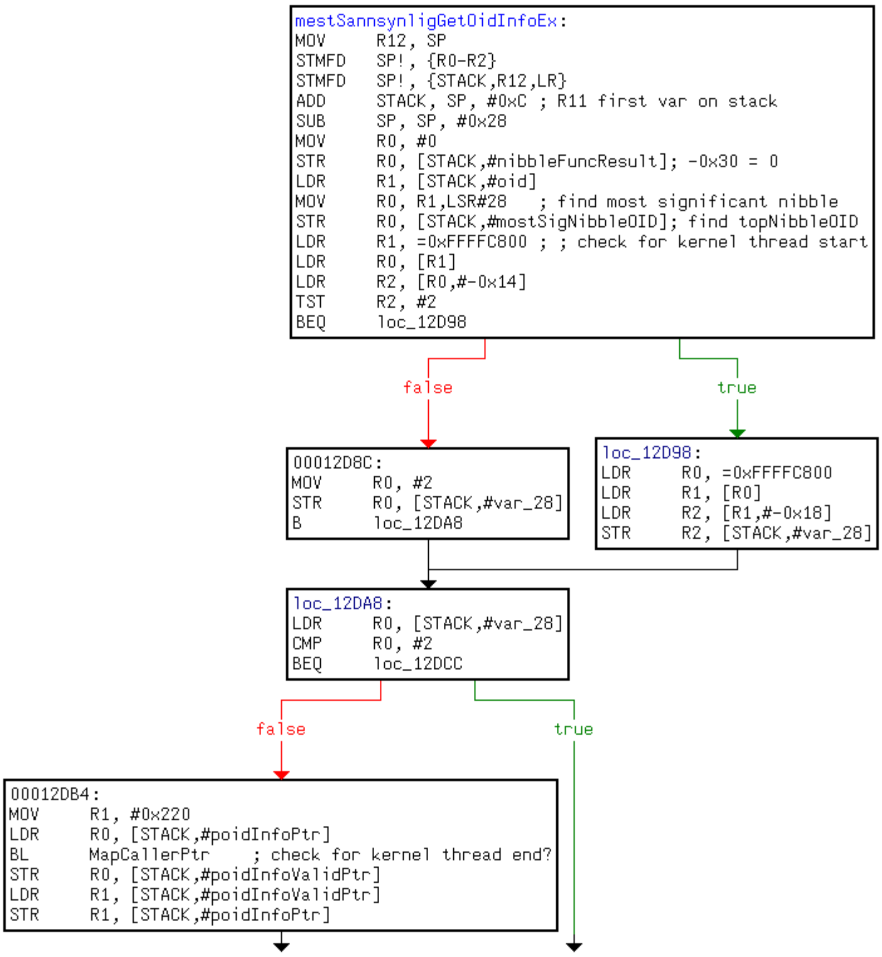
a) Without register renaming

```
000268E0 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦
000268E0 oidInfoNibble0Func                      ; CODE XREF:
000268E0                                         ; sub_3ABDC+
000268E0                 STMFD   SP!, {R4-R7,LR}
000268E4                 MOV     R4, R0
000268E8                 MOV     R6, R1
000268EC                 MOV     R5, R2
000268F0                 MOV     R7, #0
000268F4                 BL      hash_
000268F8                 CMP     R0, #0
000268FC                 BNE     loc_26910
00026900
00026900 goodbye                                 ; CODE XREF:
00026900                 MOV     R0, #0x57
00026904                 BL      SetLastError
00026908                 MOV     R0, R7
0002690C                 LDMFD   SP!, {R4-R7,PC}
00026910 ; ------------------------------------------------------
00026910
00026910 loc_26910                               ; CODE XREF:
00026910                 LDR     R0, [R0,#-0xC]
00026914                 MOV     R1, R0,LSR#28
00026918                 CMP     R1, #4
0002691C                 BEQ     loc_26970
00026920                 CMP     R1, #5
00026924                 BEQ     loc_26960
00026928                 CMP     R1, #7
0002692C                 BEQ     loc_2694C
00026930                 CMP     R1, #8
00026934                 BNE     goodbye
00026938                 MOV     R2, R5
0002693C                 MOV     R1, R6
00026940                 MOV     R0, R4
00026944                 BL      sub_334D8
00026948                 B       loc_2697C
```

b) With register renaming

```
000268E0 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦
000268E0 oidInfoNibble0Func                      ; CODE XREF:
000268E0                                         ; sub_3ABDC+
000268E0 nibbleFuncArgument = R4
000268E0 oid = R6
000268E0 poidInfoPtr = R5
000268E0                 STMFD   SP!, {nibbleFuncArgument-R7,LR}
000268E4                 MOV     nibbleFuncArgument, R0
000268E8                 MOV     oid, R1
000268EC                 MOV     poidInfoPtr, R2
000268F0                 MOV     R7, #0
000268F4                 BL      hash_
000268F8                 CMP     R0, #0
000268FC                 BNE     loc_26910
00026900
00026900 goodbye                                 ; CODE XREF:
00026900                 MOV     R0, #0x57
00026904                 BL      SetLastError
00026908                 MOV     R0, R7
0002690C                 LDMFD   SP!, {nibbleFuncArgument-R7,PC}
00026910 ; ------------------------------------------------------
00026910
00026910 loc_26910                               ; CODE XREF:
00026910                 LDR     R0, [R0,#-0xC]
00026914                 MOV     R1, R0,LSR#28
00026918                 CMP     R1, #4
0002691C                 BEQ     loc_26970
00026920                 CMP     R1, #5
00026924                 BEQ     loc_26960
00026928                 CMP     R1, #7
0002692C                 BEQ     loc_2694C
00026930                 CMP     R1, #8
00026934                 BNE     goodbye
00026938                 MOV     R2, poidInfoPtr
0002693C                 MOV     R1, oid
00026940                 MOV     R0, nibbleFuncArgument
00026944                 BL      sub_334D8
00026948                 B       loc_2697C
```

Figure 4.41: Interactive register renaming.



```
mestSannsynligGetOidInfoEx:
MOV     R12, SP
STMFD   SP!, {R0-R2}
STMFD   SP!, {STACK,R12,LR}
ADD     STACK, SP, #0xC  ; R11 first var on stack
SUB     SP, SP, #0x28
MOV     R0, #0
STR     R0, [STACK,#nibbleFuncResult]; -0x30 = 0
LDR     R1, [STACK,#oid]
MOV     R0, R1,LSR#28    ; find most significant nibble
STR     R0, [STACK,#mostSigNibbleOID]; find topNibbleOID
LDR     R1, =0xFFFFC800 ; ; check for kernel thread start
LDR     R0, [R1]
LDR     R2, [R0,#-0x14]
TST     R2, #2
BEQ     loc_12D98
```

```
00012D8C:
MOV     R0, #2
STR     R0, [STACK,#var_28]
B       loc_12DA8
```

```
loc_12D98:
LDR     R0, =0xFFFFC800
LDR     R1, [R0]
LDR     R2, [R1,#-0x18]
STR     R2, [STACK,#var_28]
```

```
loc_12DA8:
LDR     R0, [STACK,#var_28]
CMP     R0, #2
BEQ     loc_12DCC
```

```
00012DB4:
MOV     R1, #0x220
LDR     R0, [STACK,#poidInfoPtr]
BL      MapCallerPtr     ; check for kernel thread end?
STR     R0, [STACK,#poidInfoValidPtr]
LDR     R1, [STACK,#poidInfoValidPtr]
STR     R1, [STACK,#poidInfoPtr]
```

Figure 4.42: Code control flow graph.

4.2.6 Testing

The testing phase of this strategy was somewhat different from the black box strategy. The idea this time was to find a suitable target for the disassembling, meaning a procedure that was able to locate objects from object identifiers, and follow this procedure as far as we were able to.

4.2.6.1 Test 7

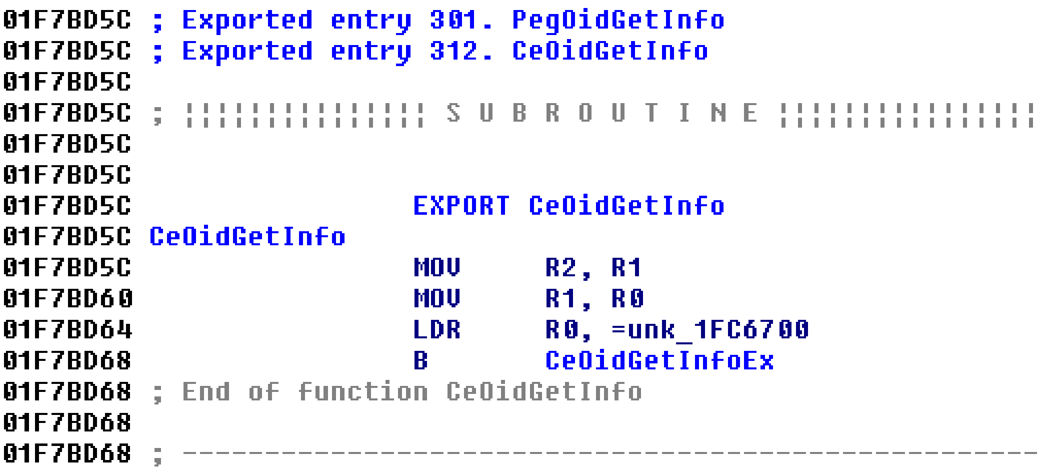
Create test

From the Windows CE API we had the procedure *CeOidGetInfo* (figure 4.11):
This procedure fulfilled our criteria, and was picked to be used in the disassembling.

Extract info/Adjust PSM

The next obstacle was then to locate the actual implementation of this procedure. According to "Windows CE 3.0 - Application Programming" [21], most Windows CE APIs are exported by *coredll.dll*. This file is part of Windows Mobile's system files, and we were not able to reach this file on the phone for a simple copy/paste from the phone to the computer. Instead we had to inspect the flash partition were the operating system files are located, knowing that we had the tools to make a complete image of this area. To get hold of the file, we needed to re-construct the file from this flash image. Writing a tool to perform this task was now our primary task. Luckily, we were able to find a tool which did exactly what we needed. The DumpRom tool by Willem Jan Hengeveld[7] takes a flash image as input, and tries to re-generate all the files seen in the image. The tool did what it claimed, and we were able to get *coredll.dll* from the flash image with less effort than we had expected. It was time to start disassembling *coredll.dll* using IDA. We will base the disassembling on some of the ideas from Sychronized Refinement described in section 3.2.3. We will analyse the code in parallel with the process of building up a description of what the code accomplishes by focusing on heavy commenting. We will also keep dynamic expectation lists that will guide our analysis towards an adequate understanding.

IDA presented a menu with a list of procedures, from where we could easily locate *CeOidGetInfo*. Figure 4.43 shows the code found. As we can

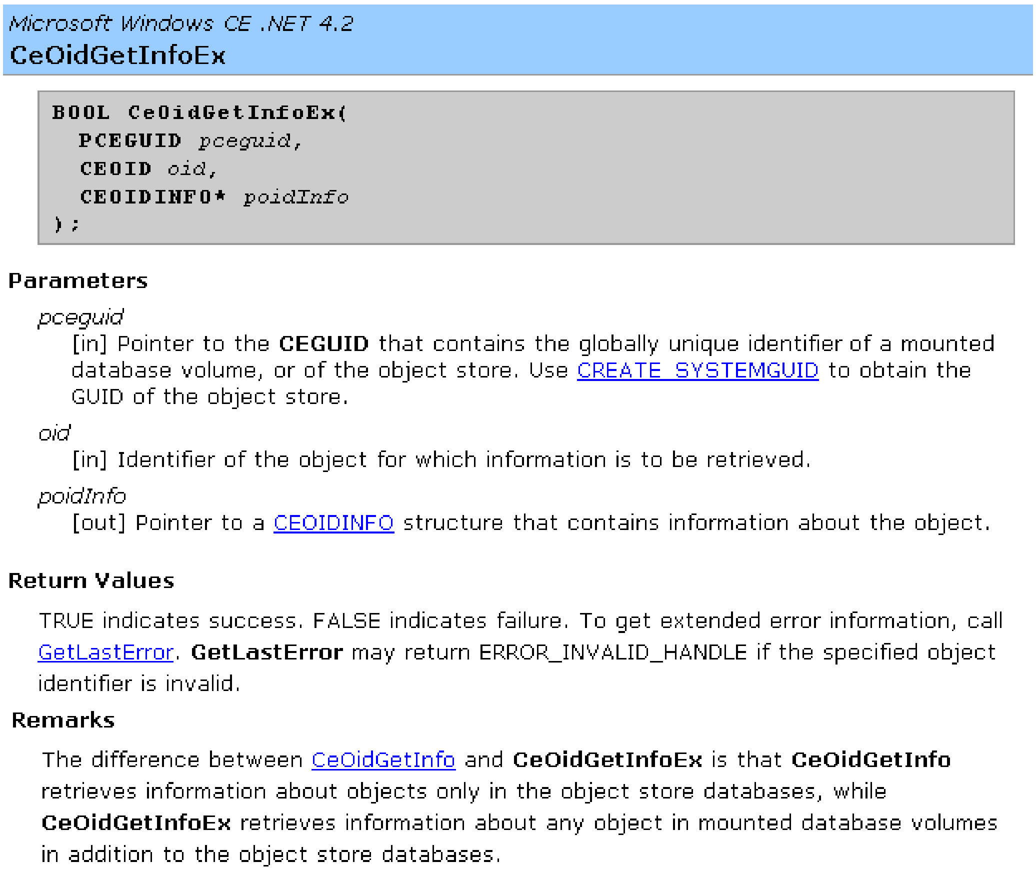


```
01F7BD5C ; Exported entry 301. PegOidGetInfo
01F7BD5C ; Exported entry 312. CeOidGetInfo
01F7BD5C
01F7BD5C ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
01F7BD5C
01F7BD5C
01F7BD5C                 EXPORT CeOidGetInfo
01F7BD5C CeOidGetInfo
01F7BD5C                 MOV     R2, R1
01F7BD60                 MOV     R1, R0
01F7BD64                 LDR     R0, =unk_1FC6700
01F7BD68                 B       CeOidGetInfoEx
01F7BD68 ; End of function CeOidGetInfo
01F7BD68
01F7BD68 ; ---------------------------------------------------
```

Figure 4.43: Test 7 - CeOidGetInfo in *coredll.dll*.

see, this code segment is called for both *PegOidGetInfo* and *CeOidGetInfo*, where *PegOidGetInfo* is used for applications on Windows CE vesions 1.0 and 1.01. Remembering the definition of *CeOidGetInfo*, we know that it takes two parameters, a *CEOID* and a pointer to a *CEOIDINFO* structure (which will be explored in detail below). Upon a procedure call, these parameters will be placed in registers *R0* and *R1* (R0, R1, R2, and R3 are used for argument passing), from where they can be reached within the procedure. What happens in the code segment above is that the second parameter is instead moved to *R2*, the first parameter is moved to *R1*, and some value is loaded into *R0* (the name unk_1FC6700 is inserted by IDA. A branch is then made to *CeOidGetInfoEx*. Looking at *CeOidGetInfoEx* in the Windows CE API, we found the definition in figure(4.44).

As we can see, *CeOidGetInfoEx* has been given an additional first parameter, a *PCEGUID*. The *PCEGUID* is a pointer to a globally unique identifier of the database volume, or of the object store. From this, we know what is done in the code segment above (figure 4.43). The two parameters of *CeOidGetInfo* has been shifted one position to fit their positions in *CeOidGetInfoEx*. This is what happens with the two *MOV* operations. The value placed in *R0* is the *PCEGUID* used in *CeOidGetInfoEx*. As this was a call to *CeOidGetInfo*, the *PCEGUID* will always point to the identifier of the object store. That is what allows this value to be placed into the code itself. At the end of this code segment, the original call to *CeOidGetInfo* has been adjusted to fit *CeOidGetInfoEx*, which now means that the same code can



Microsoft Windows CE .NET 4.2
CeOidGetInfoEx

```
BOOL CeOidGetInfoEx(
  PCEGUID pceguid,
  CEOID oid,
  CEOIDINFO* poidInfo
);
```

Parameters

pceguid
[in] Pointer to the **CEGUID** that contains the globally unique identifier of a mounted database volume, or of the object store. Use CREATE_SYSTEMGUID to obtain the GUID of the object store.

oid
[in] Identifier of the object for which information is to be retrieved.

poidInfo
[out] Pointer to a CEOIDINFO structure that contains information about the object.

Return Values

TRUE indicates success. FALSE indicates failure. To get extended error information, call GetLastError. **GetLastError** may return ERROR_INVALID_HANDLE if the specified object identifier is invalid.

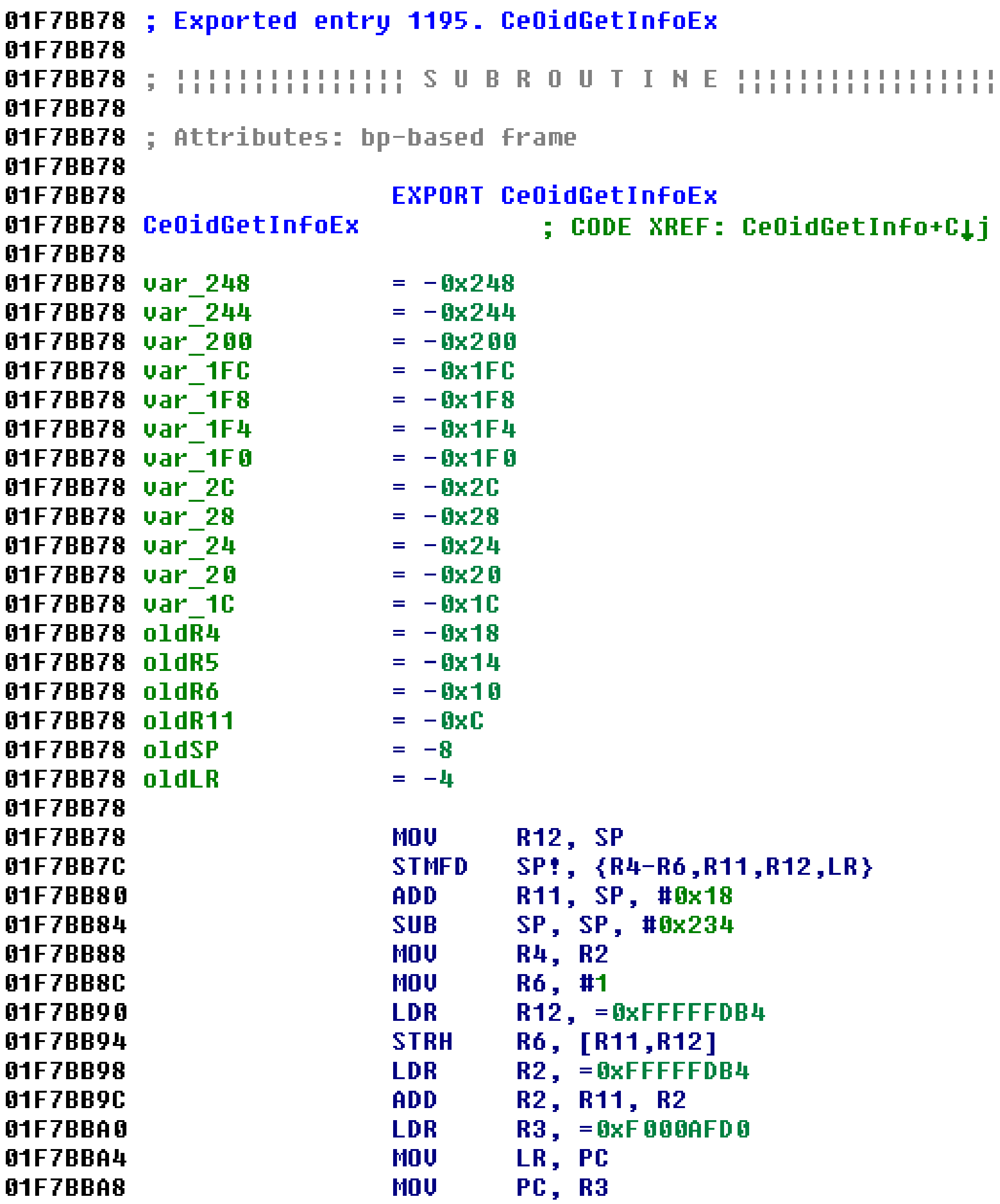
Remarks

The difference between CeOidGetInfo and **CeOidGetInfoEx** is that **CeOidGetInfo** retrieves information about objects only in the object store databases, while **CeOidGetInfoEx** retrieves information about any object in mounted database volumes in addition to the object store databases.

Figure 4.44: Windows CE API - CeOidGetInfoEx

be used for both. A branch is therefore made to *CeOidGetInfoEx*.

As we located *CeOidGetInfoEx*, we found a larger code segment. The beginning of this procedure is shown in figure 4.45. The first two instructions (*MOV*, *STMFD*) together form the entry stub, which saves register values, stack pointer and return address, and moves the stack pointer accordingly. Next, *"ADD R11, SP, #0x18"* assigns *R11* with the previous stack pointer value, the value seen before the entry stub. This is done so that *R11* can be used as a base in later references to the stack. To see why an *ADD* instruction is used instead of a *SUB* instruction, we need to remember that the stack grows downwards, towards lower memory addresses. Thus, the entry stub has assigned the stack pointer with a smaller value. *"SUB SP, SP, #0x234"* assigns 564 bytes (234 hexadecimal) to the stack. This makes room

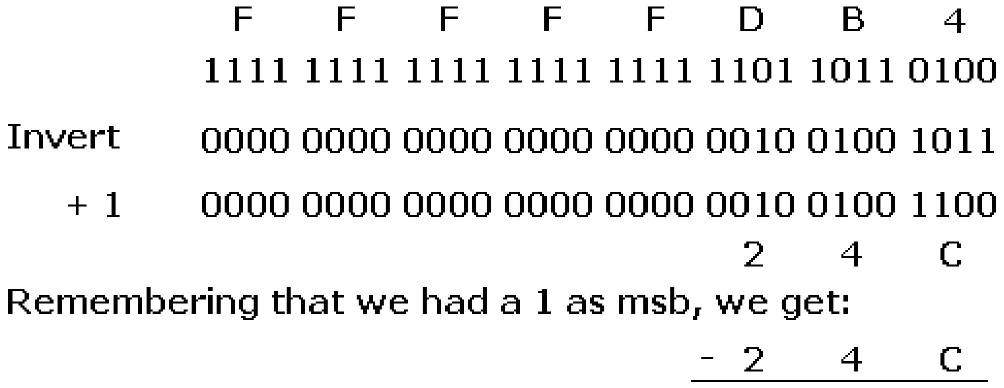


```
01F7BB78 ; Exported entry 1195. CeOidGetInfoEx
01F7BB78
01F7BB78 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
01F7BB78
01F7BB78 ; Attributes: bp-based frame
01F7BB78
01F7BB78                 EXPORT CeOidGetInfoEx
01F7BB78 CeOidGetInfoEx            ; CODE XREF: CeOidGetInfo+C↓j
01F7BB78
01F7BB78 var_248         = -0x248
01F7BB78 var_244         = -0x244
01F7BB78 var_200         = -0x200
01F7BB78 var_1FC         = -0x1FC
01F7BB78 var_1F8         = -0x1F8
01F7BB78 var_1F4         = -0x1F4
01F7BB78 var_1F0         = -0x1F0
01F7BB78 var_2C          = -0x2C
01F7BB78 var_28          = -0x28
01F7BB78 var_24          = -0x24
01F7BB78 var_20          = -0x20
01F7BB78 var_1C          = -0x1C
01F7BB78 oldR4           = -0x18
01F7BB78 oldR5           = -0x14
01F7BB78 oldR6           = -0x10
01F7BB78 oldR11          = -0xC
01F7BB78 oldSP           = -8
01F7BB78 oldLR           = -4
01F7BB78
01F7BB78                 MOV     R12, SP
01F7BB7C                 STMFD   SP!, {R4-R6,R11,R12,LR}
01F7BB80                 ADD     R11, SP, #0x18
01F7BB84                 SUB     SP, SP, #0x234
01F7BB88                 MOV     R4, R2
01F7BB8C                 MOV     R6, #1
01F7BB90                 LDR     R12, =0xFFFFFDB4
01F7BB94                 STRH    R6, [R11,R12]
01F7BB98                 LDR     R2, =0xFFFFFDB4
01F7BB9C                 ADD     R2, R11, R2
01F7BBA0                 LDR     R3, =0xF000AFD0
01F7BBA4                 MOV     LR, PC
01F7BBA8                 MOV     PC, R3
```

Figure 4.45: Test 7 - The start of CeOidGetInfoEx in *coredll.dll*.

for a *CEOIDINFOEX* structure that is to be filled by the procedure (figure 4.19). This is an extended version of the *CEOIDINFO* structure, containing one additional value (*WORD wVersion*). This structure is used in a third version of the procedure, called *CeOidGetInfoEx2*. When *CeOidGetInfo* or

CeOidGetInfoEx are used, the values in the *CEOIDINFOEX* structure, except the new version field, will get copied into the *CEOIDINFO* structure before returning from the procedure. *"MOV R4, R2"* stores the value of the *CEOIDINFO** argument into *R4*, so that *R2* is free to be used for argument passing in a new procedure call. *"MOV R6, #1"* places the value 1 into *R6*. *"LDR R12, =0xFFFFFDB4"* then assigns the value 0xFFFFFDB4 to *R12*. Looking at the following instruction, *"STRH R6, [R11,R12]"*, we see that this value is used as offset from the base stored in *R11*, to store the value 1 found in *R6* on the stack. This offset value is a 2's Complement value. We can decode 0xFFFFFDB4 into binary as follows in figure 4.46.



```
          F    F    F    F    F    D    B    4
        1111 1111 1111 1111 1111 1101 1011 0100

Invert  0000 0000 0000 0000 0000 0010 0100 1011

   + 1  0000 0000 0000 0000 0000 0010 0100 1100
                                   2    4    C
Remembering that we had a 1 as msb, we get:
                                 - 2    4    C
```

Figure 4.46: Test 7 - Converting from 2's Complement to binary.

0x24C equals 0x18 + 0x234. As seen from the base value in *R11*, -0x24C is the position of the last assigned location on the stack. This is where the code places the value 1. The *STRH* instruction stores a half word, which equals 2 bytes on a ARM processor. If we go back to the definition of the *CEOIDINFOEX* structure in the Windows CE API (figure 4.19), we see that the first field in the structure is the *WORD wVersion*. The size of this word is 2 bytes, and according to the comment in the API, this field always needs to be set to 1. This is exactly what has happened in the assembly so far. Figure 4.47 shows the changes seen on the stack so far in *CeOidGetInfoEx*.

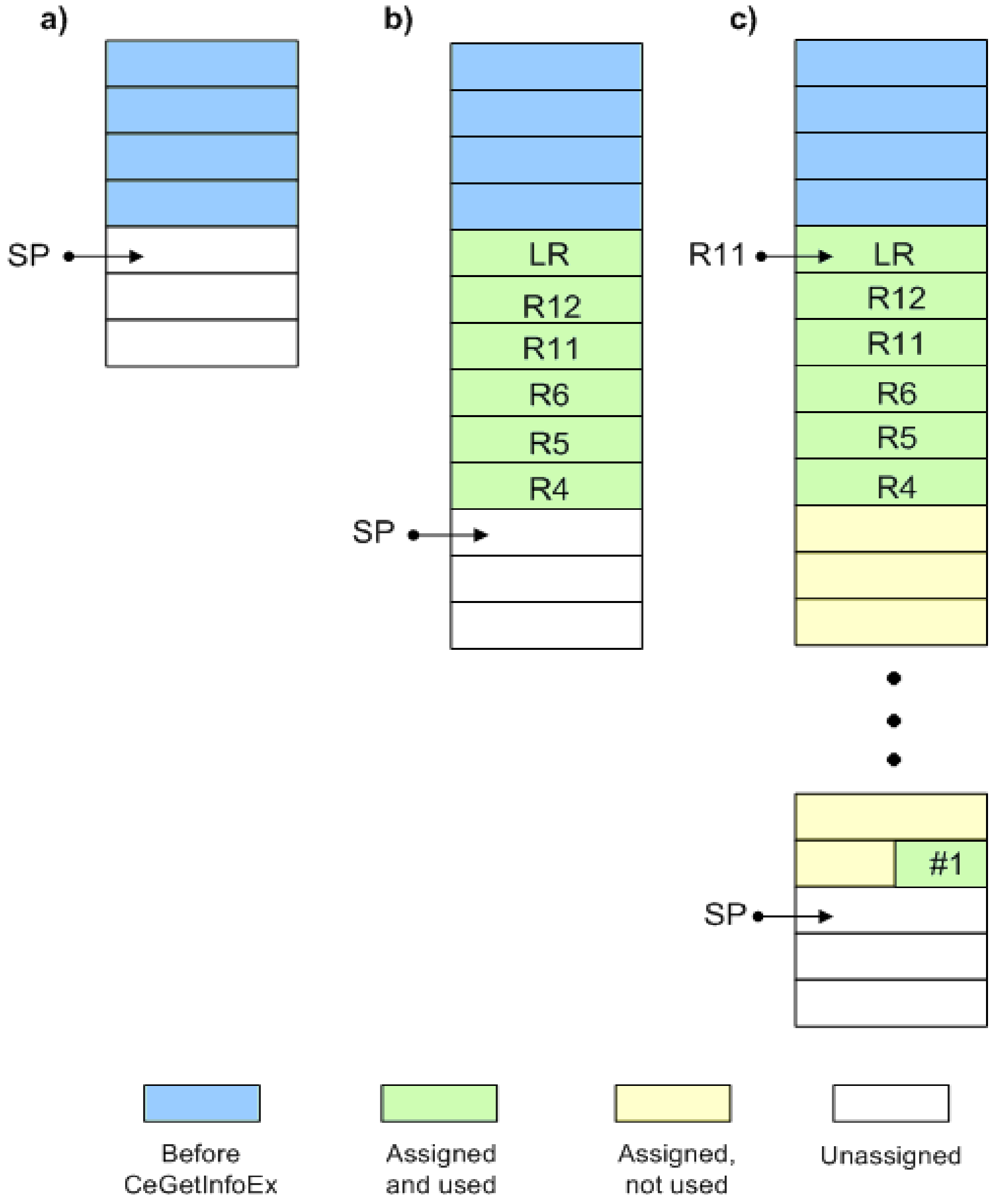


Figure 4.47: Test 7 - a) Before calling CeOidGetInfoEx. b) After executing the first two instructions, *LR*, *R12*, *R11*, *R6*, *R5*, and *R4* have been pushed on the stack, and the stack pointer has been moved accordingly. c) *R11* has been set as base, the stack pointer has assigned 0x234 more bytes to the stack, and the value 1 has been inserted at offset -0x24C. The STRH instruction only stores 2 bytes.

The next two instructions (*LDR* and *ADD*) calculates this memory location again, and places it in register *R2* so that it can be used as an argument. Then, if we look at what happens in the last three instructions in figure 4.45, a new unknown value(*0xF000AFD0*) is introduced, and the program counter ends up getting this value. Thus, we tried to search the phones memory to see what could be located at this address, but had no luck. We also searched different sources at the Internet, looking for any references to this memory address, but did not get any answers. For some reason the code jumped to this location, but it seemed impossible to find the code that was executed there. The memory location did not even seem to be in use.

Then we discovered an article at the ”Windows CE Base Team Blog” called ”Inside Windows CE API calls” [31] which contained valuable information. This included the following explanation of Windows CE API calls:

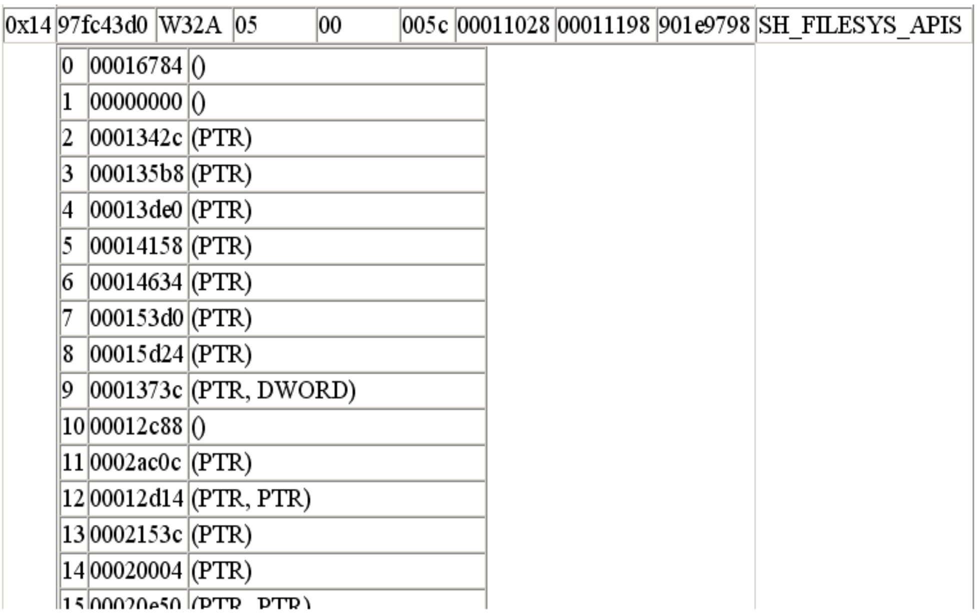
- Windows CE APIs are implemented by a set of server processes (*nk.exe*, *filesys.exe*, *gwes.exe*, *device.exe*, *services.exe*).
- When an application calls an API in one of these servers, the app thread jumps into the server process.
- Most Windows CE APIs are exported by *coredll.dll*, which all Windows CE applications link against, but *coredll.dll* is just a small wrapper, re-directing the call to the server process containing the actual implementation.
- The re-direction of the call is done by making a jump to an invalid address. This causes an exception. The invalid address value is not an arbitrary invalid address, but contains information about what server process the call should be re-directed to. When the exception is caught, the value is recognized and decoded.

From the XDA-developers web site [32] we found more information, including the formula to decode the invalid address value:

- A call is made to an invalid address in the range 0xf0000000 - 0xf0010000.
- The system call number is determined by 0xf0010000-(256*apiset+apinr)*4.
- The api set handles are defined in PUBLIC/COMMON/SDK/INC/K-FUNCS.H and PUBLIC/COMMON/OAK/INC/psyscall.h.

- The aipnrs are defined in several files, for example SH_GDI calls are defined in PUBLIC/COMMON/OAK/INC/mwinuser.h.

Decoding *0xF000AFD0* according to this rule, we get *API set = 20* and *API nr = 12*. The same article from XDA-developers also contained a list with all system calls found on a Pocket PC Phone Edition 2003, another version of Windows CE. Instead of trying to make such a list for our device, we decided to first see what we could find by using the list found. Despite the fact that our device is running the Second Edition, the OS was the same. Scrolling down the list to set number 20(0x14), we found that it is called SH_FILESYS_APIS (figure 4.48). Since we were looking at a procedure to get information about files, this seemed reasonable.



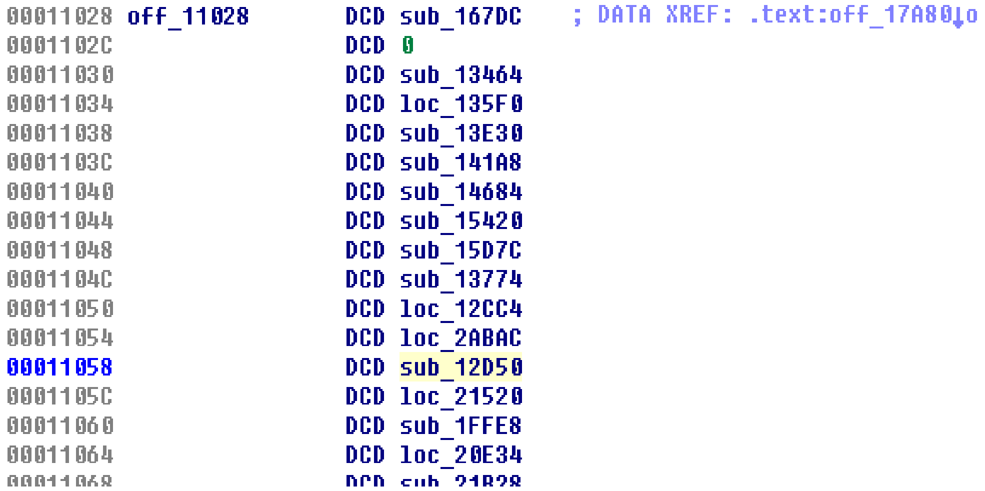
0x14	97fc43d0	W32A	05	00	005c	00011028	00011198	901e9798	SH_FILESYS_APIS

0	00016784	()
1	00000000	()
2	0001342c	(PTR)
3	000135b8	(PTR)
4	00013de0	(PTR)
5	00014158	(PTR)
6	00014634	(PTR)
7	000153d0	(PTR)
8	00015d24	(PTR)
9	0001373c	(PTR, DWORD)
10	00012c88	()
11	0002ac0c	(PTR)
12	00012d14	(PTR, PTR)
13	0002153c	(PTR)
14	00020004	(PTR)
15	00020e50	(PTR, PTR)

Figure 4.48: Test 7 - List of API sets.

It also backed up our suspicion of finding the implementation in the server process *filesys.exe*. As a consequence, we decided to start disassembling *filesys.exe*, which was also available from the flash image.

At the top of the file created by IDA we saw the table shown in figure 4.49.

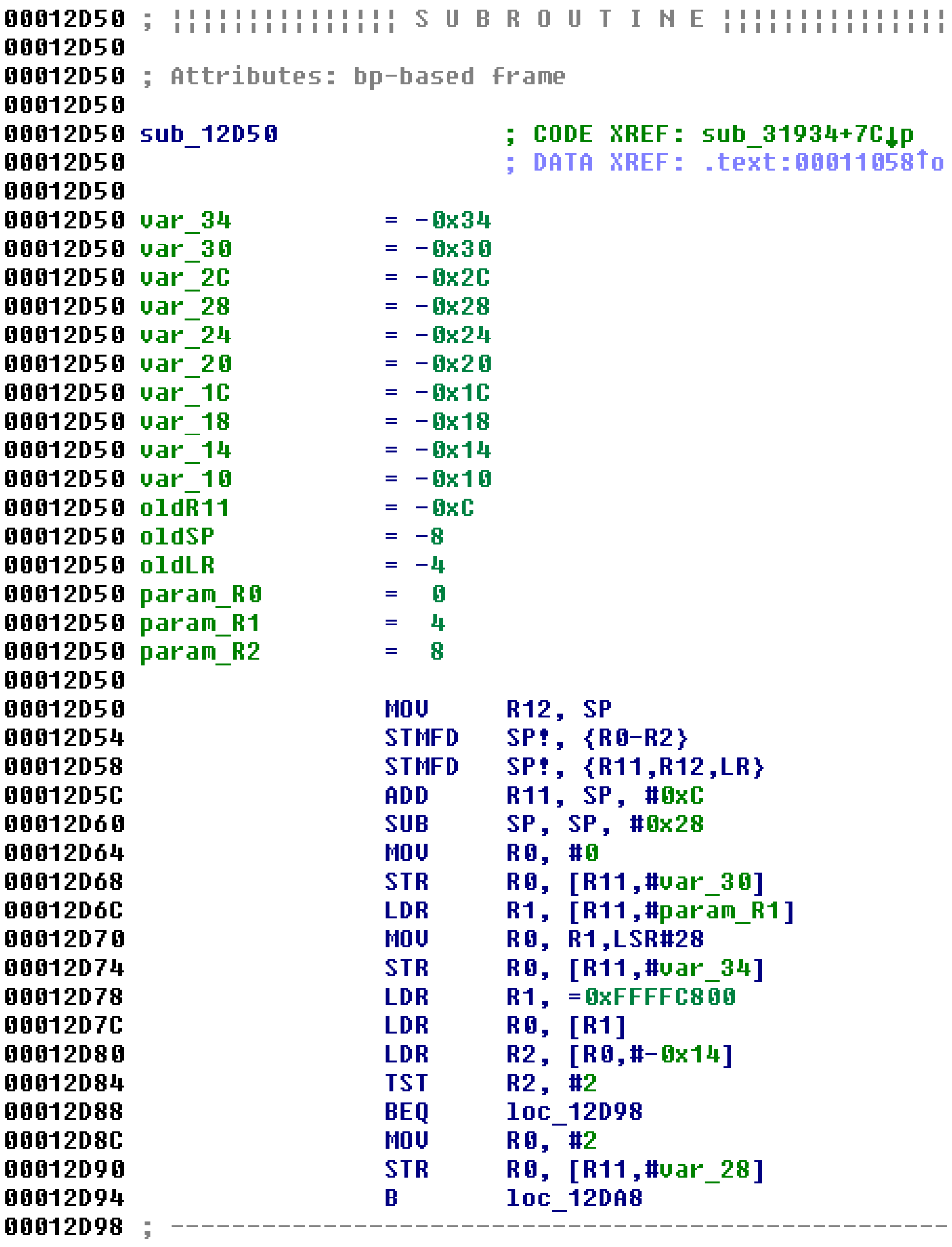


```
00011028 off_11028       DCD sub_167DC    ; DATA XREF: .text:off_17A80↓o
0001102C                 DCD 0
00011030                 DCD sub_13464
00011034                 DCD loc_135F0
00011038                 DCD sub_13E30
0001103C                 DCD sub_141A8
00011040                 DCD sub_14684
00011044                 DCD sub_15420
00011048                 DCD sub_15D7C
0001104C                 DCD sub_13774
00011050                 DCD loc_12CC4
00011054                 DCD loc_2ABAC
00011058                 DCD sub_12D50
0001105C                 DCD loc_21520
00011060                 DCD sub_1FFE8
00011064                 DCD loc_20E34
00011068                 DCD sub_21B28
```

Figure 4.49: Test 7 - Table at the beginning of filesys.exe.

The table contains memory addresses for several subroutines, and the addresses seemed quite similar to the addresses found in 4.48. Though they were not exactly alike, the list in 4.48 had been created with a different phone, which made us assume that this list might be what we were looking for. We had already found the API number to be 12. Counting from 0, the subroutine at index 12 is highlighted. It appears to be located at address 0x12D50, and figure 4.50 shows the code located there. As before, the first instructions form the entry stub. *R11* is set to be used as base, and the stack pointer is moved to make room for some additional values on the stack. *"MOV R0, #0"* and *"STR R0, [R11,#var_30]"* initializes one of the new variables on the stack to zero. As we will see below, this variable is what ends up being the return value received in *coredll.dll*. If it is still zero when received by *coredll.dll*, the procedure call will be interpreted as failed. The numeric value zero will then also be returned by *coredll.dll*. Recall from the API definition of *CeOidGetInfoEx* (4.44) that the return value will be either TRUE or FALSE. As we know, the numeric value zero is interpreted as FALSE, which according to the API correctly indicates failure.

We know from the analysis of *coredll.dll* that the second parameter, which is found in *R1*, is the object identifier (*CEOID*). *"LDR R1, [R11,#param_R1]"* gets the value of this parameter from its newly assigned slot on the stack, and puts it into *R1*. This may seem unnecessary, and in fact it is, since *R1* has not been changed. The value to be stored in *R1* is already there. The reason for this strange behavior is that the compiler, which translates



```
00012D50 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
00012D50
00012D50 ; Attributes: bp-based frame
00012D50
00012D50 sub_12D50                ; CODE XREF: sub_31934+7C↓p
00012D50                          ; DATA XREF: .text:00011058↑o
00012D50
00012D50 var_34          = -0x34
00012D50 var_30          = -0x30
00012D50 var_2C          = -0x2C
00012D50 var_28          = -0x28
00012D50 var_24          = -0x24
00012D50 var_20          = -0x20
00012D50 var_1C          = -0x1C
00012D50 var_18          = -0x18
00012D50 var_14          = -0x14
00012D50 var_10          = -0x10
00012D50 oldR11          = -0xC
00012D50 oldSP           = -8
00012D50 oldLR           = -4
00012D50 param_R0        =  0
00012D50 param_R1        =  4
00012D50 param_R2        =  8
00012D50
00012D50                 MOV     R12, SP
00012D54                 STMFD   SP!, {R0-R2}
00012D58                 STMFD   SP!, {R11,R12,LR}
00012D5C                 ADD     R11, SP, #0xC
00012D60                 SUB     SP, SP, #0x28
00012D64                 MOV     R0, #0
00012D68                 STR     R0, [R11,#var_30]
00012D6C                 LDR     R1, [R11,#param_R1]
00012D70                 MOV     R0, R1,LSR#28
00012D74                 STR     R0, [R11,#var_34]
00012D78                 LDR     R1, =0xFFFFC800
00012D7C                 LDR     R0, [R1]
00012D80                 LDR     R2, [R0,#-0x14]
00012D84                 TST     R2, #2
00012D88                 BEQ     loc_12D98
00012D8C                 MOV     R0, #2
00012D90                 STR     R0, [R11,#var_28]
00012D94                 B       loc_12DA8
00012D98 ; ---------------------------------------------------
```

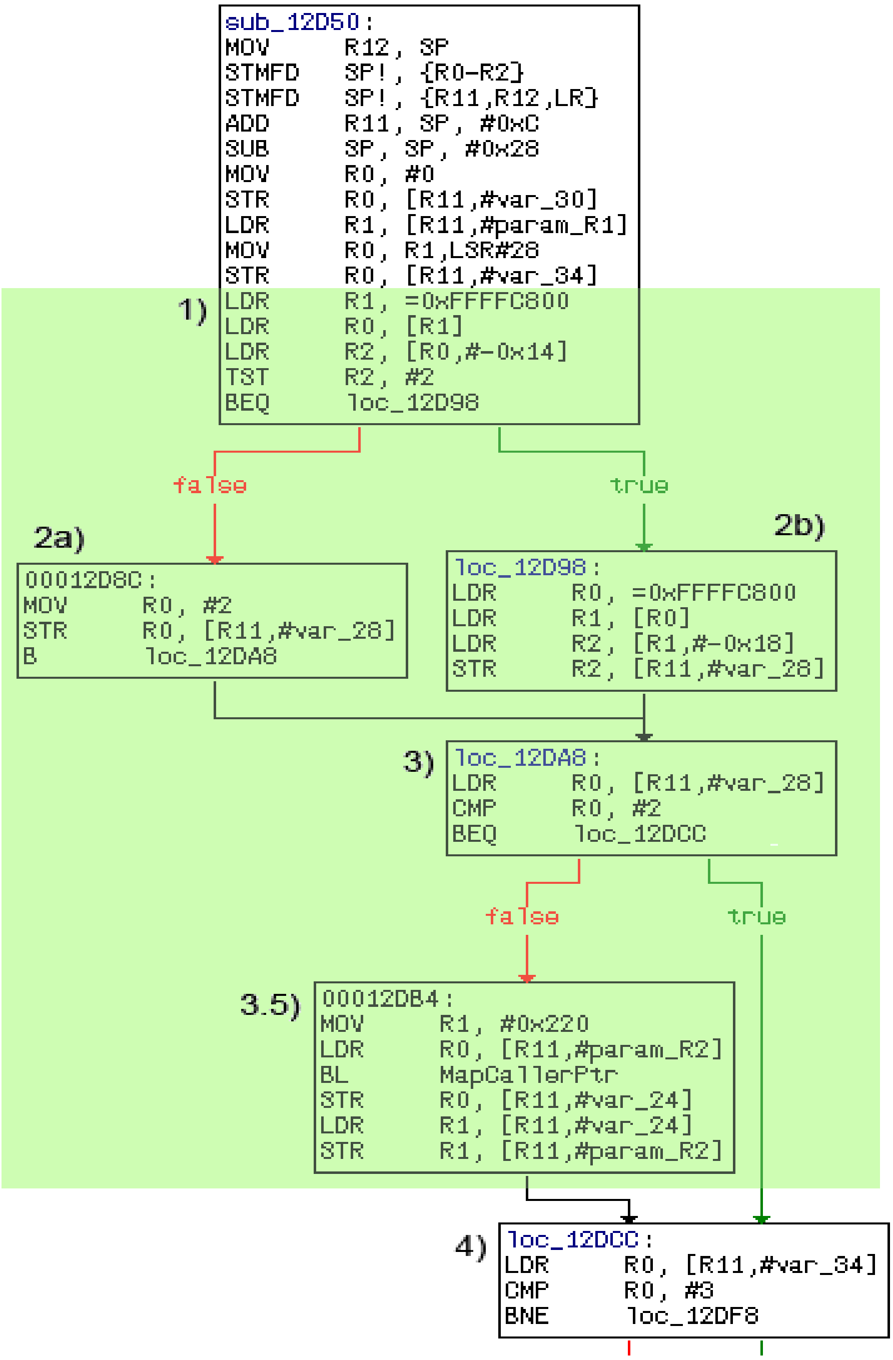
Figure 4.50: Test 7 - Beginning of code segment found when jumping to address 0x00012D50.

from high level language into assembly, may not always find optimal solutions. This may result in some unnecessary instructions at times, and we will see similar examples as we move along. *"MOV R0, R1,LSR#28"* performs a logical shift right operation on the object identifier value, shifting the value 28 bits to the right. The 4 previous MSBs of the object identifier, which is what is left after the shift operation, is then stored in *R0*. *"STR R0, [R11,#var_34]"* then stores this value into the last assigned slot on the stack so far.

What happens next is best explained by a flowchart. The flowchart (figure 4.51) itself has been generated by IDA, and we will now analyze the instructions within the green area. Continuing our analysis, *"LDR R1, =0xFFFFC800"* and *"LDR R0, [R1]"* loads some value from the memory address *0xFFFFC800*. We found several references to this memory location, including one at the XDA-developers web site [32]. *0xFFFFC800* is the location of a kernel data structure called *KDataStruct*. *KDataStruct* is defined in PRIVATE/WINCEOS/COREOS/NK/INC/NKARM.H, and in this file we also noticed a comment confirming the memory location of the KDataStruct.

As the memory location read was the same as the location of the *KDataStruct*, we knew that we were interested in the first 4 bytes of the structure. We found this field at the first position: *"LPDWORD lpvTls; /* 0x000 Current thread local storage pointer */"*. From the comment we knew that the value loaded was a pointer to the current threads local storage. *"LDR R2, [R0,#-0x14]"* uses this pointer to load the desired value into *R2*. The next instruction tests to see if the value loaded equals 2, and depending on the outcome of the test, the "branch if equal" instruction decides which code to execute next. We did not know exactly what this value was, but we had some suspicions. Anyhow, as we will see, it was not necessary to figure out that value to understand the rest of the code. For that reason, we did not spend much time on it. What has happened in the code segment marked 1 is that a value has been loaded from the threads local storage, and a branch is made on the condition of whether or not this value is 2.

As we analyze further, we can follow the flowchart. In the case where the value is 2, a new value is loaded with the same pointer as offset, and stored on the stack. If it was not 2, the value 2 is stored on that same stack location instead. Then, no matter the result from the previous conditional branch, step 3 merges the two paths and checks to see if the stack value is 2. That will be the case if either the previous condition failed, or if it succeeded and the new value found with *"LDR R2, [R1,#-0x18]"* was 2.



```
sub_12D50:
MOV     R12, SP
STMFD   SP!, {R0-R2}
STMFD   SP!, {R11,R12,LR}
ADD     R11, SP, #0xC
SUB     SP, SP, #0x28
MOV     R0, #0
STR     R0, [R11,#var_30]
LDR     R1, [R11,#param_R1]
MOV     R0, R1,LSR#28
STR     R0, [R11,#var_34]
LDR     R1, =0xFFFFC800
LDR     R0, [R1]
LDR     R2, [R0,#-0x14]
TST     R2, #2
BEQ     loc_12D98
```

```
00012D8C:
MOV     R0, #2
STR     R0, [R11,#var_28]
B       loc_12DA8
```

```
loc_12D98:
LDR     R0, =0xFFFFC800
LDR     R1, [R0]
LDR     R2, [R1,#-0x18]
STR     R2, [R11,#var_28]
```

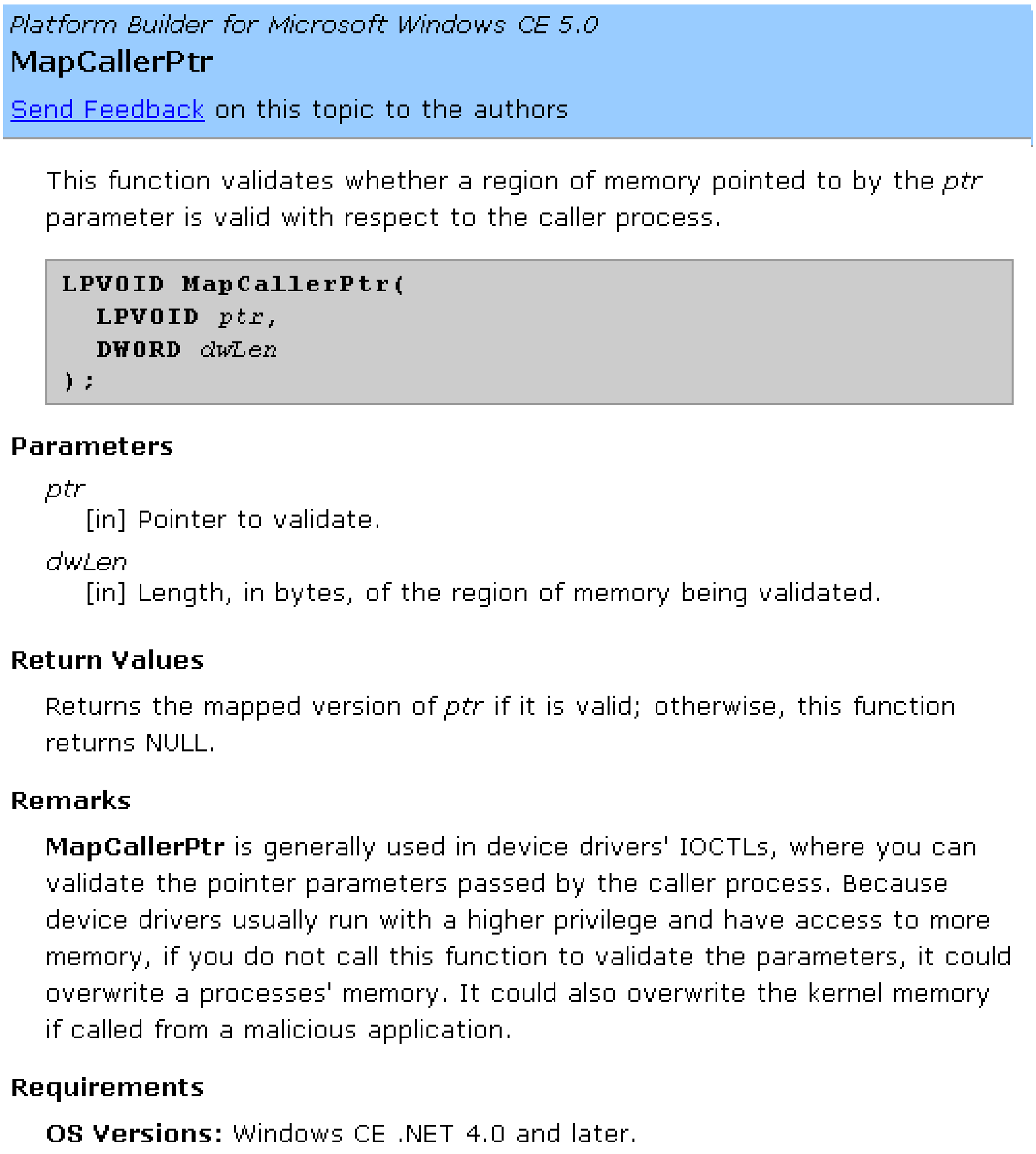
```
loc_12DA8:
LDR     R0, [R11,#var_28]
CMP     R0, #2
BEQ     loc_12DCC
```

```
00012DB4:
MOV     R1, #0x220
LDR     R0, [R11,#param_R2]
BL      MapCallerPtr
STR     R0, [R11,#var_24]
LDR     R1, [R11,#var_24]
STR     R1, [R11,#param_R2]
```

```
loc_12DCC:
LDR     R0, [R11,#var_34]
CMP     R0, #3
BNE     loc_12DF8
```

Figure 4.51: Test 7 - Assembly flow graph.

We now come to what really separates the paths taken. If the last condition fails, we need one extra step, denoted 3.5 in figure 4.51. A procedure named *MapCallerPtr* is called, with the two parameter registers *R0* and *R1* prepared with corresponding values. *R0* is given the value of the second argument to *CeOidGetInfoEx*, which was a pointer to the *CEOIDINFO* structure to be filled. *R1* is given a value matching the size assigned for the structure. Figure 4.52 shows the definition of *MapCallerPtr*.



Platform Builder for Microsoft Windows CE 5.0

MapCallerPtr

Send Feedback on this topic to the authors

This function validates whether a region of memory pointed to by the *ptr* parameter is valid with respect to the caller process.

```
LPVOID MapCallerPtr(
  LPVOID ptr,
  DWORD dwLen
);
```

Parameters

ptr
[in] Pointer to validate.

dwLen
[in] Length, in bytes, of the region of memory being validated.

Return Values

Returns the mapped version of *ptr* if it is valid; otherwise, this function returns NULL.

Remarks

MapCallerPtr is generally used in device drivers' IOCTLs, where you can validate the pointer parameters passed by the caller process. Because device drivers usually run with a higher privilege and have access to more memory, if you do not call this function to validate the parameters, it could overwrite a processes' memory. It could also overwrite the kernel memory if called from a malicious application.

Requirements

OS Versions: Windows CE .NET 4.0 and later.

Figure 4.52: Windows CE API - MapCallerPtr

As we can see from the API, *MapCallerPtr* is used to guarantee that the initiator of the procedure call has access to write to the desired memory location.

If we summarize the code shown in the green area of figure 4.51 in short, we could say that some information about the thread is checked to see whether or not the access rights to the memory location needs to be validated. We simply assumed that the unknown value that was checked had to do with the access rights of the thread, and whether or not it runs in kernel mode. But as stated before, for our purpose of finding how files are located, having the exact answer was unnecessary.

Figure 4.53 shows the code following the access check analyzed above. At this point we saw some signs telling us that we might be on the right track, as the assembly code started using the object identifier to make decisions. The 4 MSBs of the object identifier that was previously stored on the stack are now loaded back into *R0* with *"LDR R0, [R11,#var_34]"*. The values of these bits are tested against four different values on lines 0x00012DD0, 0x00012E3C, 0x00012E48, and 0x00012E58. The options are 3, 0, 1, and 0xE, and from this point on the execution take separate paths depending on the value of these 4 MSBs. From this we could already see that the object identifier was not just a random unique numeric value, but instead actually held information encoded in the identifier itself. After reverse engineering all the paths, we were able to see the difference:

3: Objects with a global identifier other than 0. This supports objects found on a mounted database volume other than the object store databases.

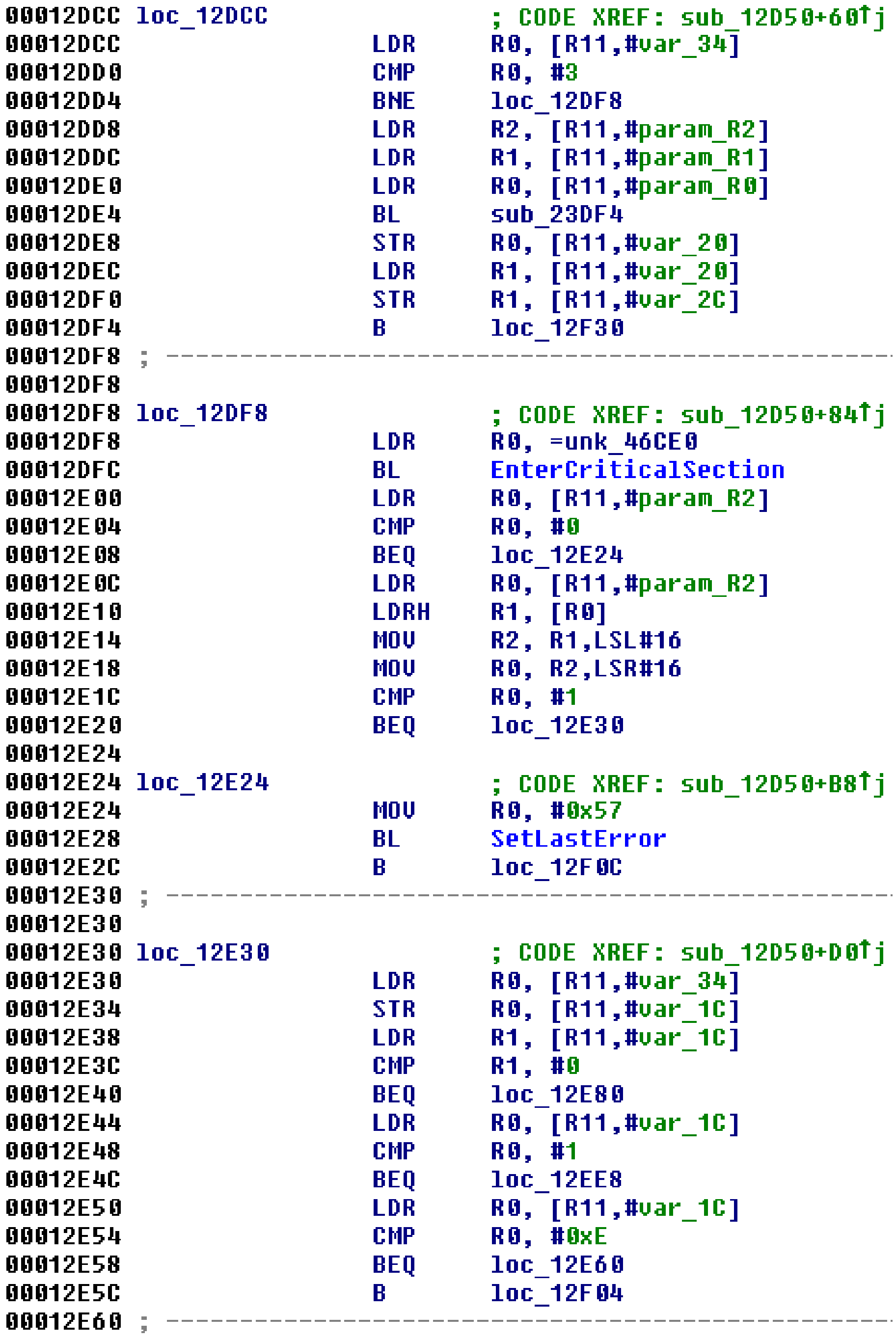
0: Objects that are located within the object store.

1: Objects located in ROM.

0xE: Special case objects.

Since our task was to figure out how to locate files in the object store, we will only present the details of the path concerning objects with identifiers having 0 in the 4 MSBs.

Starting at the top on figure 4.53, the test against 3 will then obviously fail,

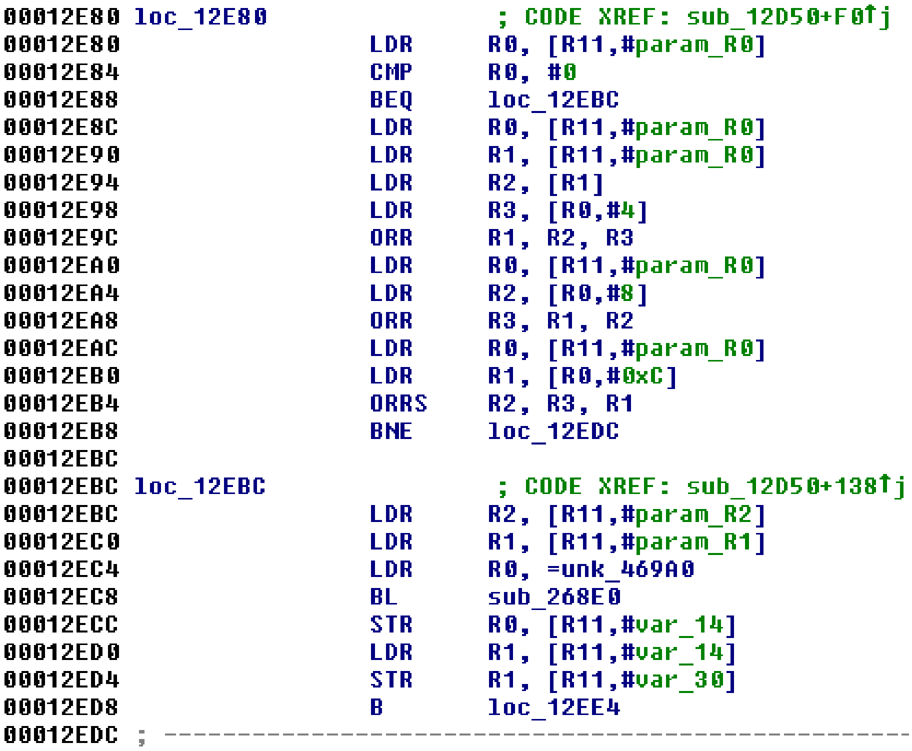


```
00012DCC loc_12DCC                     ; CODE XREF: sub_12D50+60↑j
00012DCC                LDR     R0, [R11,#var_34]
00012DD0                CMP     R0, #3
00012DD4                BNE     loc_12DF8
00012DD8                LDR     R2, [R11,#param_R2]
00012DDC                LDR     R1, [R11,#param_R1]
00012DE0                LDR     R0, [R11,#param_R0]
00012DE4                BL      sub_23DF4
00012DE8                STR     R0, [R11,#var_20]
00012DEC                LDR     R1, [R11,#var_20]
00012DF0                STR     R1, [R11,#var_2C]
00012DF4                B       loc_12F30
00012DF8 ; ---------------------------------------------------------------
00012DF8
00012DF8 loc_12DF8                     ; CODE XREF: sub_12D50+84↑j
00012DF8                LDR     R0, =unk_46CE0
00012DFC                BL      EnterCriticalSection
00012E00                LDR     R0, [R11,#param_R2]
00012E04                CMP     R0, #0
00012E08                BEQ     loc_12E24
00012E0C                LDR     R0, [R11,#param_R2]
00012E10                LDRH    R1, [R0]
00012E14                MOV     R2, R1,LSL#16
00012E18                MOV     R0, R2,LSR#16
00012E1C                CMP     R0, #1
00012E20                BEQ     loc_12E30
00012E24
00012E24 loc_12E24                     ; CODE XREF: sub_12D50+B8↑j
00012E24                MOV     R0, #0x57
00012E28                BL      SetLastError
00012E2C                B       loc_12F0C
00012E30 ; ---------------------------------------------------------------
00012E30
00012E30 loc_12E30                     ; CODE XREF: sub_12D50+D0↑j
00012E30                LDR     R0, [R11,#var_34]
00012E34                STR     R0, [R11,#var_1C]
00012E38                LDR     R1, [R11,#var_1C]
00012E3C                CMP     R1, #0
00012E40                BEQ     loc_12E80
00012E44                LDR     R0, [R11,#var_1C]
00012E48                CMP     R0, #1
00012E4C                BEQ     loc_12EE8
00012E50                LDR     R0, [R11,#var_1C]
00012E54                CMP     R0, #0xE
00012E58                BEQ     loc_12E60
00012E5C                B       loc_12F04
00012E60 ; ---------------------------------------------------------------
```

Figure 4.53: Test 7 - Seperating MSB paths.

and a branch is made to 0x00012DF8. What happens there is that a call is made to *EnterCriticalSection* to ensure mutual-exclusion synchronization so that no changes are made to the object while retrieving information. Then two tests are performed on the *CEOIDINFO* pointer.

The first checks that the pointer is not 0. The second checks if the version field of the structure is set to 1. If either of the two tests fails, execution is continued from 0x00012E24, which sets the last error value to 0x57, leaves the critical section, and return with an error. Error message 0x57 is defined as ERROR_INVALID_PARAMETER (in *WinError.h*). If all goes well, execution continues from 00012E30, where the three remaining paths (0, 1, and 0xE) are separated. In our case, the *"CMP R1, #0"* instruction will find the two values to be equal, which cause a branch to 0x00012E80.

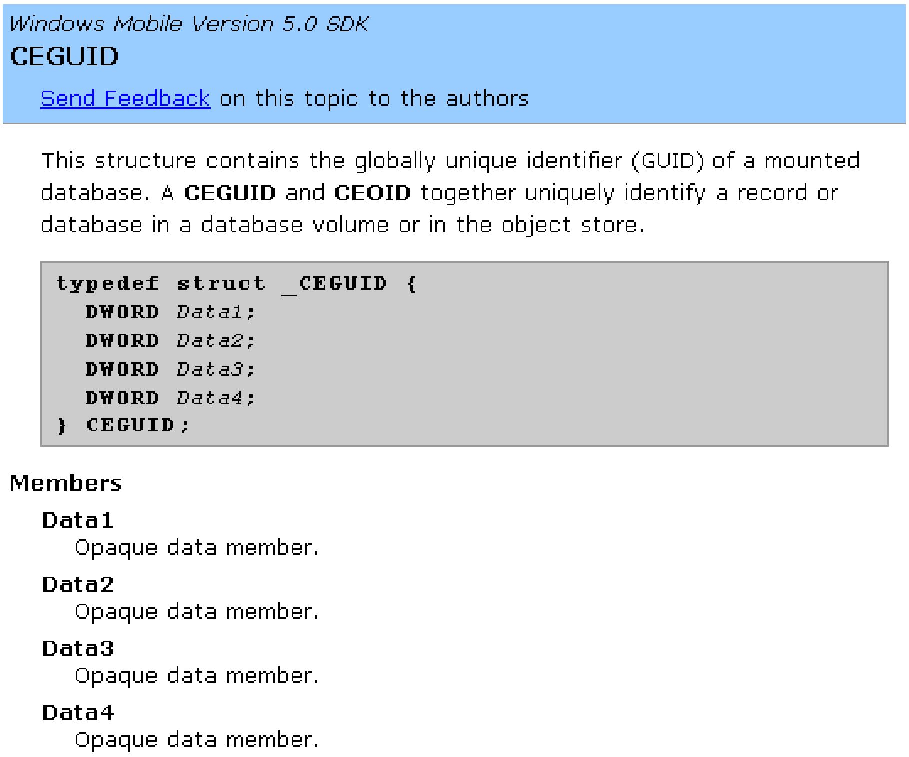


```
00012E80 loc_12E80                       ; CODE XREF: sub_12D50+F0↑j
00012E80                 LDR     R0, [R11,#param_R0]
00012E84                 CMP     R0, #0
00012E88                 BEQ     loc_12EBC
00012E8C                 LDR     R0, [R11,#param_R0]
00012E90                 LDR     R1, [R11,#param_R0]
00012E94                 LDR     R2, [R1]
00012E98                 LDR     R3, [R0,#4]
00012E9C                 ORR     R1, R2, R3
00012EA0                 LDR     R0, [R11,#param_R0]
00012EA4                 LDR     R2, [R0,#8]
00012EA8                 ORR     R3, R1, R2
00012EAC                 LDR     R0, [R11,#param_R0]
00012EB0                 LDR     R1, [R0,#0xC]
00012EB4                 ORRS    R2, R3, R1
00012EB8                 BNE     loc_12EDC
00012EBC
00012EBC loc_12EBC                       ; CODE XREF: sub_12D50+138↑j
00012EBC                 LDR     R2, [R11,#param_R2]
00012EC0                 LDR     R1, [R11,#param_R1]
00012EC4                 LDR     R0, =unk_469A0
00012EC8                 BL      sub_268E0
00012ECC                 STR     R0, [R11,#var_14]
00012ED0                 LDR     R1, [R11,#var_14]
00012ED4                 STR     R1, [R11,#var_30]
00012ED8                 B       loc_12EE4
00012EDC ; ---------------------------------------------------
```

Figure 4.54: Test 7 - Validating that object store path is correct.

At address 0x00012E80 we found the code in figure 4.54. The first parameter to *CeOidGetInfoEx*, which was a pointer to a global identifier (*PCEGUID*), is loaded from the stack and compared with 0. If the *PCEGUID* is 0, there is no global identifier, which in turn indicates that the file is located in the object store. A branch is then made to 0x00012EBC.

If, on the other hand, the *PCEGUID* is defined, the code needs to check if this global identifier actually is the identifier of the object store. Figure 4.55 shows the definition of the global identifier (*CEGUID*). It is defined as a structure with four dwords. These values are loaded one at a time, and then *OR*ed together, before *"ORRS R2, R3, R1"* finally, in addition to performing an *OR*-instruction, sets the zero flag if *R2* ends up with 0. As the final value in *R2* is decided by an *OR* of all the four dwords in the *CEGUID*, it will only be 0 if all these dwords are 0. If this is the case, the branch is not made, and execution continues from 0x00012EBC as it did when the *PCEGUID* was 0. If *R2* does not end up with 0, a branch is made to a location where an error code is determined, and the procedure returns as failed.



Windows Mobile Version 5.0 SDK

CEGUID

Send Feedback on this topic to the authors

This structure contains the globally unique identifier (GUID) of a mounted database. A **CEGUID** and **CEOID** together uniquely identify a record or database in a database volume or in the object store.

```
typedef struct _CEGUID {
  DWORD Data1;
  DWORD Data2;
  DWORD Data3;
  DWORD Data4;
} CEGUID;
```

Members

Data1
Opaque data member.

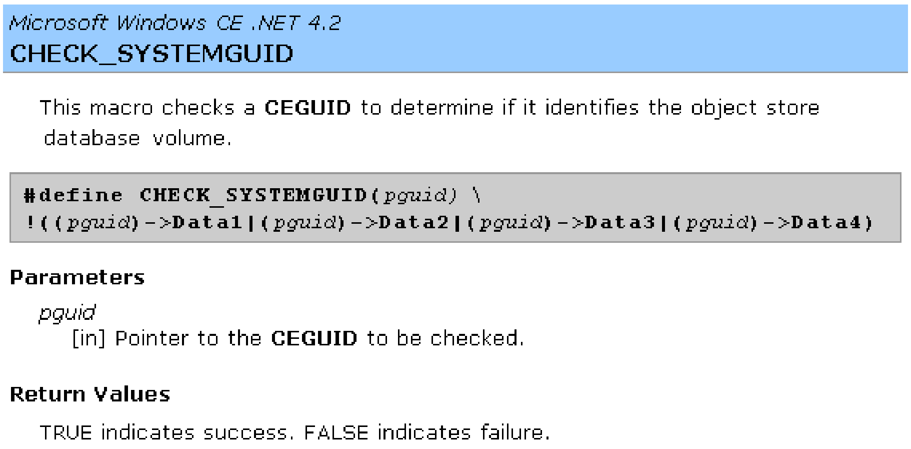
Data2
Opaque data member.

Data3
Opaque data member.

Data4
Opaque data member.

Figure 4.55: Windows CE API - CEGUID structure

The Windows CE API defines a macro to perform a check on a *CEGUID* to see if it identifies the object store. Figure 4.56 shows the macro. Comparing this macro with the assembly code just analyzed, we see that this check is exactly what happens in the code. When execution continues from address 0x00012EBC, we now know that the object is located in the object store.



Microsoft Windows CE .NET 4.2

CHECK_SYSTEMGUID

This macro checks a **CEGUID** to determine if it identifies the object store database volume.

```
#define CHECK_SYSTEMGUID(pguid) \
!((pguid)->Data1|(pguid)->Data2|(pguid)->Data3|(pguid)->Data4)
```

Parameters

pguid
[in] Pointer to the **CEGUID** to be checked.

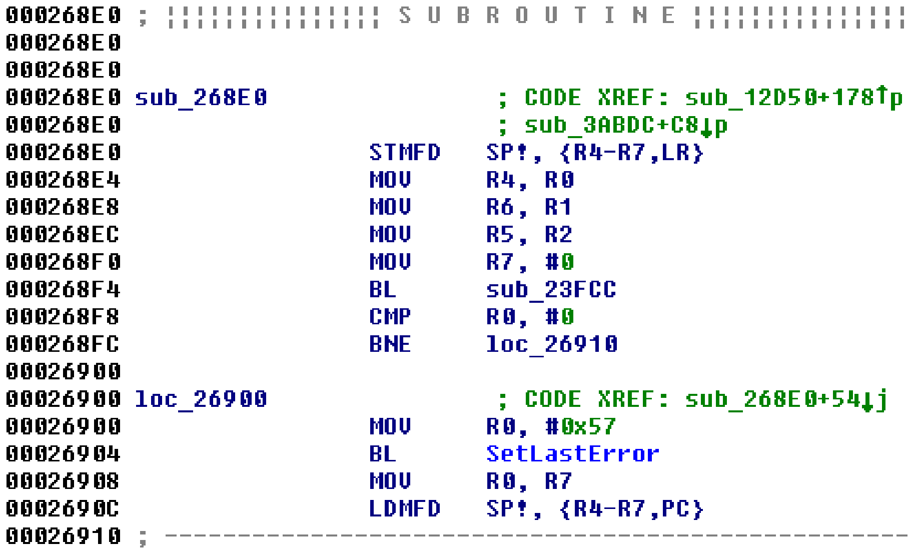
Return Values

TRUE indicates success. FALSE indicates failure.

Figure 4.56: Windows CE API - CHECK_SYSTEMGUID macro

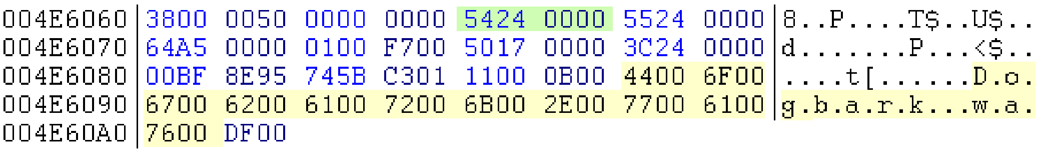
At address 0x00012EBC we see the object identifier (*CEOID*) and the pointer to the file info structure (*CEOIDINFOEX**) being loaded back into *R1* and *R2*, and some global variable loaded into *R0*. A branch is then made to 0x000268E0 (figure 4.57) with these three parameters. At 0x000268E0 there is just a redirection to a new subroutine located at 0x00023FCC. The result from this subroutine is compared with 0, where a 0 indicates failure, and any other value cause a branch to 0x00026910.

Figure 4.59 shows the subroutine found at address 0x00023FCC. We noticed that several calculations and comparisons were made on the object identifier value itself, so we decided that the best way to keep up with what happens in the code segment was to simulate the code on paper with some real parameter values. For the object identifier in *R1*, we used the identifier of the meta-data object for the file *Dogbark.wav* which is one of the phones default files.



```
000268E0 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
000268E0
000268E0
000268E0 sub_268E0                   ; CODE XREF: sub_12D50+178↑p
000268E0                             ; sub_3ABDC+C8↓p
000268E0                 STMFD   SP!, {R4-R7,LR}
000268E4                 MOV     R4, R0
000268E8                 MOV     R6, R1
000268EC                 MOV     R5, R2
000268F0                 MOV     R7, #0
000268F4                 BL      sub_23FCC
000268F8                 CMP     R0, #0
000268FC                 BNE     loc_26910
00026900
00026900 loc_26900                   ; CODE XREF: sub_268E0+54↓j
00026900                 MOV     R0, #0x57
00026904                 BL      SetLastError
00026908                 MOV     R0, R7
0002690C                 LDMFD   SP!, {R4-R7,PC}
00026910 ; ---------------------------------------------------
```

Figure 4.57: Test 7 - Redirecting to 0x00023FCC.

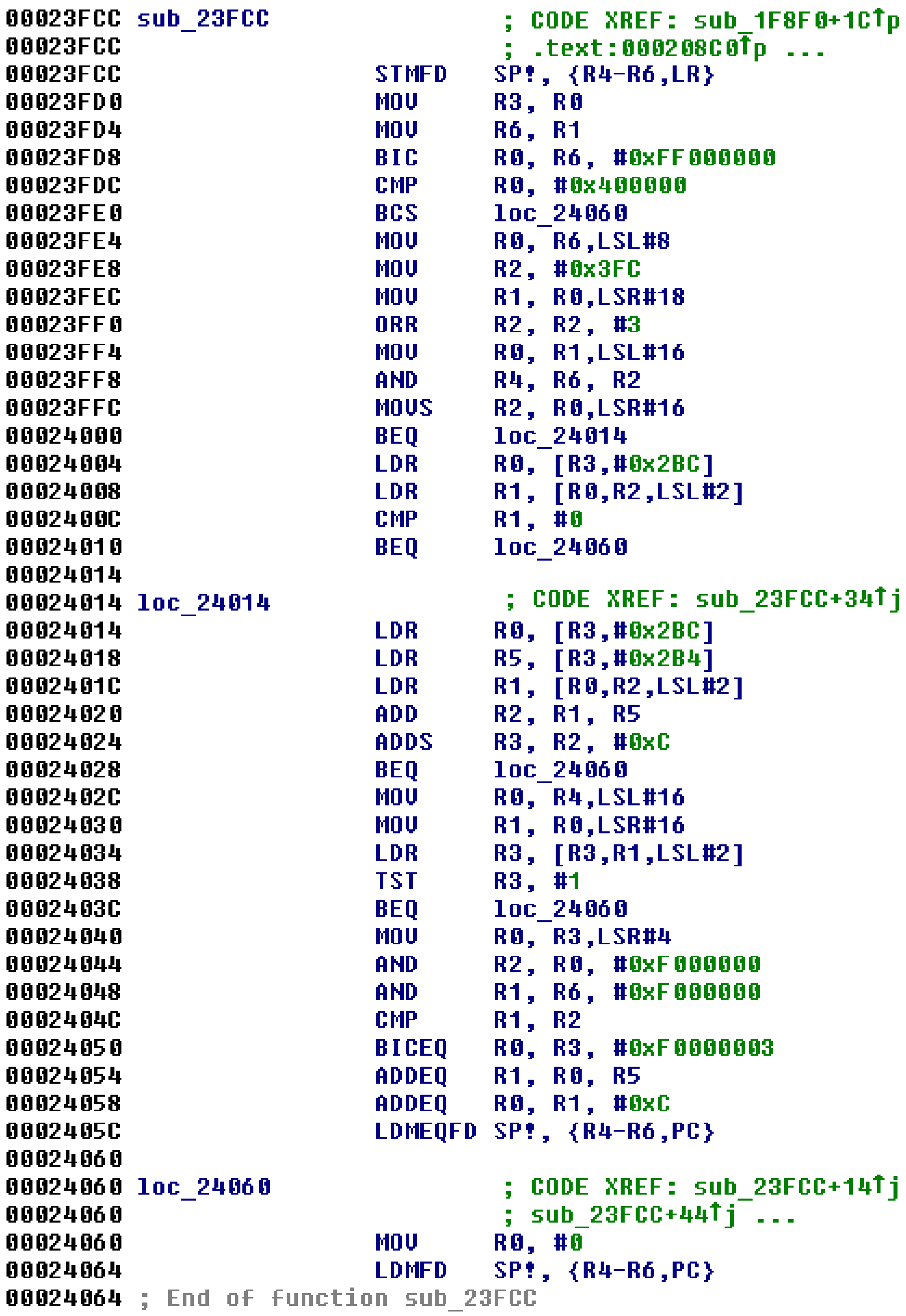


```
004E6060|3800 0050 0000 0000 5424 0000 5524 0000|8..P....T$..U$..
004E6070|64A5 0000 0100 F700 5017 0000 3C24 0000|d.......P...<$..
004E6080|00BF 8E95 745B C301 1100 0B00 4400 6F00|....t[......D.o.
004E6090|6700 6200 6100 7200 6B00 2E00 7700 6100|g.b.a.r.k...w.a.
004E60A0|7600 DF00
```

Figure 4.58: Test 7 - The file *dogbark.wav* used in the simulation.

We located the file in memory to find its identifier. The identifier of this object was *0x00002454*, shown in green on figure 4.58. *R2* contains a pointer to the structure to be filled. As this is an out parameter, we did not need a precise value for it, but kept the definition of the *CEOIDINFOEX* structure (figure 4.60) in mind.

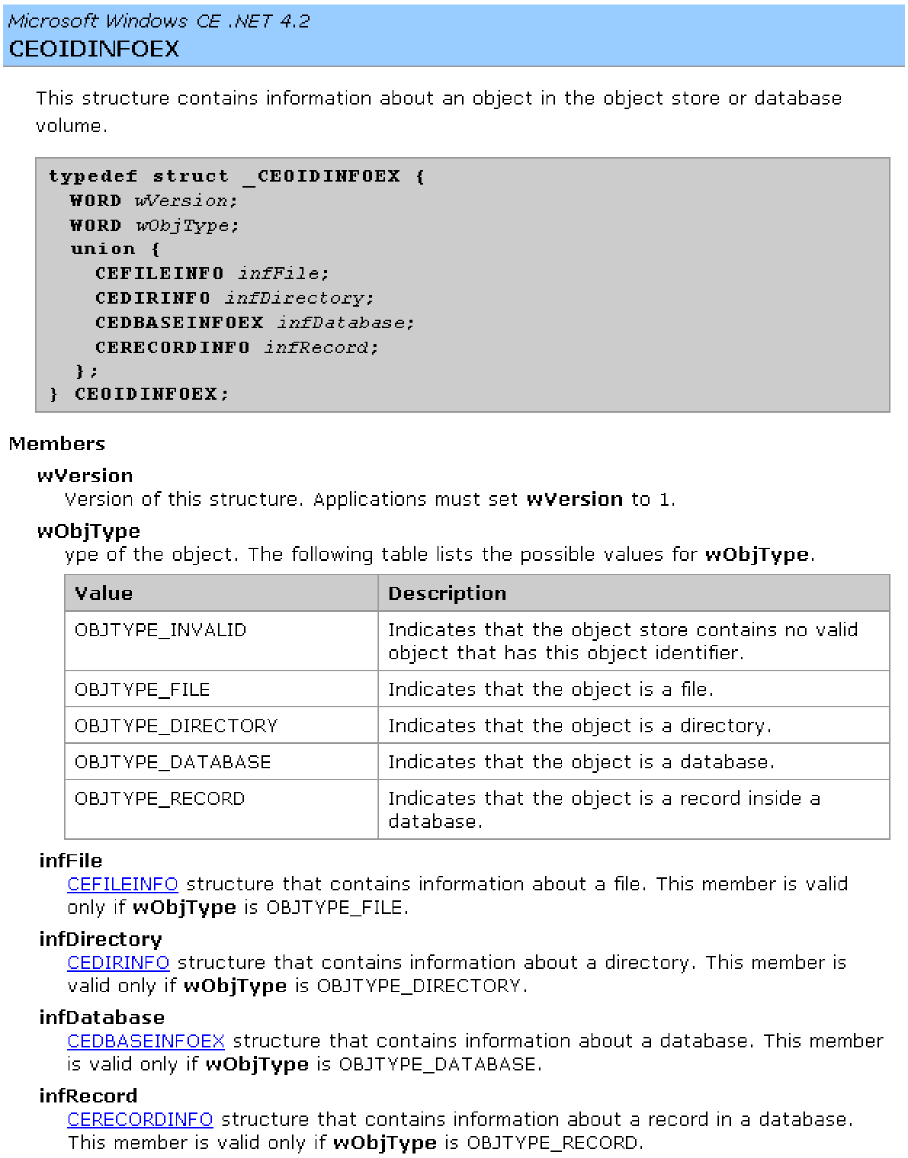
For *R0*, we needed to locate the global variable that was loaded before the subroutine call in figure 4.57. IDA noted that the variable was located at address 0x00012F44. When locating this memory address, we needed to make sure that we read the address from *filesys.exe's* perspective. By using the Itsutils[7] tool called pps, which dumps information about processes, we found the filesys.exe process to be located at address 0x04000000. With 0x00012F44 as offset from the start of *filesys.exe's* memory slot, we found the value of the global variable to be *0x00469a0* (which we learned was exactly the value used by IDA to name the variable on line 0x00012EC4 in figure 4.54).



```
00023FCC sub_23FCC                       ; CODE XREF: sub_1F8F0+1C↑p
00023FCC                                 ; .text:000208C0↑p ...
00023FCC                 STMFD   SP!, {R4-R6,LR}
00023FD0                 MOV     R3, R0
00023FD4                 MOV     R6, R1
00023FD8                 BIC     R0, R6, #0xFF000000
00023FDC                 CMP     R0, #0x400000
00023FE0                 BCS     loc_24060
00023FE4                 MOV     R0, R6,LSL#8
00023FE8                 MOV     R2, #0x3FC
00023FEC                 MOV     R1, R0,LSR#18
00023FF0                 ORR     R2, R2, #3
00023FF4                 MOV     R0, R1,LSL#16
00023FF8                 AND     R4, R6, R2
00023FFC                 MOVS    R2, R0,LSR#16
00024000                 BEQ     loc_24014
00024004                 LDR     R0, [R3,#0x2BC]
00024008                 LDR     R1, [R0,R2,LSL#2]
0002400C                 CMP     R1, #0
00024010                 BEQ     loc_24060
00024014
00024014 loc_24014                       ; CODE XREF: sub_23FCC+34↑j
00024014                 LDR     R0, [R3,#0x2BC]
00024018                 LDR     R5, [R3,#0x2B4]
0002401C                 LDR     R1, [R0,R2,LSL#2]
00024020                 ADD     R2, R1, R5
00024024                 ADDS    R3, R2, #0xC
00024028                 BEQ     loc_24060
0002402C                 MOV     R0, R4,LSL#16
00024030                 MOV     R1, R0,LSR#16
00024034                 LDR     R3, [R3,R1,LSL#2]
00024038                 TST     R3, #1
0002403C                 BEQ     loc_24060
00024040                 MOV     R0, R3,LSR#4
00024044                 AND     R2, R0, #0xF000000
00024048                 AND     R1, R6, #0xF000000
0002404C                 CMP     R1, R2
00024050                 BICEQ   R0, R3, #0xF0000003
00024054                 ADDEQ   R1, R0, R5
00024058                 ADDEQ   R0, R1, #0xC
0002405C                 LDMEQFD SP!, {R4-R6,PC}
00024060
00024060 loc_24060                       ; CODE XREF: sub_23FCC+14↑j
00024060                                 ; sub_23FCC+44↑j ...
00024060                 MOV     R0, #0
00024064                 LDMFD   SP!, {R4-R6,PC}
00024064 ; End of function sub_23FCC
```

Figure 4.59: Test 7 - Utilizing the rest of the object identifier.

We were now ready to begin the simulation. Table 4.5 shows how we stepped through the code segment one instruction at a time.



Microsoft Windows CE .NET 4.2

CEOIDINFOEX

This structure contains information about an object in the object store or database volume.

```
typedef struct _CEOIDINFOEX {
  WORD wVersion;
  WORD wObjType;
  union {
    CEFILEINFO infFile;
    CEDIRINFO infDirectory;
    CEDBASEINFOEX infDatabase;
    CERECORDINFO infRecord;
  };
} CEOIDINFOEX;
```

Members

wVersion
Version of this structure. Applications must set **wVersion** to 1.

wObjType
ype of the object. The following table lists the possible values for **wObjType**.

Value	Description
OBJTYPE_INVALID	Indicates that the object store contains no valid object that has this object identifier.
OBJTYPE_FILE	Indicates that the object is a file.
OBJTYPE_DIRECTORY	Indicates that the object is a directory.
OBJTYPE_DATABASE	Indicates that the object is a database.
OBJTYPE_RECORD	Indicates that the object is a record inside a database.

infFile
CEFILEINFO structure that contains information about a file. This member is valid only if **wObjType** is OBJTYPE_FILE.

infDirectory
CEDIRINFO structure that contains information about a directory. This member is valid only if **wObjType** is OBJTYPE_DIRECTORY.

infDatabase
CEDBASEINFOEX structure that contains information about a database. This member is valid only if **wObjType** is OBJTYPE_DATABASE.

infRecord
CERECORDINFO structure that contains information about a record in a database. This member is valid only if **wObjType** is OBJTYPE_RECORD.

Figure 4.60: Windows CE API - CEOIDINFOEX structure

Instruction	Result	Comment
STMFD SP!, R4-R6,LR		Entry stub
MOV R3, R0	R3 = 0x469A0	
MOV R6, R1	R6 = 0x2454	The identifier.
BIC R0, R6, #0xFF000000	R0 = 0x469A0	Clears the 8MSBs, which have done their part of the lookup
CMP R0, #0x400000		Checks that the OID is less than the limit 0x400000
BCS loc_24060		Branches to return 0 if OID is incorrect
MOV R0, R6,LSL#8	R0 = 0x245400	
MOV R2, #0x3FC	R2 = 0x3FC	
MOV R1, R0,LSR#18	R1 = 0x9	b23-b10 of OID
ORR R2, R2, #3	R2 = 0x3FF	
MOV R0, R1,LSL#16	R0 = 0x90000	
AND R4, R6, R2	R4 = 0x54	b9-b0 of OID
MOVS R2, R0,LSR#16	R2 = 0x9	
BEQ loc_24014		Branches if R2 is 0
LDR R0, [R3,#0x2BC]	R0 = 0x42001000	Uses the global variable in R3 as base, in a memory load from offset 0x2BC
LDR R1, [R0,R2,LSL#2]	R1 = 0x4A9C68	Uses the value found in the previous load as a base in a new memory load. The offset is b23-b10 of the OID, shifted 2 positions to the left. This equals a multiplication by 4.
CMP R1, #0		
BEQ loc_24060		If the value found and placed in R1 is 0, the procedure failed
LDR R0, [R3,#0x2BC]	R0 = 0x42001000	Uses the global variable to load from offset 0x2BC as before
LDR R5, [R3,#0x2B4]	R5 = 0x42005000	Uses the global variable to load a new value from offset 0x2B4

LDR R1, [R0,R2,LSL#2]	R1 = 0x4A9C68	Loads the same address as before
ADD R2, R1, R5	R2 = 0x424AEC68	
ADDS R3, R2, #0xC	R3 = 0x424AEC74	
BEQ loc_24060		Branches and returns failed if R3 is 0
MOV R0, R4,LSL#16	R0 = 0x540000	
MOV R1, R0,LSR#16	R1 = 0x54	
LDR R3, [R3,R1,LSL#2]	R3 = 0x4E1061	Uses the 10 LSBs of the OID as offset from R3.
TST R3, #1		
BEQ loc_24060		
MOV R0, R3,LSR#4	R0 = 0x4E106	
AND R2, R0, #0xF000000	R2 = 0	
AND R1, R6, #0xF000000	R1 = 0	
CMP R1, R2		
BICEQ R0, R3, #0xF0000003	R0 = 0x4E1060	
ADDEQ R1, R0, R5	R1 = 0x424E6060	
ADDEQ R0, R1, #0xC	R0 = 0x424E606C	

Table 4.5: Test 7 - Simulating a file lookup.

The result of the simulation shows the value *0x424E606C* being returned by the subroutine. Surprisingly, this value was exactly the memory address of the data held by *Dogbark.wav's* metadata object, with address 0x42000000 being the start of the object store. The object identifier had been split into pieces, where the pieces themselves were used as offsets into memory to locate the file. We had now found the way in which files are located, which had been the missing link in our reverse engineering process. But to get a better impression of what actually happened in the code, we needed to transform this assembly code from the PSM into a more abstract definition to include in the PIM.

Transform to PIM

We sketched what we had just experienced, and made the model shown in figure 4.61. Two values had been loaded from a structure pointed to by the global variable that was passed as an argument. The offset values into this structure were 0x2B4 and 0x2BC. Both of these loaded values *0x42001000* and *0x42005000* pointed to a table, which we have called *Object table list* and *Object table*. Bits 10 to 23 were used as index into the table pointed to by the value found at structure index 0x2BC (*Object table list*). The value found there were used as a new offset from the value found at structure index 0x2B4, to locate the correct *Object table* where the object address could be found. From the *Object table*, bits 0 to 9 were used as index to locate the actual address of the object.

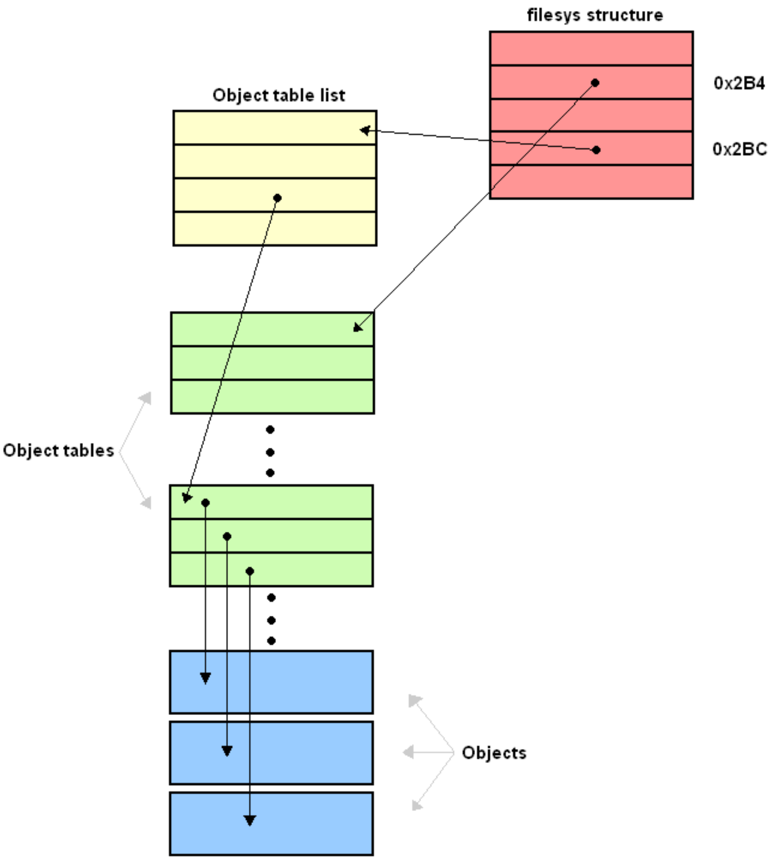


Figure 4.61: Test 7 - Locating an object.

If we combine this lookup with the connection we have seen previously between metadata, filedata list, and filedata objects, we get the complete file lookup shown in figure 4.62.

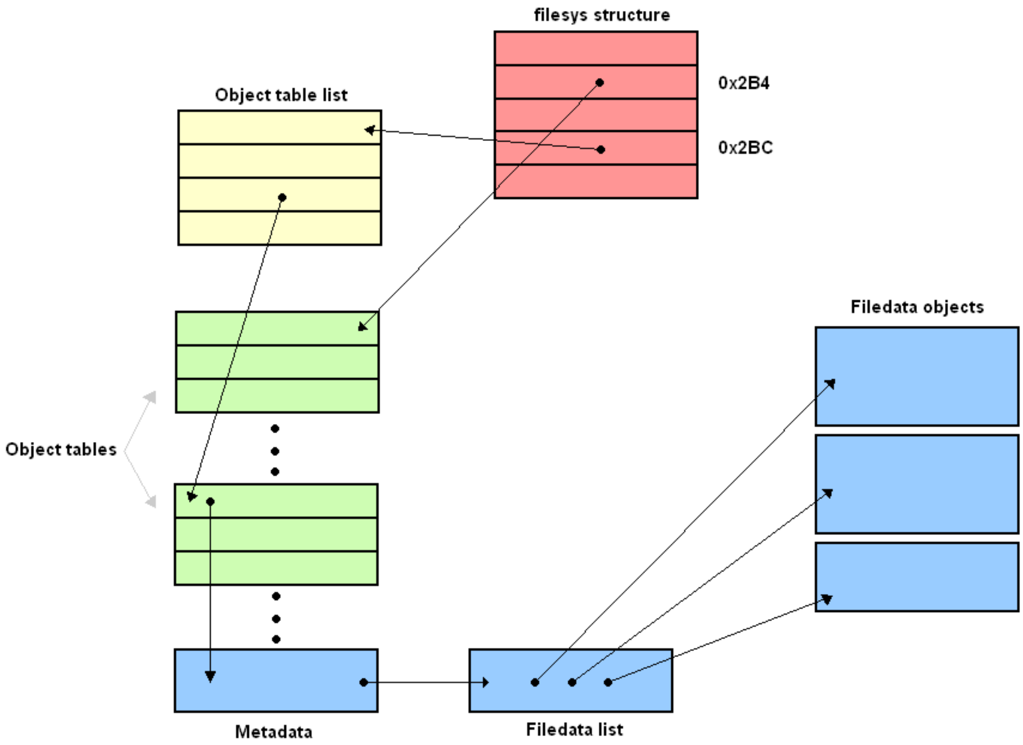


Figure 4.62: Test 7 - Locating a file.

4.2.7 Defining a Strategy - Third Loop

Now that we had gained a good understanding of both the storage format of objects and how they were located, we started to get a good overview of how the object store is built. This made us discover things that we did not see at earlier stages. As we had found how the object store uses tables to locate objects, we wanted to test how these tables were affected when objects were deleted. In addition, when inspecting the memory dumps, we had seen some memory addresses that kept appearing at the beginning of unused spaces. Now that we had a better understanding of the object store, we also wanted to inspect these addresses further. By looking at the data pointed to by these addresses, we often found the addresses pointing to new addresses in a way as if it were a linked list.

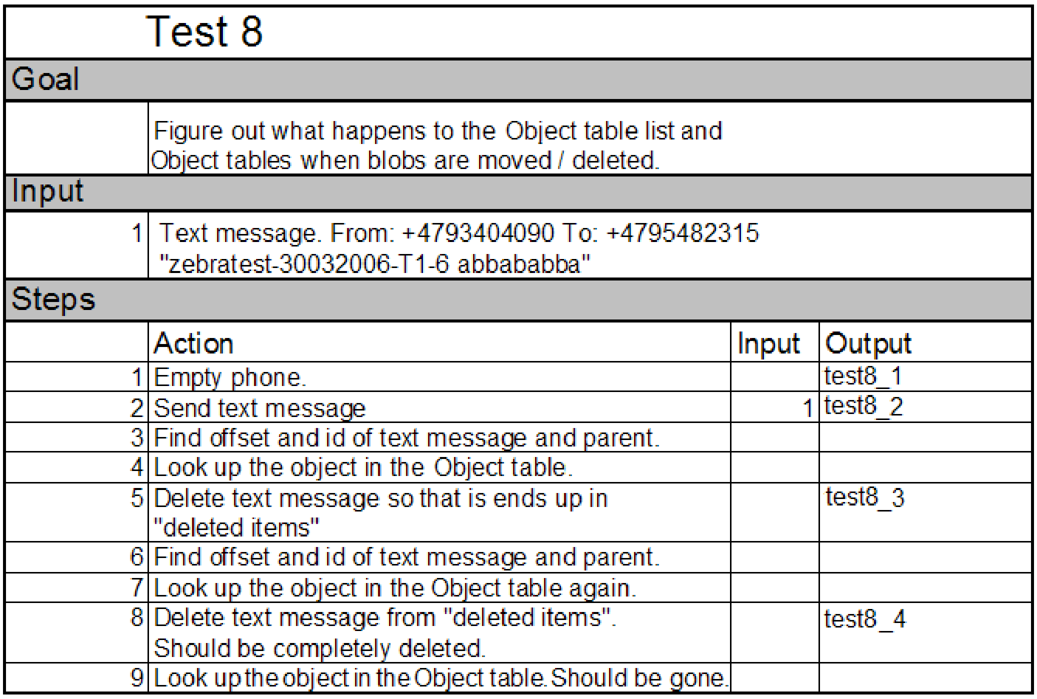
We decided to inspect these aspects further, and went back to the black box strategy as before.

4.2.8 Testing

4.2.8.1 Test 8

Create test Now that have located the *Object table list* and the *Object tables* with the help of disassembly we would like to understand how they are updated when files are moved or deleted. The point of this is to figure out if there is any way we can extract blobs that have been ″deleted″ by the user. It is very uncommon for such data to be overwritten at once at deletion. In most storage systems the data is only marked as ready for re-allocation.

Figure 4.63 contains the eight test.

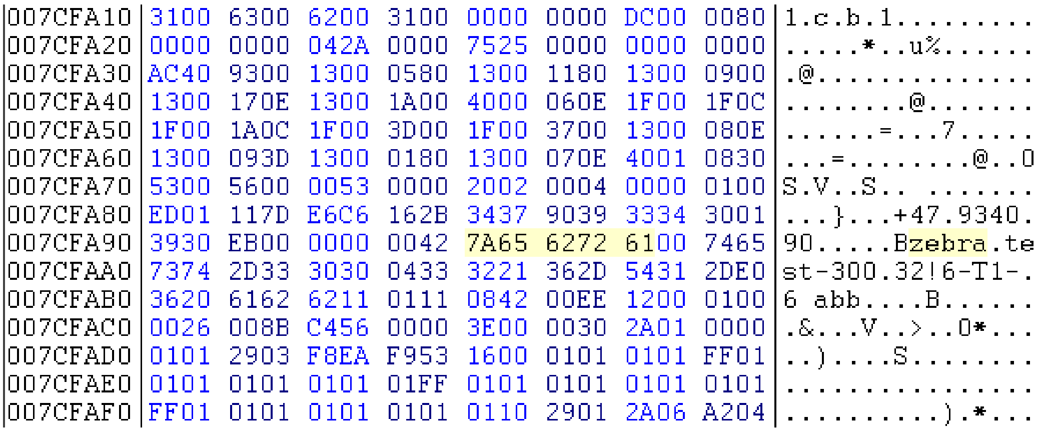


	Test 8		
Goal			
	Figure out what happens to the Object table list and Object tables when blobs are moved / deleted.		
Input			
1	Text message. From: +4793404090 To: +4795482315 "zebratest-30032006-T1-6 abbababba"		
Steps			
	Action	Input	Output
1	Empty phone.		test8_1
2	Send text message	1	test8_2
3	Find offset and id of text message and parent.		
4	Look up the object in the Object table.		
5	Delete text message so that is ends up in "deleted items"		test8_3
6	Find offset and id of text message and parent.		
7	Look up the object in the Object table again.		
8	Delete text message from "deleted items". Should be completely deleted.		test8_4
9	Look up the object in the Object table. Should be gone.		

Figure 4.63: Test 8 - Object table list and Object tables.

Extract info/Adjust PSM

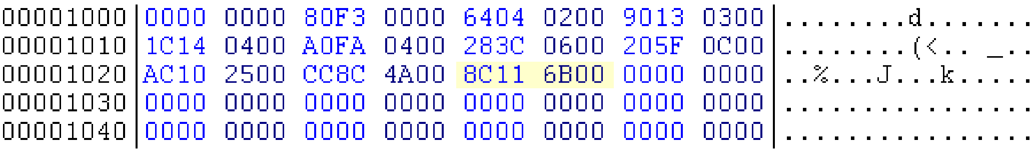
Searching for ”zebra” we find our text message at offset 0x007CFA1C (figure 4.64). Its id is *0x2A04* and its parent id is *0x2575* (Inbox database).



```
007CFA10|3100 6300 6200 3100 0000 0000 DC00 0080|1.c.b.1.........
007CFA20|0000 0000 042A 0000 7525 0000 0000 0000|.....*..u%......
007CFA30|AC40 9300 1300 0580 1300 1180 1300 0900|.@..............
007CFA40|1300 170E 1300 1A00 4000 060E 1F00 1F0C|........@.......
007CFA50|1F00 1A0C 1F00 3D00 1F00 3700 1300 080E|......=...7.....
007CFA60|1300 093D 1300 0180 1300 070E 4001 0830|...=........@..0
007CFA70|5300 5600 0053 0000 2002 0004 0000 0100|S.V..S.. .......
007CFA80|ED01 117D E6C6 162B 3437 9039 3334 3001|...}...+47.9340.
007CFA90|3930 EB00 0000 0042 7A65 6272 6100 7465|90.....Bzebra.te
007CFAA0|7374 2D33 3030 0433 3221 362D 5431 2DE0|st-300.32!6-T1-.
007CFAB0|3620 6162 6211 0111 0842 00EE 1200 0100|6 abb....B......
007CFAC0|0026 008B C456 0000 3E00 0030 2A01 0000|.&...V..>..0*...
007CFAD0|0101 2903 F8EA F953 1600 0101 0101 FF01|..)....S........
007CFAE0|0101 0101 0101 01FF 0101 0101 0101 0101|................
007CFAF0|FF01 0101 0101 0101 0110 2901 2A06 A204|..........).*...
```

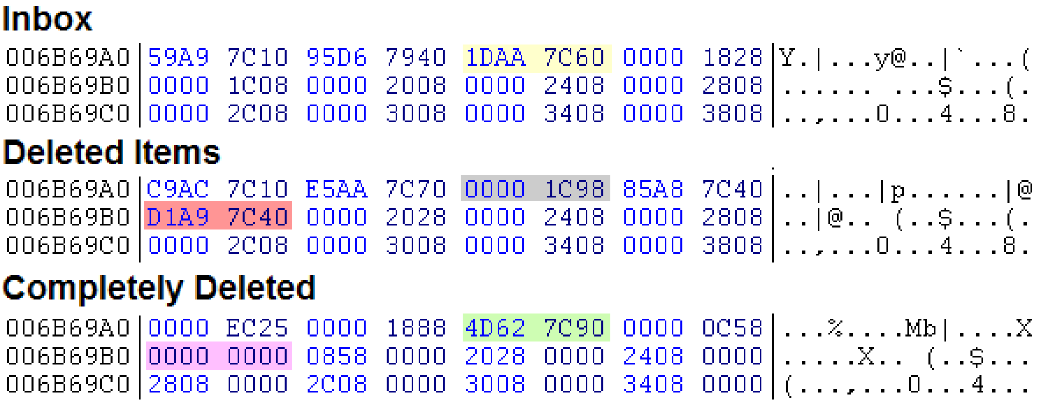
Figure 4.64: Test 8 - Zebra found.

Doing the same calculations as earlier in the disassembly we get that the index into the *Object table list* should be 0x0A and the index into the *Object table* should be 0x204. From the disassembly we know that the object store starts at virtual address 0x42000000, the *Object table list* starts at 0x42001000 and the data in the object store starts at 0x42005000. In our memory dumps of the object store however, we start counting from 0, not 0x42000000. Therefore the object page table is located at offset 0x1000 and the data starts at 0x5000. Dumping the *Object table list* from offset 0x1000 in figure 4.65 we see that this gives us the value *0x6B118C* at offset 0xA*4(marked with yellow). This is the offset to the *Object table* for our blob. We add 0x5000 to get to the start of data and get 0x6B618C. Next we add 0x204*4, which is the *Object table* index, and 0xC to skip the header of the *Object table*, which gives us 0x6B69A8. This location is where the offset



```
00001000|0000 0000 80F3 0000 6404 0200 9013 0300|........d.......
00001010|1C14 0400 A0FA 0400 283C 0600 205F 0C00|........(<.. _..
00001020|AC10 2500 CC8C 4A00 8C11 6B00 0000 0000|..%...J...k.....
00001030|0000 0000 0000 0000 0000 0000 0000 0000|................
00001040|0000 0000 0000 0000 0000 0000 0000 0000|................
```

Figure 4.65: Test 8 - Object table list.



Inbox

```
006B69A0|59A9 7C10 95D6 7940 1DAA 7C60 0000 1828|Y.|...y@..|`...(
006B69B0|0000 1C08 0000 2008 0000 2408 0000 2808|...... ...$...(.
006B69C0|0000 2C08 0000 3008 0000 3408 0000 3808|..,...0...4...8.
```

Deleted Items

```
006B69A0|C9AC 7C10 E5AA 7C70 0000 1C98 85A8 7C40|..|...|p......|@
006B69B0|D1A9 7C40 0000 2028 0000 2408 0000 2808|..|@.. (..$...(.
006B69C0|0000 2C08 0000 3008 0000 3408 0000 3808|..,...0...4...8.
```

Completely Deleted

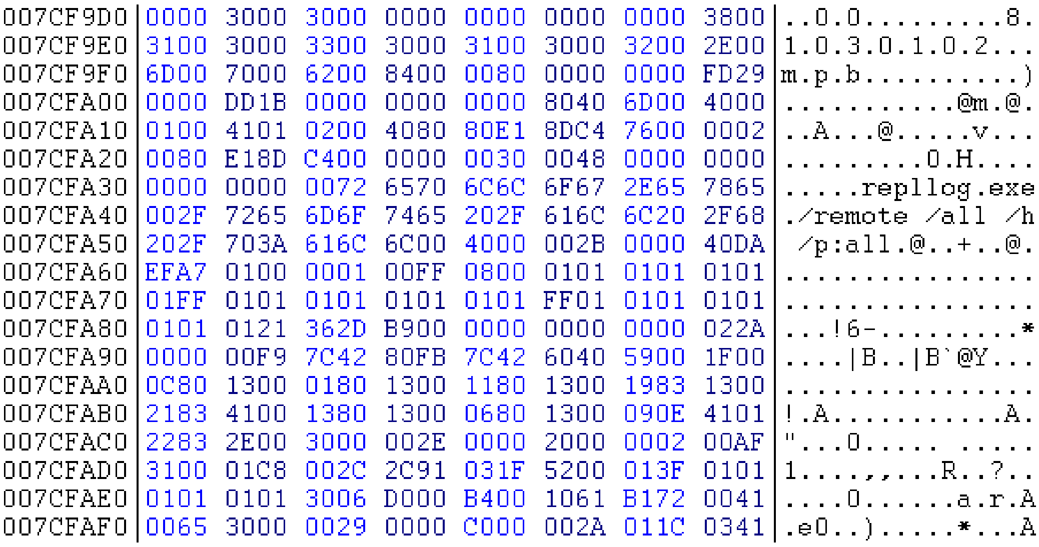
```
006B69A0|0000 EC25 0000 1888 4D62 7C90 0000 0C58|...%....Mb|....X
006B69B0|0000 0000 0858 0000 2028 0000 2408 0000|.....X.. (..$...
006B69C0|2808 0000 2C08 0000 3008 0000 3408 0000|(...,...0...4...
```

Figure 4.66: Test 8 - Object table

of the blob with id *0x2A04* is stored. As we see in figure 4.66, yellow markings, this is now *0x607CAA1D*. The algorithm from the disassembly states that the next step is to clear the 2 lowest bits and the highest byte of this. The result is 0x7CAA1C to which we finally add 0x5000 to get 0x7CFA1C, which is the offset to our blob.

Now what happens when we delete the message from ”Inbox”? We know from using the phone that the message is actually just moved to another folder called ”Deleted Items”. How is this reflected in the object store? Let’s look at the memory dump taken after the text message was moved to ”Deleted Items”. A search for ”zebra” shows us that text message now is located at offset 0x007CF9D0 instead. It has been moved and has partially written over the old version that was stored at offset 0x007CFA1C (figure 4.64). There is a new id, *0x2A06*, and doing the same calculations as the last paragraph for this new id tells us that the offset to this blob is stored in the *Object table* at location 0x6B69B0(4.66, red markings). Clearing the two lowest bits and the highest byte and adding 0x5000 again gives us the correct offset to our new blob. Notice that location 0x6B69A8, the slot for our first blob in the *Object table*, has been cleared (4.66, grey markings). This means that when a text message is moved from ”Inbox” to ”Deleted Items” the data is copied to a completely new blob and the old blob is deleted. The reference to the old deleted blob is also erased from the *Object table*.

Next we deleted the text message completely from the device. Inspecting the old offset of the text message, 0x007CF9D0 in figure 4.67, we now see that the blob has been completely overwritten by new data. The purple markings in figure 4.66 also shows us that the blob has been erased from the *Object table*. Notice also that the id *0x2A04*, at the green offset 0x006B69A8, has been reused and a pointer to a new blob has been inserted here.



```
007CF9D0|0000 3000 3000 0000 0000 0000 0000 3800|..0.0.........8.
007CF9E0|3100 3000 3300 3000 3100 3000 3200 2E00|1.0.3.0.1.0.2...
007CF9F0|6D00 7000 6200 8400 0080 0000 0000 FD29|m.p.b..........)
007CFA00|0000 DD1B 0000 0000 0000 8040 6D00 4000|...........@m.@.
007CFA10|0100 4101 0200 4080 80E1 8DC4 7600 0002|..A...@.....v...
007CFA20|0080 E18D C400 0000 0030 0048 0000 0000|.........0.H....
007CFA30|0000 0000 0072 6570 6C6C 6F67 2E65 7865|.....repllog.exe
007CFA40|002F 7265 6D6F 7465 202F 616C 6C20 2F68|./remote /all /h
007CFA50|202F 703A 616C 6C00 4000 002B 0000 40DA| /p:all.@..+..@.
007CFA60|EFA7 0100 0001 00FF 0800 0101 0101 0101|................
007CFA70|01FF 0101 0101 0101 0101 FF01 0101 0101|................
007CFA80|0101 0121 362D B900 0000 0000 0000 022A|...!6-.........*
007CFA90|0000 00F9 7C42 80FB 7C42 6040 5900 1F00|....|B..|B`@Y...
007CFAA0|0C80 1300 0180 1300 1180 1300 1983 1300|................
007CFAB0|2183 4100 1380 1300 0680 1300 090E 4101|!.A...........A.
007CFAC0|2283 2E00 3000 002E 0000 2000 0002 00AF|"...0..... .....
007CFAD0|3100 01C8 002C 2C91 031F 5200 013F 0101|1....,,...R..?..
007CFAE0|0101 0101 3006 D000 B400 1061 B172 0041|....0......a.r.A
007CFAF0|0065 3000 0029 0000 C000 002A 011C 0341|.e0..).....*...A
```

Figure 4.67: Test 8 - Objects completely deleted.

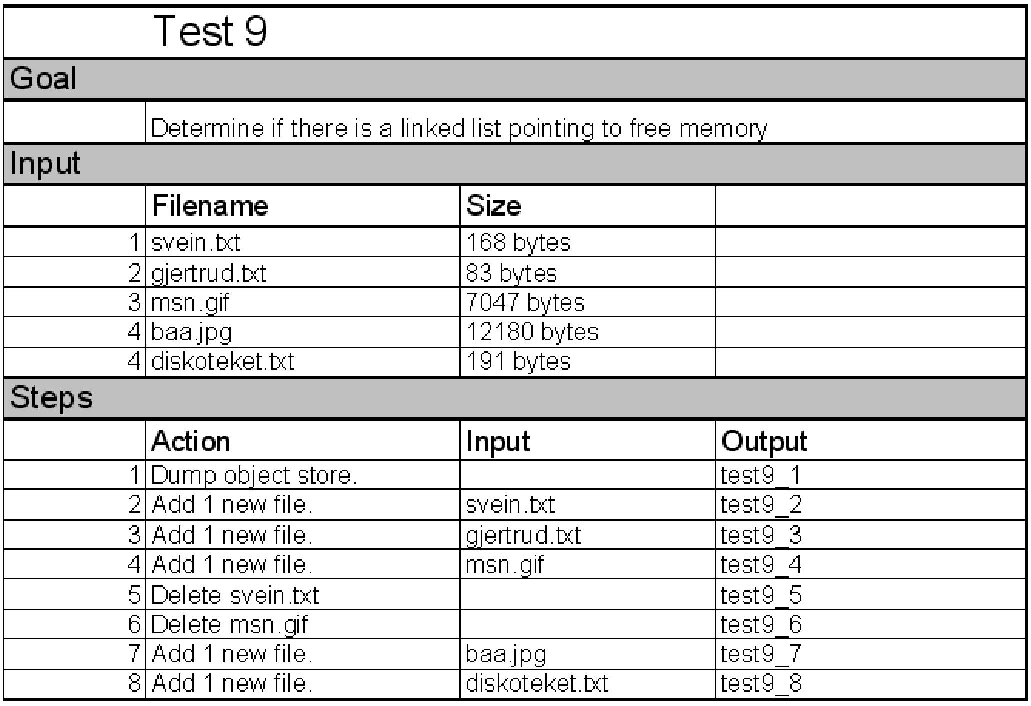
To summarize: when a blob is deleted the data of the blob can be overwritten and the pointer to the blob in the *Object table* is erased. Sooner or later the object id is reused and a new pointer is placed in the id's slot in the *Object table*.

4.2.8.2 Test 9

Create test

The memory addresses that seemed to form a linked list were suspected to be a linked list pointing to all the spaces of unused memory. Such a list could be used to find the first space of memory large enough to store new objects. If this was the case, we would have an additional way to locate areas with possibly deleted data. We wanted this test to find out if our suspicion was true by adding and deleting some files. The sizes of the files were picked to help determine this. We would first add some small files (*svein.txt* and *gjertrud.txt*), and then a fairly large file (*msn.gif*). After deleting *msn.gif* and *svein.txt*, we first added a file larger then all others (*baa.jpg*), and finally one more small file (*diskoteket.txt*). We expected *diskoteket.txt* to be placed at the position of the deleted file *msn.gif*.

Figure 4.68 shows the ninth test. We needed to make the memory dumps from *filesys.exe*'s view, in order to use the addresses as found.



Test 9

Goal

	Determine if there is a linked list pointing to free memory

Input

	Filename	Size	
1	svein.txt	168 bytes	
2	gjertrud.txt	83 bytes	
3	msn.gif	7047 bytes	
4	baa.jpg	12180 bytes	
4	diskoteket.txt	191 bytes	

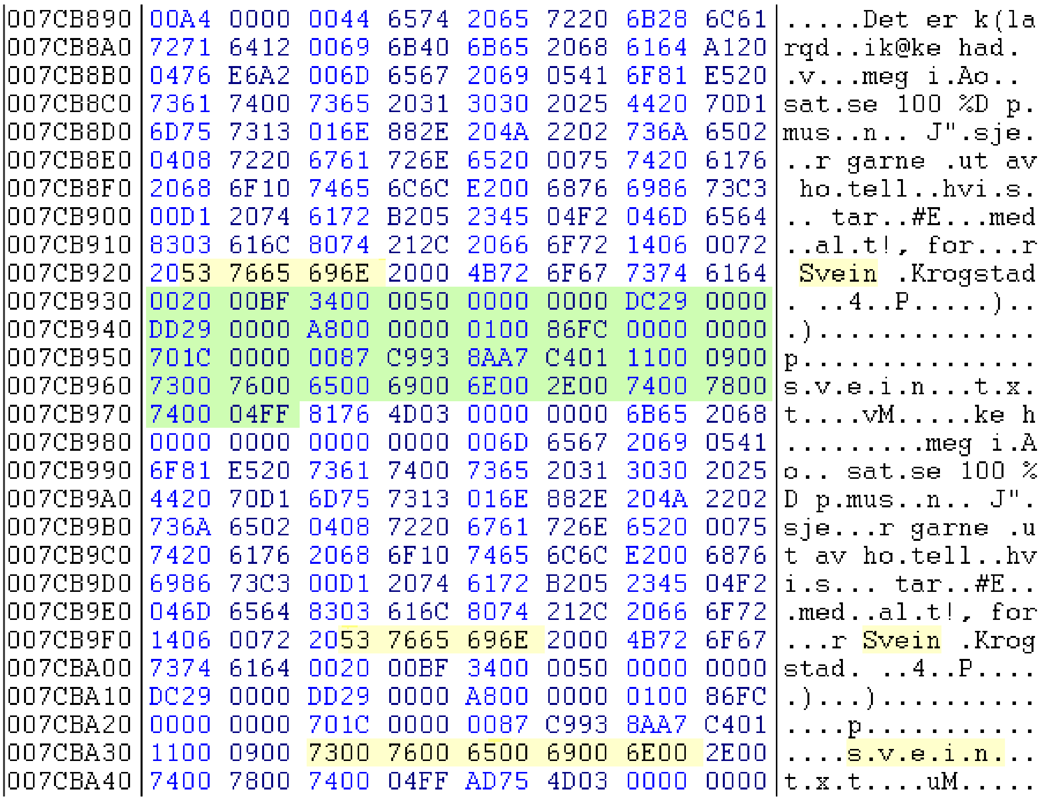
Steps

	Action	Input	Output
1	Dump object store.		test9_1
2	Add 1 new file.	svein.txt	test9_2
3	Add 1 new file.	gjertrud.txt	test9_3
4	Add 1 new file.	msn.gif	test9_4
5	Delete svein.txt		test9_5
6	Delete msn.gif		test9_6
7	Add 1 new file.	baa.jpg	test9_7
8	Add 1 new file.	diskoteket.txt	test9_8

Figure 4.68: Test 9 - Pointers to deleted space.

Extract info/Adjust PSM

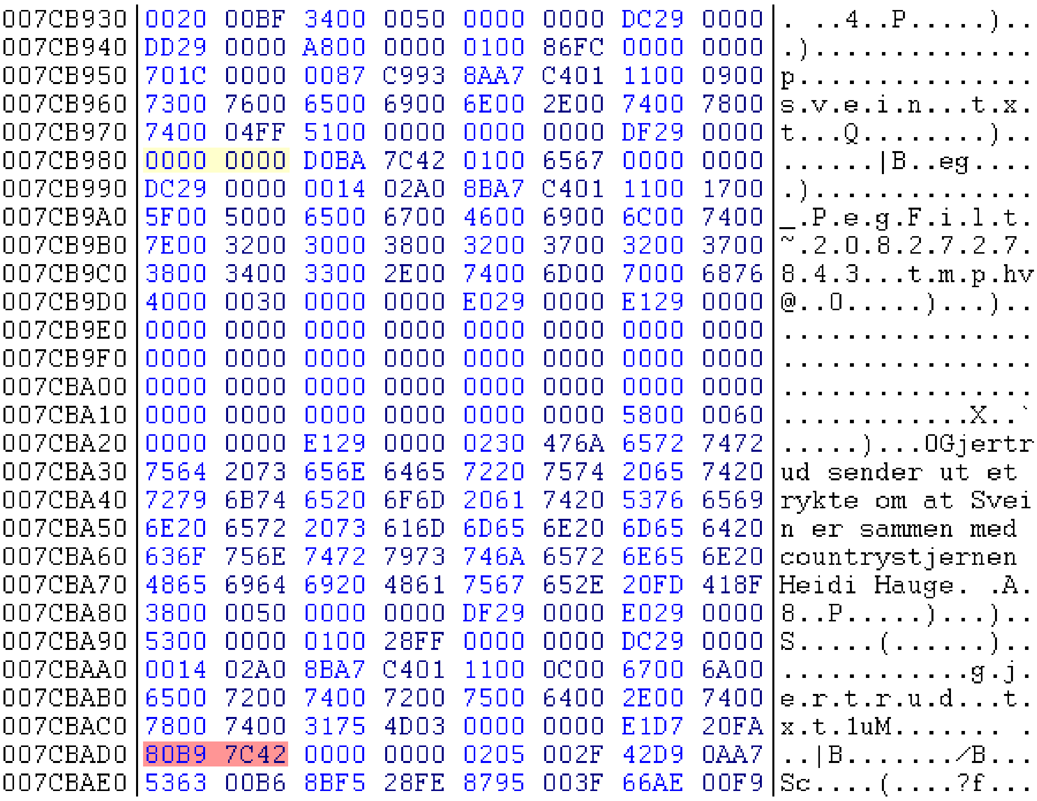
We started inspecting *test9_2*, where only *svein.txt* has been added. We located the objects related to *svein.txt*, shown in figure 4.69. The last object related to *svein.txt*, the metadata object, is highlighted in green. Assuming that no other objects has been added without our knowledge by the operating system, the free memory space should be starting right after the metadata object of *svein.txt*. We can see that the data following *svein.txt* are clearly not using any blob pattern we have seen so far, so we assume this could in fact be the beginning of the free space. We also notice that parts of the data related to *svein.txt* are found again right below the correct data objects. From this we made the assumption that the objects related to *svein.txt* may have been moved, and the remnants below are showing the previous position of *svein.txt*.



```
007CB890|00A4 0000 0044 6574 2065 7220 6B28 6C61|.....Det er k(la
007CB8A0|7271 6412 0069 6B40 6B65 2068 6164 A120|rqd..ik@ke had.
007CB8B0|0476 E6A2 006D 6567 2069 0541 6F81 E520|.v...meg i.Ao..
007CB8C0|7361 7400 7365 2031 3030 2025 4420 70D1|sat.se 100 %D p.
007CB8D0|6D75 7313 016E 882E 204A 2202 736A 6502|mus..n.. J".sje.
007CB8E0|0408 7220 6761 726E 6520 0075 7420 6176|..r garne .ut av
007CB8F0|2068 6F10 7465 6C6C E200 6876 6986 73C3| ho.tell..hvi.s.
007CB900|00D1 2074 6172 B205 2345 04F2 046D 6564|.. tar..#E...med
007CB910|8303 616C 8074 212C 2066 6F72 1406 0072|..al.t!, for...r
007CB920|2053 7665 696E 2000 4B72 6F67 7374 6164| Svein .Krogstad
007CB930|0020 00BF 3400 0050 0000 0000 DC29 0000|. ..4..P.....)..
007CB940|DD29 0000 A800 0000 0100 86FC 0000 0000|.).............
007CB950|701C 0000 0087 C993 8AA7 C401 1100 0900|p...............
007CB960|7300 7600 6500 6900 6E00 2E00 7400 7800|s.v.e.i.n...t.x.
007CB970|7400 04FF 8176 4D03 0000 0000 6B65 2068|t....vM.....ke h
007CB980|0000 0000 0000 0000 006D 6567 2069 0541|.........meg i.A
007CB990|6F81 E520 7361 7400 7365 2031 3030 2025|o.. sat.se 100 %
007CB9A0|4420 70D1 6D75 7313 016E 882E 204A 2202|D p.mus..n.. J".
007CB9B0|736A 6502 0408 7220 6761 726E 6520 0075|sje...r garne .u
007CB9C0|7420 6176 2068 6F10 7465 6C6C E200 6876|t av ho.tell..hv
007CB9D0|6986 73C3 00D1 2074 6172 B205 2345 04F2|i.s... tar..#E..
007CB9E0|046D 6564 8303 616C 8074 212C 2066 6F72|.med..al.t!, for
007CB9F0|1406 0072 2053 7665 696E 2000 4B72 6F67|...r Svein .Krog
007CBA00|7374 6164 0020 00BF 3400 0050 0000 0000|stad. ..4..P....
007CBA10|DC29 0000 DD29 0000 A800 0000 0100 86FC|.)...)..........
007CBA20|0000 0000 701C 0000 0087 C993 8AA7 C401|....p...........
007CBA30|1100 0900 7300 7600 6500 6900 6E00 2E00|....s.v.e.i.n...
007CBA40|7400 7800 7400 04FF AD75 4D03 0000 0000|t.x.t....uM.....
```

Figure 4.69: Test 9 - The objects of *svein.txt*.

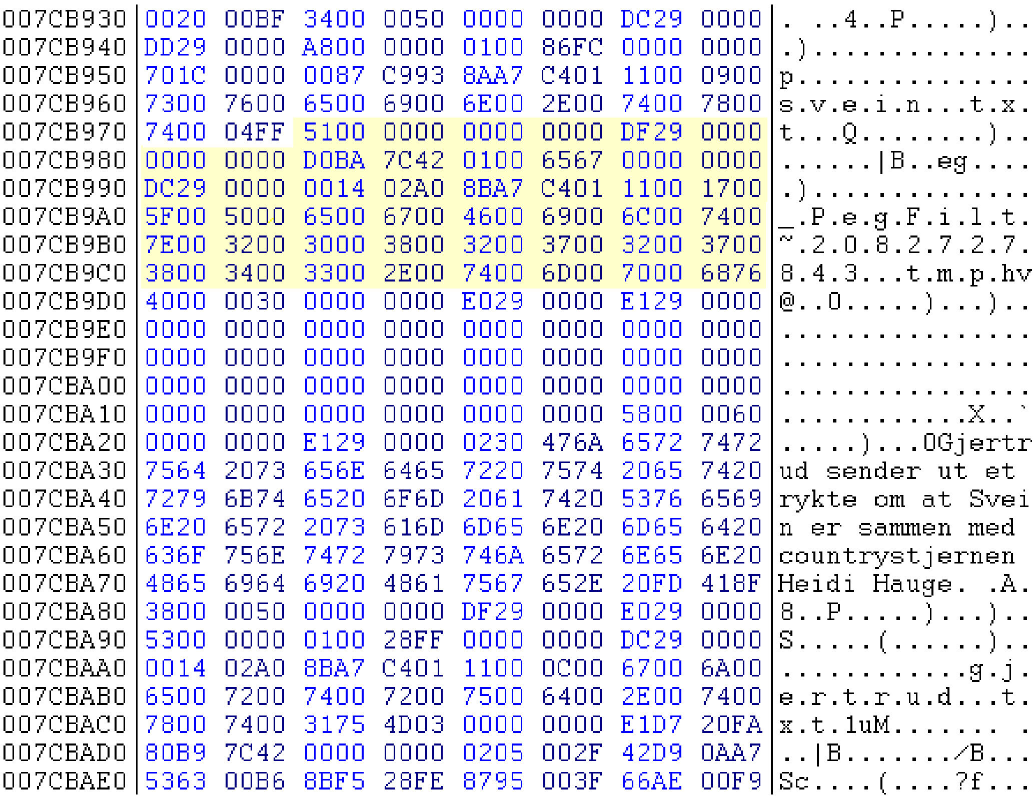
To inspect further, we looked at the file *test9_3*, where *gjertrud.txt* has been added. This time, we found a pointer following *gjertrud.txt*'s metadata object. The value of the pointer was *0x427CB980* (marked in red on figure 4.70). As before, the memory is dumped from the start of the object store(0x42000000), which means that this address is seen as 0x7CB980 on figure 4.70. If we take a look at what is found at this address, we see the value *0x00000000* (marked in yellow). But right next to it, we find the address *0x427CBAD0*. As we can see, it points back to the first pointer. This appears to be a linked list as expected, where two pointers are used to point to the *next* and *previous* node. If this is the case, we have two separate spaces of free memory on figure 4.70. In addition, these spaces need to be the only spaces of free memory, since one space has a value in the *next* field but not the *previous*, and the other space has the opposite. If this is in fact a linked list, these are the only two nodes.



```
007CB930 0020 00BF 3400 0050 0000 0000 DC29 0000 . ..4..P.....)..
007CB940 DD29 0000 A800 0000 0100 86FC 0000 0000 .)..............
007CB950 701C 0000 0087 C993 8AA7 C401 1100 0900 p...............
007CB960 7300 7600 6500 6900 6E00 2E00 7400 7800 s.v.e.i.n...t.x.
007CB970 7400 04FF 5100 0000 0000 0000 DF29 0000 t...Q.........)..
007CB980 0000 0000 D0BA 7C42 0100 6567 0000 0000 ......|B..eg....
007CB990 DC29 0000 0014 02A0 8BA7 C401 1100 1700 .)..............
007CB9A0 5F00 5000 6500 6700 4600 6900 6C00 7400 _.P.e.g.F.i.l.t.
007CB9B0 7E00 3200 3000 3800 3200 3700 3200 3700 ~.2.0.8.2.7.2.7.
007CB9C0 3800 3400 3300 2E00 7400 6D00 7000 6876 8.4.3...t.m.p.hv
007CB9D0 4000 0030 0000 0000 E029 0000 E129 0000 @..0.....)...)..
007CB9E0 0000 0000 0000 0000 0000 0000 0000 0000 ................
007CB9F0 0000 0000 0000 0000 0000 0000 0000 0000 ................
007CBA00 0000 0000 0000 0000 0000 0000 0000 0000 ................
007CBA10 0000 0000 0000 0000 0000 0000 5800 0060 ............X..`
007CBA20 0000 0000 E129 0000 0230 476A 6572 7472 .....)...0Gjertr
007CBA30 7564 2073 656E 6465 7220 7574 2065 7420 ud sender ut et
007CBA40 7279 6B74 6520 6F6D 2061 7420 5376 6569 rykte om at Svei
007CBA50 6E20 6572 2073 616D 6D65 6E20 6D65 6420 n er sammen med
007CBA60 636F 756E 7472 7973 746A 6572 6E65 6E20 countrystjernen
007CBA70 4865 6964 6920 4861 7567 652E 20FD 418F Heidi Hauge. .A.
007CBA80 3800 0050 0000 0000 DF29 0000 E029 0000 8..P.....)...)..
007CBA90 5300 0000 0100 28FF 0000 0000 DC29 0000 S.....(......)..
007CBAA0 0014 02A0 8BA7 C401 1100 0C00 6700 6A00 ............g.j.
007CBAB0 6500 7200 7400 7200 7500 6400 2E00 7400 e.r.t.r.u.d...t.
007CBAC0 7800 7400 3175 4D03 0000 0000 E1D7 20FA x.t.1uM....... .
007CBAD0 80B9 7C42 0000 0000 0205 002F 42D9 0AA7 ..|B......./B...
007CBAE0 5363 00B6 8BF5 28FE 8795 003F 66AE 00F9 Sc....(....?f...
```

Figure 4.70: Test 9 - Locating *gjertrud.txt*.

So how do we determine if this is the case? If we take a look at the data found at this second space(marked yellow in figure 4.71), we see that



```
007CB930|0020 00BF 3400 0050 0000 0000 DC29 0000|. ..4..P.....)..
007CB940|DD29 0000 A800 0000 0100 86FC 0000 0000|.).............
007CB950|701C 0000 0087 C993 8AA7 C401 1100 0900|p...............
007CB960|7300 7600 6500 6900 6E00 2E00 7400 7800|s.v.e.i.n...t.x.
007CB970|7400 04FF 5100 0000 0000 0000 DF29 0000|t...Q.......)..
007CB980|0000 0000 D0BA 7C42 0100 6567 0000 0000|......|B..eg....
007CB990|DC29 0000 0014 02A0 8BA7 C401 1100 1700|.)..............
007CB9A0|5F00 5000 6500 6700 4600 6900 6C00 7400|_.P.e.g.F.i.l.t.
007CB9B0|7E00 3200 3000 3800 3200 3700 3200 3700|~.2.0.8.2.7.2.7.
007CB9C0|3800 3400 3300 2E00 7400 6D00 7000 6876|8.4.3...t.m.p.hv
007CB9D0|4000 0030 0000 0000 E029 0000 E129 0000|@..0.....)...)..
007CB9E0|0000 0000 0000 0000 0000 0000 0000 0000|................
007CB9F0|0000 0000 0000 0000 0000 0000 0000 0000|................
007CBA00|0000 0000 0000 0000 0000 0000 0000 0000|................
007CBA10|0000 0000 0000 0000 0000 0000 5800 0060|............X..`
007CBA20|0000 0000 E129 0000 0230 476A 6572 7472|.....)...0Gjertr
007CBA30|7564 2073 656E 6465 7220 7574 2065 7420|ud sender ut et
007CBA40|7279 6B74 6520 6F6D 2061 7420 5376 6569|rykte om at Svei
007CBA50|6E20 6572 2073 616D 6D65 6E20 6D65 6420|n er sammen med
007CBA60|636F 756E 7472 7973 746A 6572 6E65 6E20|countrystjernen
007CBA70|4865 6964 6920 4861 7567 652E 20FD 418F|Heidi Hauge. .A.
007CBA80|3800 0050 0000 0000 DF29 0000 E029 0000|8..P.....)...)..
007CBA90|5300 0000 0100 28FF 0000 0000 DC29 0000|S.....(......)..
007CBAA0|0014 02A0 8BA7 C401 1100 0C00 6700 6A00|............g.j.
007CBAB0|6500 7200 7400 7200 7500 6400 2E00 7400|e.r.t.r.u.d...t.
007CBAC0|7800 7400 3175 4D03 0000 0000 E1D7 20FA|x.t.1uM....... .
007CBAD0|80B9 7C42 0000 0000 0205 002F 42D9 0AA7|..|B......./B...
007CBAE0|5363 00B6 8BF5 28FE 8795 003F 66AE 00F9|Sc....(....?f...
```

Figure 4.71: Test 9 - A temporary object.

it is found right beneath *svein.txt*'s metadata object. This is the area we assumed to be the start of the free space in our first inspection of *test9_2*. Since the three objects related to *gjertrud.txt* has not been placed starting from this position, some other object must have been placed there if our assumptions are correct. This object would have to have been created by the operating system or some other process. Looking at the data following the metadata of *svein.txt*, the size field of this object would then be 0x51. As we have seen earlier, an odd value for the size appears to indicate that the object is deleted, which in this case needs to be true if our theory about the pointers is correct.

If we look at the id field of the deleted object, which is still present, the id of this object would have been *0x000029DF*. This is in fact the id following those used by *svein.txt*. From this we can see that some object has been created between the insertion of *svein.txt* and *gjertrud.txt*. *gjertrud.txt*'s metadata object also uses this id, so the object must also have been deleted before the insertion of *gjertrud.txt*.

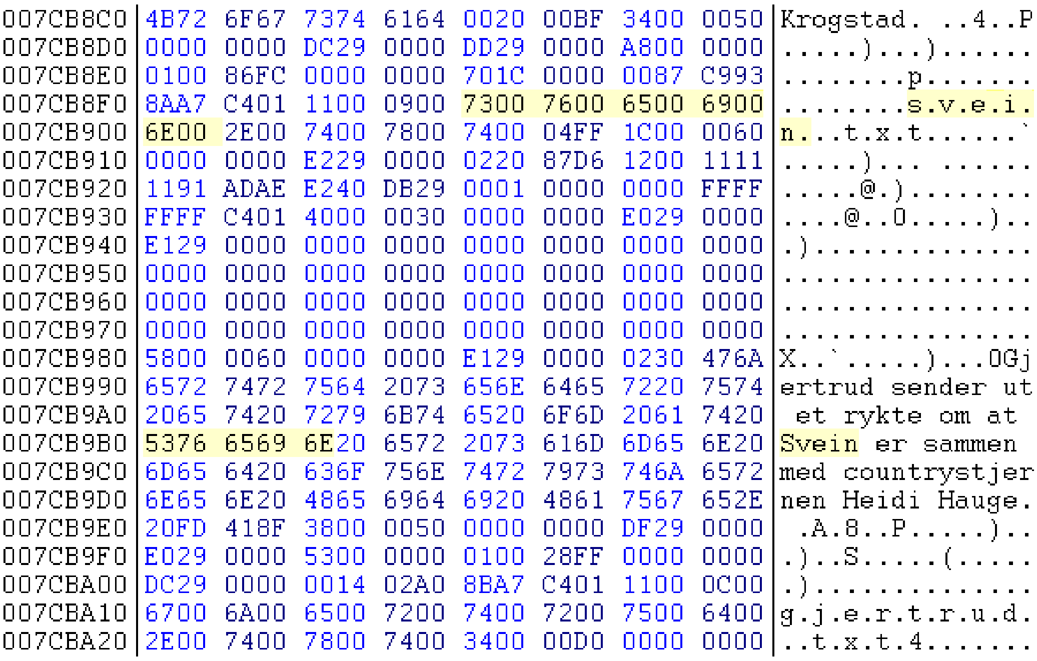
If we look at what appears to be the name of the file, its file extension shows that it was a temporary file (*.tmp*). This backs up the idea that the object has been deleted, and that the space is now free to be reused. We find that the size of this free space is 0x50, and if we ignore the 1 in the size field, which we assume has to do with the object being deleted, this equals the value found there.

Having seen that the size was found in the first bytes of the free space, we took another look at the first space following *gjertrud.txt*. The value found to occupy the first two dwords there is *0x034D7531*. Ignoring the 1 in the LSB as before, we find this value to equal 55407920 in decimal. This seemed like a reasonable value for the free space on the phone, as it translates to approximately 52.8 MB. When we inspected the phones settings, we actually found this to be exactly the amount of memory said to be free. This was a strong indicator telling us that our theory about these pointers was correct. We took another look at *test9_2* that was inspected previously (figure 4.69), and found that in the positions where the pointers should be located (0x7CB980 and 0x7CB984), we saw the value *0x00000000* at both positions. This seems reasonable, meaning that there is only one large space of free memory, hence no other spaces to point to.

The next file inserted in the test was *msn.gif*. The results from this insertion could be inspected in *test9_4*. We located *svein.txt* (figure 4.72), and found that some changes had been made. The object store appeared to have been reorganized, and the deleted temporary file was no longer present. When we located *msn.gif* (metadata object shown in green on figure 4.73), we found that both pointers were now indeed given the value *0x00000000*, indicating that a reorganization had been performed. This reorganization makes sense, in order to prevent the object store from being too fragmented.

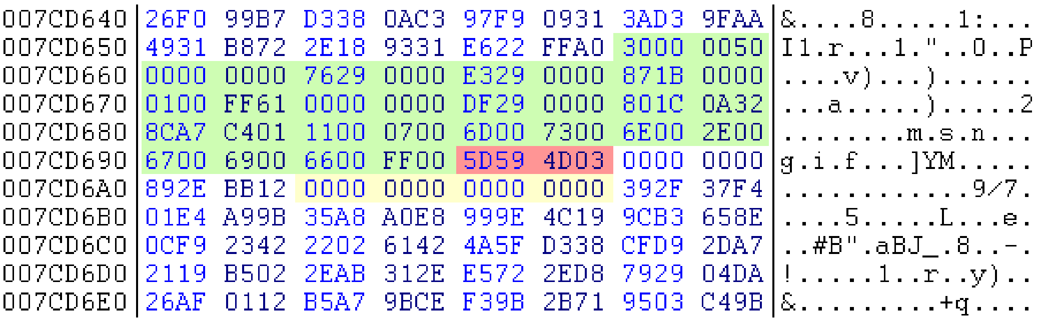
The rest of the test files showed the same behavior we had seen so far. The linked list kept pointing to the free spaces, the size field was updated, and from time to time the object store was reorganized. In fact, we never saw more than two spaces of free memory before a new reorganization was performed. Because of this reorganization, *diskoteket.txt* had been placed last in the main free space area just as the other files.

The test had confirmed our suspicion about the pointers.



```
007CB8C0|4B72 6F67 7374 6164 0020 00BF 3400 0050|Krogstad. ..4..P
007CB8D0|0000 0000 DC29 0000 DD29 0000 A800 0000|.....)...)......
007CB8E0|0100 86FC 0000 0000 701C 0000 0087 C993|........p.......
007CB8F0|8AA7 C401 1100 0900 7300 7600 6500 6900|........s.v.e.i.
007CB900|6E00 2E00 7400 7800 7400 04FF 1C00 0060|n...t.x.t......`
007CB910|0000 0000 E229 0000 0220 87D6 1200 1111|.....)... ......
007CB920|1191 ADAE E240 DB29 0001 0000 0000 FFFF|.....@.)........
007CB930|FFFF C401 4000 0030 0000 0000 E029 0000|....@..0.....)..
007CB940|E129 0000 0000 0000 0000 0000 0000 0000|.)..............
007CB950|0000 0000 0000 0000 0000 0000 0000 0000|................
007CB960|0000 0000 0000 0000 0000 0000 0000 0000|................
007CB970|0000 0000 0000 0000 0000 0000 0000 0000|................
007CB980|5800 0060 0000 0000 E129 0000 0230 476A|X..`.....)...0Gj
007CB990|6572 7472 7564 2073 656E 6465 7220 7574|ertrud sender ut
007CB9A0|2065 7420 7279 6B74 6520 6F6D 2061 7420| et rykte om at
007CB9B0|5376 6569 6E20 6572 2073 616D 6D65 6E20|Svein er sammen
007CB9C0|6D65 6420 636F 756E 7472 7973 746A 6572|med countrystjer
007CB9D0|6E65 6E20 4865 6964 6920 4861 7567 652E|nen Heidi Hauge.
007CB9E0|20FD 418F 3800 0050 0000 0000 DF29 0000| .A.8..P.....)..
007CB9F0|E029 0000 5300 0000 0100 28FF 0000 0000|.)..S.....(.....
007CBA00|DC29 0000 0014 02A0 8BA7 C401 1100 0C00|.)..............
007CBA10|6700 6A00 6500 7200 7400 7200 7500 6400|g.j.e.r.t.r.u.d.
007CBA20|2E00 7400 7800 7400 3400 00D0 0000 0000|..t.x.t.4.......
```

Figure 4.72: Test 9 - Object store reorganized



```
007CD640|26F0 99B7 D338 0AC3 97F9 0931 3AD3 9FAA|&....8.....1:...
007CD650|4931 B872 2E18 9331 E622 FFA0 3000 0050|I1.r...1."..0..P
007CD660|0000 0000 7629 0000 E329 0000 871B 0000|....v)...)......
007CD670|0100 FF61 0000 0000 DF29 0000 801C 0A32|...a.....).....2
007CD680|8CA7 C401 1100 0700 6D00 7300 6E00 2E00|........m.s.n...
007CD690|6700 6900 6600 FF00 5D59 4D03 0000 0000|g.i.f...]YM.....
007CD6A0|892E BB12 0000 0000 0000 0000 392F 37F4|............9/7.
007CD6B0|01E4 A99B 35A8 A0E8 999E 4C19 9CB3 658E|....5.....L...e.
007CD6C0|0CF9 2342 2202 6142 4A5F D338 CFD9 2DA7|..#B".aBJ_.8..-.
007CD6D0|2119 B502 2EAB 312E E572 2ED8 7929 04DA|!.....1..r..y)..
007CD6E0|26AF 0112 B5A7 9BCE F39B 2B71 9503 C49B|&.........+q....
```

Figure 4.73: Test 9 - Locating *msn.gif*.

Transform to PIM

Figure 4.74 summarizes what we found. At the initial state shown in a), the object store has been organized in order to have all unused memory gathered into one big space. When an object is deleted, a linked list connects the spaces of free memory as shown in b). Whenever the object store is reorganized, the free space of the object store goes back to the state shown in a).

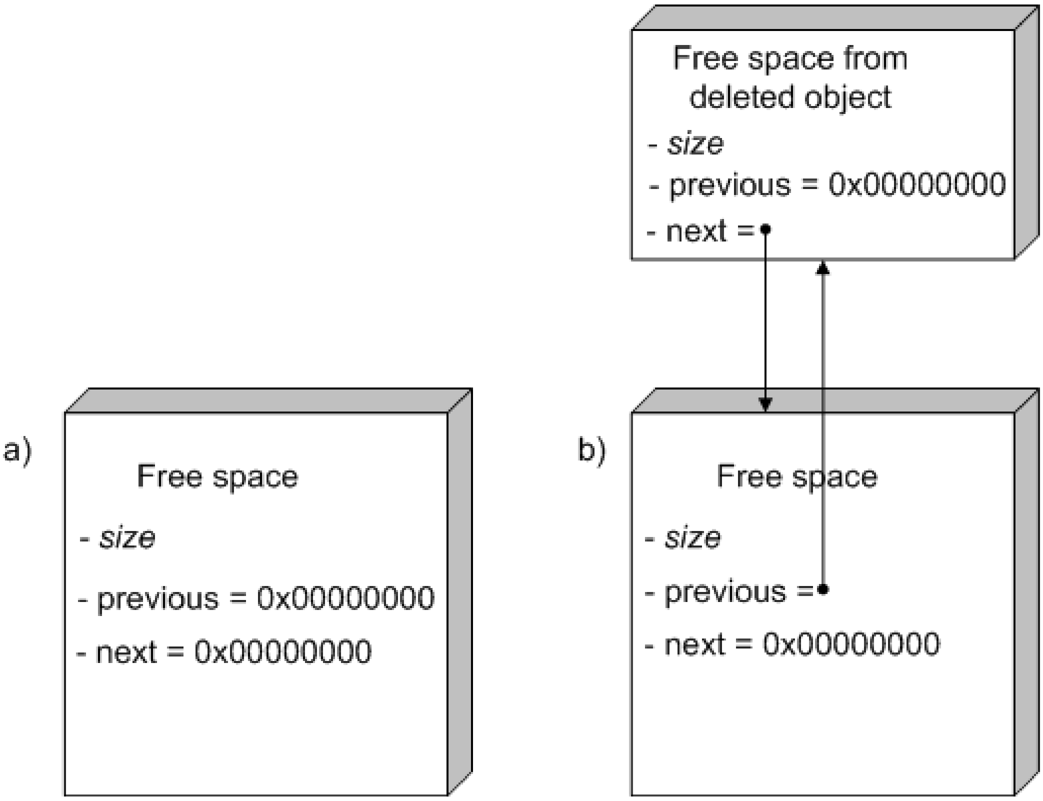


Figure 4.74: Test 9 - Linked list pointing to free space.

4.2.9 Evaluate PIM/CIM Consistency

During this loops evaluation of the consistency between our PIM and CIM, we found our reverse engineering to be adequate. We had covered both of the two major parts of the CIM: the object allocation table and the objects storage format. The success criteria appeared to be met, so we decided to enter the validation phase in order to test if this was in fact true.

4.3 Validation Phase

According to our methodology, the validation should be performed by utilizing what is called *reverse reverse engineering*. This meant that we had to enhance our validation tool (BlobExtractor) in order to test all the success criteria described in section 4.1.

The second success criterion is: ”Understand the format used to store object, in order to extract data and attributes.” As we have performed our tests, we have steadily been building a list of structure definitions on the various blobs found in the object store. The last version of these structures was presented in listing 4.10. In order to test this criterion we implemented these structures and the logic to extract them from a given location offset of the memory dump in our validation tool. With this we could, if we had the starting location of a blob, extract it directly from the memory dump without utilizing the Windows CE API at all. For most of the blob types we could also interpret the data in the blobs and differentiate between data and metadata.

The problem now was to find the starting location of the blobs. Our first approach was to scan through the file sequentially byte by byte. We combined this with several heuristics to detect the start of a new blob. The heuristics were:

1. Size check. The size of the blob should be divisible by 2 and over 0.
2. ID check. The id of the blob should be over 0 and under 30000. (This was the largest value we had seen as an id and then some.)
3. Type check. The type of the blob should not be 0 and should only use the highest nibble of the 2 byte wide type field.
4. Flags check. The flag field should be set to 0.

As we saw in test 6 this worked pretty well, but there were several downsides with this approach. First of all, it is time consuming. Scanning through the memory dumps and applying the heuristics for every byte took up to 30 minutes, which is a lot considering that they are only 64 MB in size. Second, and the worst problem, is that we cannot guarantee that we have found all the blobs because our heuristics may be wrong. This leads us to success criterion number 3.

According to this criterion we had to ”Make sure that we have covered the entire object store”. In order to do this we needed a much better approach to finding blobs than simple heuristics. Thanks to the disassembly in test 7 and further testing in test 8 we found the coveted lookup table that maps between object ids and offsets to actual blobs in the object store. Combining the fact that every object in the object store must have an id and the fact that we have the table that maps all ids to blobs, we now had a way to make sure we knew where all blobs in the object store was located. We implemented code to extract the lookup table from the memory dumps and used this table to find all the blobs which could then be extracted.

Now that we know where all the blobs in the object store are, we also know where they are not. We use this knowledge to figure out where all the possible locations for deleted data. They are basically any place there is data which is not connected to a blob that can be found through the lookup table. This fulfills the first criterion: ”Be able to distinguish between deleted and non-deleted data.” In our validation tool we have implemented code that recognizes these locations. In addition we have also implemented code that utilizes the other way to locate deleted data. This method is mentioned in test 9. It basically entails following a double linked list of freed memory (which can contain deleted data) that the object store maintains.

By enhancing our validation tool to use all the information in the PIM we have successfully used *reverse reverse engineering* to validate that our model has produced enough information to be able to say that all the specific success criteria for our object store analysis has been fulfilled. To summarize:

1. **Be able to distinguish between deleted and non-deleted data.** This came as a result of fulfilling criterion 2 and 3. We have two ways of fulfilling this criterion. 1) All data that is not pointed to by the object store is possibly deleted. We know everything the object store points to and thus can deduct what it doesn’t point to. 2) Follow the freed memory double linked list.

2. **Understand the format used to store objects, in order to extract data and attributes.** The black box tests enabled us to extract this information.

3. **Make sure that we have covered the entire object store.** We found the lookup table used by the object store by disassembling parts of

the operating system. This table contains pointer to all valid blobs and thus all valid data in the object store.

Chapter 5

Discussion

This project has had two parallel objectives. On one hand we wanted to analyze Windows Mobile's object store in order to improve the tools used for forensic analysis of evidence from devices running Windows Mobile. At the same time we were interested in the establishment of a general methodology for forensic analysis of an unknown embedded device. The following discussion will consider each of these objectives in separate sections.

5.1 The Object Store

During the final validation phase of the methodology (4.3) we concluded that all the success criteria for the analysis had been met. By extensive experimentation we had gained enough knowledge about the object store to extract the information we wanted. We had found a way to locate objects from the identifiers. As we also knew the value space of the identifiers, we could either loop through all identifier values, or we could more elegantly loop through the tables containing identifiers of all active objects. This made us capable of locating all active objects. By subtracting the memory occupied by these objects from the rest of the object store memory, we were left with only the parts of memory possibly containing deleted information. In addition, we had found a second way to find the deleted parts of memory. We located a linked list structure pointing to all areas of free memory, which equals the areas where deleted data can be found. This gave us two different methods to verify the results.

Of the objects themselves, we had almost complete knowledge of the attributes, and we were able to extract the data from the objects. The few fields missing were left because they did not appear to contain valuable information for our purpose, and hence were not worth spending time on. This is the kind of choice any reverse-engineer will have to make, unless interested in complete knowledge of the entire device.

The most important classes of BlobExtractor is listed in appendix B. These are the classes that do all the heavy work involved with parsing a memory dump and extracting the objects and free space areas from the objects store. The complete program is much larger and also includes code for visualizing the content of the object store in different ways. Its source code is included in the zip file accompanying this thesis.

Appendix B also include the extensions made to the Judas Forensic Tool during our analysis. The complete tool is included in the zip file.

5.1.1 Future Work

Our work has given future tools the ability to extract a lot more accurate information about Windows Mobile phones content, and even accurate information concerning remaining parts of deleted information on the phones. [6] discussed existing tools, and showed that they all lacked the ability to extract deleted information from such phones. With the results from this project, a tool can now be made that could easily outperform the existing tools, giving a detailed view of all remnants of deleted data on the phone. This could have an important forensic value.

5.2 The Methodology

Our decision to sketch a methodology before moving on to the actual analysis of the device was problematic, since we were not experienced with reverse engineering. We still found this to be the best solution in order to get a good basis for the discussion of a general methodology that was applicable to forensic analysis of other unknown embedded devices. Instead of just attacking the object store immediately and at the end summarize our experiences from this process; we believe that our approach made us more qualified to discuss methodology.

It was natural to base the methodology on previous work by other research teams. This would make use of years of research, and by using concepts familiar to the reverse-engineer, the methodology framework were more likely to be well understood and accepted.

The methodology sketched in section 3.3.2 guided us through our reverse engineering process, and we found it to be a good support throughout. The methodology itself will never solve the problem. Reverse engineering such a system is hard, and we often found ourselves stuck in situations where solutions seemed far away. But in such situations it is a good support to have a well-defined methodology to lean on. The methodology helped us maximize our focus on the problem areas, instead of just fumbling around in the dark.

Of particular value was our immediate focus on building a knowledge base, continuously updated with all the information we found to be relevant for our analysis. There were many situations during the analysis where progress was made because we somehow recognized data we had documented in the knowledge base, and thereby made connections to define new knowledge. Building such a knowledge base should have a high priority in any reverse engineering process.

The methodology aimed at being both general enough to be applicable with other devices, and at the same time be specific enough to be useful in the process. We solved this by separating the specific choice of strategy from the more general aspects of the process. The strategies are chosen dynamically to adapt to the current situation. Over time, experience and learning from other projects will help the reverse engineer choose accurate strategies.

Model-Driven Reverse Engineering was chosen as basis for the general procedure. This supported the idea of documenting all information found in a structured way, by using models to represent information. The models represented different abstraction levels, which we believe served as a good way to both drive the process and to validate the adequacy of the results. As defining a standard for the representation was outside the scope of this project, and a big task in itself, we did not make any attempt at this. This meant that we had to choose the representation we found to be the most suitable on the fly. Though this was inevitable, it meant that we could not test the effects of using a standard modeling language.

By separating information on different abstraction levels, the top-level models representing the CIM can be re-used in similar projects. This may be a good help for a reverse-engineer, who can search previous projects in order to find top-level models for similar projects that could either be used directly, or slightly modified. This way, experience from previous projects are used directly in the new project.

As we were two reverse-engineers working together on this project, we also got a glimpse of how the methodology can be used to improve cooperation. The models were used as basis for our discussions throughout the process, and we believe they played an important role in order to prevent misunderstandings and to make sure we both dragged in the same direction.

With a defined methodology you get another possible benefit. Courses and training material could be made, aimed at improving the use of the methodology. This could help improve both quality and efficiency.

5.2.1 Future Work

Whether or not the methodology is a good sketch for a general approach is hard to determine from a single case study. Though our experiences support the methodology, additional cases should be targeted with the methodology in order to establish its relevance.

We have discussed the importance of the knowledge base, and mentioned how the representation of information is of great importance. In order to get advanced tools more involved in the process, this should be targeted in future work. In [14], Rugaber discuss representation when reverse engineering programs and specifies several requirements. We believe that his thoughts on this serves as a good foundation for future work on this task:

- **Requirements Related to the Information Content of the Representation:** The representation must be able to contain a variety of types of information. These include informal rationale and annotations, program segments, pointers to other documentation, and application descriptions. Most importantly, it must be able to represent the organization of the program in terms of detected abstractions. In fact, the reverse engineer constructs a complex information structure that describes the organization of the program and the interrelationships of its pieces. There must be a place in the representation to hold observations made by the reverse engineer during his process.

- **Requirements Related to the Relationships Among the Data Being Represented:** The representation is constructed incrementally by the reverse engineer. It must allow an observation concerning a section of code to be associated both with related sections of code and with the overall functional description being constructed. This includes both hierarchical connections among abstractions and heterarchical (cross-reference) associations. Finally, the representation should support instances where a section of code contains several components interleaved together.

- **Requirements Related to How the Representation is Constructed:** The representation needs to be easy to construct incrementally, both computationally and from a user interface point of view. Additionally, it should be language independent in the sense that it can be used during the reverse engineering of programs written in a variety of languages and programming paradigms.

- **Requirements Related to How the Representation is Used:** The representation must be formal enough to support automatic manipulation. For example, after a program has been reverse engineered into the representation, it should be possible to apply tools to adapt segments for reuse. This process is called *transformational programming*, and a variety of such transformations exist.

- **Requirements Related to How the Representation is Accessed and Viewed:** A predominant use of the representation will be to facilitate program browsing. That is, a maintenance programmer desiring to fix a bug or make an enhancement needs to be able to pursue the information structure either to answer specific questions (which functions call a given function), obtain an architectural overview (in graphical form), or locate a specific section of the code (where are all of the statements that could affect the final value of a given output variable). The representation must, at the same time, be independent of any particular design method or notation and be capable of generating information in any of a variety of formats.

Chapter 6

Conclusion

The steadily increasing number of new digital device models is putting a strain on law enforcement agencies' ability to acquire evidence in criminal cases. The devices are often highly advanced and poorly documented which makes it very time consuming to analyze them completely. One of the more dire consequences of this is that there might be under development a new kind of digital free haven for criminals. They can utilize the new abilities of the devices to optimize their nefarious purposes safe in the knowledge that police will not be able to catch them doing it.

The solution to this problem lays in making sure the analysis of the new devices can be done more efficiently and well-structured than before. In our thesis we have tried to do just this. First we constructed a methodology for doing such analysis. This was based on prior research and assumptions we made of the context of our task. Then we tested our methodology on a specific analysis of a Qtek Windows Mobile phone. This device was chosen because the direct results of this analysis can be applied to any device utilizing this operating system, which is predicted by Gartner to be around 20 million in 2008. Our analysis utilized with great effect the well known techniques of black box testing and disassembly. One of the most important lessons learned has been that understanding every single detail of a unknown device is extremely time consuming and in most cases completely unnecessary. One should take great care in defining, in advance, when one knows enough to get the job done. This is essential in figuring out when to decide enough is enough. Our resulting developed tool, BlobExtractor, can be used to extract most data from any device running Windows Mobile or Windows CE. Some details remain, but the time and cost involved in understanding them completely was found to be too high.

While the detailed analysis of Windows Mobile is interesting, the other important part of our task was to evaluate whether our methodology and model was general enough or could be generalized to work with other types of digital devices. Based on our own success in utilizing it, we believe that this method could very well be suitable for others. Its iterative approach of analyzing low level details and abstracting them to higher levels of documentation to further understanding lends itself to widespread use. The exact techniques used to do the low level analysis are device dependant and not specified in the methodology. It is also important to note that the methodology takes into account the importance of documenting all knowledge firmly in a knowledge base, to enable better cooperation between several reverse engineers. In the future, reverse engineering task will be so complex that one cannot expect single individuals to perform them well in short amounts of time.

We hope that our thesis is a contribution in keeping the tidal wave of new digital devices from swamping the ability of law enforcement agencies to do their jobs.

Bibliography

[1] Gartner: *Forecast: Mobile Terminals, Worldwide, 2000-2009*, 2005

[2] Microsoft Corp.: *Microsoft Shared Source Initiative*, http://www.microsoft.com/resources/sharedsource/ 2006

[3] Noblett, Pollitt, Presley: *Recovering and Examining Computer Forensic Evidence*, Forensic Science Communications, Volume 2, Number 4, October 2000.

[4] Chikofsky, Cross: *Reverse Engineering and Design Recovery: A Taxonomy*, IEEE Software, pp 13-17, IEEE Computer Society, January 1990.

[5] Biggerstaff : *Design Recovery for Maintenance and Reuse*, Computer, pp 36-49, July 1989

[6] Eide: *Mobile Forensics*, http://www.idi.ntnu.no/ jarlee/m_forensics.pdf, 2005

[7] Hengeveld: *Itsutil*, http://www.xs4all.nl/ itsme/projects/xda/-tools.html, 2005

[8] Stirewalt: *Model-Driven Reverse Engineering*, Georgia Institute of Technology, 2004

[9] Miller, Mukerji: *MDA Guide Version 1.0.1*, Object Management Group, 2003

[10] Fowler, Scott: *UML Destilled. Second Edition. A Brief User Guide to the Standard Object Modeling Language.*, Addison Wesley Longman Inc., 1999

[11] Raymond: *Reference Model of Open Distributed Processing (RM-ODP): Introduction*, University of Queensland

[12] Kamper, Rugaber: *A Reverse Engineering Methodology For Data Processing Applications*, Georgia Institute of Technology, 1990

[13] Jackson: *A System Development Method*, http://www.ferg.org/papers/jackson–a_system_development_method.pdf, 1981

[14] Rugaber: *Program Comprehension For Reverse Engineering*, Georgia Institute of Technology, 1992

[15] Qtek: wwww.qtek.nu

[16] Microsoft Corp.: *Microsoft Developers Network*, http://msdn.microsoft.com/

[17] Berkeley Wireless Research Center :*ARM instruction Set Quick Reference*, http://bwrc.eecs.berkeley.edu/Research/Pico_Radio/Test_Bed/Hardware/Documentation/ARM/ARM_Instruction_Set.pdf 2005

[18] ARM Ltd.: *ARM architecture*, http://www.arm.com/miscPDFs/8031.pdf 2005

[19] Wikipedia: *"Black box testing — Wikipedia The Free Encyclopedia"*, http://en.wikipedia.org/w/index.php?title=Black_box_testing, 2006

[20] Microsoft Corp.: *Windows CE 3.0 Features*, http://msdn.microsoft.com/embedded/prevver/ce3/feature/

[21] Grattan, Brain: *Windows CE 3.0 Application Programming*, Prentice Hall PTR, 2001

[22] VCOM Company: *Sourcer*, http://www.partitioncommander.com/company

[23] Cronos/Terminus One: *BORG*, http://www.caesum.com

[24] Jimnez: *BDASM*, http://www.bdasm.com/

[25] URSoft: *W32Dasm*, http://www.ursoftware.com

[26] unknown: *PEDasm*, http://www.geocities.com/SiliconValley/Lab/6307/PEDasm.htm

[27] DoggySoft: *Diss*, http://www.doggysoft.co.uk

[28] IOTA: *Disassembler*, http://www.iota.demon.co.uk/psion/disassembler/disassembler.html

[29] Delosoft: *ARMDis*, http://www.delosoft.com/

[30] DataRescue : *DataRescue*, http://www.datarescue.com/

[31] Loh: *Windows CE Base Team Blog: Inside Windows CE API Calls*, http://blogs.msdn.com/ce_base/archive/2006/02/02/Inside_Windows_CE_API_Calls.aspx, 2006

[32] XDA Developers: *XDA Developers*, http://xda-developers.com, 2006

Appendix A

Qtek S110 basics

Platform	Dimension
• PDA form factor integrated GSM/GPRS, Bluetooth, and 1.3 mega-pixel camera • Microsoft Windows PocketPC Phone Second Edition	• 108.2 mm(L) x 58 mm(W) x 18.2 mm(T) • 150 g with battery pack
Processor/Chipset	**Memory**
• Intel Bulverde 416 MHz	• ROM: 64 MB • RAM: 128 MB SDRAM
LCD Module	**GSM/GPRS Function**
• 2.8″ 240 x 320 dots resolution • 64K-color TFT Transflective LCD with white LED back light • Sensitive Touch Screen	• Internal antenna • Tri-Band (900/1800/1900MHz) • GPRS Functionality • Multi-slot standard class 10 • SIM • 3V operation • SIM Application Toolkit release 96 • Over the air programming

Notification	**Audio**
• Vibration for notification • Notification by sound, message on the display	• Built-in Microphone • Receiver • Loud speaker for Hands-Free supported
Camera	**Interface**
• Colors CMOS 1.3 mega-pixel camera with dust-proof cover • Preview Mirror	• Infrared IrDA SIR • SDIO/MMC card slot with door (top) • 3V SIM card • 2.5 D stereo audio jack

Table A.1: Qtek S110 specification

Appendix B

Source Code

B.1 BlobExtractor

Listing B.1: BlobExtractor: *Blob.cs*

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;

namespace BlobExtractor
{
    public class Blob : TreeNode,IBlob
    {
        #region Fields
        private long startOffset;
        private long endOffset;
        private long analysisID;
        const long headerSize = 2 + 2 + 4 + 4;


        private UInt32 blobDataSize;
        private UInt16 blobType;
        private UInt32 flags;
        private UInt32 blobID;
        private byte[] data;

        #endregion

        #region Properties

        public long AnalysisID
        {
            get { return analysisID; }
            set { analysisID = value; }
        }
        public long StartOffset
        {
            get { return startOffset; }
            set { startOffset = value; }
        }

        public long EndOffset
        {
            get { return endOffset; }
            set { endOffset = value; }
        }

        public long HeaderSize
        {
            get { return headerSize; }
        }

        public long TotalSize
        {
            get{return HeaderSize + blobDataSize;}
        }

        public UInt32 BlobDataSize
        {
            get { return blobDataSize; }
            set { blobDataSize = value; }
        }


        public UInt16 BlobType
        {
            get { return blobType; }
            set { blobType = value; }
        }

        public String BlobTypeString
        {
            get
            {
                return BlobFactory.GetBlobTypeString(blobType);
            }
        }

        public UInt32 Flags
        {
            get { return flags; }
            set { flags = value; }
        }


        public UInt32 BlobID
        {
            get { return blobID; }
            set { blobID = value; }
        }


        public byte[] Data
        {
            get { return data; }
            set { data = value; }
        }
        #endregion

        #region Null properties
        public virtual Blob ChildBlob
        {
            get
            {
                return null;
            }
        }

        public virtual Blob ParentBlob
        {
            get
            {
                return null;
            }
        }

        public virtual Blob NeighbourBlob
        {
            get
            {
                return null;
            }
        }
        #endregion

        #region Constructor
        public Blob(long startOffset, long endOffset, UInt16 dataSize, UInt16 type, UInt32 flags, UInt32 id, byte[] data, long analysisID)
        {
            this.startOffset = startOffset;
            this.endOffset = endOffset;
            this.blobDataSize = dataSize;
            this.blobType = type;
            this.flags = flags;
            this.blobID = id;
            this.data = data;
            this.analysisID = analysisID;
            this.Text = ""+blobID;

            this.ContextMenu = new ContextMenu();
            MenuItem mItemStartOffset = new MenuItem();
            mItemStartOffset.Tag = this;
            mItemStartOffset.Text = "Copy␣start␣offset␣to␣clipboard";
            mItemStartOffset.Click += new EventHandler(mItemStartOffset_Click);
            this.ContextMenu.MenuItems.Add(mItemStartOffset);

        }
        #endregion

        void mItemStartOffset_Click(object sender, EventArgs e)
        {
            Blob b = (sender as MenuItem).Tag as Blob;
            if (b != null)
            {
                Clipboard.SetText(b.StartOffset.ToString());
            }
        }

        public override string ToString()
        {
            return "" + blobID;
        }

        #region Debug

        public virtual string DebugInfo
        {
            get
            {
                return String.Format("{0}\t{1}\t0x{2:X}\t0x{3:X}\t{4}\t0x{5:X}\t0x{6:X}\t0x{7:X}", analysisID, blobID, blobType, flags, blobDataSize, TotalSize, startOffset, endOffset);
            }
        }

        public virtual string Debug
        {
            get
            {
                return "debug";
            }
        }
        public static string DebugInfoHeader
        {
            get
            {
                return "AnalID\tID\tType\tFlags\tDataSize\tTotalSize\tStartOfs\tEndOfs";

            }
        }

        #endregion
    }
```

```
}
```

Listing B.2: BlobExtractor: *BlobExtractor.cs*

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Collections;

namespace BlobExtractor
{
    class BlobExtractor
    {
        const long HIGH_LOOKUP = 0x1000;
        const long BASE_LOOKUP = 0x5000;
        const long BLOB_HEADER_SIZE = 0xC;
        const long START_OFFSET = 0x0;

        static long counter = 0;
        StreamWriter sw = new StreamWriter("c:\\BlobExtractor.log");
        Hashtable h = new Hashtable();
        byte[] highLookUpTable = new byte[0x04000];
        byte[][] lowTables;
        string filename = "";

        public BlobExtractor(string filename)
        {
            this.filename = filename;
        }

        public UInt32 GetBlobPointer(UInt32 blobID)
        {
            return GetBlobPointer(GetHighIndexValue(blobID), GetLowIndexValue(blobID));
        }

        public UInt32 GetBlobPointer(UInt32 highIndex, UInt32 lowIndex)
        {
            UInt32 indexValue = BitConverter.ToUInt32(lowTables[highIndex], (int)(lowIndex * 4));
            indexValue = indexValue & 0x00FFFFFC;
            return indexValue;
        }

        public long GetBlobPointerFileOffset(UInt32 blobID)
        {
            return GetBlobPointerFileOffset(GetHighIndexValue(blobID), GetLowIndexValue(blobID));
        }

        public long GetBlobPointerFileOffset(UInt32 highIndex, UInt32 lowIndex)
        {
            long seekValue = GetBlobPointer(highIndex, lowIndex)+ BASE_LOOKUP;
            return seekValue;
        }

        public static UInt32 GetHighIndexValue(UInt32 blobID)
        {
            return (blobID & 0xFC00)>>10;
        }

        public static UInt32 GetLowIndexValue(UInt32 blobID)
        {
            return blobID & 0x3FF;
        }

        public void ExtractExact(string filename)
        {
            this.filename = filename;
            FileStream fs = File.OpenRead(filename);
            BinaryReader br = new BinaryReader(fs);
            long fileLength = fs.Length;

            fs.Seek(HIGH_LOOKUP, SeekOrigin.Begin);

            int count = fs.Read(highLookUpTable, 0, 0x4000);
            if (count != 0x4000 || highLookUpTable == null || BitConverter.ToUInt32(highLookUpTable, 0) == 0xFFFFFFFF)
            {
                Console.WriteLine("couldn't␣read␣hightable");
                return;
            }

            int numLowTables = 1;
            UInt32 indexValue = 1;

            while(true){ //get count
                indexValue = BitConverter.ToUInt32(highLookUpTable, numLowTables*4);
                if (indexValue != 0)
                    numLowTables++;

                else
                    break;
            }

            lowTables = new byte[numLowTables][];
            for (int i = 0; i < numLowTables; i++)
            {
                indexValue = BitConverter.ToUInt32(highLookUpTable, i * 4);

                long seekHeaderValue = BASE_LOOKUP + indexValue;
                fs.Seek(seekHeaderValue, SeekOrigin.Begin);
                readBlob(br);

                long seekDataValue = BASE_LOOKUP + indexValue + + BLOB_HEADER_SIZE;
                fs.Seek(seekDataValue, SeekOrigin.Begin);
                byte[] lowTable = new byte[0x1000];
                fs.Read(lowTable, 0, 0x1000);
                lowTables[i] = lowTable;
            }

            int emptySkips = 0;
            int largeSkips = 0;
            int okNormal = 0;

            for (int j = 0; j < numLowTables; j++)
            {
                for (int i = 0; i < 0x400; i++)
                {
                    indexValue = BitConverter.ToUInt32(lowTables[j], i * 4);
                    indexValue = indexValue& 0x00FFFFFC;
                    if(indexValue != 0)
                    {
                        long seekValue = indexValue + BASE_LOOKUP;
                        fs.Seek(seekValue, SeekOrigin.Begin);
                        readBlob(br);
                        okNormal++;
                    }
                    else if (indexValue != 0)
                    {
                        //Console.WriteLine("skipping large entry: [" + j + "," + i + "]");
                        largeSkips++;
                    }
                    else
                    {
                        //Console.WriteLine("skipping empty entry: [" + j + "," + i + "]");
                        emptySkips++;
                    }
                }
            }
            Console.WriteLine("high " + numLowTables);
            Console.WriteLine("empty " + emptySkips);
            Console.WriteLine("large " + largeSkips);
            Console.WriteLine("normal " + okNormal);
        }

        private void handleHighLookup(byte[] highLookupTable, int highIndex)
        {
        }

        public void Extract(string filename)
        {
            FileStream fs = File.OpenRead(filename);
            BinaryReader br = new BinaryReader(fs);
            long fileLength = fs.Length;

            fs.Seek(START_OFFSET, SeekOrigin.Begin);
            Console.WriteLine("starting extraction");
            sw.WriteLine(Blob.DebugInfoHeader);

            Blob blob = null;
            Blob last = null;
            bool done = false;
            do
            {
                if (fs.Position < fileLength - 14)
                    blob = readBlob(br);
                else
                    blob = null;

                if (blob != null)
                {
                    if (last != null) //check for holes
                    {
                        long diff = blob.StartOffset - last.EndOffset;
                        if (diff > 0)
                        {
                            String hole = string.Format("Memory␣hole.␣\t\t\t\t{0}\t{1:X}\t{2:X}", diff, last.EndOffset, blob.StartOffset);
                        }
                    }
                    dumpBlob(blob);
                    last = blob;
                }
                else
                {
                    fs.Seek(2, SeekOrigin.Current);
                    if (fs.Position >= fileLength -12)
                    {
                        done = true;
                    }
                }

            } while (!done);
            Console.WriteLine("done␣extracting");
            sw.Flush();
            sw.Close();
        }


        public UInt32 ExtractFreeBlobEndpoint(UInt32 startOffset, bool goNext)
        {
            FileStream fs = File.OpenRead(filename);
            BinaryReader br = new BinaryReader(fs);
            UInt32 result = getFreeBlobEndpoint(br, startOffset, goNext);
            br.Close();
            return result;
        }


        private UInt32 getFreeBlobEndpoint(BinaryReader br, UInt32 offset, bool goNext)
        {
            br.BaseStream.Seek(offset, SeekOrigin.Begin);

            UInt32 blobSize = br.ReadUInt32();
            UInt32 flags = br.ReadUInt32();
            UInt32 blobID = br.ReadUInt32();
            UInt32 next = br.ReadUInt32();
            UInt32 prev = br.ReadUInt32();

            if (goNext)
            {
                if ((next & 0xFFFFFF) == 0)
                    return offset;
                else
                    return getFreeBlobEndpoint(br, (next - 0xC) & 0xFFFFFF, true);
            }else
            {
                if ((prev & 0xFFFFFF) == 0)
                    return offset;
                else
                    return getFreeBlobEndpoint(br, (prev - 0xC) & 0xFFFFFF, false);
            }

        }

        private void dumpBlob(Blob blob)
        {
            String debugInfo = blob.DebugInfo;
            //Console.WriteLine(debugInfo);
            sw.WriteLine(debugInfo);
        }

        private Blob readBlob(BinaryReader br)
        {
            byte[] data;
            long startOffset = br.BaseStream.Position;
            UInt16 blobSize = br.ReadUInt16();
            UInt16 blobType = br.ReadUInt16();
            UInt32 flags = br.ReadUInt32();
            UInt32 blobID = br.ReadUInt32();

            bool sizeOK = (blobSize > 0) && (blobSize % 2 == 0);
            bool idOK = (blobID >= 0 && blobID < 30000);
            bool typeOK = ((blobType & 0x0FFF) == 0x000) && (blobType != 0);
            bool flagsOK = (flags == 0 || flags == 0x20000000);
            if (sizeOK && idOK && typeOK && flagsOK)
            {
                data = br.ReadBytes(blobSize);
            }
            else
            {
                //Console.WriteLine("Got error at : 0x{0:X} Rewinding to : 0x{1:X}", br.BaseStream.Position, startOffset);
                br.BaseStream.Seek(-12,SeekOrigin.Current);
                return null;
            }

            //if(
            long analysisID = counter++;
            Blob blob = BlobFactory.Create(startOffset, br.BaseStream.Position, blobSize, blobType, flags, blobID, data, analysisID);

            /*Blob blob = new Blob();
            blob.StartOffset = startOffset;
            blob.EndOffset = br.BaseStream.Position;
            blob.BlobDataSize = blobSize;
            blob.BlobType = blobType;
            blob.Flags = flags;
            blob.BlobID = blobID;
            blob.Data = data;
            blob.AnalysisID = analysisID; */

            return blob;
        }
    }
}
```

Listing B.3: BlobExtractor: *BlobFactory.cs*

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Text;
using System.Windows.Forms;
namespace BlobExtractor
{


    class BlobFactory
    {
        public static Dictionary<UInt32,Blob> FileDirIDToBlob = new Dictionary<UInt32, Blob>();
        public static Dictionary<UInt32, Blob> DatabaseIDToBlob = new Dictionary<UInt32, Blob>();
        public static Dictionary<UInt32, Blob> OtherIDToBlob = new Dictionary<UInt32, Blob>();
        public static Dictionary<UInt32, Blob> AllIDToBlob = new Dictionary<UInt32, Blob>(); //everything
        public static Dictionary<long, Blob> OffsetToFreeBlobs = new Dictionary<long, Blob>();

        public static Blob LowestOffsetBlob;
        public static Blob HighestOffsetBlob;

        public static Blob Create(long startOffset, long endOffset, UInt16 dataSize, UInt16 type, UInt32 flags, UInt32 id, byte[] data, long analysisID)
        {
            Blob b = null;

            switch (type)
            {
                case 0xFFFF:
                  //  b = new FreeBlob(startOffset, endOffset, dataSize, type, flags, id, data, analysisID);
                    break;

                case 0x4000: //"DIRECTORYNAME_ENTRY"
                    b = new DirectoryBlob(startOffset, endOffset, dataSize, type, flags, id, data, analysisID);
                    if (FileDirIDToBlob.ContainsKey(b.BlobID))
                        Console.WriteLine("FileDirIDToBlob already has key " + b.BlobID);
                    FileDirIDToBlob[b.BlobID] = b;
                    break;

                case 0x5000: //FILENAME_ENTRY
                    b = new FileBlob(startOffset, endOffset, dataSize, type, flags, id, data, analysisID);
                    if (FileDirIDToBlob.ContainsKey(b.BlobID))
                        Console.WriteLine("FileDirIDToBlob already has key " + b.BlobID);
                    FileDirIDToBlob[b.BlobID] = b;
                    break;

                case 0x7000: //database stuff
                b = new DatabaseBlob(startOffset, endOffset, dataSize, type, flags, id, data, analysisID);
                if (DatabaseIDToBlob.ContainsKey(b.BlobID))
                    Console.WriteLine("DatabaseIDToBlob␣already␣has␣key␣" + b.BlobID);
                DatabaseIDToBlob[b.BlobID] = b;
                break;
            case 0x8000: //database stuff
                b = new DatabaseRecordBlob(startOffset, endOffset, dataSize, type, flags, id, data, analysisID);
                if (DatabaseIDToBlob.ContainsKey(b.BlobID))
                    Console.WriteLine("DatabaseIDToBlob␣already␣has␣key␣" + b.BlobID);
                DatabaseIDToBlob[b.BlobID] = b;
                break;
            case 0xE000:
                b = new DatabaseIndexBlob(startOffset, endOffset, dataSize, type, flags, id, data, analysisID);
                if (DatabaseIDToBlob.ContainsKey(b.BlobID))
                    Console.WriteLine("DatabaseIDToBlob␣already␣has␣key␣" + b.BlobID);
                DatabaseIDToBlob[b.BlobID] = b;
                break;
            case 0x2000: //"SUPERBLOCK";
                b = new Blob(startOffset, endOffset, dataSize, type, flags, id+0xFF000000, data, analysisID);
                if (OtherIDToBlob.ContainsKey(b.BlobID))
```

```
                        Console.WriteLine("OtherIDToBlob already has key " + b.BlobID);
                    OtherIDToBlob[b.BlobID] = b;
                    break;
                default:
                case 0x3000: //"FILLER-BLOCK";
                case 0x6000: //"DATA(FILE ONLY?)";
                case 0xC000: //"REGISTER_STRING";
                case 0xD000: //"REGISTER_KEY";
                    b = new Blob(startOffset, endOffset, dataSize, type, flags, id, data, analysisID);
                    if(OtherIDToBlob.ContainsKey(b.BlobID))
                        Console.WriteLine("OtherIDToBlob already has key " + b.BlobID);
                    OtherIDToBlob[b.BlobID] = b;
                    break;

            }

            if (b != null)
            {
                if(AllIDToBlob.ContainsKey(b.BlobID))
                    Console.WriteLine("AllIDToBlob already has key " + b.BlobID);
                AllIDToBlob[b.BlobID] = b;

                if (LowestOffsetBlob == null || b.StartOffset < LowestOffsetBlob.StartOffset)
                    LowestOffsetBlob = b;
                if (HighestOffsetBlob == null || b.StartOffset > HighestOffsetBlob.StartOffset)
                    HighestOffsetBlob = b;
            }

            return b;
```

```
        }

        public static string GetBlobTypeString(UInt16 blobType)
        {
            switch (blobType)
            {
                case 0x2000:
                    return "SUPERBLOCK";
                    break;

                case 0x3000:
                    return "DATALIST";
                    break;

                case 0x4000:
                    return "DIRECTORYNAME";
                    break;

                case 0x5000:
                    return "FILE";
                    break;

                case 0x6000:
                    return "DATA";
                    break;

                case 0x7000:
                    return "Database";
                    break;

                case 0x8000:
                    return "DatabaseRecord";
                    break;

                case 0xE000:
                    return "DatabaseIndex";
                    break;

                case 0xC000:
                    return "REGISTER_STRING";
                    break;

                case 0xD000:
                    return "REGISTER_KEY";
                    break;

            }
            return "";
        }

        /// <summary>
        /// Get all blobs with given blob as parent.
        /// </summary>
        /// <param name="parent"></param>
        /// <returns></returns>
        public static List<Blob> GetParent(Blob parent,Dictionary<UInt32,Blob> table)
        {
            List<Blob> nodes = new List<Blob>();

            foreach (Blob blob in table.Values)
            {
                if (blob.ParentBlob == parent)
                    nodes.Add(blob);
            }
            return nodes;

        }

        /// <summary>
        /// Get all blobs with given ID as parent.
        /// </summary>
        /// <param name="blobId"></param>
        /// <returns></returns>
        public static List<Blob> GetParent(UInt32 parentID, Dictionary<UInt32, Blob> table)
        {
            Blob parent = table[parentID];
            return GetParent(parent, table);
        }
    }
```

```
}
```

Listing B.4: BlobExtractor: *DatabaseRecordBlob.cs*

```
using System;
using System.Collections.Generic;
using System.Text;

namespace BlobExtractor
{
    class DatabaseRecordBlob : Blob, IBlob
    {
        #region Fields
        UInt32 parentID;
        UInt32 lowOrderTime;
        UInt32 highOrderTime;
        UInt16 nameLength;
        string name;
        Property[] properties;

        Blob parent = null;

        #endregion
        #region Properties
        public UInt32 ParentID
        {
            get { return parentID; }
            set { parentID = value; }
        }
        #endregion

        public override Blob ParentBlob
        {
            get
            {
                if (parent == null && parentID != BlobID && BlobFactory.DatabaseIDToBlob.ContainsKey(parentID))
                    parent = BlobFactory.DatabaseIDToBlob[parentID];
                return parent;
            }
        }

        #region Constructor
        public DatabaseRecordBlob(long startOffset, long endOffset, UInt16 dataSize, UInt16 type, UInt32 flags, UInt32 id, byte[] data, long analysisID)
            : base(startOffset, endOffset, dataSize, type, flags, id, data, analysisID)
        {


            parentID = BitConverter.ToUInt32(data, 0);
            properties = PropertyFactory.Create(data, this);

            /*neighbourID = BitConverter.ToUInt32(data, 16);
            lowOrderTime = BitConverter.ToUInt32(data, 20);
            highOrderTime = BitConverter.ToUInt32(data, 24);
            nameLength = BitConverter.ToUInt16(data, 30);
            name = new string(Encoding.Unicode.GetChars(data, 32, nameLength * 2));*/
            string general = string.Format("{0:X}:{1}(db{2:X} size{3})", BlobID, name, type, TotalSize);
            this.Text = general + "\n" + getPropertyString();
        }
        #endregion

        private string getPropertyString()
        {
            StringBuilder sb = new StringBuilder();
            foreach (Property property in properties)
            {
                sb.Append("--");
                sb.Append(property.ToString());
            }
            return sb.ToString();
        }

        public override string Debug
        {
            get
            {
                return "db" + BlobID + ":" + name + " parent:" + parentID;
            }
        }

    }
}
```

Listing B.5: BlobExtractor: *FileBlob.cs*

```
using System;
using System.Collections.Generic;
using System.Text;

namespace BlobExtractor
{
    class FileBlob : Blob, IBlob
    {
        #region Fields
        UInt32 dataListID;
        UInt32 directoryID;
        UInt32 neighbourID;
        UInt32 lowOrderTime;
        UInt32 highOrderTime;
        UInt16 nameLength;
        string name;
        Blob dataList;
        Blob directory;
        Blob neighbour;
        #endregion

        #region Properties
        public UInt32 DataListID
        {
            get { return dataListID; }
            set { dataListID = value; }
        }


        public UInt32 DirectoryID
        {
            get { return directoryID; }
            set { directoryID = value; }
        }


        public UInt32 NeighbourID
        {
            get { return neighbourID; }
            set { neighbourID = value; }
        }


        public UInt16 NameLength
        {
            get { return nameLength; }
            set { nameLength = value; }
        }


        public string Name
        {
            get { return name; }
            set { name = value; }
        }


        public override Blob ChildBlob //datalist
        {
            get
            {
                if (dataList == null && BlobFactory.FileDirIDToBlob.ContainsKey(dataListID))
                    dataList = BlobFactory.FileDirIDToBlob[dataListID];
                return dataList;
            }
        }

        public override Blob ParentBlob
        {
            get
            {
                if (directory == null && directoryID != BlobID && BlobFactory.FileDirIDToBlob.ContainsKey(directoryID))
                    directory = BlobFactory.FileDirIDToBlob[directoryID];
                return directory;
            }
        }

        public override Blob NeighbourBlob
        {
            get
            {
                if (neighbour == null && BlobFactory.FileDirIDToBlob.ContainsKey(neighbourID))
                    neighbour = BlobFactory.FileDirIDToBlob[neighbourID];
                return neighbour;
            }
        }
        #endregion

        #region Constructor
        public FileBlob(long startOffset, long endOffset, UInt16 dataSize, UInt16 type, UInt32 flags, UInt32 id, byte[] data, long
            analysisID)
              : base(startOffset, endOffset, dataSize,
                 type, flags, id, data, analysisID)
          {
              dataListID = BitConverter.ToUInt32(data,
                 0);
              directoryID = BitConverter.ToUInt32(data,
                 12);
              neighbourID = BitConverter.ToUInt32(data,
                 16);
              lowOrderTime = BitConverter.ToUInt32(data,
                  20);
              highOrderTime = BitConverter.ToUInt32(data
                 , 24);
              nameLength = BitConverter.ToUInt16(data,
                 30);


              name = new string(Encoding.Unicode.
                 GetChars(data, 32, nameLength * 2));
              this.Text = BlobID + ":" + name + "(f)";
          }
          #endregion

          public override string Debug
          {
              get
              {
                  return "file" + BlobID + ":" + name +"
                     ␣parent:" + directoryID;
              }
          }
      }
}
```

Listing B.6: BlobExtractor: *FreeBlob.cs*

```
using System;
using System.Collections.Generic;
using System.Text;

```

```
namespace BlobExtractor
{
    class FreeBlob : Blob, IBlob
    {
        const int CeDB_MAXDBASENAMELEN = 32;

        #region Fields
        UInt32 nextOffset;
        UInt32 prevOffset;
        Blob next = null;
        Blob prev = null;

        #endregion
        #region Properties

        public UInt32 NextOffset
        {
            get { return nextOffset; }
            set { nextOffset = value; }
        }

        public UInt32 PreviousOffset
        {
            get { return prevOffset; }
            set { prevOffset = value; }
        }


        #endregion

        public override Blob NextBlob
        {
            get
            {
                if (next == null && nextOffset != StartOffset && BlobFactory.OffsetToFreeBlobs.ContainsKey(nextOffset))
                    next = BlobFactory.OffsetToFreeBlobs[nextOffset];
                return next;
            }
        }

        public override Blob NextBlobsy
        {
            get
            {
                if (next == null && nextOffset != StartOffset && BlobFactory.OffsetToFreeBlobs.ContainsKey(nextOffset))
                    next = BlobFactory.OffsetToFreeBlobs[nextOffset];
                return next;
            }
        }

        #region Constructor
        public FreeBlob(long startOffset, long endOffset, UInt16 dataSize, UInt16 type, UInt32 flags, UInt32 id, byte[] data, long analysisID)
            : base(startOffset, endOffset, dataSize, type, flags, id, data, analysisID)
        {
            parentID = BitConverter.ToUInt32(data, 0);
            char[] nameChars = Encoding.Unicode.GetChars(data, 8, CeDB_MAXDBASENAMELEN * 2);
            name = new string(nameChars);
            name = name.Substring(0, name.IndexOf('\0'));


            this.Text = string.Format("{0:X}:{1}(db{2:X},size{3})", BlobID, name, type, TotalSize);
        }
        #endregion

        public override string Debug
        {
            get
            {
                return "db" + BlobID + ":" + name + " parent:" + parentID;
            }
        }
}
```

Listing B.7: BlobExtractor: *Property.cs*

```
using System;
using System.Collections.Generic;
using System.Text;

namespace BlobExtractor
{
    public enum PropertyTypeEnum : ushort
    {
        I2 =            0x02,
        UI2 =           0x12,
        I4 =            0x03,
        UI4 =           0x13,
        TIME =          0x40,
        STRING =        0xF1,
        BIN =           0x41,
        BOOL =          0x0B,
        R8 =            0x05,
        ERROR =         0xFF,
        UNKNOWN =       0xFE,
    }

    class Property
    {
        #region Fields
        PropertyTypeEnum type;
        object value;
        int id;

        #endregion

        #region Properties
        public PropertyTypeEnum PropertyType
        {
            get { return type; }
            set { type = value; }
        }
        public object PropertyValue
        {
            get { return this.value; }
            set { this.value = value; }
        }

        public int ID
        {
            get { return this.id; }
            set { this.id = value; }
        }
        #endregion

        #region Constructor


        public Property(PropertyTypeEnum type, int id, object value)
        {
            this.type = type;
            this.id = id;
            this.value = value;
        }
        #endregion

        #region Overrides
        public override string ToString()
        {
            string s = "(" + id + ":" + type.ToString();
            if (value != null)
                s += ":" + value.ToString();
            s += ")";
            return s;
        }
        #endregion
    }
}
```

Listing B.8: BlobExtractor: *PropertyFactory.cs*

```
using System;
using System.Collections.Generic;
using System.Text;

namespace BlobExtractor
{
    class PropertyFactory
    {
        /// <summary>
        /// Create properties from blob data.
        /// </summary>
        /// <param name="data"></param>
        /// <returns></returns>
        public static Property[] Create(byte[] data, Blob blob)
        {
            int aaa;
            if ((blob.BlobID == 0x2a36) || (blob.BlobID == 0x362a))
                aaa = 0;

            int intSize = 4;
            int index = 8;
            UInt16 headerBytes = BitConverter.ToUInt16(data, index);
            index += 2;
            int propertiesDataSize = BitConverter.ToUInt16(data, index);
            index += 2;
            int numProperties = (data.Length - propertiesDataSize - 12) / 4;
            Property[] properties = new Property[numProperties];
            for (int i = 0; i < numProperties; i++)
            {
                PropertyTypeEnum type = (PropertyTypeEnum)data[index];
                index += 2;
                int id = BitConverter.ToInt16(data, index);
                index += 2;
                properties[i] = new Property(type, id, null);
            }

            bool unknown = false;
            if ((headerBytes & 0x4000) == 0x4000)
            {
                index += 2; //skip 2 bytes if this bit is set.
                intSize = 2;
            }
            else
            {
                intSize = 4;
            }

            for (int i = 0; i < numProperties; i++)
            {
                Property property = properties[i];
                if(unknown){
                    property.PropertyType = PropertyTypeEnum.ERROR;
                    continue;
                }

                try
                {
                    switch (property.PropertyType)
                    {
                        case PropertyTypeEnum.BIN:
                            index += 16;
                            break;
                        case PropertyTypeEnum.I2:
                            property.PropertyValue = BitConverter.ToInt16(
                        data, index);
                    index += 2;
                    break;
                case PropertyTypeEnum.I4:
                    if (intSize == 2)
                    {
                        property.PropertyValue = BitConverter.ToInt16(data, index);
                        index += 2;
                    }
                    else if(intSize == 4)
                    {
                        property.PropertyValue = BitConverter.ToInt32(data, index);
                        index += 4;
                    }


                    break;
                case PropertyTypeEnum.STRING:
                    string stringValue = null;
                    if (data[index + 3] == 0x0) //unicode
                    {
                        UInt16 length = BitConverter.ToUInt16(data, index);
                        index += 2;
                        stringValue = new string(Encoding.Unicode.GetChars(data, index, length));
                        index += length;
                    }
                    else //ascii
```

```
                        {

                            byte length = (byte)(data[index] / 2);
                            index += 1;
                            stringValue = new string(Encoding.ASCII.GetChars(data, index, length));
                            if (length % 2 == 0)
                                index += length + 1;
                            else
                                index += length;
                        }
                        property.PropertyValue = stringValue;
                        break;
                    case PropertyTypeEnum.TIME:
                        byte[] date = new byte[4];
                        date[0] = data[index++];
                        date[1] = data[index++];
                        date[2] = data[index++];
                        date[3] = data[index++];
                        property.PropertyValue = date;
                        break;
                    case PropertyTypeEnum.UI4:
                        if (intSize == 2)
                        {
                            property.PropertyValue = BitConverter.ToUInt16(data, index);
                            index += 2;
                        }
                        else if (intSize == 4)
                        {
                            property.PropertyValue = BitConverter.ToUInt32(data,
                                   index);
                                index += 4;
                            }
                            break;
                        default:
                            int o = 4;
                            property.PropertyType =
                               PropertyTypeEnum.
                              UNKNOWN;
                            unknown = true;
                            break;
                    }
                }
                catch (Exception e)
                {
                    property.PropertyType =
                       PropertyTypeEnum.ERROR;
                    unknown = true;
                }
            }
            return properties;
        }
    }
}
```

B.2 Extensions to the Judas Forensic Tool

Listing B.9: Extensions to the Judas Forensic Tool

```
void perform_OID_request()
{
        printf("****** Enter file name: ");
        char file_name[MAX_FILENAME_SIZE];
        scanf("%hs", &file_name);
        get_oid(file_name);
}

void perform_object_request()
{
        printf("****** Enter OID (hex): ");
        int object_id;
        scanf("%x",&object_id);

        if(object_id & 0xffff0000)
        {
                get_object(object_id);
        }else
        {
                DWORD cnt = 0x1000000;
                while((object_id < 0xFF00000) && !get_object(object_id))
                {
                        printf("tried 0x%X\n", object_id);
                        object_id += cnt;
                }
        }
}

DWORD flip_dword(DWORD dw_in)
{
        DWORD dw_rev_in = dw_in << 24;
        dw_rev_in = ((dw_in & 0x0000ff00) << 8) | dw_rev_in;
        dw_rev_in = ((dw_in & 0x00ff0000) >> 8) | dw_rev_in;
        dw_rev_in = dw_rev_in | (dw_in >> 24);
        return dw_rev_in;
}

void get_oid(char *file_name)
{

        wchar_t wide_file_name[MAX_FILENAME_SIZE];
        mbstowcs(wide_file_name,file_name, MAX_FILENAME_SIZE);

        CE_FIND_DATA find_data;
        HANDLE hSearch;
        hSearch = CeFindFirstFile(wide_file_name, &find_data);
        if(hSearch != INVALID_HANDLE_VALUE){

                char szFilename[256];
                wcstombs(szFilename,find_data.cFileName,256);

                printf("File name: %s\n",szFilename);
                printf("Object ID: %x\n",find_data.dwOID);
                print_time_data(find_data.ftLastWriteTime);
                print_file_attributes(find_data.dwFileAttributes);

                return;
        }else{
                hSearch = CeFindFirstDatabase(0); // Parameter value zero means that all database types are enumerated
                if(hSearch != INVALID_HANDLE_VALUE)
                {
                        CEOID ceoid = CeFindNextDatabase(hSearch);
                        char szFilename[256];
                        while(ceoid != 0)
```

```
            {
                CEOIDINFO ceoidinfo;
                CeOidGetInfo(ceoid,&ceoidinfo);
                wcstombs(szFilename,ceoidinfo.infDatabase.szDbaseName,256);
                printf("Name : %s\n",szFilename);
                printf("Type : 0x%X\n",ceoidinfo.infDatabase.dwDbaseType);
                printf("OID  : 0x%X\n",ceoid);

        ceoid = CeFindNextDatabase(hSearch);
            }
        }else{
            printf("Error");
            return;
        }
    }
    CeCloseHandle(hSearch);

}

void print_file_attributes(DWORD attr)
{
    printf("File Attributes (0x%X):\n",attr);
    printf("\n");
    printf("Archive      : ");        (FILE_ATTRIBUTE_ARCHIVE & attr)? printf("X\n") : printf("\n");
    printf("Compressed   : ");        (FILE_ATTRIBUTE_COMPRESSED & attr)? printf("X\n") : printf("\n");
    printf("Directory    : ");        (FILE_ATTRIBUTE_DIRECTORY & attr)? printf("X\n") : printf("\n");
```

```
	printf("Has children : ");		(FILE_ATTRIBUTE_HAS_CHILDREN & attr)? printf("X\n") : printf("\n");
	printf("Hidden       : ");		(FILE_ATTRIBUTE_HIDDEN & attr)? printf("X\n") : printf("\n");
	printf("In ROM       : ");		(FILE_ATTRIBUTE_INROM & attr)? printf("X\n") : printf("\n");
	printf("Normal       : ");		(FILE_ATTRIBUTE_NORMAL & attr)? printf("X\n") : printf("\n");
	printf("Read only    : ");		(FILE_ATTRIBUTE_READONLY & attr)? printf("X\n") : printf("\n");
	printf("ROM module   : ");		(FILE_ATTRIBUTE_ROMMODULE & attr)? printf("X\n") : printf("\n");
	printf("System       : ");		(FILE_ATTRIBUTE_SYSTEM & attr)? printf("X\n") : printf("\n");
	printf("Temporary    : ");		(FILE_ATTRIBUTE_TEMPORARY & attr)? printf("X\n") : printf("\n");
}

void print_db_flags(DWORD flags)
{
	printf("Database flags (%X):\n",flags);
	printf("\n");
	printf("Valid modified time : ");		(CEDB_VALIDMODTIME & flags)? printf("X\n") : printf("\n");
	printf("Valid name          : ");		(CEDB_VALIDNAME & flags)? printf("X\n") : printf("\n");
	printf("Valid type          : ");		(CEDB_VALIDTYPE & flags)? printf("X\n") : printf("\n");
	printf("Valid sort spec     : ");		(CEDB_VALIDSORTSPEC & flags)? printf("X\n")
```

```
                : printf("\n");
        printf("Valid␣flags␣␣␣␣␣␣␣␣␣:␣");          (CEDB_VALIDDBFLAGS & flags)? printf("X\n") : printf("\n");
        printf("No␣compress␣␣␣␣␣␣␣␣␣:␣");          (CEDB_NOCOMPRESS & flags)? printf("X\n") : printf("\n");
}

void print_time_data(FILETIME filetime)
{
        SYSTEMTIME s_time;
        if(FileTimeToSystemTime(&filetime, &s_time)){
                printf("Modified␣:␣%i.%i.%i␣%i:%i\n", s_time.wDay, s_time.wMonth, s_time.wYear, s_time.wHour, s_time.wMinute);
                DWORD dw_low = filetime.dwLowDateTime;
                DWORD dw_high = filetime.dwHighDateTime;
                printf("FILETIME␣:␣%x␣%x\n",dw_high, dw_low);
                DWORD dw_rev_low = flip_dword(dw_low);
                DWORD dw_rev_high = flip_dword(dw_high);
                printf("Reversed␣:␣%x␣%x\n",dw_rev_low,dw_rev_high);
        }
}

void print_sort_order_specs(SORTORDERSPEC sos[], WORD num_sos)
{
        printf("\n");
        printf("Sort␣order␣specs\n");
        for(int i=0;i<num_sos;i++)
        {
                printf("\n");
                printf("Spec␣#%i:\n",i);
                print_sort_flags(sos[i].dwFlags);
        }
```

```
}

void print_sort_flags(DWORD flags)
{
        printf("Descending       : "); (CEDB_SORT_DESCENDING & flags)? printf("X\n") : printf("\n");
        printf("Case insensitive : "); (CEDB_SORT_CASEINSENSITIVE & flags)? printf("X\n") : printf("\n");
        printf("Unknown first    : "); (CEDB_SORT_UNKNOWNFIRST & flags)? printf("X\n") : printf("\n");
}

bool get_object(int object_id)
{
        CEOID ceoid = object_id;
        CEOIDINFO ceoidinfo;
        if(CeOidGetInfo(ceoid,&ceoidinfo))
        {
                char name[256];
                switch(ceoidinfo.wObjType)
                {
                case OBJTYPE_INVALID:
                        printf("Object type: INVALID\n");
                        break;
                case OBJTYPE_FILE:
                        wcstombs(name, ceoidinfo.infFile.szFileName,256);
                        printf("Name       : %s\n", name);
                        printf("Object type: FILE\n");
                        printf("Object ID  : 0x%X\n", ceoid);
                        printf("Parent ID  : 0x%X\n", ceoidinfo.infFile.oidParent);
                        print_time_data(ceoidinfo.infFile.ftLastChanged);
                print_file_attributes(ceoidinfo.infFile.dwAttributes);
                break;
            case OBJTYPE_DIRECTORY:
                wcstombs(name,ceoidinfo.infDirectory.szDirName,256);
                printf("Name       : %s\n",name);
                printf("Object type: DIRECTORY\n");
                printf("Object ID  : 0x%X\n",ceoid);
                printf("Parent ID  : 0x%X\n",ceoidinfo.infDirectory.oidParent);
                print_file_attributes(ceoidinfo.infDirectory.dwAttributes);
        break;
            case OBJTYPE_DATABASE:
                wcstombs(name,ceoidinfo.infDatabase.szDbaseName,256);
                printf("Name           : %s\n",name);
                printf("Object type    : DATABASE\n");
                printf("Object ID      : 0x%X\n",ceoid);
                print_time_data(ceoidinfo.infDatabase.ftLastModified);
                print_db_flags(ceoidinfo.infDatabase.dwFlags);
                printf("\n");
                printf("Database info\n");
                printf("Database type         : 0x%X\n",ceoidinfo.infDatabase.dwDbaseType);
```

```
                        printf("Database␣size␣␣␣␣␣␣␣␣␣:␣0x%X\n",ceoidinfo.infDatabase.dwSize);
                        printf("#␣records␣␣␣␣␣␣␣␣␣␣␣␣␣:␣0x%X\n",ceoidinfo.infDatabase.wNumRecords);
                        printf("Number␣of␣sort␣orders␣:␣0x%X\n",ceoidinfo.infDatabase.wNumSortOrder);
                        if(CEDB_VALIDSORTSPEC & ceoidinfo.infDatabase.dwFlags)
                        {
                                print_sort_order_specs(ceoidinfo.infDatabase.rgSortSpecs, ceoidinfo.infDatabase.wNumSortOrder);
                        }

                        break;
                case OBJTYPE_RECORD:
                        printf("Object␣type␣:␣RECORD\n");
                        printf("OID␣parent␣␣:0x%X\n", ceoidinfo.infRecord.oidParent);
                        break;
                }
                return true;
        }else
        {
                return false;
        }
}

void perform_attributes_request()
{
        printf("******␣Enter␣file␣name:␣");
        char search_name[MAX_FILENAME_SIZE];
        scanf("%hs", &search_name);

        wchar_t wide_search_name[MAX_FILENAME_SIZE];
        mbstowcs(wide_search_name,search_name,MAX_FILENAME_SIZE);

        CE_FIND_DATA find_data;
        HANDLE hSearch;
        hSearch = CeFindFirstFile(wide_search_name, &find_data);
        if(hSearch != INVALID_HANDLE_VALUE){

                char szFilename[256];
                wcstombs(szFilename,find_data.cFileName,256);

                printf("File␣name:␣%s\n",szFilename);
                printf("Object␣ID:␣%x\n",find_data.dwOID);
                print_time_data(find_data.ftLastWriteTime);
                char at;
                while(true){
                        print_file_attributes(find_data.dwFileAttributes);
                        printf("\n");
                        printf("Alter␣attribute␣((a)rchive,␣(h)idden,␣(n)ormal,␣(r)ead␣only,␣(s)ystem,␣(t)emporary).␣Enter␣c␣to␣cancel:␣");
                        printf("\n");
                        scanf("%c",&at);
                        scanf("%c",&at);
                        switch(at)
                        {
                        case 'a':
                                alter_attribute(find_data,
                                        FILE_ATTRIBUTE_ARCHIVE
                                        );
                                    break;
                        case 'h':
                                    alter_attribute(
                                        find_data,
                                        FILE_ATTRIBUTE_HIDDEN
                                        );
                                    break;
                        case 'n':
                                    alter_attribute(
                                        find_data,
                                        FILE_ATTRIBUTE_NORMAL
                                        );
                                    break;
                        case 'r':
                                    alter_attribute(
                                        find_data,
                                        FILE_ATTRIBUTE_READONLY
                                        );
                                    break;
                        case 's':
                                    alter_attribute(
                                        find_data,
                                        FILE_ATTRIBUTE_SYSTEM
                                        );
                                    break;
                        case 't':
                                    alter_attribute(
                                        find_data,
                                        FILE_ATTRIBUTE_TEMPORARY
                                        );
                                    break;
                        case 'c':
                                    CeCloseHandle(hSearch)
                                        ;
                                    return;
                        default:
                                    printf("Error:␣
                                        incorrect␣input");
                                    return;
```

```
                }
                CeCloseHandle(hSearch);
                hSearch = CeFindFirstFile(wide_search_name, &find_data);
                if(hSearch == INVALID_HANDLE_VALUE) return;
            }
            return;
        }else
        {
            printf("Error:␣No␣matching␣file␣found");
            return;
        }
        CeCloseHandle(hSearch);
}

void alter_attribute(CE_FIND_DATA find_data, DWORD attr)
{
        wchar_t ws[256];
        ws[0] = '\\';

        int cnt = 1;
        for(int i=0;find_data.cFileName[i]!=0;i++)
        {
            ws[i+1] = find_data.cFileName[i];
            cnt++;
        }
        ws[cnt] = 0;

        (find_data.dwFileAttributes & attr) ?
        CeSetFileAttributes(ws,(find_data.dwFileAttributes & !attr)):
        CeSetFileAttributes(ws, (find_data.dwFileAttributes | attr));
}
```

Appendix C

ARM Instruction Set Quick Reference Card

ARM Instruction Set
Quick Reference Card

Key to Tables		
	`{cond}`	Refer to Table **Condition Field {cond}**
	`<Oprnd2>`	Refer to Table **Oprnd2**
	`{field}`	Refer to Table **Field**
	`S`	Sets condition codes (optional)
	`B`	Byte operation (optional)
	`H`	Halfword operation (optional)
	`T`	Forces address translation. Cannot be used with pre-indexed addresses
	`<a_mode2>`	Refer to Table **Addressing Mode 2**
	`<a_mode2P>`	Refer to Table **Addressing Mode 2 (Privileged)**
	`<a_mode3>`	Refer to Table **Addressing Mode 3**
	`<a_mode4L>`	Refer to Table **Addressing Mode 4 (Load)**
	`<a_mode4S>`	Refer to Table **Addressing Mode 4 (Store)**
	`<a_mode5>`	Refer to Table **Addressing Mode 5**
	`#32bit_Imm`	A 32-bit constant, formed by right-rotating an 8-bit value by an even number of bits
	`<reglist>`	A comma-separated list of registers, enclosed in braces ({ and })

Operation		Assembler	S updates	Action	Notes
Move	Move	`MOV{cond}{S} Rd, <Oprnd2>`	N Z C	Rd:= <Oprnd2>	
	NOT	`MVN{cond}{S} Rd, <Oprnd2>`	N Z C	Rd:= 0xFFFFFFFF EOR <Oprnd2>	
	SPSR to register	`MRS{cond} Rd, SPSR`		Rd:= SPSR	*Architecture 3, 3M and 4 only*
	CPSR to register	`MRS{cond} Rd, CPSR`		Rd:= CPSR	*Architecture 3, 3M and 4 only*
	register to SPSR	`MSR{cond} SPSR{field}, Rm`		SPSR:= Rm	*Architecture 3, 3M and 4 only*
	register to CPSR	`MSR{cond} CPSR{field}, Rm`		CPSR:= Rm	*Architecture 3, 3M and 4 only*
	immediate to SPSR flags	`MSR{cond} SPSR_f, #32bit_Imm`		SPSR:= #32-bit_Imm	*Architecture 3, 3M and 4 only*
	immediate to CPSR flags	`MSR{cond} CPSR_f, #32bit_Imm`		CPSR:= #32-bit_Imm	*Architecture 3, 3M and 4 only*
ALU	Arithmetic				
	Add	`ADD{cond}{S} Rd, Rn, <Oprnd2>`	N Z C V	Rd:= Rn + <Oprnd2>	
	with carry	`ADC{cond}{S} Rd, Rn, <Oprnd2>`	N Z C V	Rd:= Rn + <Oprnd2> + Carry	
	Subtract	`SUB{cond}{S} Rd, Rn, <Oprnd2>`	N Z C V	Rd:= Rn - <Oprnd2>	
	with carry	`SBC{cond}{S} Rd, Rn, <Oprnd2>`	N Z C V	Rd:= Rn - <Oprnd2> - NOT(Carry)	
	reverse subtract	`RSB{cond}{S} Rd, Rn, <Oprnd2>`	N Z C V	Rd:= <Oprnd2> - Rn	
	reverse subtract with carry	`RSC{cond}{S} Rd, Rn, <Oprnd2>`	N Z C V	Rd:= <Oprnd2> - Rn - NOT(Carry)	
	Multiply	`MUL{cond}{S} Rd, Rm, Rs`	N Z	Rd:= Rm * Rs	*Not in Architecture 1*
	accumulate	`MLA{cond}{S} Rd, Rm, Rs, Rn`	N Z	Rd:= (Rm * Rs) + Rn	*Not in Architecture 1*
	unsigned long	`UMULL{cond}{S} RdLo, RdHi, Rm, Rs`	N Z	RdHi:= (Rm*Rs)[63:32] RdLo:= (Rm*Rs)[31:0]	*Architecture 3M and 4 only*
	unsigned accumulate long	`UMLAL{cond}{S} RdLo, RdHi, Rm, Rs`	N Z	RdLo:= (Rm*Rs)[31:0] + RdLo RdHi:=(Rm*Rs)[63:32] + RdHi + CarryFrom ((Rm*Rs)[31:0] +RdLo))	*Architecture 3M and 4 only*
	signed long	`SMULL{cond}{S} RdLo, RdHi, Rm, Rs`	N Z	RdHi:= signed(Rm*Rs)[63:32] RdLo:= signed(Rm*Rs)[31:0]	*Architecture 3M and 4 only*
	signed accumulate long	`SMLAL{cond}{S} RdLo, RdHi, Rm, Rs`	N Z	RdLo:= signed(Rm*Rs)[31:0] + RdLo RdHi:= signed(Rm*Rs)[63:32] + RdHi + CarryFrom ((Rm*Rs)[31:0] +RdLo))	*Architecture 3M and 4 only*
	Compare	`CMP{cond} Rd, <Oprnd2>`	N Z C V	CPSR flags:= Rn - <Oprnd2>	
	negative	`CMN{cond} Rd, <Oprnd2>`	N Z C V	CPSR flags:= Rn + <Oprnd2>	
	Logical				
	Test	`TST{cond} Rn, <Oprnd2>`	N Z C	CPSR flags:= Rn AND <Oprnd2>	
	Test equivalence	`TEQ{cond} Rn, <Oprnd2>`	N Z C	CPSR flags:= Rn EOR <Oprnd2>	
	AND	`AND{cond}{S} Rd, Rn, <Oprnd2>`	N Z C	Rd:= Rn AND <Oprnd2>	
	EOR	`EOR{cond}{S} Rd, Rn, <Oprnd2>`	N Z C	Rd:= Rn EOR <Oprnd2>	
	ORR	`ORR{cond}{S} Rd, Rn, <Oprnd2>`	N Z C	Rd:= Rn OR <Oprnd2>	
	Bit Clear	`BIC{cond}{S} Rd, Rn, <Oprnd2>`	N Z C	Rd:= Rn AND NOT <Oprnd2>	
	Shift/Rotate				See Table **Oprnd2**

ARM Instruction Set
Quick Reference Card

Operation		Assembler	Action	Notes
Branch	Branch	B{cond} label	R15:= address of label	Address calculated pc-relative
	with link	BL{cond} label	R14:=R15-4, R15:= address of label	
	and exchange instruction set	BX{cond} Rn	R15:=Rn, T bit:= Rn[0]	*Architecture 4 with Thumb only* to Thumb state; Rn[0] = 1 to ARM state; Rn[0] =0
Load	Word	LDR{cond} Rd, <a_mode2>	Rd:= [address]	
	with user-mode privilege	LDR{cond}T Rd, <a_mode2P>		
	Byte	LDR{cond}B Rd, <a_mode2>	Rd:= [byte value from address] Loads bits 0 to 7 and sets bits 8-31 to 0	
	with user-mode privilege	LDR{cond}BT Rd, <a_mode2P>		
	signed	LDR{cond}SB Rd, <a_mode3>	Rd:= [signed byte value from address] Loads bits 0 to 7 and sets bits 8-31 to bit 7	*Architecture 4 only*
	Halfword	LDR{cond}H Rd, <a_mode3>	Rd:= [halfword value from address] Loads bits 0 to 15 and sets bits 16-31 to 0	*Architecture 4 only*
	signed	LDR{cond}SH Rd, <a_mode3>	Rd:= [signed halfword value from address] Loads bits 0 to 15 and sets bits 16-31 to bit 15	*Architecture 4 only*
	Multiple			
	Block data operations			
	Increment Before	LDM{cond}IB Rd{!}, <reglist>{^}		! sets the W bit (updates the base register after the transfer) ^ sets the S bit
	Increment After	LDM{cond}IA Rd{!}, <reglist>{^}		
	Decrement Before	LDM{cond}DB Rd{!}, <reglist>{^}		
	Decrement After	LDM{cond}DA Rd{!}, <reglist>{^}		
	Stack operations	LDM{cond}<a_mode4L> Rd{!}, <reglist>	Stack manipulation (pop)	! sets the W bit (updates the base register after the transfer)
	and restore CPSR	LDM{cond}<a_mode4L> Rd{!}, <reglist+pc>^		
	User registers	LDM{cond}<a_mode4L> Rd, <reglist>^		
Store	Word	STR{cond} Rd, <a_mode2>	[address]:= Rd	
	with user-mode privilege	STR{cond}T Rd, <a_mode2P>		
	Byte	STR{cond}B Rd, <a_mode2>	[address]:= byte value from Rd	
	with user-mode privilege	STR{cond}BT Rd, <a_mode2P>		
	Halfword	STR{cond}H Rd, <a_mode3>	[address]:= halfword value from Rd	*Architecture 4 only*
	Multiple			
	Block data operations			
	Increment Before	STM{cond}IB Rd{!}, <reglist>{^}		! sets the W bit (updates the base register after the transfer) ^ sets the S bit
	Increment After	STM{cond}IA Rd{!}, <reglist>{^}		
	Decrement Before	STM{cond}DB Rd{!}, <reglist>{^}		
	Decrement After	STM{cond}DA Rd{!}, <reglist>{^}		
	Stack operations	STM{cond}<a_mode4S> Rd{!}, <reglist>	Stack manipulation (push)	
	User registers	STM{cond}<a_mode4S> Rd{!}, <reglist>^		
Swap	Word	SWP{cond} Rd, Rm, [Rn]		*Not in Architecture 1 or 2*
	Byte	SWP{cond}B Rd, Rm, [Rn]		*Not in Architecture 1 or 2*
Coprocessors	Data operations	CDP{cond} p<cpnum>, <op1>, CRd, CRn, CRm, <op2>		*Not in Architecture 1*
	Move to ARM reg from coproc	MRC{cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>		
	Move to coproc from ARM reg	MCR{cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>		
	Load	LDC{cond} p<cpnum>, CRd, <a_mode5>		
	Store	STC{cond} p<cpnum>, CRd, <a_mode5>		
Software Interrupt		SWI 24bit_Imm	Causes a software interrupt processor exception	24-bit immediate value encoded within the instruction.

ARM Addressing Modes
Quick Reference Card

Addressing Mode 2	
Immediate offset	[Rn, #+/-12bit_Offset]
Register offset	[Rn, +/-Rm]
Scaled register offset	[Rn, +/-Rm, LSL #5bit_shift_imm]
	[Rn, +/-Rm, LSR #5bit_shift_imm]
	[Rn, +/-Rm, ASR #5bit_shift_imm]
	[Rn, +/-Rm, ROR #5bit_shift_imm]
	[Rn, +/-Rm, RRX]
Pre-indexed offset	
Immediate	[Rn, #+/-12bit_Offset]!
Register	[Rn, +/-Rm]!
Scaled register	[Rn, +/-Rm, LSL #5bit_shift_imm]!
	[Rn, +/-Rm, LSR #5bit_shift_imm]!
	[Rn, +/-Rm, ASR #5bit_shift_imm]!
	[Rn, +/-Rm, ROR #5bit_shift_imm]!
	[Rn, +/-Rm, RRX]!
Post-indexed offset	
Immediate	[Rn], #+/-12bit_Offset
Register	[Rn], +/-Rm
Scaled register	[Rn], +/-Rm, LSL #5bit_shift_imm
	[Rn], +/-Rm, LSR #5bit_shift_imm
	[Rn], +/-Rm, ASR #5bit_shift_imm
	[Rn], +/-Rm, ROR #5bit_shift_imm
	[Rn, +/-Rm, RRX]

Addressing Mode 2 (Privileged)	
Immediate offset	[Rn, #+/-12bit_Offset]
Register offset	[Rn, +/-Rm]
Scaled register offset	[Rn, +/-Rm, LSL #5bit_shift_imm]
	[Rn, +/-Rm, LSR #5bit_shift_imm]
	[Rn, +/-Rm, ASR #5bit_shift_imm]
	[Rn, +/-Rm, ROR #5bit_shift_imm]
	[Rn, +/-Rm, RRX]
Post-indexed offset	
Immediate	[Rn], #+/-12bit_Offset
Register	[Rn], +/-Rm
Scaled register	[Rn], +/-Rm, LSL #5bit_shift_imm
	[Rn], +/-Rm, LSR #5bit_shift_imm
	[Rn], +/-Rm, ASR #5bit_shift_imm
	[Rn], +/-Rm, ROR #5bit_shift_imm
	[Rn, +/-Rm, RRX]

Addressing Mode 3 - Signed Byte and Halfword Data Transfer	
Immediate offset	[Rn, #+/-8bit_Offset]
Pre-indexed	[Rn, #+/-8bit_Offset]!
Post-indexed	[Rn], #+/-8bit_Offset
Register	[Rn, +/-Rm]
Pre-indexed	[Rn, +/-Rm]!
Post-indexed	[Rn], +/-Rm

Addressing Mode 5 - Coprocessor Data Transfer	
Immediate offset	[Rn, #+/-(8bit_Offset*4)]
Pre-indexed	[Rn, #+/-(8bit_Offset*4)]!
Post-indexed	[Rn], #+/-(8bit_Offset*4)

Addressing Mode 4 (Load)			
Addressing Mode		Stack Type	
IA	Increment After	FD	Full Descending
IB	Increment Before	ED	Empty Descending
DA	Decrement After	FA	Full Ascending
DB	Decrement Before	EA	Empty Ascending

Addressing Mode 4 (Store)			
Addressing Mode		Stack Type	
IA	Increment After	EA	Empty Ascending
IB	Increment Before	FA	Full Ascending
DA	Decrement After	ED	Empty Descending
DB	Decrement Before	FD	Full Descending

Oprnd2	
Immediate value	#32bit_Imm
Logical shift left	Rm LSL #5bit_Imm
Logical shift right	Rm LSR #5bit_Imm
Arithmetic shift right	Rm ASR #5bit_Imm
Rotate right	Rm ROR #5bit_Imm
Register	Rm
Logical shift left	Rm LSL Rs
Logical shift right	Rm LSR Rs
Arithmetic shift right	Rm ASR Rs
Rotate right	Rm ROR Rs
Rotate right extended	Rm RRX

Field		
Suffix	Sets	
_c	Control field mask bit	(bit 3)
_f	Flags field mask bit	(bit 0)
_s	Status field mask bit	(bit 1)
_x	Extension field mask bit	(bit 2)

Condition Field {cond}	
Suffix	Description
EQ	Equal
NE	Not equal
CS	Unsigned higher or same
CC	Unsigned lower
MI	Negative
PL	Positive or zero
VS	Overflow
VC	No overflow
HI	Unsigned higher
LS	Unsigned lower or same
GE	Greater or equal
LT	Less than
GT	Greater than
LE	Less than or equal
AL	Always