

PATENT APPLICATION SERIAL NO _____

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

02/22/2006 NNGUYEN1 00000093 60774406

01 FC:2005

100.00 OP

PTO-1556

(5/87)

021706

1831 U.S. PTO

PTO/SB/16 (10-01)

Approved for use through 10/31/2002. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

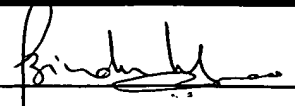
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PROVISIONAL APPLICATION FOR PATENT COVER SHEET

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53(c).

Express Mail Label No. EQ165635822US

INVENTOR(S)					
Given Name (first and middle [if any])		Family Name or Surname		Residence (City and either State or Foreign Country)	
Robert C.		Daley		13 Middle Dunstable Road, Nashua, NH 030	
Deven P.		Shah		5030 East Tenderrow Place, F, Orange, CA	
Karen		Chan		Toronto, Canada	
<input type="checkbox"/> Additional inventors are being named on the _____ separately numbered sheets attached hereto					
TITLE OF THE INVENTION (500 characters max)					
DIAGNOSTICS AND MONITORING SERVICES IN A MOBILE NETWORK FOR A MOBILE DEVICE					
Direct all correspondence to: CORRESPONDENCE ADDRESS					
<input type="checkbox"/> Customer Number		[]		Place Customer Number Bar Code Label here	
OR		Type Customer Number here			
<input type="checkbox"/> Firm or Individual Name		McAndrews, Held & Malloy, Ltd			
Address		500 West Madison Street, 34th Floor			
Address					
City		Chicago	State	IL	ZIP 60661
Country		Telephone	312-775-8108	Fax	
ENCLOSED APPLICATION PARTS (check all that apply)					
<input checked="" type="checkbox"/>	Specification	Number of Pages	58	<input type="checkbox"/>	CD(s), Number
<input checked="" type="checkbox"/>	Drawing(s)	Number of Sheets	3	<input type="checkbox"/>	Other (specify)
<input type="checkbox"/>	Application Data Sheet. See 37 CFR 1.76				
METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT					
<input checked="" type="checkbox"/>	Applicant claims small entity status. See 37 CFR 1.27.				FILING FEE AMOUNT (\$)
<input checked="" type="checkbox"/>	A check or money order is enclosed to cover the filing fees				
<input type="checkbox"/>	The Commissioner is hereby authorized to charge filing fees or credit any overpayment to Deposit Account Number: []				
<input type="checkbox"/>	Payment by credit card. Form PTO-2038 is attached.				\$100.00
The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.					
<input checked="" type="checkbox"/>	No.				
<input type="checkbox"/>	Yes, the name of the U.S. Government agency and the Government contract number are: _____				

Respectfully submitted,
 SIGNATURE 
 TYPED or PRINTED NAME Bindu Rama Rao
 TELEPHONE 949-234-7027

Date 02/17/2006
 REGISTRATION NO. []
 (if appropriate)
 Docket Number: 101USMD134

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

This collection of information is required by 37 CFR 1.51. The information is used by the public to file (and by the PTO to process) a provisional application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 8 hours to complete, including gathering, preparing, and submitting the complete provisional application to the PTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Washington, D.C. 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Box Provisional Application, Assistant Commissioner for Patents, Washington, D.C. 20231.

60774406



021706

PROVISIONAL APPLICATION COVER SHEET

Additional Page

PTO/SB/16 (02-01)
Approved for use through 10/31/2002. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

Docket Number 101USMD134

INVENTOR(S)/APPLICANT(S)		
Given Name (first and middle (if any))	Family or Surname	Residence (City and either State or Foreign Country)
Bindu Rama	Rao	21 Henley Drive, Laguna Niguel, CA 92677

Number 2 of 2

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

Applicant or Patentee: Bitfone Corp.
Serial or Patent No.: Not assigned
Filed or Issued: Herewith
For:

Attorney: Christopher C. Winslade
Docket No.: 101USMD134

VERIFIED STATEMENT (DECLARATION) CLAIMING SMALL ENTITY STATUS (37 CFR 1.9 (f) and 1.27 (c) -SMALL BUSINESS CONCERN

I hereby declare that I am

- the owner of the small business concern identified below:
- an official of the small business concern empowered to act on behalf of the concern identified below:

Bitfone, Corp.
(a Delaware Corporation)
32451 Golden Lantern, Suite 301
Laguna Niguel, CA 92677
United States of America
949-234-7000

I hereby declare that the above identified small business concern qualifies as a small business concern as defined in 13 CFR 121.3-18, and reproduced in 37 CFR 1.9 (d), for purposes of paying reduced fees under section 41(a) and (b) of Title 35, United States Code, in that the number of employees of the concern, including those of its affiliates, does not exceed 500 persons. For purposes of this statement, (1) the number of employees of the business concern is the average over the previous fiscal year of the concern of the persons employed on a full-time, part-time or temporary basis during each of the pay periods of the fiscal year, and (2) concerns are affiliates of each other when either, directly or indirectly, one concern controls or has the power to control the other, or a third party or parties controls or has the power to control both.

I hereby declare that rights under contract or law have been conveyed to and remain with the small business concern identified above with regard to the invention, entitled DIAGNOSTICS AND MONITORING SERVICES IN A MOBILE NETWORK FOR A MOBILE DEVICE, by inventor(s) Robert C. Daley, Deven P. Shah, Karen Chan and Bindu Rama Rao, described in

- the specification filed herewith.
- application serial no. _____, filed _____.

If the rights held by the above identified small business concern are not exclusive, each individual, concern or organization having rights to the invention is listed below* and no rights to the invention are held by any person, other than the inventor, who could not qualify as small business concern under 37 CFR 1.9 (d) or a nonprofit organization under 37 CFR 1.9 (e).

*NOTE: Separate verified statements are required from each named person, concern or organization having rights to the invention averring to their status as small entities. (37 CFR 1.27)

NAME : Bitfone, Corp.
ADDRESS 32451 Golden Lantern, Suite 301, Laguna Niguel, CA 92677.
 INDIVIDUAL SMALL BUSINESS CONCERN NONPROFIT ORGANIZATION

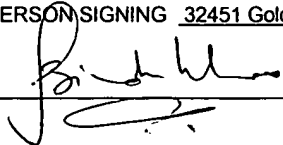
I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate. (37 CFR 1.28 (b))

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application, any patent issuing thereon, or any patent to which this verified statement is directed.

NAME OF PERSON SIGNING Bindu R. Rao

TITLE OF PERSON OTHER THAN OWNER Manager for IPR and Standards

ADDRESS OF PERSON SIGNING 32451 Golden Lantern, Suite 301, Laguna Niguel, CA 92677

SIGNATURE  DATE February 17, 2006

ABSTRACT

The present invention makes it possible to obtain debug and other diagnostic information from mobile devices in the operator network. A Log Management object provides support for logging diagnostic data. A log file is employed to collect information on various device features for which tracing or debugging is turned on in a mobile device. It is also used to selectively collect information on specific events that are monitored, device specific data being collected, network performance data, etc. The diagnostic agent in the mobile device is a client side application that runs on a the mobile device when required (or always as a monitoring application) and which manages and collects tracing information wirelessly to a server using cellular data networks. A diagnostic client can also be downloaded and executed to collect diagnostic data from applications, etc. Traps can also be set and data collected from them. In general, the Log file can be retrieved from then server side in pull or push mode.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

TITLE

**DIAGNOSTICS AND MONITORING SERVICES IN A MOBILE
NETWORK FOR A MOBILE DEVICE**

INVENTOR

**Robert C. Daley
13 Middle Dunstable Road
Nashua, NH 03062
Citizenship: USA**

**Deven P. Shah
5030 East Tenderrow Place, F
Orange, CA 92867
Citizenship: India**

**Karen Chan
Citizenship: Canada**

**Bindu Rama Rao
21 Henley Drive
Laguna Niguel, CA 92677
Citizenship: India**

CERTIFICATE OF EXPRESS MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service "Express Mail Post Office to addressee" Service under 37 C.F.R. Sec. 1.10 addressed to: Assistant Commissioner for Patents, Box Patent Application, Washington, D.C. 20231, on Feb. 17, 2006.

Express Mailing Label No.: EQ165635822US


Bindu R. Rao

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

**TITLE: DIAGNOSTICS AND MONITORING SERVICES IN A
 MOBILE NETWORK FOR A MOBILE DEVICE**

SPECIFICATION

CROSS REFERENCE TO RELATED APPLICATIONS

The present application is a continuation US provisional patent application titled "DEVICE CLIENT SPECIFICATION", attorney docket number 101USMD117, filed on March 21, 2005, which is hereby incorporated by reference in it's entirety

BACKGROUND

1. Technical Field

The present invention relates generally to the management of mobile devices, and, more specifically, to the use of managed objects for diagnostics and monitoring.

2. Related Art

Electronic devices, such as mobile phones and personal digital assistants (PDA's), often contain firmware and application software that are either provided by the manufacturers of the electronic devices, by telecommunication carriers, or by third parties. If firmware or firmware components are to be changed in electronic devices, it is often very tricky to update the firmware components.

It is often difficult to determine what is wrong with a device when a problem is encountered. Quite often, a customer care representative for an operator does not have answers to a customer's problem and is not able to fix it. Determination of problems with a customer's mobile device is a big problem for operators. Answering customer care

calls is quite expensive. Especially so if at the end of such a call, the customer care representative is unable to determine what is wrong with the device.

Different devices have different set of resources, different sets of parameters, etc. Managing mobile devices in a heterogenous network is a huge problem. Figuring out what parameters need to be set or diagnosed as a potential error is also a problem.

Debugging software problems in mobile devices that occur “in the field” is a challenging endeavor. A user or a software developer cannot step through code using a debugger, s/he must rely on debug information recorded in memory of the mobile device during field use of the device in order to diagnose problems with any software in the device. This information may consist of trace information, logs of data taken from the radio subsystem, stack dumps, error flags, or anything the developer has stored in specific memory reserved for debugging purposes. Currently, after trace data is recorded, the information is manually transferred from memory into a PC via a USB cable, and the developer examines the debug data file on the PC. This process suffers from a number of drawbacks, however. First, the user or software developer and the problematic device may not be co-located, so obtaining the debug file is nontrivial. Second, the device users (typically field or beta testers) may be inconsistent about retrieving debug files from the memory of the mobile device in a timely manner, and debug information is overwritten after a period of time, so the user or software developer may never receive the needed debug information. Existing diagnostic clients work on proprietary protocols, and often on a cable/ wire based solution. These are not for diagnosing widely reported problems with millions of deployed devices.

2. Brief Description of the Diagrams

The numerous objects and advantages of the present invention may be better understood by those skilled in the art by reference to the accompanying figures in which:

Figure 1 is a perspective block diagram of a mobile network that is capable of conducting remote diagnostics on an electronic device wherein the electronic device supports several features, including diagnostics features and wherein applications can be downloaded and installed on the mobile device to monitor applications and diagnose problems if needed.

Figure 2 is a perspective block diagram of a section of the DM Tree used by a DM client or diagnostic agent to provide support for diagnostics and monitoring services.

Figure 3 depicts an exemplary log file that supports reporting of diagnostic data.

Appendix A – Diagnostics Support for Treo.

3. Detailed Description of the Diagrams

Figure 1 is a perspective block diagram of a mobile network 105 that is capable of conducting remote diagnostics on an electronic device 107 wherein the electronic device 107 supports several features, including diagnostics features and wherein applications can be downloaded and installed on the electronic device 107 to monitor applications and diagnose problems if needed.

The electronic device 107, which is typically a mobile device 107, can be used to request customer care service via a customer care server 157, using the device capability information as one of the parameters provided. A customer service representative (CSR) provides a service to the customer using the mobile device 107, after determining the device capability information that is retrieved from the mobile device 107, thereby making it unnecessary for a customer to provide such information himself to a CSR. The mobile network 105 is capable of supporting remote diagnostics by a CSR via the customer care server 157. It can also support a diagnostic data collection request from a diagnostic server 129 and return the collected diagnostics data to the diagnostics server 129 or to any other authorized server in the mobile network 105. The customer / subscriber of the mobile device 107 might be having problems and may need some help in diagnosing the problems - the mobile network 105 facilitates diagnosis by a CSR via the customer care server 157, as well as by the diagnostic server 129.

The present invention makes it possible to obtain debugging and other diagnostic information from mobile devices 107 in the operator network. A Log management object (MO) is proposed (in Figure 2) that provides support for collecting and logging diagnostic data. A log file (described in Figure 3) is employed to collect information on

various device features for which diagnosis data collection or tracing / debugging is turned on in a mobile device 107. It is also used to selectively collect information on specific events that are monitored, device specific data being collected, network performance data, etc. The diagnostic agent 171 in the mobile device 107 is a client side application that runs on the mobile device 107 when required (or always as a monitoring application) and which manages and collects tracing information wirelessly to a server using cellular data networks. A diagnostic client 121 can also be downloaded and executed to collect diagnostic data from applications, etc. Traps 125 can also be set and data collected from them. In general, the Log file can be retrieved from then server side in pull or push mode.

The mobile network 105 comprises a device management (DM) server 109, the customer care server 157, a download server 151, the mobile device 107 and a diagnostics server 129. The mobile device 107 is capable of updating an application software 127, an operating system (OS) 119, or a firmware 117 in the mobile device 107 employing an update package delivered by the download server 151. It is also capable of receiving provisioning information from the customer care server 157 to fix configuration problems or reconfigure software and hardware. The device capability information stored in the device is modified as necessary when this occurs.

The mobile device 107 is capable of applying updates using one or more update agents 115 that are each capable of processing update packages or subsets thereof. The mobile device 107 is capable of receiving update packages. It comprises the update agent(s) 115 that is capable of updating the mobile device 107, the diagnostic client 121 that facilitates remote diagnosis and the traps client 125 that facilitates setting traps and

retrieving collected information. The mobile device 107 also comprises a DM client 163 that is capable of interacting with the provisioning client 123, a diagnostic client 121 and a traps client 125. The DM client 163 typically received DM commands from the DM server 109 and implements them. The download server 151 is used to download firmware and software updates.

In one embodiment, a log file is employed to collect information on various device features for which tracing or debugging is turned on. It is also used to selectively collect information on specific events that are monitored, device specific data being collected, network performance data, etc. The present invention makes it possible to obtain debug and other diagnostic information from mobile devices 107 in the field. The diagnostic agent 171 in the mobile device 107 is a client side application that runs on the mobile device when required (or always as a monitoring application) and which manages and collects tracing information wirelessly to a server using cellular data networks 105. It is typically an embedded software capable of collecting data when requested to, and capable of saving it in a log file or in some other storage structure. The diagnostic server 129 is a server side application which receives, processes, and stores / presents the information obtained from the mobile device 107 in an easily readable fashion. In one embodiment, a PC is used by a user to accept the collected tracing information from the mobile device 107 through a USB connection, a Bluetooth connection or via an SD card (such as when a wireless data service for OTA transfer is unavailable).

The diagnostic client 121 is a downloadable diagnostic client that executes specific diagnostic tasks, such as monitoring a task and collecting diagnostics data for it,

collecting configuration data for network connectivity, application configuration, user preferences, etc., or collecting network performance data. The traps client 125 facilitates setting traps, each trap being associated with a collection method and an optional reporting method. Traps can be used to collect data such as errors, faults encountered while operating the mobile device 107, network performance data, call setup data, etc. In one embodiment, the traps client 125 and the diagnostic agent 171 are combined into one embedded diagnostic client component capable of supporting traps as well as collecting diagnostic data and configuration information, etc. for eventual transfer to the diagnostic server 129 or the customer care server 157.

The present invention provides for a device management (DM) approach for mobile diagnostics wherein management objects (MOs) are created and used for diagnosing each feature domain or application, and the diagnostics related MOs comprise of parameters to be monitored, configuration to be set, tracing information to be collected, etc. Each application installed in the mobile device 107 will provide an associated diagnostics MOs, that are stored as part of a configuration or management related MOs (say as subnodes of a associated application MO) for that application. Alternatively, they are stored as subnodes of the DevDetail MO in a DM tree in the mobile device 107.

Operators / service providers associated with the mobile network 105 will be able to enable / disable device tracing of the applications, as needed. For example, even if the mobile device 107 supports debugging of an application settings and application resource usage, the operator may be able to disable (temporarily or permanently) some of the tracing features, or configure them as they need.

The tracing information is retrieved from the device, in one embodiment as a log file, and in another as trap information sent by a trap set on specific devices. In the case of a log file, it can be retrieved as an XML file, as a binary file that is uploaded from the device, as a structured file (formatted as per a well defined format and structure), or as device specific formatted content.

Figure 2 is a perspective block diagram of a section of the DM Tree used by a DM client or Diagnostic client to provide support for diagnostics and monitoring services. The Log MO makes it possible to collect diagnostic information and store it or manipulate it as part of the DM Tree that is managed by the DM client in the mobile device 107.

In general, the present invention provides a method by which the following is supported:

- Logging diagnostic data and retrieving it in pull or push mode.
- Configuring of the diagnostic data collection,
- Configuring and collecting data related to RF parameters, applications, network parameters, connectivity parameters, resource used by applications, etc.
- Optional Preconfigured set of traps that can be activated, enabled or disabled, etc.
- Support for Get command on the Log MO to retrieve diagnostics data via a DM server
- Support for an Exec command on the Log MO in order to start the logging process based on configured set of diagnostic choices and parameters

- Support for Reset of the collected Log file, wherein Reset can delete or reset all collected data that might be necessary.
- Reporting based on size limits and or duration of collection of diagnostics data.
- Support for enabling and disabling diagnostics tasks.

In general, via the Log MO, all parameters settable on device by a user or by the DM server can be monitored, retrieved and saved in the associated log file. In one embodiment, a brief default set of data is logged periodically in the device when enabled – this includes device ID, basic information about device, and tracing information on battery level, dropped calls, connectivity parameters, all provided in the log file in XML format. In another related embodiment, radio diagnostic data and data services provided to a subscriber over IP are collected in the log file. The diagnostic agent is capable of tracing calls and crash handling capabilities.

In one embodiment, the upload of the log file also results in the automatic deletion of the log file in the mobile device 107. In another, the log file is not deleted, even after it has been transferred, until the server side (via the DM server) requests such a delete, such as by means of a Reset node being invoked on the management object associated with the log file (using a Reset parameter on a Log MO).

There are a number of ways that diagnostics data is generated on the mobile device 107. The diagnostic agent 171 and the diagnostic client 121 employ APIs provided on the mobile device 107 to receive debugging data, or some device specific information. They also employ other APIs to store or manage the data on the mobile device 107, optionally transmit the data to an off-device storage area such as an SD card

or SIM card, and optionally transfer the tracing data by OTA means to the diagnostic server 129, such OTA means being proprietary or standard based (such as OMA-DM or http).

In one embodiment, debug messages or trace messages included in the software of the device, (such as during software development) are collected on the mobile device 107 by enabling tracing (remotely from a DM server or locally by user using a UI screen). When tracing is invoked, software modules (such as functions, subroutines, library code, etc.) write specific data into memory or into the log file. In a related embodiment, the trace messages are assembled in memory (RAM) of the mobile device 107 before being written into the log file in FLASH memory. Binary data as well as text data can thus be stored in the log. Error codes encountered are also logged. Thus, tracing can be turned off or turned on (enabled or disabled) using a management object to control tracing. The concept of trace levels is also supported. Tracing can be automatically enabled by the diagnostic agent of the mobile device 107 (or by other agents) when a software is known to crash in the mobile device 107 – such anticipatory tracing makes it possible to collect data for a problem that is expected to occur based upon previous encounters. In one related embodiment, the tracing data transferred by the mobile device to the diagnostic server 129 is processed and provided as viewable data by the diagnostic server 129, for analysis by a human engineer for subsequent corrective steps.

Figure 3 depicts an exemplary log file 307 that supports reporting of diagnostic data. The log file is created, managed and used to store diagnostics related information by the Log MO 207 described with reference to Figure 2 above. The Log MO 207 is managed by means of the DM client 163 in the mobile device 107.

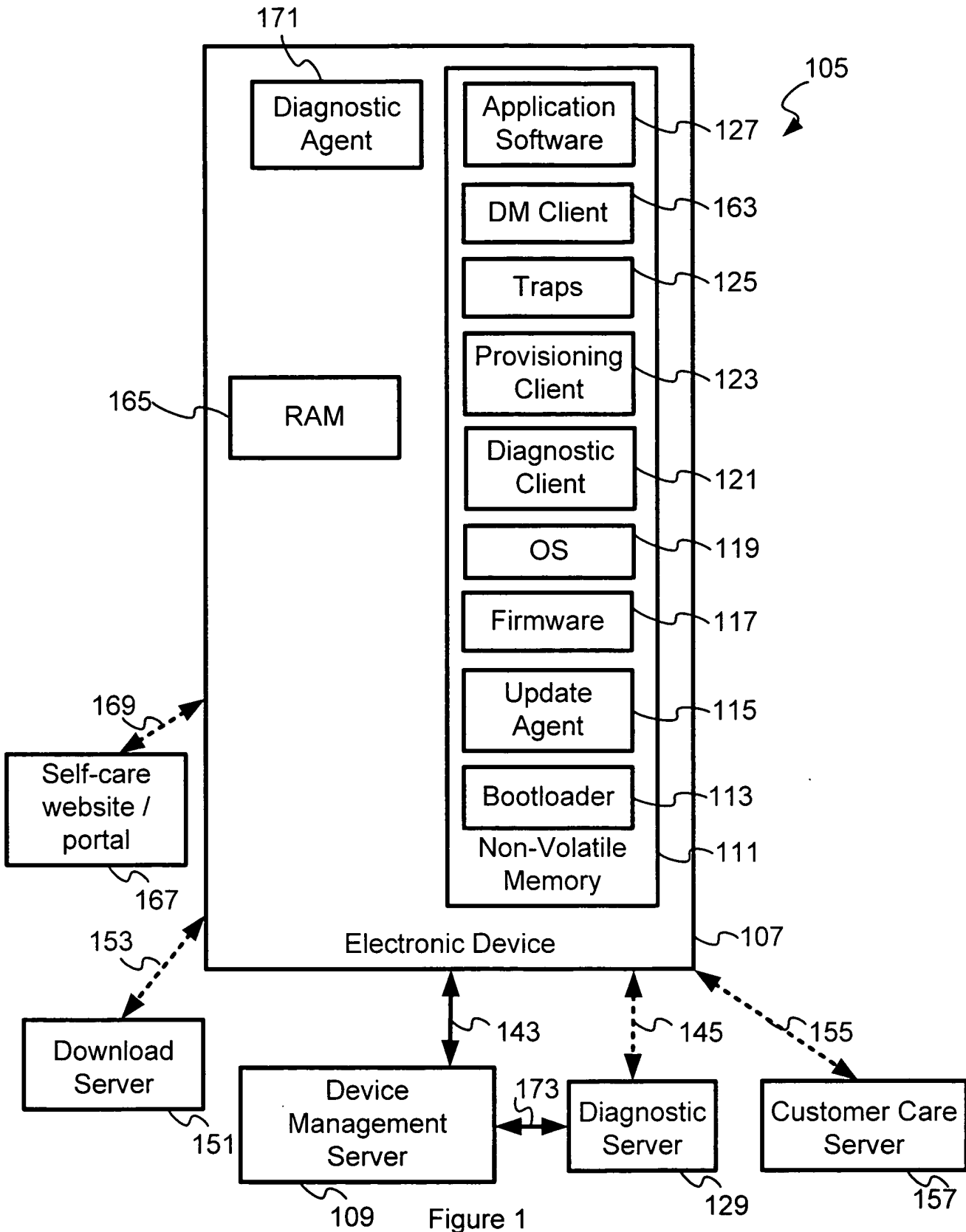
Diagnosis of several user agents or applications need to be supported by a mobile device 107. For example, the user agents for firmware update over-the-air (FOTA), push-to-talk over cellular (PoC), instant messaging (IM), browsing etc. are employed in a typical mobile device 107 and they can encounter problems. Each of these are associated with their own MO (or subnodes) and the diagnosis of the parameters set in these MOs is supported by the diagnostic agent, and associated data collected in stored in the Log file 307, in a structured format. This structured format provides for categorization and storage of diagnostic data, with and without security. Some categories of data require encryption for security, others require encoding for transport flexibility. In addition, diagnostic clients that downloaded and installed also store their data in the same Log file 307.

In one embodiment, the Log file 307 makes it possible to collect categories of diagnostic data that are each associated with a security mechanism and a format for layout. The security mechanism and the format information are also provides in the Log file such that a consumer of the log file can use them to retrieve and process the diagnostic data.

The log file can be digitally signed for security, and its signature 315 is included I the log file 307. The log file can be transferred to a diagnostic server over an ftp connection, an http connection, a bluetooth connection, etc. In one embodiment, the log file 307 is communicated by the DM client in the mobile device 107 over an http connection by means of an http "POST" mechanism. In another related embodiment, the log file 307 is communicated by the DM client 163 in the mobile device 107 by means of

an ftp mechanism. In yet another embodiment, it is communicated over the large object download mechanism of the OMA DM protocol supported by the DM client 163.

Although a system and method according to the present invention has been described in connection with the preferred embodiment, it is not intended to be limited to the specific form set forth herein, but on the contrary, it is intended to cover such alternative, modifications, and equivalents, as can be reasonably included within the spirit and scope of the invention as defined by this disclosure and appended diagrams.



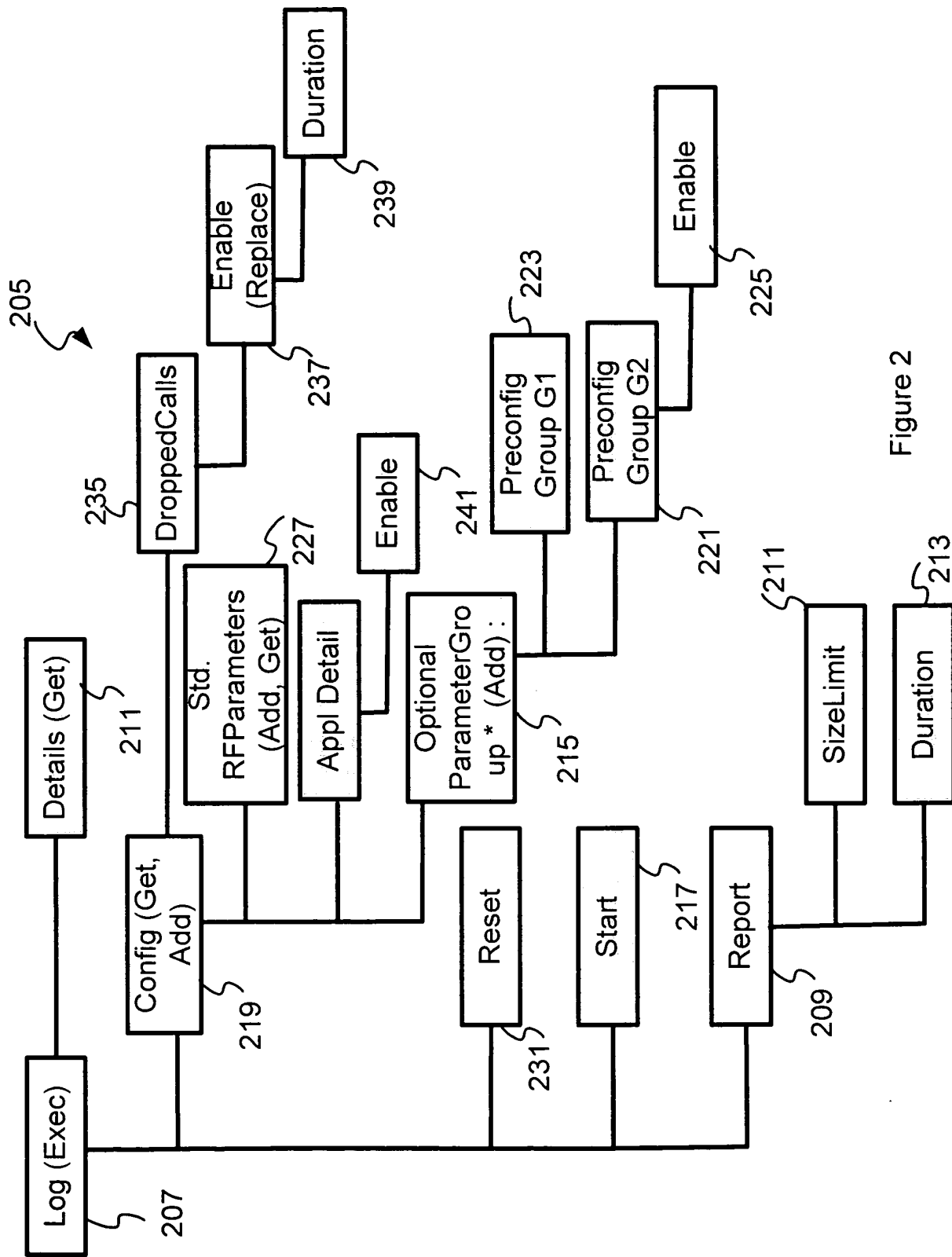


Figure 2

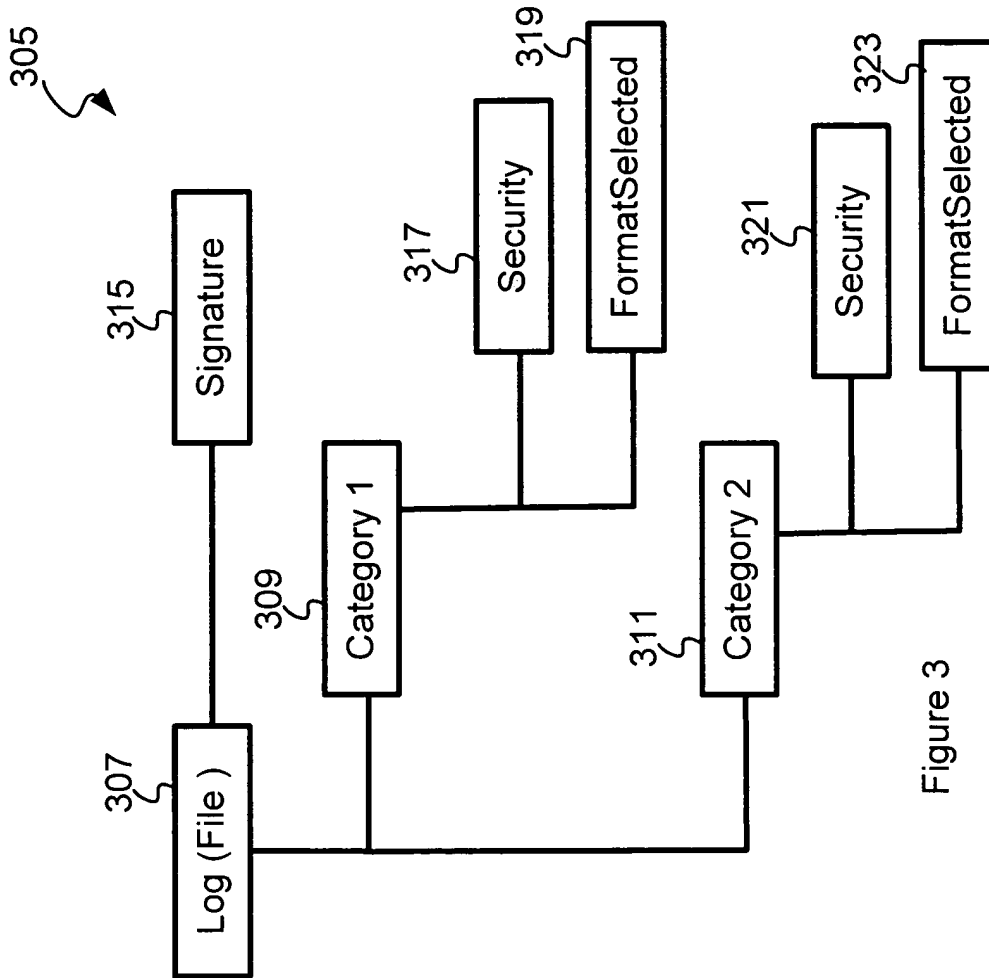


Figure 3



32451 Golden Lantern
Laguna Niguel, CA 92677
(949) 234-7000
<http://www.bitfone.com>

Appendix A – Diagnostics Support for Treo

Treo Trace Architecture and Design

8 February, 2006

Document ID:	Bluebird_DSD_CLI_SVR
Status:*	Proposal
Document Version:	1.0
Author:	Bob Daley, Karen Chan
Reviewed to a Proposal:	Jason Penkethman
Approved:	

Note: We use the following classification of deliverables:

Draft: Unfinished document representing author's views

Proposal: Reviewed by the project manager, represents the views of the project group.

Final: Deliverable that has formally approved by the Senior Vice President of Engineering.

Revision History

Date	Document Version	Description	Author
01/12/2006	0.1	Initial Document	Karen Chan
01/26/2006	0.2	Revised the client section in conformance with information and decisions from the meeting with Palm in Cupertino on 18 January 2006	Bob Daley
01/27/2006	0.3	Revised the server section	Karen Chan
01/30/2006	0.4	Revised the format of the Log Message Transmission header. Expanded on the server section	Karen Chan
01/31/2006	0.5	Added screen shots into the server section	Karen Chan
01/31/2006	0.6	Updated the server section to reflect the discussion from the Feb 1 st meeting with Palm and to include the necessary functionality for Phase 1 Stage 1 of the project	Karen Chan
02/06/2006	0.7	Updated the client section to reflect the changes from the meeting with Palm on 2/1/2006	Bob Daley
02/08/2006	0.8	Added a client requirements section and changed the document layout to put all requirements first	Bob Daley
02/09/2006	0.9	Updated the Trace Data screens to split the Text data onto a different line. Added the Trace Data Detail screen.	Karen Chan
2/10/2006	1.0	Clarification and review	Jason Penkethman

- 1 INTRODUCTION 1**
 - 1.1 Purpose, Intended Audience, and Use of This Document..... 1
 - 1.2 Functionality Overview 1
 - 1.3 Definitions, Acronyms, and Abbreviations 2
 - 1.4 Product Terminology and Definitions 2
 - 1.5 Relevant Documents 3

- 2 OVERVIEW AND REQUIREMENTS 4**
 - 2.1 Architecture Platform for Treo Trace 4
 - 2.2 Treo Trace Client Requirements 4
 - 2.3 Treo Trace Server Requirements 7
 - 2.4 Treo Trace Client Overview..... 9

- 3 TREO TRACE CLIENT OPERATION 11**
 - 3.1 Log Message Receiver (AKA the Logger)..... 11
 - 3.1.1 Log Message Filtering..... 11
 - 3.1.2 Event Codes 13
 - 3.1.3 Real Time Logging Mode 13
 - 3.1.4 Internal Log Memory Management 13
 - 3.1.5 Internal Flash Transfer Thread 15
 - 3.1.6 Log Buffer Contention Management 15
 - 3.1.7 Logger Interfaces 16
 - 3.2 Log Transfer Module (AKA the Uploader) 18
 - 3.2.1 Server Communications Module 20
 - 3.2.2 PC Communications Module 21
 - 3.3 SmartCare Agent for Treo Trace 21
 - 3.4 Device Configuration Agent..... 21

- 4 TREO TRACE SERVER 22**
 - 4.1 Overview of Treo Trace Server 22
 - 4.2 Implementation Details 22

4.3	Devices, Users and User Groups, Subscribers and Subscriber Classes.....	22
4.3.1	User Management	23
4.3.2	Subscriber Management	25
4.3.3	Device Management.....	28
4.4	Trace Data Management	31
4.4.1	Trace Data Transfer Format	31
4.4.2	Servlet and JMS Listener for parsing and receiving Trace Data	33
4.4.3	Screen for filtering and displaying Trace Data for a Device	35
4.4.4	Triggering a Trace Data Transfer on the Device	37
4.4.5	Clearing all Trace Data on the Device	37
4.4.6	Displaying Trace Data for one or more Devices	38
4.5	Displaying and Modifying Device Information and Settings	39

1 Introduction

1.1 Purpose, Intended Audience, and Use of This Document

The purpose of this document is to describe the requirements, architecture and design details for Phase 1, Stage 1 of the Treo Trace project. This document covers the design of the Treo Trace Client and the Treo Trace Server.

The intended audience for this document includes the engineers, professional services engineers, and the quality assurance team who will be working on the project. This document will also be presented to Palm for discussion, validation and acceptance. Upon acceptance by Palm, this document will serve as the basis for mutual understanding of the core design requirements. Subsequent changes and additions will be managed through the Engineering Change Request process (ECR), according to the Bitfone Product Development SOP.

1.2 Functionality Overview

The Phase 1 Treo Trace deliverables are developed in two Stages.

For Stage 1, the required functionality is as follows:

- **Settable device tracing parameters** on the device, including variable buffer sizing, debug levels, components to debug, data transmission method (TCP/IP, SD card, USB), data transmission trigger, and Clear Buffer command.
- **Functional server communication** - Server displays device ID, basic information about device, and trace data file in XML format. All trace data is transferred over the air to the server, and can also be dumped to SD card via the SDIO slot on the device, and to a PC via the device's USB connection.
- **Functional client-side application** - as per Palm Treo Trace SOW, with tracing calls, component and trace level filtering, crash handling, tracing in interrupt routines and multiple threads, and all required data triggering capabilities functional (triggers based on trace level, component, immediate dump request, periodic trigger, or buffer fullness condition).
- **Acceptable real-time performance of the system** - under heavy loading conditions (to be agreed upon) there must not be data corruption, and must not interfere with normal operation of the Treo device. The software must be designed assuming that a client-side call will be made into a trace function every 10ms, and each call will transfer a maximum of 1KB of data.
- **Functionality and performance** – to be demonstrated on the "Camino" device

The contents of this material are confidential and proprietary to Bitfone Corporation and may not be reproduced, published, or disclosed to others without the prior written consent of Bitfone. © 2000-2005 Bitfone, Inc. All Rights Reserved. Bitfone Corporation, the Bitfone logos and all other Bitfone product or service names are trademarks of Bitfone Corporation. This product may be covered by U.S. Patent Nos. 6,941,453 and 6,832,373 and 6,785,707.

1.3 Definitions, Acronyms, and Abbreviations

Table 1: Acronyms and Abbreviations

API	Application programming interface
Device ID	Refers to the IMEI (International Device Equipment Identity), the device ID within a GSM network.
GSM	Global System for Mobile Phones
MSISDN	“Mobile Station Integrated Services Data Network”, commonly used as “mobile phone number on a GSM network”.
SMS	Short Message Service – a simple store and forward text messaging system.
EDGE	Enhanced Data rates for Global Evolution (EDGE) - a radio based high-speed mobile data standard. It allows data transmission speeds of 384 kbps to be achieved when all eight timeslots are used
USB	Universal Serial Bus - an external bus standard that supports data transfer rates of 12 Mbps. A single USB port can be used to connect up to 127 peripheral devices, such as mice, modems, and keyboards.
UTC	Coordinated Universal Time - a time scale that couples Greenwich Mean Time, which is based solely on the Earth's inconsistent rotation rate, with highly accurate atomic time. When atomic time and Earth time approach a one second difference, a leap second is calculated into UTC. UTC was devised on January 1, 1972 and is coordinated in Paris by the International Bureau of Weights and Measures. UTC, like Greenwich Mean Time, is set at 0 degrees longitude on the prime meridian.

1.4 Product Terminology and Definitions

Table 2: Terms and Definitions

Device Instance	Represents a single device identified on the mobile network by its "Device ID" (IMEI or ESN). Under normal circumstances, each device instance is associated with a specific subscriber MSISDN or MIN, except for "borrowed" or "stolen" devices discovered on a GSM mobile network.
User	A person who is allowed to login to the Treo Trace system to

	review trace information and to perform administrative tasks.
User Group	A entity used for grouping users with common attributes
Subscriber	A mobile phone account owner in a carrier's network, known to the system by the mobile phone number.
Subscriber Class	Defines a set of subscriber attributes, common to a set of subscribers. A Subscriber class is generally used to define provisioning attributes associated with geographic region and a class of subscriber service (e.g., consumer, corporate, pre-paid, basic, premier, elite, etc.).

1.5 Relevant Documents

- Treo Trace Statement of Work – Arun Mathias and Chris Cadwell, Oct 6, 2005
- Palm Treo Trace Proposal – Bitfone, Oct 28, 2005
- Exhibit 1, Specification for Schedule A -1 – Bitfone, Dec 20, 2005
- Exhibit 2, Development Schedule to Schedule A -1 – Bitfone, Dec 20, 2005

2 Overview and Requirements

2.1 Architecture Platform for Treo Trace

The Treo Trace Client and Server shall be based on Bitfone's existing diagnostic platform and product line, called SmartCare. The advantage of using SmartCare as the base platform is that it provides a commercially proven framework enabling the server to collect, configure and analyze information of mobile devices over-the-air.

Specific client and server requirements, and design extensions to the SmartCare client, server and architecture, are detailed in this section. Requirements and design for the PC application component for Treo Trace are documented separately.

2.2 Treo Trace Client Requirements

The Treo Trace client represents a significant extension to the existing SmartCare client to support device initiated logging and uploading of logged information to the Treo Trace Server as well as to a USB attached PC, and a removable handset SD card. The specific requirements for Phase 1 Stage 1 of the Treo Trace client are as follows:

FRD ID	Feature and Requirement	Reference
TREO_CLI_01	The Treo Trace client shall provide a native shared library service that can be called from other applications, other shared libraries, and also during interrupt time, to record errors, warnings, and other device events of interest.	
TREO_CLI_02	Each log entry shall be recorded with the component ID, the Sub-component ID, and the severity level of the event. The ability to include optional binary data (e.g., a stack trace) and/or a text message shall be provided.	
TREO_CLI_03	Message logging shall be controlled by a filter which shall specify the component and sub-component pairs and associated severity level of the messages to be logged. If the component and sub-component ID is not matched within the filter, or the severity level of a matched filter entry is lower (less severe) than the message to be logged, the message is discarded and the Logger shall return to its caller immediately.	
TREO_CLI_04	To the greatest extent feasible, logging performance shall not visibly impact normal user device operation. Filtering shall be accomplished via the fastest means possible, such that messages that are not to be logged take as little CPU time as possible. All	

	<p>caller thread operations shall be written to a dedicated RAM buffer by native ARM code. All movement of log messages to other, slower storage targets shall be performed by lower priority threads.</p>	
TREO_CLI_05	<p>The main RAM log buffer shall be implemented as a “circular buffer” such that only the oldest log messages are lost should the processes for moving data to other storage targets fall behind, or these other storage targets not be available.</p>	
TREO_CLI_06	<p>The option to configure a larger internal flash “backup” buffer shall be provided. This internal flash buffer shall be supported by a background thread that regularly moves log data from RAM to internal flash to free RAM log space.</p> <p>This thread shall run at a priority lower than the priority of any caller of the logging service, but higher than the thread that moves log data from internal device memory to an external target.</p> <p>The information moved to internal flash shall be maintained as a set of flash files, and is not to exceed 64KB in length for flash performance reasons. The log message shall not be split across these flash files. Should the amount of internal flash memory be exceeded, the oldest flash file shall be overwritten to ensure that the newest log information is always preserved.</p>	
TREO_CLI_07	<p>A “real time” logging mode shall be provided in which all log messages are written directly to a USB attached PC. When real time mode is active, log information is not logged to internal memory, except possibly for buffering purposes. Once the log message is successfully received by the PC, it is no longer retained in internal memory in the device.</p>	
TREO_CLI_08	<p>It shall be possible, via the use of a “special” key sequence to invoke a dialog for setting log mode, “trigger” criteria, buffer sizes, external upload target(s), and filter criteria. The user interface for this dialog shall be modeled after the existing log configuration dialog with appropriate additions and changes. It must be possible to enter component and sub-component IDs that were not known at the time the operating system was built, when the descriptive name is not known.</p>	
TREO_CLI_09	<p>It shall be possible to receive and process a complete set of configuration information from the Treo Trace server to configure all parameters that are configurable locally.</p>	

TREO_CLI_10	The logging service shall expose a shared library interface for changing and reconfiguring log filters, settings, and trigger criteria. This service can be used both by the device logging configuration dialog, handling server configuration request messages, and also by local device-side diagnostics or testing software.	
TREO_CLI_11	The following “triggers” shall be supported for initiating log data uploading to one or more configured external targets: <ul style="list-style-type: none"> • Specific requests from the server over SMS. • Log data reaching a pre-specified log message threshold. • A message matching specific filter criteria (a specifically flagged component, subcomponent, and severity level). • Periodic triggers, based on a configured time interval. 	
TREO_CLI_12	The logging service shall expose an interface for retrieval of log messages from the internal log buffer. This interface is to be used for device side acceptance testing, to check that the messages that should be in the log are there, with no “unwanted” information.	
TREO_CLI_13	The client shall provide the ability to configure OTA connectivity settings over SMS, using standard SmartCare methods.	
TREO_CLI_14	The logging “transfer” service shall provide the means of moving internal log data (RAM or flash) to one or more of the following external targets, in priority order: <ol style="list-style-type: none"> 1. The Treo Trace Server 2. A USB attached PC 3. A removable device SD card. <p>When multiple targets are configured, “blocks” of data are written to each target in priority order. If all three external targets were to be configured, these blocks would be written in the following order:</p> <ol style="list-style-type: none"> 1. Block 1 to Server 2. Block 1 to PC 3. Block 1 to SD card 4. Block 2 to Server 5. Block 2 to PC 6. etc. 	
TREO_CLI_15	Successful transfer of a block of log data to any external target shall release (erase) the internal storage for that log data.	
TREO_CLI_16	Transfer of log data to an external target shall run at a lower	

	priority than the thread that moves data from RAM to internal flash (when internal flash is configured). This requirement assures that storage for RAM log data continues to be available for new log messages.	
TREO_CLI_17	Each log message transfer shall be provided by a header that includes such information such as: <ul style="list-style-type: none"> • Message version, format, and type • Transfer block message length • Unique "boot" number • Time information in device precision and locality • Device ID (ESN, IMEI, etc.) • Subscriber phone number when available 	
TREO_CLI_18	The logging service shall provide a means of clearing all internal log buffers, either from the configuration dialog or via an SMS request by the server.	

2.3 Treo Trace Server Requirements

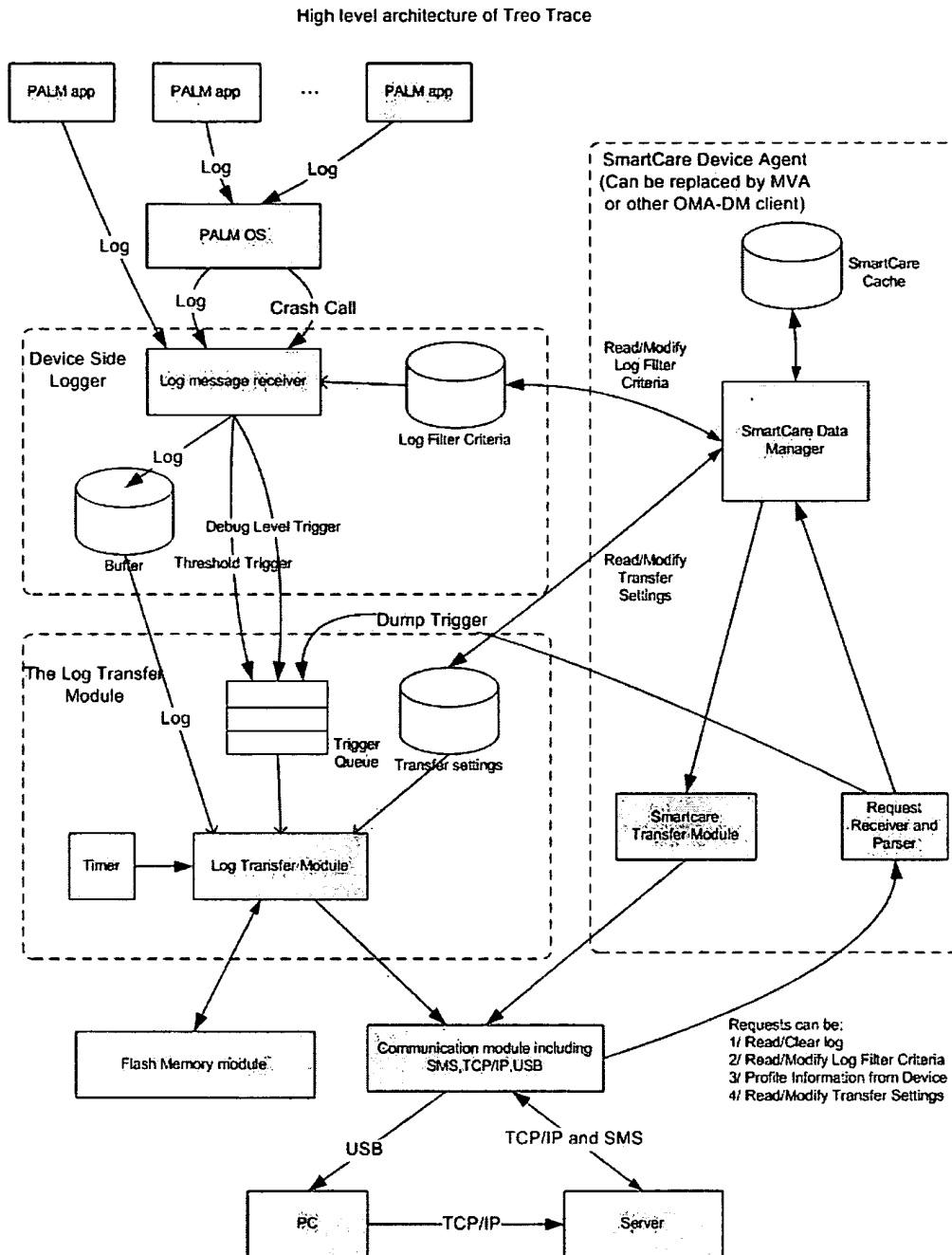
The Treo Trace server represents a new service to be built upon the existing SmartCare server architecture to support the management of, and data collection from, Treo Trace enabled devices. The following are mandatory requirements for the Treo Trace Server for Phase 1 Stage 1 of the project.

FRD ID	Feature and Requirement	Reference
TREO_SVR_01	The Treo Trace server must be able to trigger a log data transfer on the device by sending an SMS notification to the device.	
TREO_SVR_02	The Treo Trace server must be able to clear all log data on the device by sending an SMS notification to the device.	
TREO_SVR_03	The Treo Trace server must be able to collect all available Treo Trace settings and Log Filter settings from the device OTA.	
TREO_SVR_04	The Treo Trace server must be able to configure all available Treo Trace settings and Log Filter settings on the device OTA. Treo Trace settings include all those settings detailed in client requirement TREO_CLI_08.	
TREO_SVR_05	The Treo Trace server must be able to collect Basic Device Info and Data Connection (IP) settings from the device OTA.	
TREO_SVR_06	The Treo Trace server must be able to configure Data Connection	

	settings on the device OTA. If the Data Connection is incorrectly configured and TCP/IP is not available, the server must be able to fix the GPRS Data Connection setting using SMS only.	
TREO_SVR_07	The Treo Trace server must not corrupt trace data received from the device.	
TREO_SVR_08	The Treo Trace server must provide a means for a user to historically search through Trace Data. The search criteria could be a combination of zero or more of the following: Component, Sub-component, Severity, Device Time when Trace data is logged, Server Time when Log Data is received	
TREO_SVR_09	The Treo Trace server must allow for deletion of Trace Data.	
TREO_SVR_10	The Treo Trace server must allow for export of Trace Data to a CSV file.	

2.4 Treo Trace Client Overview

The Treo Trace client was presented in the RFP response with the following diagram:



For ease of explanation, the Treo Trace Client is described here in terms of the following major components, with some names changed and modules consolidated for readability:

1. The “Log Message Receiver” is hereinafter referred simply as the “**Logger**”. The Logger is written as a shared library in native ARM code and, except for interrupts, always runs in the caller’s thread. The logger is supported by a lower priority background thread which is responsible for moving log messages from RAM into internal flash memory to free up RAM log space, **if** space for an internal flash log has been configured. The Logger may operate in one of the following modes:
 - a. A small RAM buffer with a large internal flash “backup” buffer.
 - b. A large RAM buffer with (perhaps) no internal flash memory configured.
 - c. “Real time” mode, where all log messages are sent directly to a USB attached PC, using little or no internal log buffer space. (i.e., once a message is sent to the PC successfully, it is no longer maintained in internal device memory.)
2. The Log Transfer Module is hereinafter referred to as the “**Uploader**” and includes the associated SD card manager, USB PC communication, and server communication modules. The Uploader is responsible for moving log messages from internal log memory to one or more external recipients. The Uploader is also written as a shared library in ARM native code and runs at a priority lower than the thread that moves messages from RAM to internal flash memory. (*When internal flash is configured, it is more important to continue to free up RAM log memory than free up internal flash memory.*)
3. **SmartCare Device Agent** as customized for Treo Trace. This component includes the SmartCare Transfer Module and the Request message receiver defined in the RFP response.. The agent runs as an application and presents an end user interface as necessary. It is preferable that the agent be written in ARM native code, but retaining this module as 68K code is acceptable if necessary to maintain committed schedules
4. SmartCare Data Manager, hereinafter referred to as the Treo Trace “**Configuration Agent**”, is responsible for processing both client initiated and server initiated configuration requests. The configuration agent is an extension to the SmartCare device agent, specific to the Treo Trace project, runs as an application, and present a UI as necessary.

Each of these major components is described in greater detail in the following sections.

3 Treo Trace Client Operation

This section provides the detailed design requirements and overall operation of the Treo Trace client.

3.1 Log Message Receiver (AKA the Logger)

The Logger is the central logging interface for message logging for all Palm OS components and applications. It may be called by any OS or application component, and also under the following special conditions:

1. Upon OS startup, after system crash. In this case, the OS has already saved critical log information at the time of the crash and will make a normal call to the Logger after the OS is restarted. No special handling is required by the Logger for this case, as the recovery logic is to be managed by Palm.
2. The logger may also be called by OS code that is running at “interrupt” time. If the RAM log is in use during such a call, the log data must be captured in a dedicate “interrupt buffer” and moved into the normal log, once access to the RAM log again becomes available.

It is expected that Logger is able to sustain a peak rate of 10KB every 10 milliseconds. Since the Log Message Receiver is such a high duty cycle component, it is very important that its operation does not adversely affect the performance of the Treo device. The prototype for a typical logging call is:

```
int TraceBinWithText      // Main logging function
( BYTE    byLevel,       // Severity Level
  UINT32  u32Component,  // ID of component reporting the event
  UINT32  u32SubComp,    // ID of sub-component reporting the event
  UINT16  u16Datalen,    // Length of optional binary data or 0
  BYTE*   pbyData,       // Optional binary data or 0 (NULL)
  char*   pszTextData ); // Optional text data
```

Several shorter versions of the above call are provided for situations where there is no binary data, and/or no text data. A variant that provides formatting is also provided. Note that no formatting is done for messages that do not “pass” the log filter, to avoid unnecessary overhead.

3.1.1 Log Message Filtering

Upon receiving a logging request, the Logger has to check the following parameters to determine whether or not the message should be logged, based on a set of parameters that match both the message severity and the component and sub-component pair specified by the caller. Thus all of the following criteria must be matched for the message to be logged:

1. **Component and subcomponent ID** – Each component has a unique 32-bit ID, commonly expressed using constants that represent readable characters (e.g., 'COMM'). A component may consist of one or more subcomponents, identified again by a 32-bit integer, which is unique only within the context of its parent component. The logger treats these two parameters as a pair, and the message is not logged unless the filter contains the specific pair of component and subcomponent IDs.
2. **Message Severity** – FAULT, ERROR, WARNING, INFO, DEBUG, represented by the values 0, 1, 2, 3, and 4 respectively. If the severity specified by a Logger call is greater than the severity level established as **part of the log component and subcomponent “filter”**, the message is not logged and the Logger returns immediately.

Note that the severity level “FAULT” shall always be logged, regardless of other criteria, as this level should only be used for the most serious of errors (e.g., a imminent crash is likely).

To check if a message should be logged, the component and subcomponent are combined into the following structure to provide look-up in a sorted list:

```
struct logFilter          // Log filter table entry
{
    uint64 compIDs;       // Component & subcomponent ID pair
    BYTE   bySeverity;    // Severity level
    BYTE   bySetTrigger;  // "1" if this message triggers an upload
};
```

Since 64-bit integers are not generally supported on Palm devices, the following structure is used to represent these values. In the case of component and subcomponent pairs, the component ID is the “high” word and the subcomponent is the “low” word.

```
struct uint64            // Structure for emulating a 64-bit word
{
    UINT32  high;        // High 32-bits, e.g., component ID
    UINT32  low;         // Low 32-bits, e.g., sub-component
};
```

A **single** bsearch is used to look-up the 64-bit combined component and subcomponent pair. If the pair is not found in the pre-specified list of IDs, the Logger returns immediately. If the search succeeds, the value of `bySeverity` in the entry found is used to determine if the message is to be logged. If the severity level specified by the caller is less than or equal to `bySeverity`, the message is logged, otherwise the logger returns immediately.

Any message that passes the filter test may be used to trigger an upload of all log messages held in “internal” log storage. If the value of `bySeverity` has the low order bit set, the logging of this message notifies the Uploader thread to move all internal log data to the appropriate external receivers.

The filter criteria and other logging parameters (e.g., log buffer sizes) may be set by the handset user using a secret key sequence to launch a logging configuration dialog described below. The server may also set these same parameters over-the air using SMS. The logger is built with an internal default set of parameters which are used until the parameters are reset via one of these two mechanisms.

3.1.2 Event Codes

Component independent event code" can be used to capture discreet events such as low/high signal or roaming transitions. Event codes are implemented by using the artificial component ID 'EVNT', and assigning event-specific codes as subcomponent IDs. This allows a Logger call to specify the system wide unique "event" being reported, regardless of the component actually reporting the event. An event code could be assigned to the 'EVNT' component for "call drop" and another for "Call Origination Failure" occurring anywhere within the handset.

A list of common event codes should be established as soon as possible to allow developers to be able to make use of them.

3.1.3 Real Time Logging Mode

When real time logging mode is turned on, all logging messages are sent directly to the USB attached PC and not written to internal log memory. If the USB connection to the PC is lost, or becomes unresponsive, log messages are again written to internal log message memory to avoid log message data loss. If the PC connection again becomes available, real time logging to the PC resumes, but messages written to internal memory are not be resent to the PC.

3.1.4 Internal Log Memory Management

For highest performance, the Logger always writes new log messages sequentially into a dedicated, pre-allocated circular RAM buffer. The bytes of each log message are written sequentially into the RAM buffer using the following format:

```
struct logEntry
{
    BYTE    byTag;           // Designates message type and version
    BYTE    byLevel;        // Severity level of this message
    UINT16  u16Sequence;    // Sequence # 1-FFFF, zero for "reset"
    uint64  u64Time;        // Time when this event was reported
    UINT32  u32Component;   // ID of component reporting this event
    UINT32  u32SubComp;     // ID of sub-component reporting this event
    UINT16  u16OptDataLen;  // Optional data length, 0 == none
    UINT16  u16OptTextLen;  // Optional string text length, 0 == none
    BYTE    byOptData[1];  // -> first byte of any optional data
};
```

The 64-bit time is stored as two 32-bit numbers and represents a device-specific, high precision time value. Such time values may be in local or UTC time, and may be subject to device user intervention. Thus the time value is mostly useful to record the relationship of a series of log messages which may occur over a very short period of time. For performance reasons, conversion of log entry time values to a more readable format is handled by the Treo Trace server, not be the client. (Need the method from Palm for getting the high precision time value.)

For highest performance and to save message space, all information is maintained in binary. No indexes or other form of pointers are used to walk through all the messages. The following code walks through all messages in the log message buffer (no consideration here for wrap around messages in a circular buffer):

```
//-- Walk through all current log entries
```

```
int ndx = firstLogByte;
while (ndx <= lastLogByte)
{
    //-- Get pointer to the next log entry in the log
    logEntry* pEvt = (logEntry*) &pLog[ndx];

    //-- Do something with this log entry
    ...

    //-- Advance to the next possible log entry
    ndx += (sizeof (logEntry) - 1 +
           pEvt->u32OptDataLen + pEvt->u16OptTextLen);
}
```

The RAM buffer is managed as a sequential, circular buffer. In the case where there is no place to save the contents of the RAM buffer, older log entries are sacrificed to make room for new entries. When the RAM buffer is unloaded, the entire set of messages available at the start of the unload operation are moved to the new location, and the contiguous memory freed can then be re-used. Thru the use of a pair of “read and write Mutexes”, the uploading can proceed to move older log messages while the Logger is adding new entries, as long as the set of log messages being moved are protected from being overwritten.

The Logger uses a log threshold to trigger the uploading of log data in an attempt to free space for new messages. If internal flash memory has been configured, the Logger notifies the Internal Flash Transfer thread to move the RAM messages to internal flash memory. If there is no internal flash memory configured, the Logger notifies the Uploader to move log data to the appropriate external logging target(s). In either case, the goal is to continue to free RAM log space for new messages.

Moving data from RAM to internal flash, when configured, is an ongoing process, intended to free space in a much smaller RAM buffer. Moving data from internal memory to one or more external targets is not automatically triggered by the movement of log data from RAM to internal flash, as the log data is still considered “internal” to the logging service. Uploading to the **external** target(s) is triggered by one of the following events:

1. **Threshold Based** – The internal log threshold (combined RAM and optional Flash) is reached.
2. **Event Based** – A specific event has occurred for which its “trigger” flag has set.
3. **Periodic** – A specified time period has elapsed.
4. **Server Requested** -- The server has specifically requested an immediate upload.

Note that the thread that uploads internal log data (RAM or internal flash) to an external recipient runs at a lower priority than the thread that moves RAM log data to internal flash memory. This is intended to give priority to the thread that works continuously to free RAM memory by moving data to internal flash. When internal flash is configured, log data will move from RAM to internal flash, and later from internal flash to the external target(s). The Uploader will not need to access RAM log data, except in the case where internal flash has not been configured.

3.1.5 Internal Flash Transfer Thread

The internal flash transfer module is responsible for backing up the RAM log buffer to internal flash memory to free space in the RAM buffer, moving the log data to memory that will not be lost if the handset battery is removed. The flash transfer module only operates when internal flash “backup” space has been allocated in flash memory and one of the following conditions exist:

1. The RAM buffer has passed a pre-determined threshold
2. A timer indicates that it is now time to move data from RAM to flash for safe keeping. Note that “continuous mode” is achieved by setting the time interval to zero.

Backup flash data storage is managed as a variable number of 64KB files for performance reasons, since adding to the end of a very long flash file can be quite time consuming. The files are named “BK_001” through “BK_xxx” to provide xxx times 64KB of flash backup storage. If all backup files become full, the oldest backup file is overwritten, such that only the oldest log data is lost.

Each file contains a set of contiguous and **complete** log messages, preceded by a message header, which cannot be completely filled in until the final size is known (refer to the Uploader section for the format of the log message transfer header). Log messages must not be split across files. It is permissible to exceed the 64KB file size to complete a log message, but once a file exceeds 64Kb, no more messages may be added to it. Each 64KB file is maintained in the same format as it will be transferred to the final external target(s).

The Palm operating system will provide an every increasing boot number that is to be written into the log message transfer header. Whenever this boot number is observed to be different than the last boot number recorded by the Internal Transfer Thread, a new flash file must be started with the new boot number written to the log message transfer header.

As each block of log memory is moved to internal flash, the current flash file size maintained in the file directory must be used to determine where to write the next block of log information. This value may be cached for efficiency, but any cached value must be discarded if the device is rebooted. We are assuming here that a system crash or power loss will not end up with a corrupted directory entry, and that the flash file system only updates file length after the information is successfully written.

3.1.6 Log Buffer Contention Management

All writing to the RAM buffer is protected by a Mutex to prevent two or more threads from writing to the buffer at the same time. The holding time for this Mutex must be kept to an absolute minimum, and protect only the period of time necessary to write the bytes into the RAM buffer and update the counters. Any more extensive processing must be done on the stack of the calling thread.

When the Logger is called by an Interrupt thread (using an interrupt Specific entry point), special handling is required on Palm OS and may be useful on other OS platforms as well. When the Logger is operating during an interrupt, it can try to obtain the log Mutex, but cannot “wait” for it if another thread has it. In such a case, the Logger must use a dedicated “interrupt buffer” to store the log data. Space must be provided for approximately 5 such interrupt messages before interrupt data might potentially be lost.

When the Logger releases the RAM buffer Mutex, it checks to see if any log messages have been placed in the interrupt buffer, and if so, moves them into the main RAM buffer to free the interrupt buffer space. When updating the counters that control the interrupt buffer, the Logger must (very) briefly suspend interrupts using a method to be provided by Palm for this specific purpose, to prevent data corruption of these counters.

3.1.7 Logger Interfaces

The main Logger code any all performance sensitive code must be written in native (ARM) code. Although we would like all Treo Trace code to be written as native code, some older 68K code may need to be used in the interest of delivery time. The native code interface to the Logger is represented by the following function prototypes:

```
int SetFilters          // Set trace filtering criteria
( logFilter* pFilter,  // List of components+sub-components to log
  int      nFilter ); // .. including severity and trigger flag
```

The format of the filter list is as follows:

```
struct logFilter      // Format of a single filter list entry
{
  uint64  compIDs;    // Component + subcomponent ID pair
  BYTE    bySeverity; // Severity level
  BYTE    bySetTrigger; // '1' if this message triggers an upload
};
```

Logging functions:

```
BOOL TraceFilter      // Test if a given message will be logged
( BYTE  byLevel,      // Severity Level
  UINT32 u32Component, // ID of component reporting the event
  UINT32 u32SubComp ); // ID of sub-component reporting the event

int TraceBinWithText  // Main logging function
( BYTE  byLevel,      // Severity Level
  UINT32 u32Component, // ID of component reporting the event
  UINT32 u32SubComp,   // ID of sub-component reporting the event
  UINT16 u16Datalen,  // Length of optional binary data or 0
  BYTE*  pbyData,     // Optional binary data or 0 (NULL)
  char*  pszTextData ); // Optional text data

int TraceBinFormat    // Log full message with formatted text
( BYTE  byLevel,      // Severity Level
  UINT32 u32Component, // ID of component reporting the event
```

```
    UINT32 u32SubComp,    // ID of sub-component reporting the event
    UINT16 u16Datalen,   // Length of optional binary data or 0
    BYTE*  pbyData,      // Optional binary data or 0 (NULL)
    char*  pszFormat,    // Optional format statement
    ... );               // Optional sprintf parameters

int TraceFormat          // Log message with formatted text only
( BYTE  byLevel,        // Severity Level
  UINT32 u32Component,  // ID of component reporting the event
  UINT32 u32SubComp,    // ID of sub-component reporting the event
  char*  pszFormat,    // Optional format statement
  ... );               // Optional sprintf parameters

int TraceText           // Log message with string text only
( BYTE  byLevel,        // Severity Level
  UINT32 u32Component,  // ID of component reporting the event
  UINT32 u32SubComp,    // ID of sub-component reporting the event
  char*  pszTextData ); // Optional text data

int TraceBin            // Log message with binary data only
( BYTE  byLevel,        // Severity Level
  UINT32 u32Component,  // ID of component reporting the event
  UINT32 u32SubComp,    // ID of sub-component reporting the event
  UINT16 u16Datalen,   // Length of optional binary data or 0
  BYTE*  pbyData );    // Optional binary data or 0 (NULL)

int TraceEvent          // Log simple event message
( BYTE  byLevel,        // Severity Level
  UINT32 u32Component,  // ID of component reporting the event
  UINT32 u32SubComp );  // ID of sub-component reporting the event
```

A separate set of equivalent shim functions are provided to allow older 68K code to call the above native functions. The shim functions convert parameters and other 68K conventions and pass the call on to the native Logger functions for processing. Conversion of big Endian to little Endian format is also performed by the shim functions, such that the logger will always receive parameters in native ARM format, little Endian. Palm will provide sample code of what such shim functions need to do, based on the shims used for an older logging service.

The following variation of the above logging calls is to be used exclusively by Palm code that is running at interrupt time.

```
int TraceOnInterrupt    // Logging function for interrupts only
( BYTE  byLevel,        // Severity Level
  UINT32 u32Component,  // ID of component reporting the event
  UINT32 u32SubComp,    // ID of sub-component reporting the event
  UINT16 u16Datalen,   // Length of optional binary data or 0
  BYTE*  pbyData,      // Optional binary data or 0 (NULL)
  char*  pszTextData ); // Optional text data
```

When the Logger receives this call, it knows that it is being called at interrupt time and must use the interrupt specific logic detailed above.

The following entry point can be used to enumerate (list) all currently available log entries in internal log memory, RAM and internal flash:

```
int EnumLogEntries          // Enumerate log entries
    ( void Function (logEntry*) ); // User specified callback function
```

In stage 1, this function may enumerate only the contents of the RAM log buffer. Full support is planned for stage 2 if required.

3.2 Log Transfer Module (AKA the Uploader)

The Uploader is responsible for the actual transfer of log data from internal device storage (RAM or Flash) to one or more external targets. The Uploader runs at a lower priority than the thread that transfers RAM log data to internal flash. Thus, when internal flash is configured, the Uploader works by moving internal flash log files to the appropriate external targets. When no internal flash is configured, the Uploader works by moving log data from the RAM buffer directly to the external targets. Depending on the presence or absence of internal flash memory, the Upload will either:

1. When there is no internal flash configured – move data directly from RAM to the external targets.
2. When internal flash is configured – move data (flash files) from internal flash memory to the external targets

Since the Uploader runs at lower priority than the thread that transfers RAM log data to internal flash, and since that move to internal flash is non-blocking, the log data to be transferred externally will already have been stored as files in internal flash memory.

All such external transfers result from some type of trigger, accompanied by a notification (mailbox message) to the Uploader thread. The various triggers are specified in the [Internal Log Memory Management](#) section above. Triggers can be pre-specified during Logger configuration, or can result from an explicit SMS message request from the Treo Trace server. The Uploader services each transfer trigger using one or more of the following transfer methods as selected during Logger configuration, in the following priority order:

1. Transferring the data to the Treo Trace server OTA using TCP/IP
2. Transferring the data to an attached PC via a USB connection
3. Writing the data to an external SD card.

In each case, the transfer module uses one of the components described below to accomplish the actual data transfer, but maintains control of the overall transfer process. The Log Transfer module is capable of establishing a data connection when needed for data transfer to the server, and will take precautions to not interrupt an existing voice connection when establishing the needed data connection. The design of the Uploader will focus on minimizing the data connection and transfer times in order to minimize the visible impact to the user, since on some Palm devices, voice calls cannot be made while a data connection is in session.

Since multiple targets can be selected, the Uploader moves data in 64K blocks (same as internal flash file size) to each selected target in priority order. For example, if both the PC and SD card are selected, and there are three (3) 64K files sitting in internal flash, the following sequence would be used:

1. 1st 64K block to the PC
2. 1st 64K block to SD
3. 2nd 64K block to the PC
4. 2nd 64K block to SD
5. 3rd 64K block to the PC
6. 3rd 64K block to SD

If one of several targets is unavailable for any reason, but another is, data is written to the available target, while the unavailable target will not receive the lost log data. There is no complex logic to attempt to preserve data for an unavailable target when other targets are available. If no selected target is available (e.g., on the unavailable target is selected) the log data is retained within internal device memory. Once the log data is transferred successfully to at least one target, the data is released from device memory.

Note that 64K is a threshold, not a precise, fixed unit of transfer. Since a log message is not allowed to be split across internal flash files or transfer message blocks, it is acceptable that some transfer messages can be a bit bigger than 64K, to the extent that the last log message in the buffer crosses the 64KB threshold. Each unit of log message transfer is prefixed with the following message header:

```
struct msgHeader
{
    UINT16  u16MsgVersion;    // Message format version
    UINT16  u16MsgType;      // Message type
    UINT16  u16DevType;      // Device type
    UINT16  u16DevVersion;   // Device version, most likely the SW version
    UINT32  u32MsgLength;    // Message length, INCLUDING this header
    uint64  u64DeviceTime;   // Device dependent time of this message
    INT16   i16UTCOffset;    // Local time offset from UTC in minutes
    UINT16  u16BootNumber;   // Increments each time device is rebooted
    char    szEventID;       // SmartCare Event ID, zero terminated
    char    szDeviceID;     // Device ID e.g., IMEI/ESN, zero terminated
    char    szMSISDN;       // MSISDN or empty string, zero terminated
};
```

When a new internal flash file is written, a complete header is written to the front of the file, except that the final file size will not be known until the last set of log messages is written. Until the final file size is known, the value of `u32MsgLength` is set to zero. When moving these files to an external target, Upload must copy the header from the flash file, fill in the message (file) length, transfer (write) the corrected header to the target, followed by the log data, which can be

written directly from flash memory. In the case where no internal flash memory has been configured, the Uploader must format the above header information directly from the RAM log data.

The boot number, as maintained by the OS, is placed in the header at the time the header is first created. When using internal flash memory, the change of a boot number always starts a new file. The time value is device specific, it is in a device specific precision, and it may represent either device local time or UTC time. At the time the transfer header is created, the device time and a value representing the number of minutes offset from UTC can be calculated as follows:

1. Local time is read from the OS, usually in seconds
2. UTC, again in seconds, is calculated using available platform services
3. UTC time is subtracted from local time, and divided by 60 seconds
4. The resulting value is placed in the header as `i16UTCOffset`

The reason this is done at header creation time rather than at log time is that the conversion from local to UTC (or vice versa) often requires access to preferences that are stored in flash memory, which would dramatically slow down the Logger when writing new log message to the RAM log buffer.

The Uploader makes use of one or more of the following modules to move data to the selected external target(s):

1. **The Server Communications Module** writes the log data to the Treo Trace server. (*In this case, SmartCare configuration services must have been used to properly configure the device for OTA communication.*)
2. **PC Communications Module** writes the log data directly to a USB attached PC, in units of "64K" files/messages.
3. **The SD Card Manager** writes data to an SD card as series of "64K" files, which can be imported to a PC for processing, and possible later upload from the PC to the Treo Trace server. Note that when moving data from internal flash to the SD card, all is required is a series of file copies, in which the value of `u32MsgLength` in the file/message header is updated from the flash file directory entry.

3.2.1 Server Communications Module

The Server Communications Module contains the low level code needed to support wireless communication over TCP/IP and HTTP. This module is not concerned with message formatting or content, only the API's and protocol specifics necessary to transfer each unit of "64K" log data reliably. Note that although SMS can be used for handset configuration and control, SMS is not appropriate and not used for transferring large amounts of binary log data. For OS platforms that support TCP/IP over USB, this module may be used for USB data transfers as well. USB Attached PC.

3.2.2 PC Communications Module

For Palm OS, there is no HTTP interface for communication using HTTP protocol. For Camino Palm has agreed to provide a file transfer interface that is to be used to move 64KB files between the device and the PC over USB. It appears likely that the means of communication between the handset and the PC will vary depending on the handset platform. In any case, the actual binary log information transferred is exactly the same content and format as when communicating over the Internet, exclusive of protocol specifics.

When in “real time” mode, a different interface is needed between the device and the PC. For Camino Palm has agreed to provide a RPC-like interface that is to be used to communicate real time between the device and the PC over USB.

3.3 SmartCare Agent for Treo Trace

When the Treo Trace client is first in operation, it does may not know how to communicate with its server. The Treo Trace client requires a subset of the SmartCare IP configuration settings to establish OTA connectivity to the server. This information can be pre-provisioned at manufacturing time, or configured OTA from the Treo Trace server. A subset version of the standard SmartCare agent is used to manage the OTA configuration settings from the Treo Trace server. Note that this device agent, and all components of the Treo Trace client can either be downloaded to the device, or embedded using standard build and manufacturing procedures.

3.4 Device Configuration Agent

Until the device is configured, either from the handset dialog, or via the Treo Trace server OTA, the Trace service will operate with a minimal set of default settings. The logger can be re-configured either via an SMS message from the Treo Trace server or via a local handset dialog, invoked by a ‘secret’ keystroke sequence. The Logger configuration dialog will present the handset user with a list of all built-in component names and IDs, and allow the user to select the component and subcomponent pairs that the user wants to log, the severity level to be logged, and whether or not such an event should trigger an upload of log information. If all the subcomponents of a given component need to have their messages logged, each subcomponent must be selected. To make this easier for the user, a simple check all capability is be provided.

The dialog should also allow component and subcomponent IDs (e.g., ‘COMM’ and ‘RECV’) to be entered directly to be able to turn on logging for components that have not identified themselves to the OS loader. Once the user has selected a set of component and subcomponent filters, the new filter list is passed to the Logger via the SetFilters function, along with the user selected severity level.

The configuration dialog also provides a means for reconfiguring other Logger parameters such as the amount of RAM and optional internal flash memory to be allocated to the Logger. It is expected that changing memory parameters may introduce a few seconds delay, in order for the Logger to reconfigure log memory usage. Note that reducing the amount of memory allocated to the logger may result in the loss of some older log data.

The device-side logger configuration dialog will be seen as an extension of the existing log configuration dialog. Palm will provide the source for the current dialog implementation to be used as a model for the Treo Trace version of the configuration dialog.

4 Treo Trace Server

4.1 Overview of Treo Trace Server

The Treo Trace server is used for collecting and persisting log data from groups of devices and for OTA configuration of device-side Treo Trace Settings.

The Treo Trace server will use existing SmartCare functionality for OTA collection and modification of Treo Trace settings on the devices. The Treo Trace server initiates a collection or configuration request on the device by sending a special SMS notification to the device. Upon receiving the notification, the device agent wakes up, collects and/or configure device settings and automatically tries the most efficient delivery method (i.e. TCP/IP) to transfer data back to the server. If TCP/IP connectivity is not available, the device agent will automatically revert to SMS.

The Treo Trace server is also responsible for receiving and persisting log data collected on the device. The log information could come from two channels: 1/ directly from the device or 2/ from the PC application. Because of the size of the log data, it is not practical to use SMS as a transport mechanism. Instead, the log information should always be transferred to the server through TCP/IP.

4.2 Implementation Details

The Treo Trace server will be implemented as a J2EE application with a web-based user interface. The Struts Action framework will be used for building the web-based user interface.

The application server used for Phase 1 Stage 1 will be JBoss Application Server 4.0.3 SP1.

The database used will be Oracle 9i or higher.

4.3 Devices, Users and User Groups, Subscribers and Subscriber Classes

A subscriber is a mobile phone account owner in a carrier's network and is characterized by the mobile number. Subscribers that have common characteristics belong to the same Subscriber Class.

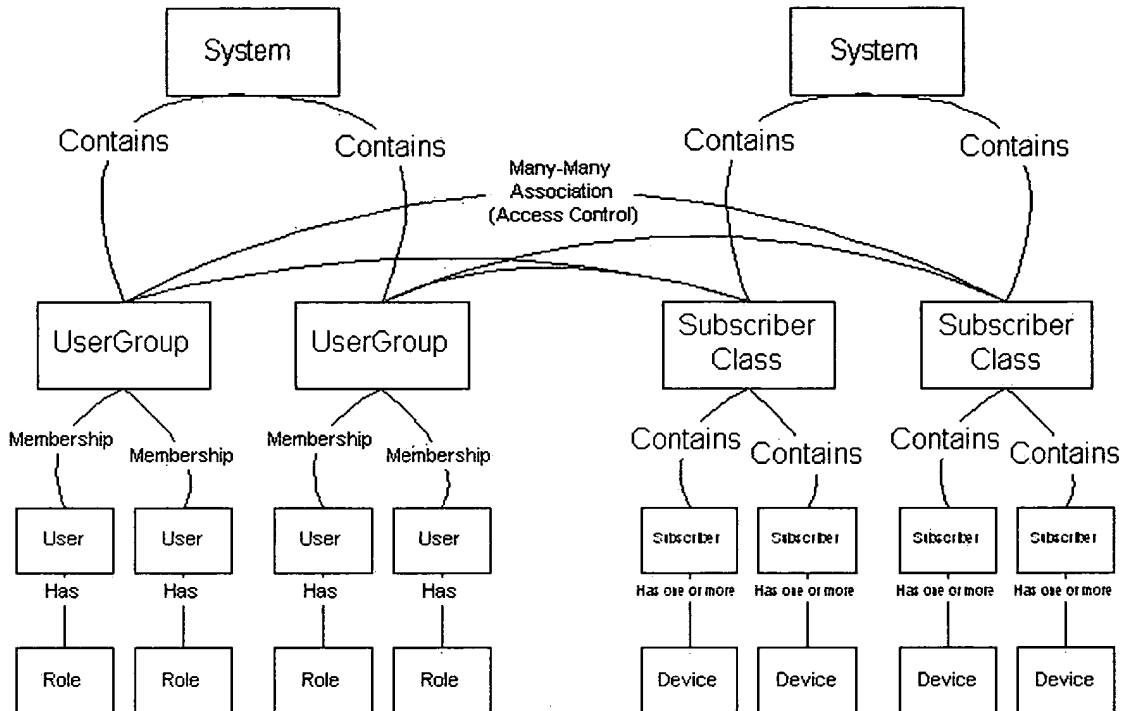
A device is a mobile phone and is characterized by the IMEI/ESN.

A subscriber can have one or more devices, but there will only be one 'last known device' associated with each subscriber.

A user is a person who logs in to the Treo Trace system to retrieve device trace logs and to perform other administrative tasks. There will be two types of users in the system - the Super User and the Regular User.

Each user will belong to a user group and each user group will be allowed to manage subscribers belonging to one or more Subscriber Classes. A user is only allowed to view and manage those subscribers that his/her user group has access to. The Super User will belong to a special DEFAULT user group and will have access to all subscribers and functionality within the system.

The above relationship is best explained by the following diagram:



In Phase 1 Stage 1, all users will be granted the Super User role. In addition, all users will belong to the same User Group and all Subscribers will belong to the same Subscriber Class. This implies that all users will be able to view all subscribers and devices in the system. User and Trace Data Partitioning will be implemented in Phase 1 Stage 2.

4.3.1 User Management

A User has the following properties: Login Name, Password, First Name, Last Name, Email, Work Phone, Mobile Phone, User Group, Time Zone, and Description.

For Phase 1 Stage 1, a default User Group will be preloaded into the database and the Group dropdown will only show the preloaded User Group.

There will be screens for Searching, Creating, Editing and Deleting Users.

User

New User
Edit User
Subscriber
Device
Trace Statistics

First Name: Last Name:

User Name: Email:

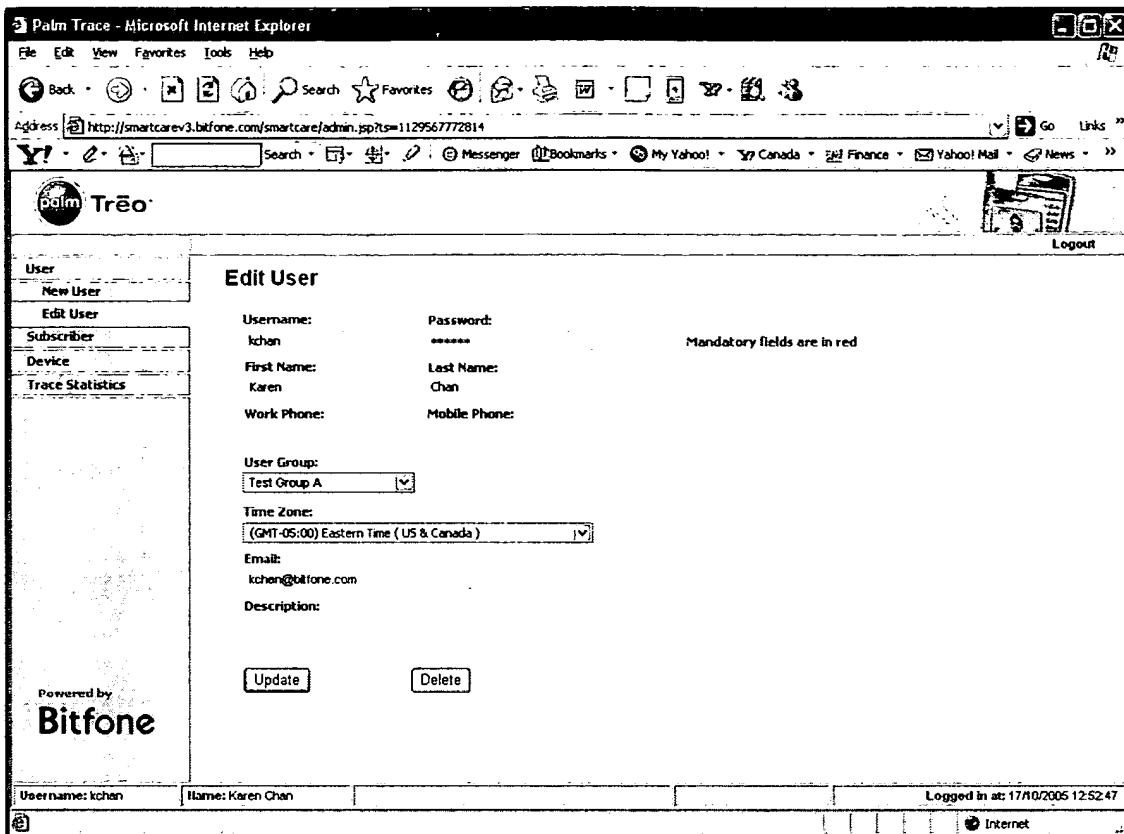
User Group:

1 records found. Show records per page Page 1 of 1

User Name	First Name	Last Name	Email	User Group
kchan	Karen	Chan	kchan@bitfone.com	Test Group A

Powered by **Bitfone**

Username: kchan Name: Karen Chan Logged in at: 17/10/2005 12:52:47



4.3.2 Subscriber Management

A subscriber has the following properties: Subscriber Class, First Name, Last Name, MSISDN, Email, Home Phone Number, and Address.

For Phase 1 Stage 1, a Subscriber Class will be preloaded into the database and the Subscriber Class dropdown will only show the preloaded Subscriber Class.

A subscriber can also have one or more devices. However, there will only be one “Last Known Device” associated with a subscriber.

There will be screens for Searching, Creating, Editing, Deleting and Importing Subscribers.

Microsoft Internet Explorer

Address: http://smartcarev3.bitfone.com/smartcare/admin.jsp?ts=1129567772814

Logout

User

Subscriber

New Subscriber

Edit Subscriber

Import Subscriber

Device

Trace Statistics

Mobile Number: Subscriber Class: Test Group A [clear] [Search]

First Name: Last Name:

Email:

1 records found. Show 10 records per page Page 1 of 1

Mobile Number	First Name	Last Name	Email	Subscriber Class
kchan	Karen	Chan	kchan@bitfone.com	Test Group A

Powered by Bitfone

User name: kchan Name: Karen Chan Logged in at: 17/10/2005 12:52:47

Bitfone Treo

Logout

User

- Subscriber
- New Subscriber
- Edit Subscriber
- Import Subscriber
- Device
- Trace Statistics

Edit Subscriber

Mandatory fields are in red

First Name: Karen
Last Name: Chan
Mobile Number: + 14168295682
Contact Number:
Address 1:
Address 2:
City:
Province:
Country:
Postal Code:
Email:
Subscriber Class: T-Mobile

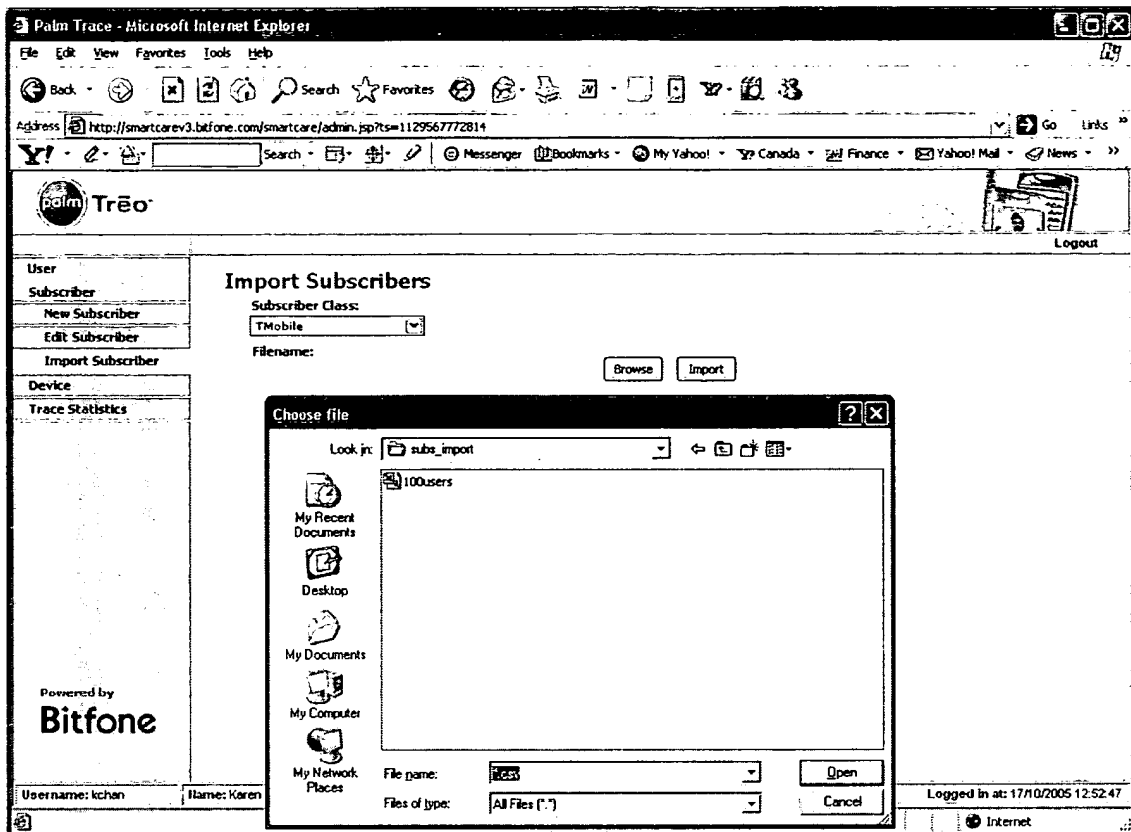
Devices:

Last Known Device	Device ID (IMEI/ESN)	Device Type
<input type="radio"/>	IMEI:004400016848428	Treo Camino
<input checked="" type="radio"/>	IMEI:004400008398725	Treo Camino

Delete Save

Powered by **Bitfone**

Username: kchan Name: Karen Chan Logged in at: 17/10/2005 12:52:47



To import a list of subscribers and to associate them with a device, click on Browse, select the CSV file from the file system and click Import.

The CSV file has to be in the following format:

[mobile number1],[optional IMEI/ESN1]

[mobile number2],[optional IMEI/ESN2]

...

If the IMEI/ESN is supplied, then the system will try to associate the subscriber with the device. If the IMEI/ESN is not supplied or is not found in the system, the system will only create the subscriber record.

4.3.3 Device Management

A device has the following properties: Device ID (IMEI/ESN), Device Type, Description and Associated Subscriber. All Device Types have to be preloaded into the Treo Trace database.

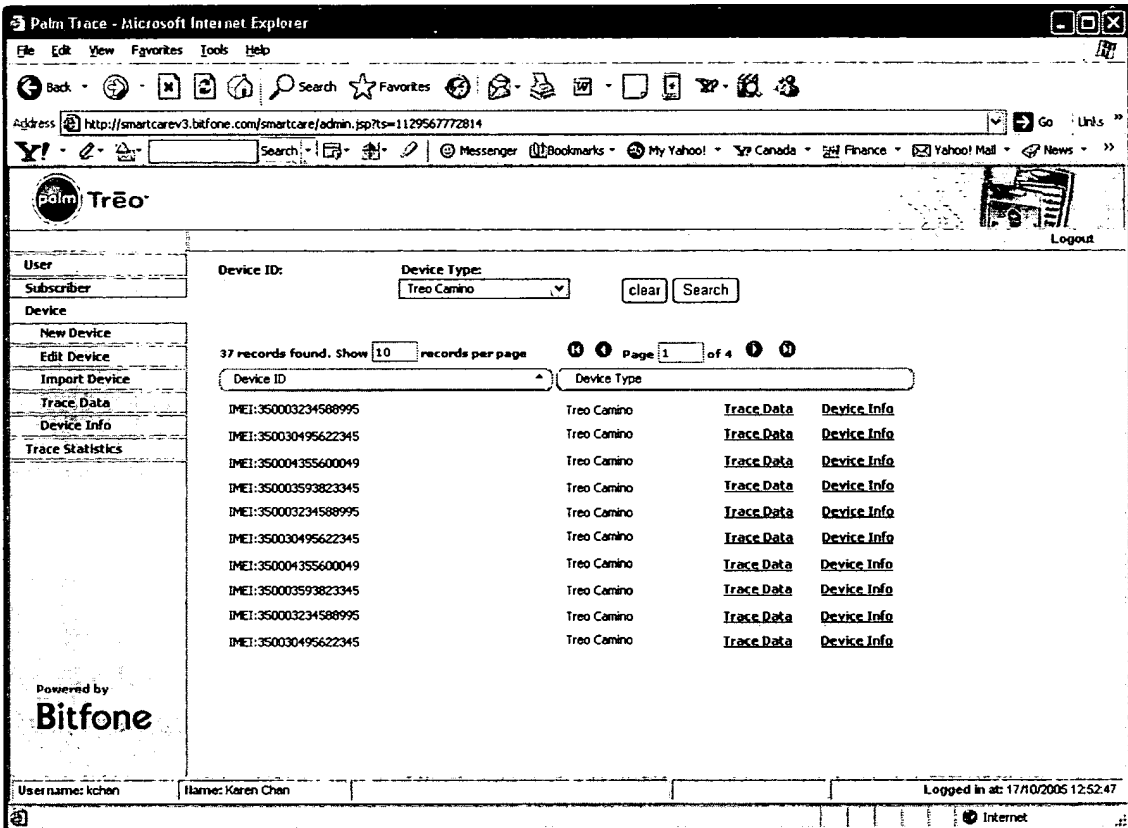
In Phase 1 Stage 1, the only Device Type that will be preloaded is the Camino.

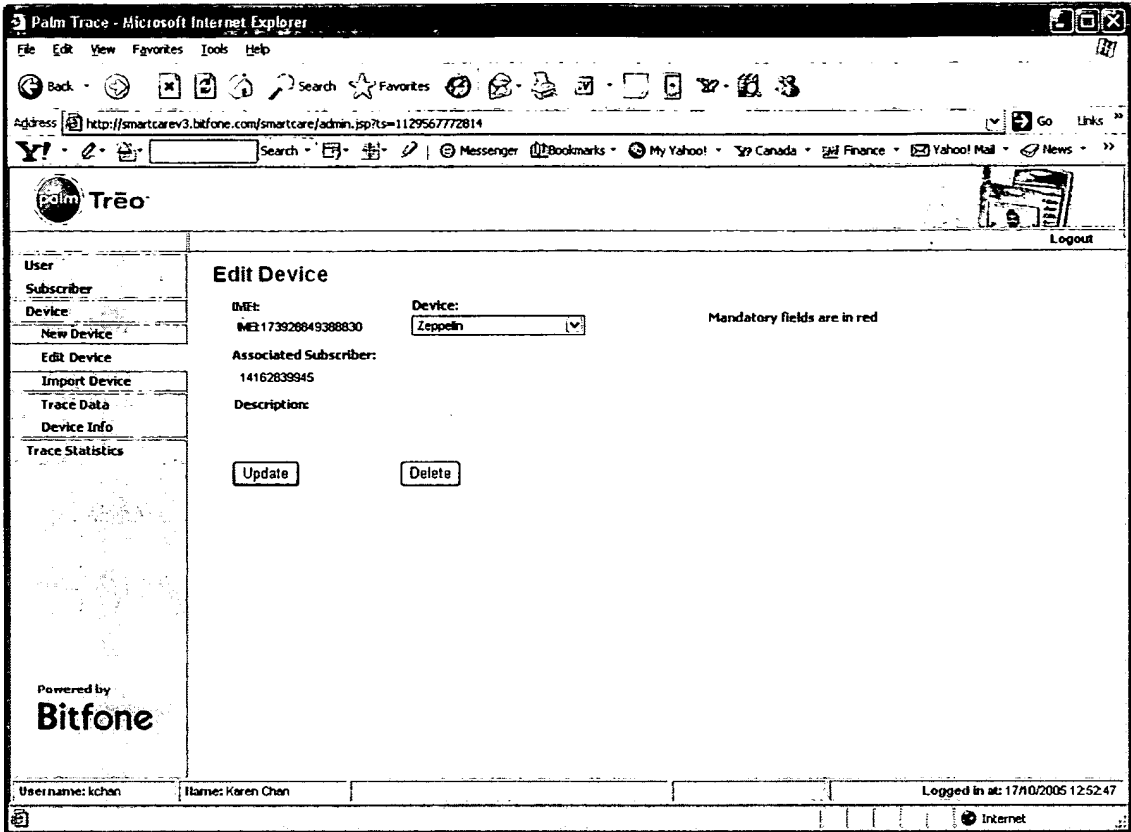
The Associated Subscriber is not editable in the Edit Device screen since the association of a device with a subscriber is performed in the Edit Subscriber screen.

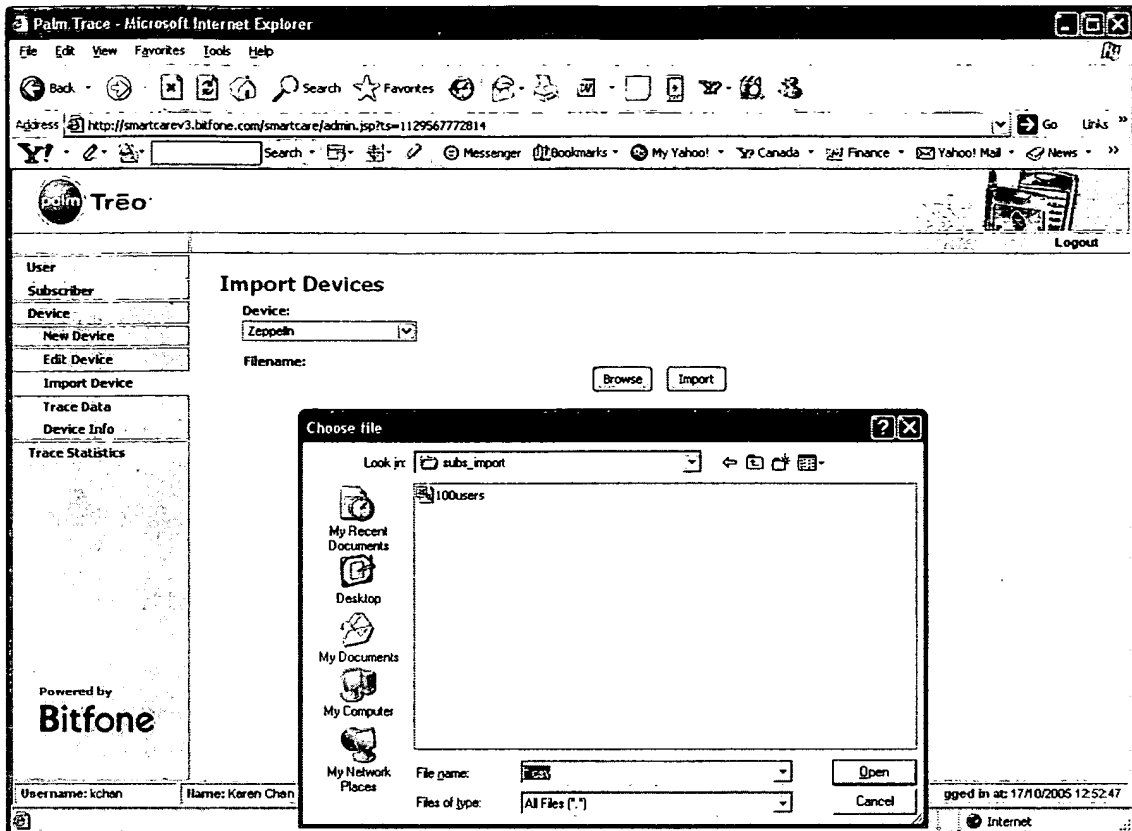
A device can be created in one of two ways:

1. Manually - by a user using the Treo Trace server user interface
2. Automatically - when trace data is sent back to the server, if the device has not been registered in the system

There will be screens for Searching, Creating, Editing, Deleting and Importing devices. Since the unique identifier of a device is the Device ID, the Device ID is not editable once the device is created.







To import a list of devices, click on Browse, select the CSV file from the file system and click Import.

The CSV file has to be in the following format:

[IMEI/ESN1]

[IMEI/ESN2]

...

4.4 Trace Data Management

Trace Data is logged on the device and transferred back to the server either through the device itself or through the PC Application. Each Trace Data transfer session is for a UNIQUE Boot Number for a SINGLE Device and should not contain any partial log entries.

4.4.1 Trace Data Transfer Format

The data should be transferred to the server using the HTTP protocol, and using byte-range in the HTTP Header.

The user-agent in the HTTP Header should state the source (i.e. "TreoPCApp" or "TreoTraceClient").

The data portion of the HTTP request must start with a log message transmission header, followed by one or more log entries.

The structure of the log message transmission header is as follows:

```
struct msgHeader
{
    UINT16    u16MsgVersion; // Message format version
    UINT16    u16MsgType;    // Message type
    UINT16    u16DevType;    // Device type
    UINT16    u16DevVersion; // Device version, most likely the SW version
    UINT32    u32MsgLength;  // Message length, INCLUDING this header
    uint64    u64DeviceTime; // Device dependent time of this transfer.
                                // Note, this is actually a structure
                                // containing two UINT32
    INT16     i16UTCOffset;  // Local time signed offset from UTC in
                                // Minutes. If the offset is 0, it means that
                                // the device is already sending the device
                                // local time in the log entries
    UINT16    u16BootNumber; // Increments each time device is rebooted
    char      szEventID;     // SmartCare Event ID, zero terminated
    char      szDeviceID;    // IMEI/ESN or other device ID, zero
                                // terminated
    char      szMSISDN;      // MSISDN or empty string, zero terminated
};
```

The structure of the log entry is as follows:

```
struct logEntry
{
    BYTE      byTag;         // Designates message type and version
    BYTE      byLevel;      // Severity level of this message
    UINT16    u16Sequence;  // Message sequence from 1-FFFF, zero for
                                // "reset"
    uint64    u64Time;      // Device dependent time of this message.
                                // Note, this is actually a structure
                                // containing two UINT32
    UINT32    u32Component; // ID of component reporting this event
    UINT32    u32SubComp;   // ID of sub-component reporting this event
    UINT16    u16OptDataLen; // Optional data length, 0 == none
    UINT16    u16OptTextLen; // Optional string text length, 0 == none
    BYTE      byOptData[1]; // Locates the first byte of any optional
                                // data. Binary data comes first, followed
                                // by text
};
```

4.4.2 Servlet and JMS Listener for parsing and receiving Trace Data

On the server side, a new servlet, a new JMS Queue and a new JMS listener will be introduced for receiving trace data. For performance purposes, the servlet should have minimum processing logic. The servlet will just read the entire binary data stream, parse the message length in the header and compare the total transmission data length. If the lengths do not match, a special HTTP status code, 400 – Bad Request, should be returned. If the lengths match, the servlet will queue the data into the JMS Queue for further processing and return HTTP status code 200 (Success).

A Trace Data Transfer session can be server-initiated; a user requesting a Trace Data dump on a device, or client-initiated; when a timer or buffer condition fullness is met on the device, or when the PC application sends device data back to the server. If a request is server-initiated the resulting trace data transfer should include the original event id in the message transmission header. If a request is client-initiated the resulting trace data transfer should include 0 as the event id in the message transmission header, and the server should generate an event id for grouping all log entries received.

The new JMS Listener will receive the message from the JMS Queue as a `javax.jms.ObjectMessage`. It will then parse the data into individual trace log entries and write them persistently into the database.

Duplicate Trace data could result when both the PC Application and the Device are trying to send back trace data to the server. The server is responsible for ignoring duplicate trace data received from the device. The server will use the Device Instance Id, Boot number, Sequence number, Device time, Component, Subcomponent, and Severity in the log entry to determine if a log entry is duplicated. When checking for duplicates, the server will NOT compare the binary or text data in the log to avoid performance problems.

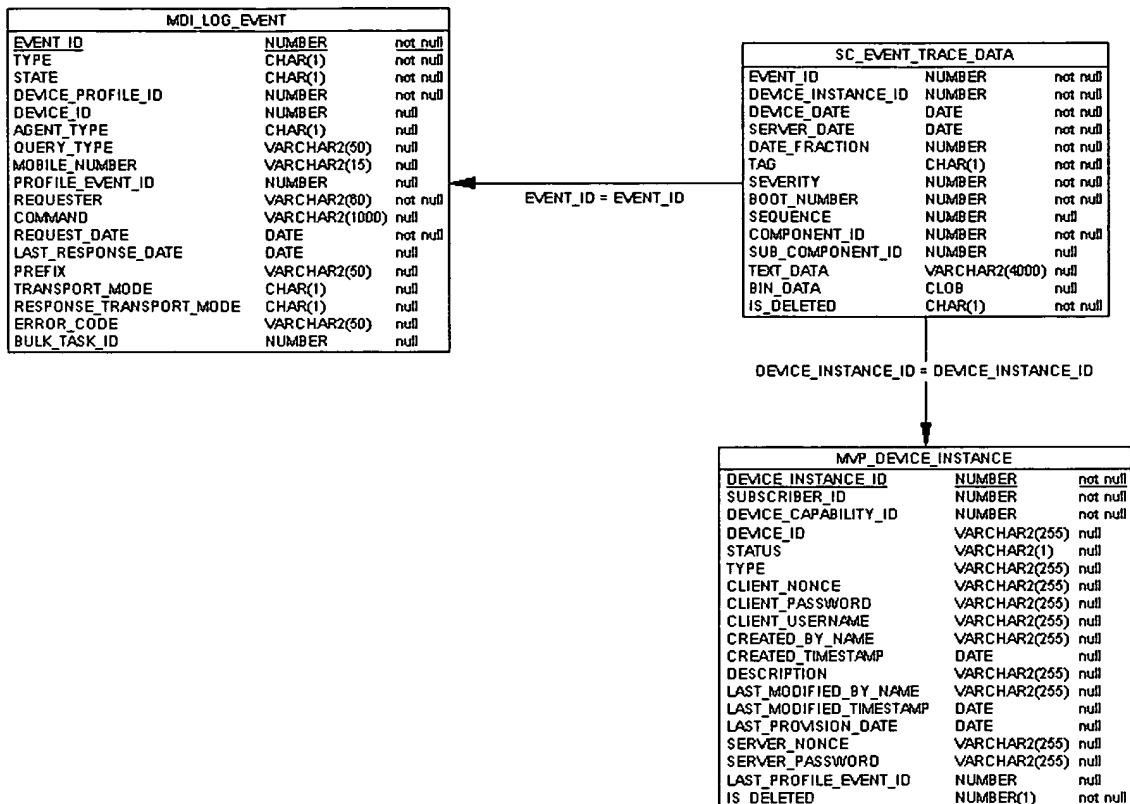
The Treo Trace server will store all trace log data into the `SC_TRACE_DATA` table. All Binary data will be stored and displayed on the screen as hex strings.

On each log entry, there is a device-dependent time (`u64Time`) to record when the log message is logged. This time is returned by the API on the device and the unit, precision, timezone and basis of the time may differ from device to device. The server will be responsible for using the appropriate device-specific adaptor to convert this time back to a datetime. In addition, the server should use the “`i16UTCOffset`” field in the log transmission header to calculate the device local time for the log entry (refer to the “Uploader” section above for details on how to use the “`i16UTCOffset`” field). The device local time for the log entry is stored in the `DEVICE_DATE` column in the `SC_EVENT_TRACE_DATA` table. Since the Oracle Datetime datatype is only precise up to seconds, addition precision (up to nanoseconds) will be stored in the `DATE_FRACTION` column.

The server is also responsible for capturing the datetime of when the log entry is received by the server. This datetime is stored as UTC in the `last_response_date` column in the `MDI_LOG_EVENT` table. If multiple transfer sessions for the same event (except for event id 0

) are received by the server, the completion time of the last transfer session will be stored in the last_response_date column.

The definition of the SC_TRACE_DATA table and its relationship to existing tables are illustrated in the following diagram:



If the message transmission header contains a valid Device ID and no such device instance exists in the database, the server is responsible for creating the device instance record in the database.

If the message transmission header contains a valid MSISDN and no such subscriber instance exists in the database, the server is responsible for creating the subscriber instance record in the database. The subscriber will be created under the DEFAULT subscriber class.

If the message transmission header contains both a valid Device ID and a valid MSISDN, the server is responsible for updating the last known device on the subscriber record and the subscriber id on the device record.

4.4.3 Screen for filtering and displaying Trace Data for a Device

There will be a screen for displaying trace data on a device-by-device basis. The user can provide one or more of the following search criteria to further streamline the results:

- Device Local Datetime of Log Record (user can specify the FROM and TO datetime to form a range; precision will be in seconds)
- Datetime when Log Record is received on the server, displayed according to the User's timezone (user can specify the FROM and TO datetime to form a range; precision will be in seconds)
- Boot Number
- Severity Level
- Component ID
- Subcomponent ID

The list of available Components and Subcomponents will be preloaded into the database in Phase 1 Stage 1. In Phase 1 Stage 2, there will be screens for searching, maintaining, and importing Components and Subcomponents.

Below are a few examples of how these search criteria can be combined to produce useful reports:

1. All FAULT messages generated by a specific Component
2. All messages, regardless of severity, generated by a particular component/subcomponent combination
3. All messages logged between Jan 1, 2006 and Jan 10, 2006, inclusive

All search results will be displayed in tabular format. If there is any binary data, the "Binary?" column will show "Y". Clicking on the magnifying class icon will bring up a Trace Data Detail screen where the binary data will be displayed as hex strings.

To delete a log entry, check the checkbox of the log entry and click the "Delete" button. To delete multiple records, check more than one checkbox and click the "Delete" button. To delete all records on the screen, check the checkbox on the header to select all records, and then click the Delete button.

Clicking on the header columns will allow the user to sort the results.

Clicking on the "Export" button will export the search results to a CSV (comma delimited) file.

The format of the CSV will be:

[Device Datetime],[Received Datetime],[tag],[Boot Number],[Sequence Number],[Severity],[Component],[Subcomponent],[Text data],[Binary data as hex]

If the data value is empty, it will be represented as an empty string. If the data value contains a comma, the entire field will be enclosed in double-quotes. If the data value contains a double-quote, the double quote will be escaped by another double-quote in front and the entire data value will be enclosed in double quote.

For example, a data record with no binary data and a text data containing double quotes and comma:

```
2006/01/23 11:23:33 2006/01/23 11:26:12 1 12 22 DEBUG RADI SGNL Signal  
"extremely low", problem
```

will be converted to the following in CSV format

```
2006/01/23 11:23:33,2006/01/23 11:26:12,1,12,22,DEBUG,RADI,SGNL,"Signal  
"extremely low", problem",
```

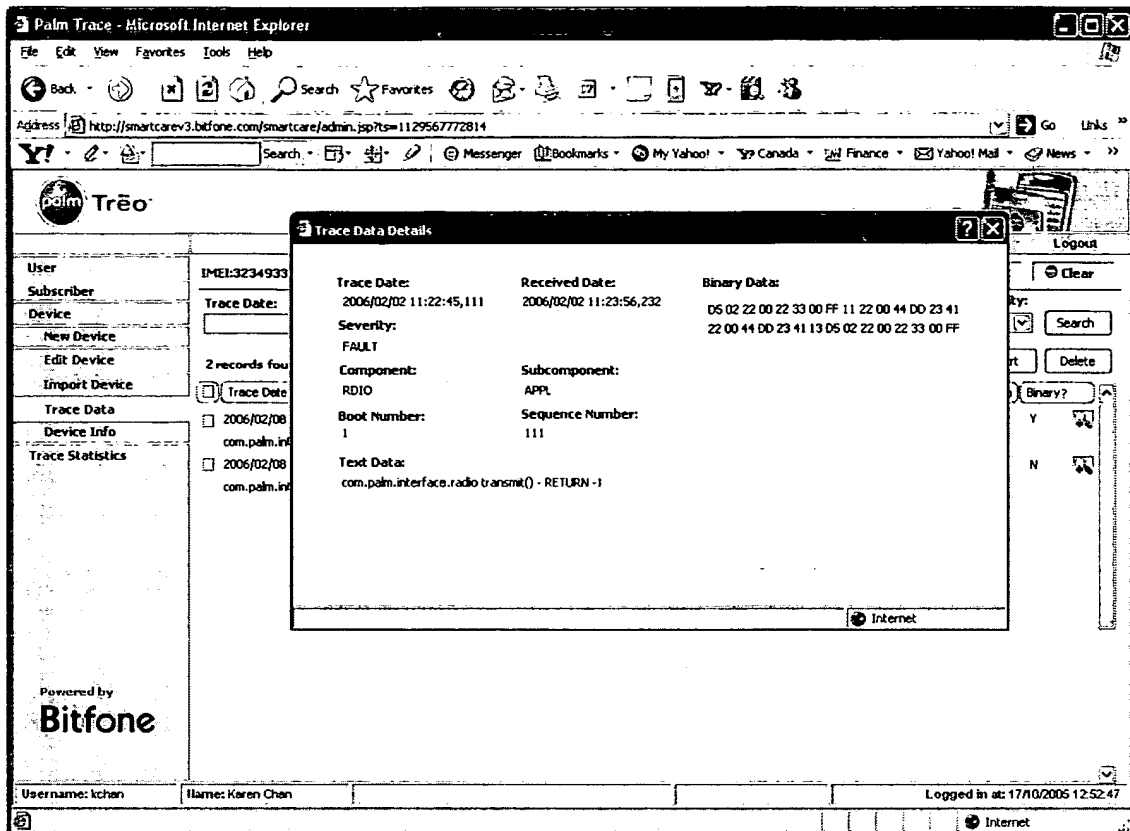
For Phase 1 Stage 1, Palm has indicated that there is no need to export or display the data in XML format. Such a need may arise in later phases, and Palm will provide the DTD for it.

The screenshot shows a web browser window titled "Palm Trace - Microsoft Internet Explorer". The address bar shows the URL: `http://smartcarev3.bitfone.com/smartcare/admin.jsp?ts=1129567772814`. The page header includes the "palm Treo" logo and a "Logout" link. Below the header, there is a navigation menu with options: "User", "Subscriber", "Device", "New Device", "Edit Device", "Import Device", "Trace Data", "Device Info", and "Trace Statistics". The main content area displays the following information:

- User: IMEI:323493349488893 (14169305949) Treo Camino Last Transmission: Jan 29, 2006 23:13:44
- Buttons: Transmit, Clear
- Filters: Trace Date, Received Date, Component (RDIO), Subcomponent (All), Severity (All), Search
- Summary: 2 records found. Show 10 records per page. Page 1 of 1. Export, Delete
- Table of records:

<input type="checkbox"/>	Trace Date	Received Date	Severity	Component	Subcomponent	Boot Num	Sequence Num	Binary?	
<input type="checkbox"/>	2006/02/08 11:22:45,111	2006/02/08 11:24:45,234	FAULT	RDIO	COMM	1	111	Y	
	com.palm.interface.radio.testTransmit() - The radio interface is not responding								
<input type="checkbox"/>	2006/02/08 11:22:45,116	2006/02/08 11:24:45,236	DEBUG	RDIO	APPL	1	112	N	
	com.palm.interface.radio.testTransmit() - Object not found. Returning -1								

At the bottom of the page, it says "Powered by Bitfone". The footer contains: "Username: kchan | Name: Karen Chan | Logged in at: 17/10/2005 12:52:47".



4.4.4 Triggering a Trace Data Transfer on the Device

Clicking the 'Transmit' button on the Trace Data screen will send a specially formatted SMS to the device to request a Trace Data Transfer. The Trace Data Transfer Request will make use of existing SmartCare protocol.

When the device receives a Trace Data Transfer Request, it should transmit all trace entries on the device for transmission. If logging is happening in parallel at the same time as the request, those log entries may or may not be included in the transfer, depending on implementation.

4.4.5 Clearing all Trace Data on the Device

Clicking the 'Clear' button on the Trace Data screen will send a specially formatted SMS to the device to clear all Trace Data. The Clear Data Request will make use of existing SmartCare protocol.

When the device receives a Clear Data Transfer Request, it should clear all trace entries on the device.

4.4.6 Displaying Trace Data for one or more Devices

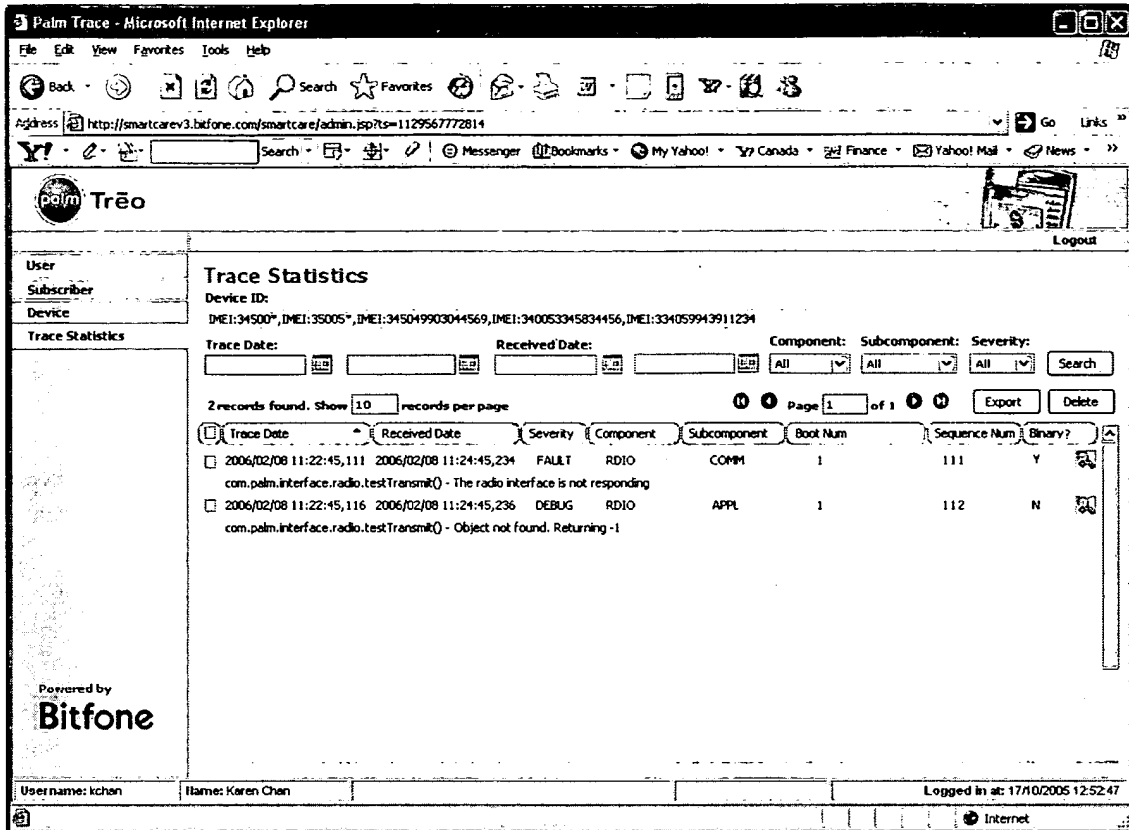
The super user will have the capability to retrieve trace data on one or more Devices. This powerful feature will allow the super user to generate diagnostics statistics for a selected group of devices.

The search criteria will include:

- One or more Device ID(s), comma separated - Wildcard character search will also be accepted. Leaving this field empty will generate a report that spans ALL devices in the system
- Device Local Datetime of Log Record (user can specify the FROM and TO datetime to form a range; precision will be in seconds)
- Datetime when Log Record is received on the server, displayed according to the User's timezone (user can specify the FROM and TO datetime to form a range; precision will be in seconds)Severity Level
- Boot Number
- Sequence Number
- Component ID
- Subcomponent ID

Below are a few examples of how these search criteria can be combined to produce useful statistical reports:

1. All Call Drop events on a particular model of device (based on a wildcard search using the TAC on the IMEI)
2. All FAULT messages generated by the RADIO component across all devices
3. All FAULT messages logged since a firmware update on Jan 14, 2006 on a particular device



4.5 Displaying and Modifying Device Information and Settings

There will be a screen for displaying the most recent device information collected. This device information includes the IMEI/ESN, Manufacturer, Model, Platform, Revision, Processor, OS Version, Free Memory, Signal Strength, Data Connection Settings, etc. Additional Device data could be collected in Phase 1 Stage 2 if APIs are available.

The most recently collected Treo Trace settings from the device will also be displayed and will include:

A Collection of Filter Criteria

- One or more Component ID, Subcomponent ID, Severity, Trigger combinations

Transfer Settings

- Buffer Fullness Threshold, URL of Treo Trace server, Real-time Mode (On/Off), Output Link to Server (On/Off), Output Link to PC (On/Off), Output Link to USB (On/Off), Periodic Trigger (On/Off), Interval between Transfer, Schedule Time for Transfer

Other Settings

- Internal Flash Buffer Size, Internal RAM Buffer size

All data will be displayed as key-value pairs on the screen and parameters will be categorized onto different tabs on the display.

To trigger a real-time collection of device data and Treo Trace settings, click on the "Profile" button.

To configure settings on the device, click on the small "edit" icon next to an attribute. This will open up the field(s) for editing. After the settings are modified and saved, click "Configure" to see a Summary of configuration changes. Click on "Configure Device" to send the configuration changes to the device.

The user would be able to configure and fix the TCP/IP connectivity by editing the settings on the Connection tab.

The system will make use of existing SmartCare protocol for all collection and modification of device data and Treo Trace settings. The transport mode for profile and configuration should be set to "Auto" so that the device will automatically try TCP/IP first and then revert to SMS if TCP/IP connectivity is not available.

