

**Vending Equipment Interface (VEI) Specification
Version 1.2
July 31, 2002**



Agent Systems, Inc.
14802 Venture Drive
Farmers Branch, Texas
75234-2426
(972) 774-0400
(972) 392-7301 FAX



Copyright 1996, 2002, Agent Systems, Inc.

Agent Systems, Inc.
14802 Venture Drive
Farmers Branch, Texas
75234-2426
(972) 774-0400
(972) 392-7301 FAX

Table of Contents

Chapter 1	
Limited License to Copy and Use	1
1.1 Statement of Copyright.	1
1.2 Rights to Copy.	1
1.3 Exclusion of Warranty.	1
1.4 Rights to Use Procedures and Messages Described in this Document.	1
1.5 Updates to this Document.	1
Chapter 2	
Executive Overview	3
Chapter 3	
Questions and Answers	5
Chapter 4	
Overview	9
4.1 Introduction	9
4.2 Network hierarchy	9
4.3 Example one	10
4.4 Example two	11
4.5 Example three	12
4.6 Example four	13
4.7 Example five	14
4.8 Example six	15
4.9 Example seven	16
Chapter 5	
Technical Overview	17
Chapter 6	
Implementation Description	19
6.1 Overview.	19
6.2 First example.	20
6.3 Second example.	21
6.4 Third example	24
6.5 Fourth example.	27
Chapter 7	
Protocol: Physical and Link Layers	37
7.1 Structure of this document	37
7.2 Overview.	37
Chapter 8	
Protocol: Transport and Session Layers	39
8.1 Overview.	39
8.2 The Agent Transport Layer (ATL).	39
8.2.1 Transport layer overview.	39

8.2.2	Transport header.	40
8.2.3	Transport synchronization.	41
8.2.3.1	Sequence diagrams.	41
8.2.3.2	State machine.	42
8.2.4	Receive side processing.	44
8.2.5	Send side processing.	45
8.2.6	Tunable parameters:	45
8.3	The Agent Session Layer (ASL).	45
8.3.1	Session layer overview.	45
8.3.2	Authentication request.	47
8.3.3	Authentication response.	47
8.3.4	Authentication acknowledgment.	48
8.3.5	After authentication.	48
 Chapter 9		
	Dialog Overview	51
9.1	Introduction	51
9.2	Dialog Classes	53
9.3	Services	53
9.4	Data formats	57
9.5	Message header	58
9.6	Response Dialog Common Fields	59
 Chapter 10		
	Condition Dialogs	61
10.1	Definition	61
10.2	Condition — Unsolicited Status	61
10.3	Condition — Report Status	62
10.4	Condition — Read Attributes	64
10.5	Condition — Alter Attributes	65
 Chapter 11		
	Variable Access Dialogs	67
11.1	Definition	67
11.2	Variable Access — Read	68
11.3	Variable Access — Write	70
11.4	Variable Access — List Structure	71
 Chapter 12		
	Log File Dialogs	73
12.1	Definition	73
12.2	Log File — Create Log	74
12.3	Log File — Initialize Log	75
12.4	Log File — Report Log Status	76
12.5	Log File — Delete Log	78
12.6	Log File — Upload Log	79
12.7	Log File — Write Log	81
 Chapter 13		

File Management Dialogs	83
13.1 Definition	83
13.2 File Management — Transfer Request	83
13.3 File Management — Transfer Block	84
13.4 File Management — Terminate Transfer	85
13.5 File Management — Read Directory	86
13.6 File Management — Rename File	88
13.7 File Management — Delete File	89
Chapter 14	
Transaction Dialogs	91
14.1 Definition	91
14.2 Transaction — Authorization	91
14.3 Transaction — Capture	94
14.4 Transaction — Cancellation	95
14.5 Transaction — Reversal	96
Chapter 15	
Program Execution Dialogs	97
15.1 Definition	97
15.2 Program Execution — Spawn	97
15.3 Program Execution — Terminate	99
15.4 Program Execution — Restart	100
15.5 Program Execution — List Program Executions	102
15.6 Program Execution — Input	104
15.7 Program Execution — Output	105
Chapter 16	
Data Type Definitions	107
16.1 Data formats	107
16.2 ValAdditionalAttribute	107
16.3 RecAdditionalAttributeList	107
16.4 RecAdditionalAttribute	108
16.5 RecAttributeModificationList	108
16.6 RecAttributeModification	108
16.7 ValAttributeID	108
16.8 Boolean	108
16.9 ValCardAccountNumber	109
16.10 ValCardType	109
16.11 RecCardTrackList	109
16.12 RecCardTrack	109
16.13 RecConditionAttributes	110
16.14 RecConditionID	110
16.15 RecConditionList	110
16.16 ValConditionPriority	111
16.17 ValConditionSeverity	111
16.18 ValConditionStatus	111
16.19 ValCounter	111
16.20 ValCustomerName	111

16.21	RecDateTime	111
16.22	RecDate	111
16.23	ValDeviceID	112
16.24	ValDeviceStatus	112
16.25	ValDirection	112
16.26	ValEncryptedPIN	112
16.27	ValFailureReason	112
16.28	ValFieldType	113
16.29	RecFileAttributeList	114
16.30	RecFileAttribute	115
16.31	ValFileName	115
16.32	ValFileTransferDirection	115
16.33	ValFreewheel	115
16.34	ValKeySerialNumber	115
16.35	RecLogFileData	115
16.36	RecLogFileRecord	116
16.37	ValLogRecordType	116
16.38	ValManufacturerID	116
16.39	RecMessageHeader	116
16.40	ValMessageIdentifier	117
16.41	ValMessageLength	117
16.42	ValMessageSequence	117
16.43	ValMessageType	117
16.44	ValMoney	117
16.45	RecParameterList	118
16.46	ValPaymentType	118
16.47	ValPermission	118
16.48	ValProcessID	118
16.49	RecProcessStatusList	118
16.50	RecProcessStatus	119
16.51	ValReadVariableMode	119
16.52	ValRecordSequence	119
16.53	ValReversalReason	119
16.54	RecSalesItemList	119
16.55	ValSecurityMode	119
16.56	ValStatusCategory	119
16.57	RecStatusReportList	120
16.58	RecStatusReport	120
16.59	RecSubsystemID	120
16.60	ValTerminationReason	120
16.61	ValTrackData	121
16.62	ValTrackID	121
16.63	ValTransactionID	121
16.64	RecVariableInterspersedList	121
16.65	RecVariableInterspersed	121
16.66	RecVariableLogFile	121
16.67	RecVariableNameList	122
16.68	ValVariableName	122
16.69	RecVariableStructureList	122

16.70	RecVariableStructure	122
16.71	RecVariableValueList	123
Chapter 17		
	Additional Data Items	125
17.1	Additional Attributes	125
17.2	Variables	129
17.2.1	Variables dealing with the device that change infrequently	129
17.2.2	Variables dealing with the configuration of the device.	130
17.2.3	Variables dealing with dates and times.	133
17.2.4	Variables dealing with counting.	133
17.2.4.1	Variables dealing with sales counts.	134
17.2.4.2	Variables dealing with product counts.	135
17.2.4.3	Variables dealing with cash counts.	137
17.2.4.4	Variables dealing with coin counts.	139
17.2.4.5	Variables dealing with counts of bills.	142
17.2.4.6	Variables dealing with counts of tokens.	144
17.2.4.7	Variables dealing with counts of credit/debit card usage.	146
17.2.4.8	Other counts.	147
17.2.4.9	Variables dealing with counts related to gates.	149
Chapter 18		
	Creating Extended Data Items	151
Appendix A		
	Glossary	153
Appendix B		
	Dialog Tutorial	161
B.1	Overview	161
B.2	Condition — Unsolicited Status	161
B.3	Condition — Report Status	162
B.4	Variable Access — Read	162
B.5	Variable Access — Write	164
B.6	Log File — Report Log Status	164
B.7	Log File — Upload Log	165
B.8	Log File — Initialize Log	166
B.9	File Management — Transfer Request / Transfer Block	167
B.10	Transaction — Authorization	169
B.11	Transaction — Capture	169
Appendix C		
	C language routines	171
Appendix D		
	Synchronizing Remote Clocks	173
Appendix E		
	Release Notes	175

E.1	Release 1.0	175
E.2	Release 1.1	175
E.3	Release 1.2	176

Chapter 1

Limited License to Copy and Use

1.1 Statement of Copyright.

This document is protected by United States and International copyright laws, and may not be copied, in whole or in part, except under the conditions set forth in this Limited License. This document was created by and all rights are owned by Agent Systems, Inc., of Dallas, Texas. Rights to copy are granted under the terms of this Limited License.

1.2 Rights to Copy.

The entire document may be copied without fee upon the following terms and conditions:

1. All copyright notices and this Limited License must be preserved and included in each copy.
2. No modification or change may be made to the document.
3. No partial copies may be made.
4. The version number of this revision must be preserved and included in each copy.

1.3 Exclusion of Warranty.

Agent Systems makes no warranty regarding the Licensed Materials, and makes no warranties, whether express, implied or statutory excluding, but not limited to, any implied conditions or warranties of merchantability and fitness for a particular purpose, and Agent Systems shall not be liable to licensee for any consequential damages resulting from the use of the Licensed Materials. However, licensee is invited to inform Agent Systems where it believes the Licensed Material may be inaccurate or incomplete.

1.4 Rights to Use Procedures and Messages Described in this Document.

Agent Systems, Inc., hereby grants to any party a royalty-free, worldwide, perpetual license to use the procedures and messages described in this document and to make, use, and sell products and services that conform to these procedures and messages.

1.5 Updates to this Document.

For a free copy of the most recent approved version of this document, contact Agent Systems, Inc., at (972) 774-0400, or see our web site at <http://www.agentsystems.com>.

Chapter 2

Executive Overview

The Vending Equipment Interface (VEI) specification was created to provide a common interface for the exchange of data between vending equipment and central computer systems. The interface defined in this standard can be used for the connection of vending equipment from any manufacturer, in any vending application.

In the last few years, vending applications in several industries have been implemented which require some form of networked monitoring and control. These industries include mass transit, parking, and merchandise vending. Several of these applications have also required electronic payment support. VEI addresses the requirements of these systems.

The VEI specification allows these networked vending applications to be based on an open, non-proprietary standard. This standard protects the investment each customer makes in these systems. In the past, each equipment manufacturer supported a different set of communications protocols and messages. These proprietary protocols and messages were incompatible with the protocols and messages of other manufacturers. This situation made equipment additions to existing systems difficult, resulting in high support costs, and high costs for replacement equipment. The VEI specification establishes a common interface that can be used by all manufacturers and users of networked vending systems.

The VEI specification was developed based on several existing standards. It is broad enough to encompass networked applications for all types of vending, and all types of technology used for data communications. Extensive testing has been conducted to ensure that all likely types of applications can be supported efficiently with the use of this standard. In addition, the VEI specification calls out the mechanisms for documenting specific implementations.

Chapter 3

Questions and Answers

1. Why should I be concerned about interface specifications?

Most modern distributed ticketing, parking control and vending systems are networked, and in the future this will be a requirement. A networked system is inherently more complicated than standalone equipment. To take full advantage of a networked system, it must work properly, efficiently, and be able to be expanded and adjusted.

To add equipment to a non-networked system, more equipment is simply purchased, plugged in and installed. To add equipment to a networked system, the additional requirement of communicating with a central network system is added. If the new equipment cannot communicate with the network, then a significant degree of functionality is lost. The choices can be to buy only from the original manufacturer of equipment, scrap all old equipment and start over, buy and install another networking system, or give up and operate new equipment on a standalone basis.

The Vending Equipment Interface (VEI) specification was assembled to propose a common "language" with which equipment from different vendors can communicate.

2. How do interface specifications affect my costs of doing business?

The goals of any enterprise should include reducing the costs of operation by minimizing capital expenditures and prolonging the useful lifetimes of capital equipment. With an open standard interface specification, vending equipment and network interface equipment can be purchased independently of each other and mixed and matched with multiple vendors of equipment and systems.

Vendors using an open standard specification do not have to expend the significant effort required to design their own interfaces and to implement multiple standards, which produces savings that can be passed along to the customer as lower prices.

3. Why can't an existing ISO, ANSI, or other standard be used?

No existing standard covers all areas of communications needed for complete management and control of remote vending devices. In addition, some standards are far too complex and would not be desirable for use as they would be too costly to implement and use in the field. Our first preference has always been to find a standard or combination of standards to use and reference. We have used several existing standards for portions of this specification. With widespread use and support, this specification can be presented to standards organizations for consideration as an international standard.

4. How does the specification protect my investment in vending equipment?

With the use of a single specification for all intercommunications between equipment and a network, you can be assured that equipment you buy five years from now can communicate with your existing systems. Old and new equipment can work together--you do not have to replace your entire system at once.

5. How does the specification protect my investment in network systems and equipment?

With the use of this standard specification you can use your existing network infrastructure and network applications with equipment from all manufacturers that support it. If you have an existing standards-compliant networking system, you can change to new equipment and not be required to buy new network hardware or software.

6. Why can't any vendor use any interface specification?

Technically, any vendor can use any interface specification that performs the needed functions. Legally, however, in some instances, one vendor can assert that they own all or part of the messages or interface specifications used in a networked system.

Through this claim of ownership, or just through failing to provide complete documentation on an interface, vendors can "lock in" customers for add-on equipment purchases. If a proprietary interface is required, the customer may be faced with an "all or nothing" decision: buy additional equipment from the original contractor, or throw away all existing equipment and go with a new company. Then the cycle may very well start over with the new vendor.

7. Can any computer system and any vending equipment use the interface standard?

Yes. This specification is independent of operating systems, local or wide area network technologies, and equipment types. The specification can be used with simple as well as very complex communications and networking scenarios.

8. What are some examples of equipment that this standard will support?

All networked, microcomputer-controlled equipment can be supported by this standard. Mass transit ticket vending, farebox and turnstile equipment, parking equipment, point-of-sale equipment, automated information signs, signals, and all forms of monitoring systems are good candidates for using this standard.

9. What is the cost to use this specification--are there license fees involved?

No costs are associated with using this specification.

10. Aren't all interface specifications for any vending network the property of the customer who purchases the system? Can't a customer allow other vendors to use this interface in the future?

Not necessarily. Unless specifically stated in a contract, interface specifications could be the property of the vendor. Vendors other than the original supplier of the system may not be able to use this interface. By having open specifications, the purchaser's rights to reuse the interface formats are protected.

11. What are the risks involved in letting a vendor use a proprietary interface specification?

Despite assurances to the contrary, even well-intentioned vendors can saddle their customers with network interfaces and messaging formats that are so arcane, poorly documented, and inflexible that future upgrades and additions of additional equipment are impossible. In the worst case, vendors can assert proprietary rights of data over crucial elements of interfaces and communications message data that could make purchases from other vendors impossible and drive up the cost of equipment that must be acquired in a noncompetitive scenario.

12. What if the standard interface specification does not include some needed function--can additional messages be added?

Nothing in the specification precludes the addition of functions or in any way reduces the flexibility of a user to design a system which best performs required duties. In fact, the VEI has been specifically designed so users can develop data extensions to the standard to fit the needs of a particular product or customer.

13. Some vendors claim they cannot support the standard interface specification and even claim their interface is better or faster or cheaper. How do I get them to use a standard interface?

The long-term benefits of supporting a standard are much greater and much more important than a simple

which-is-cheaper decision process. Of course, each vendor must invest the time and money to develop support for a standard, and some vendors may feel they are losing control over their customer by not being able to "lock in" future sales by using a nonstandard, proprietary interface. Standards must be required by the customer in bidding documents, and referenced and/or included in equipment and systems specifications. By adopting standards, vendors reduce their risk and costs. Over time, this translates into lower costs for customers.

There are cases where special technical requirements may not permit the use of standards. However, usually vendors resist adopting standards only to reduce or eliminate competition. In the end, adopting standards better serves both vendors and customers.

14. My vendor has installed a system and claims that the interface is compliant with the standard. How can I check to make sure that the system really complies with the standard?

One requirement for full compliance with this specification is the development and maintenance of documentation of messages and data used in the interface implementation. This documentation must follow a standard form in a way that allows a third party to use the interface to add equipment or network components. Testing can be done to ensure that the as-built interface complies with the written documentation delivered to the customer at the completion of the project.

15. What about older equipment--can existing equipment be made to communicate using this interface standard?

Yes, typically older equipment and application software can be reconfigured to communicate with a networking system using this standard.

16. Will using this specification give any company an advantage?

No. A standard is an open specification and is available to all vendors. All vendors develop to the same specifications, and their skills in producing equipment and application software are judged by price and performance.

17. How do standards benefit the industry (both vendors and customers)?

Benefits to the vending industry include the advantage of being able to concentrate on what they do best, which is designing and building vending equipment. This takes a great deal of risk out of projects and enables them to bid at a lower price and still maintain a fair level of profit.

Benefits to the user community include better equipment, better pricing, more predictable network performance, and the ability to mix and match equipment and network software vendors.

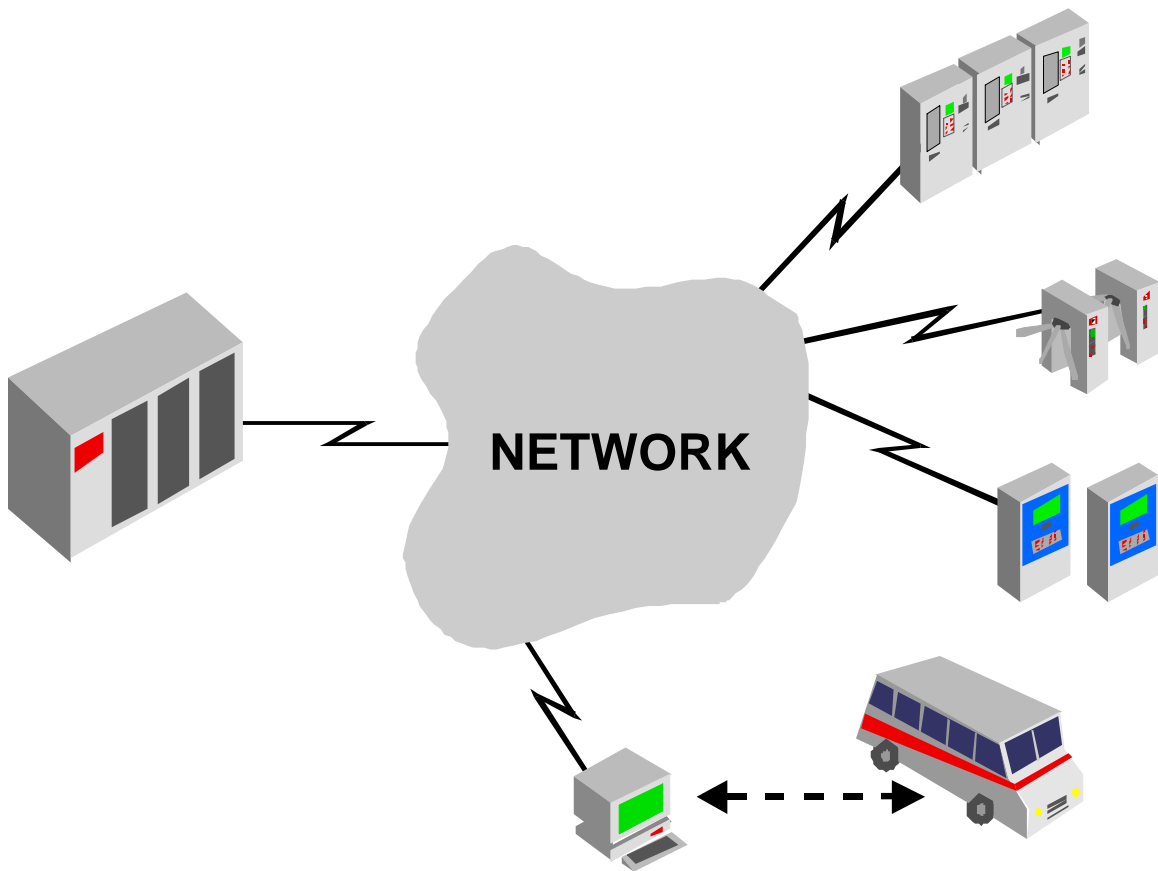
Chapter 4 Overview

4.1 Introduction

The Vending Equipment Interface Specification is explained in detail in the following chapters. It is broken down into dialog types, depending on the functionality available from particular equipment.

4.2 Network hierarchy

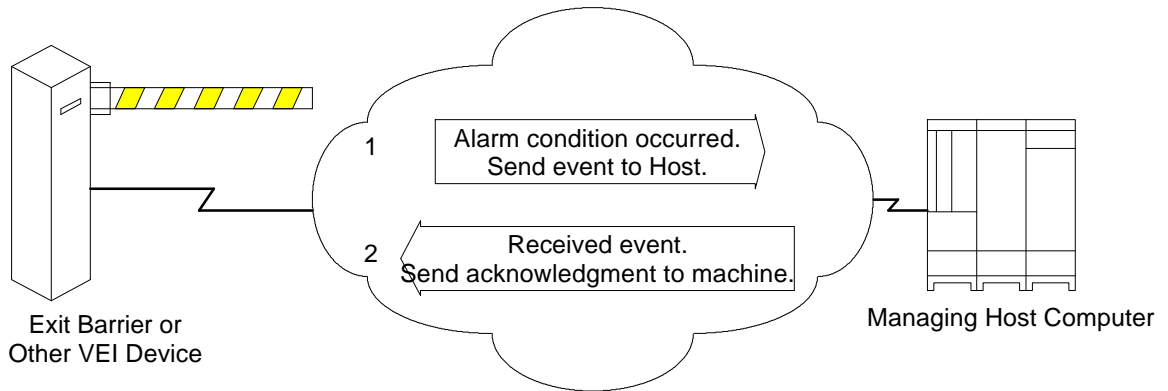
In this specification, devices (various types of vending equipment) communicate with a Managing Host Computer (MHC) over a network, as shown in the following diagram.



Sections 4.3 through 4.9 below contain examples of VEI compliant dialogs, illustrating the wide variety of VEI connections possible. It is important to understand that ***it is not required that every VEI dialog be implemented for a device interface to be VEI compliant.*** Only the VEI dialogs that are necessary to support the required device functionality need to be implemented. ***A device interface is VEI compliant if the dialogs implemented for that device interface comply with the VEI specifications.***

4.3 Example one

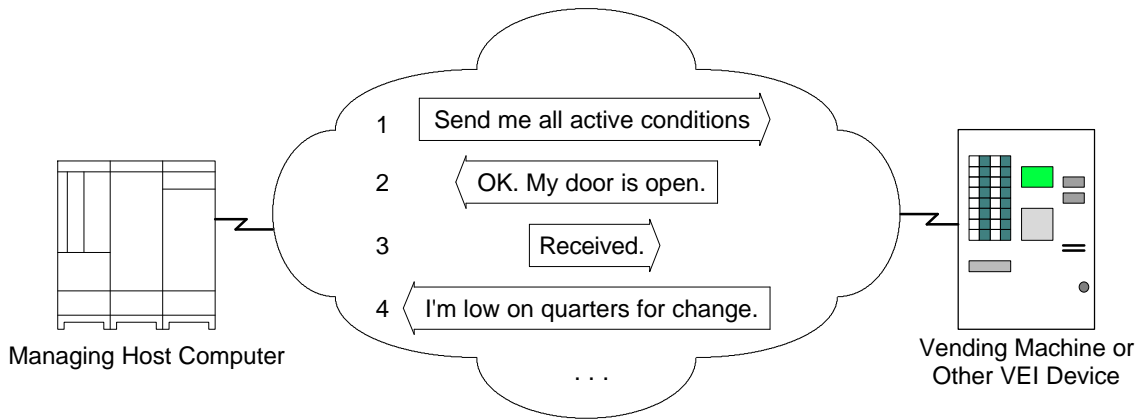
For this first example, there is only one dialog between the MHC and the remote device: alarm events. The device initiates them at the time the alarm condition occurs.



A device that implemented only this single dialog would be fully VEI compliant, if this were the only network functionality needed.

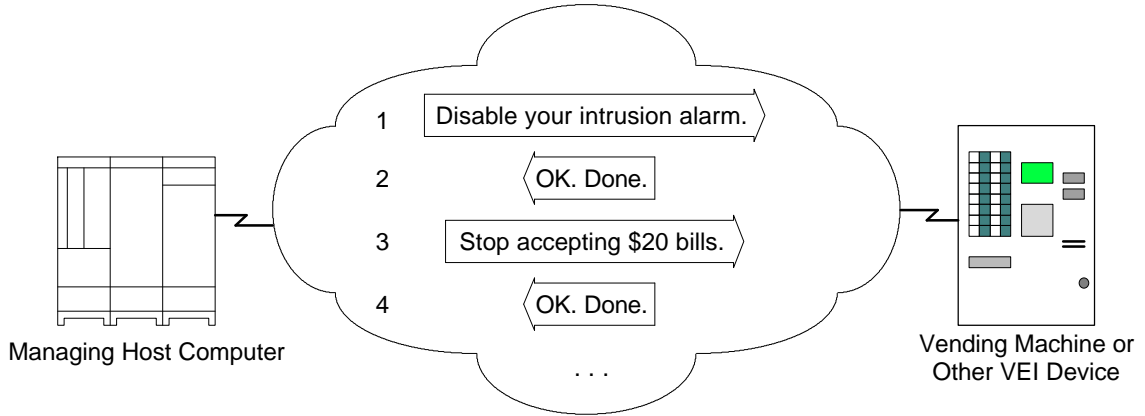
4.4 Example two

The second example is slightly more complex than the first. It involves monitoring the status of the remote device.



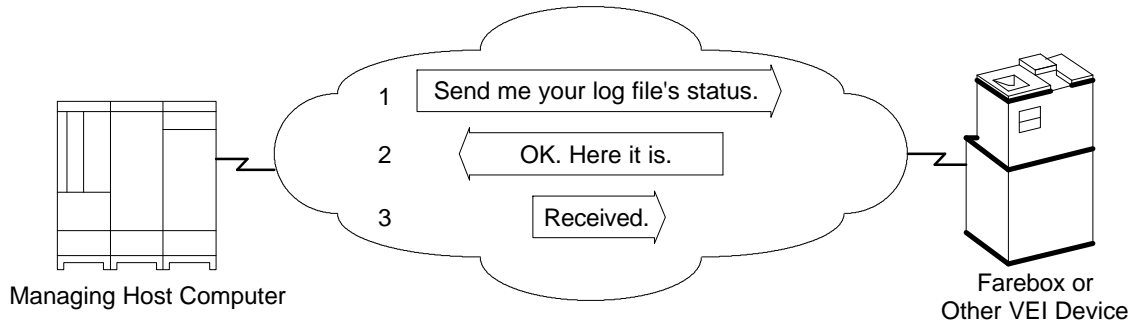
4.5 Example three

This third example demonstrates controlling the functional parameters on the remote device.

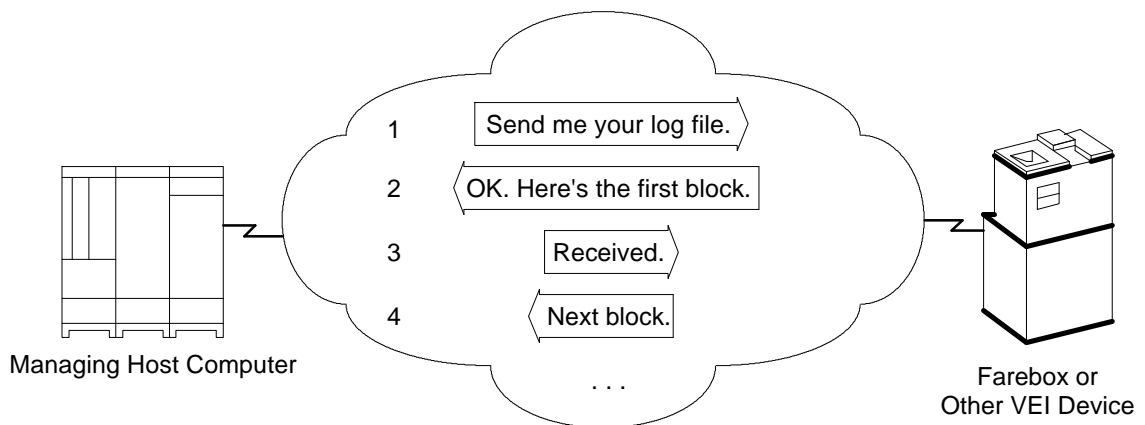


4.6 Example four

This fourth example demonstrates the flow and control of log files. As events occur, the device logs them locally to a log file. Then, the MHC can request the status of this log file.

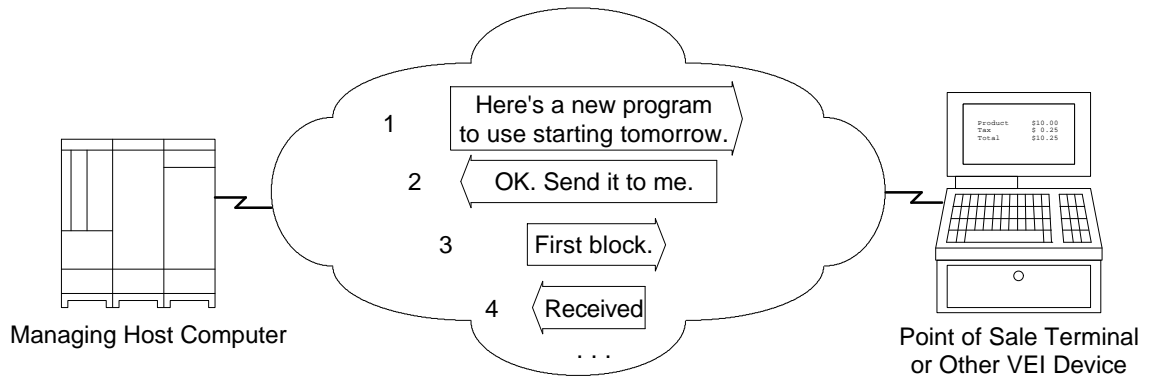
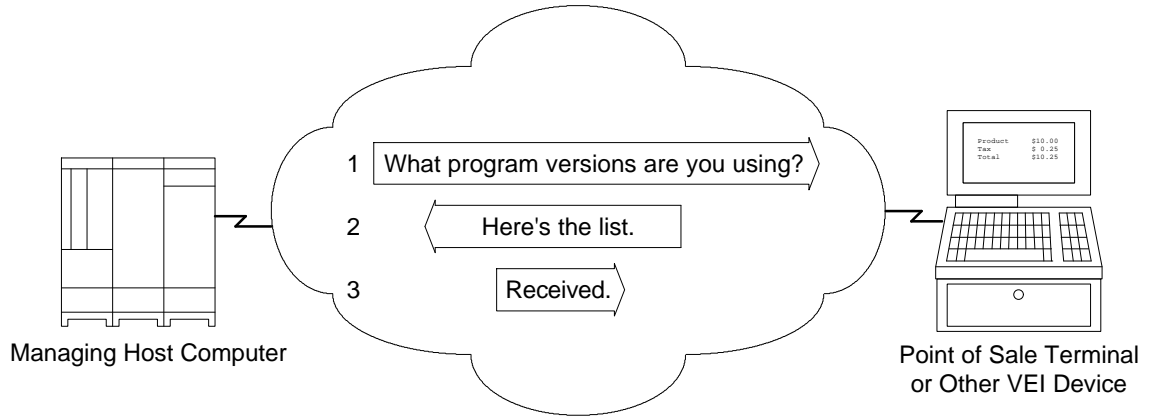


Additionally, the MHC can request that the log file be sent up line.



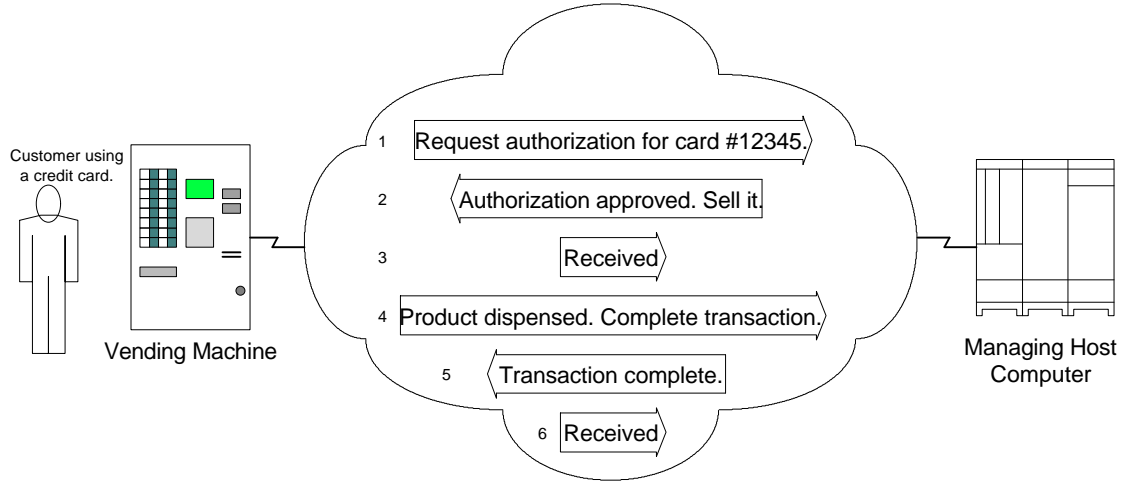
4.7 Example five

The fifth example contains several dialogs for controlling the programs and operating data that are in use on a remote device.



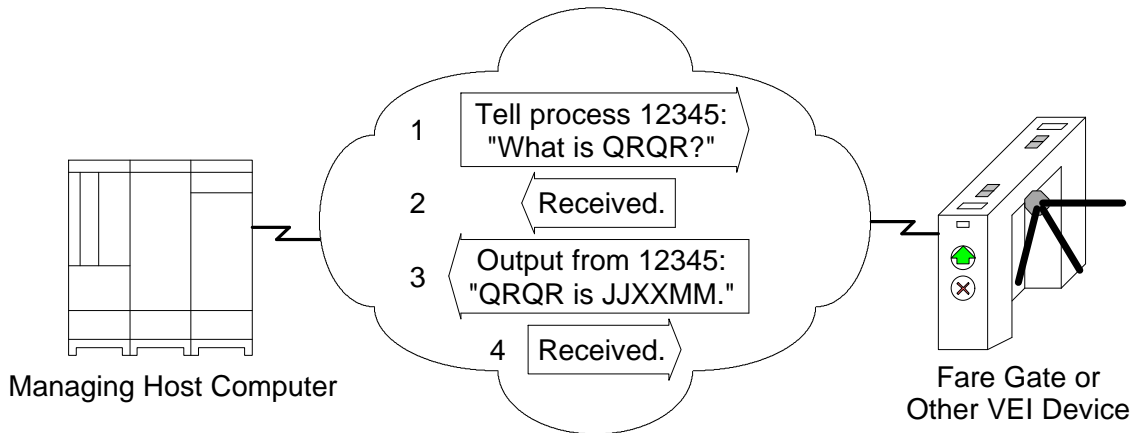
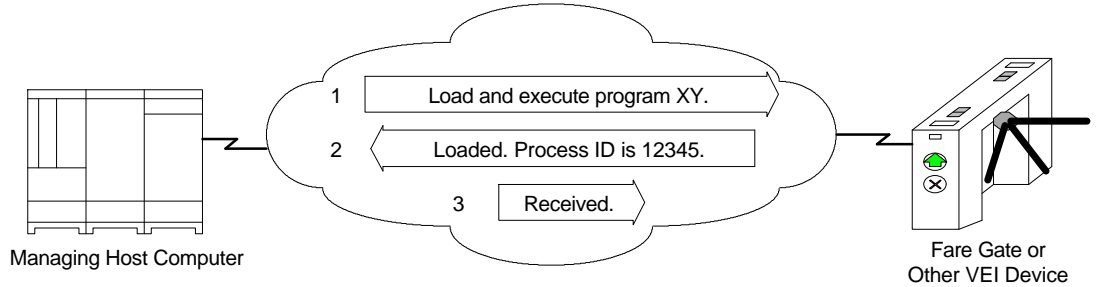
4.8 Example six

This sixth example demonstrates the flow of credit/debit card transactions:



4.9 Example seven

This seventh example shows the control and interaction of programs running on a remote device.



All the above dialogs do not have to be implemented for a given device, only what is necessary to perform its requisite functionality. For example, a device that accepts credit and debit cards might not need the program and operating data control dialogs.

Chapter 5 Technical Overview

This document contains the technical specifications for interfaces to: 1) all types of vending machines, such as mass transit ticketing machines, parking payment machines, 2) all types of attended sales terminals, such as ticket office machines, fareboxes, and parking fee computers, and 3) associated equipment such as fare gates, barriers, and ticket issuing machines.

The intended audience for this specification is: 1) users of networked vending systems such as parking, mass transit, vending operators, 2) vending equipment manufacturers wanting to connect with standard networks, and 3) network developers wanting to interconnect with standard vending equipment.

The interface specifications include communications protocols, security provisions, and message dialogs. It also includes data elements and their naming conventions.

These specifications are designed to be generic enough to encompass all types of equipment, but also complete enough to include the flexibility and extensibility needed to cover all requirements of these equipment types. Implementation can be simple, with only a few functions implemented, or more complex, with many or all functions supported. The same generic framework is used for all implementations.

Devices only use messages that are applicable to their intrinsic functionality. To simplify this, the dialogs are broken down into classes. To be compliant with the VEI specification, it is not necessary to implement any other dialogs than are needed to satisfy the intrinsic functionality. Examples of the different kinds of dialog implementation include (but are not limited to) the following:

- A. Sending only alarms. This requires implementing only one dialog.
- B. Status monitoring. This gives the Managing Host Computer (MHC) the ability to ask for the status of various conditions, and to enable/disable alarm conditions.
- C. Control of functional parameters, such as what types of payment are accepted (dimes, quarters, dollar bills, etc.) or reading and setting the date/time. This entails allowing network access to certain variable data elements, addressed by name.
- D. Local logging. With these dialogs, the device can keep a log of events as they happen, along with relevant data describing each condition. For example, when a vault is removed, the log entry should contain the date and time, the value of money within it, etc. At intervals, the MHC can upload the log file and purge it.
- E. Software and operating data version control. This involves the ability of the MHC to determine and update the software and operating data versions, via downloading.
- F. Credit/debit card support. With these dialogs, the device can request authorization and can charge vended items to credit or debit accounts.
- G. Process control and terminal I/O. This allows the MHC to control programs that are running on the device. Such control may involve executing, stopping, restarting, sending input to, and receiving output from any process running on the device.

This document is not an all-encompassing specification of all aspects of computer control of vending equipment. Such characteristics as internal vending equipment operation, vending equipment functionality, and network or vending equipment user interfaces are beyond the scope of this document.

This document has been written assuming a basic understanding of vending and computer programming concepts. This overview section is designed to be generally accessible and understandable by nontechnical personnel; however, other sections may require a basic understanding of vending equipment capabilities, hierarchical layered protocols, and general programming concepts such as state machines, bitwise operations, and code sets.

- References Other documents that might be useful include:
1. ANSI X3.92-1981 American National Standard Data Encryption Algorithm, American National Standards Institute, New York, NY, 1981.
 2. FIPS PUB 46 Data Encryption Standard, National Bureau of Standards, National Technical Information Service, Springfield, VA, 1977.
 3. ANSI X3.106-1983 American National Standard for Information Systems — Data Encryption Algorithm — Modes of Operation, American National Standards Institute, New York, NY, 1983.
 4. FIPS PUB 81 DES Modes of Operation, National Bureau of Standards, National Technical Information Service, Springfield, VA, 1980.
 5. Vending Industry Data Transfer Standard (VIDTS) Version 2, National Automatic Merchandising Association, Chicago, IL, 1992.
 6. Douglas E. Comer, Internetworking with TCP/IP, Volume I, Prentice Hall, Inc., Englewood Cliffs, NJ, 1991.
 7. CCITT, Recommendation X.25, Blue Book Volume VIII - Fascicle VIII.2, p.155 - 314, International Telecommunication Union, Geneva, 1989.
 8. CCITT, Recommendation X.509, Blue Book Volume VIII - Fascicle VIII.8, p. 48 - 81, International Telecommunication Union, Geneva, 1989.
 9. ISO/IEC 9798-2:1994, International Organization for Standardization, Geneva, 1994.

Chapter 6 Implementation Description

6.1 Overview

This chapter provides concrete examples of the steps necessary to implement the Vending Equipment Interface (VEI) Specification. These steps are as follows:

Answer Questions	1	What different device types are to be used? For example, a vending machine model XYZ.
	2	What type of network will be used? For example, TCP/IP over Ethernet.
	3	What network functionality will be used by each device type? For example, we need to send alarms from the equipment to a central location, and we need to be able to determine when inventory is getting low inside the machine.
	4	For each network function above, what dialog will be used? For example, for sending alarms, we would use the "Condition— Unsolicited Status" dialog. See Chapter 9 for an introduction to the dialogs. See Chapters 10 - 15 for details.
	5	For each dialog chosen, what data items will be used? For example, when a bank note vault is removed, the value of money in the vault should be included in the event message. See Chapters 16-18 for a thorough discussion of data items.
	6	For each data item specified, what are the valid values that it can take? For example, to specify the money in a bank note vault by denomination requires the definition of Bank Note Types (i.e., 1=one dollar bill, 2=two dollar bill, etc.).
Implement network	Implement any low-level network functions. For example, the data link protocol to be used over a dial-up phone line. (See Chapter 7).	
Implement protocols	Implement the Transport and Session layers. (See Chapter 8 and Appendix C.)	
Implement dialogs	Implement each dialog chosen using the data items and values specified.	
Create Documentation	Create a document that specifies the responses to the six questions answered in item 1.	

6.2 First example.

Answer Questions	1	The device type is an Acme Gumball Vendor, model X-22.
	2	The network consists of asynchronous serial leased line modems, using the Agent Contention Protocol.
	3	The only network functionality used is to send an intrusion alarm when the door is opened without the insertion of the security ID card into the hidden slot.
	4	The dialog that supports intrusion alarms is the “Condition—Unsolicited Status.”
	5	There are no additional data items. The following data items apply: <ul style="list-style-type: none"> a. Condition ID=1 is an intrusion alarm. b. The priority and severity will always be 99. c. Manufacturer ID is “ACX” d. Model number is X-22.
	6	There are no additional data values to specify.
Implement network	To implement the network, do the following: <ul style="list-style-type: none"> a. Purchase leased line modems, b. Setup the telephone line with our local carrier, c. Obtain or write software to implement the Agent Contention Protocol. 	
Implement protocols	To implement the Transport and Session layers, obtain or write software to do it.	
Implement dialogs	In software inside the X-22 gumball vendor, when the door is opened, check if the security ID has been inserted within the last minute. If so, the door opening is normal, and so no action needs to be taken. If not, the door opening is an intrusion, and we need to send the intrusion alarm. After an intrusion alarm is activated, if a valid security ID is inserted, send the cancellation of the intrusion alarm.	
Create Documentation	Documentation is as specified above in answers 1 - 6.	

6.3 Second example.

Answer Questions	1	The device type is an Acme Ticket Validator, model F-16.
	2	The network will consist of 10BASE2 ethernet, using TCP/IP.
	3	<p>The network functionality required is as follows:</p> <ul style="list-style-type: none"> a. Every validation results in an event message. b. The following conditions can be monitored remotely: <ul style="list-style-type: none"> i. Is the door open? ii. Is the ink cartridge removed? iii. Is there a magnetic reader/writer error? iv. Is the ticket path jammed? v. Is there an intrusion alarm? c. Intrusion alarms can be disabled remotely. d. The internal clock is synchronized over the network. e. The device saves all validation events and condition status changes to a log file. This file can be uploaded and purged periodically.
	4	<p>The following dialogs are used to meet the functionality:</p> <ul style="list-style-type: none"> a. For validation events and change of condition statuses, use the dialog "Condition — Unsolicited Status." b. For reporting of status of conditions, use the dialog "Condition— Report Status." c. For enabling/disabling intrusion alarms remotely, use the dialog "Condition— Read Attributes" and the dialog "Condition— Alter Attributes." d. For synchronizing the internal clock, use the dialogs "Variable Access— Read" and "Variable Access— Write." e. For managing the log file, use the dialogs "Log File— Report log status," "Log File— Upload log," and "Log File— Initialize log."
	5	<p>The following data items are required:</p> <ul style="list-style-type: none"> a. For validation events: <ul style="list-style-type: none"> i. _CAA032: Item Type ii. _CAA033: Serial number of item b. For all events: _CAA001: Record Sequence Number from log file. c. For synchronizing the clock, use the variable name: systemdatetime.

	<p>6 The following data values are required:</p> <ul style="list-style-type: none"> a. The following condition ID's apply: <ul style="list-style-type: none"> i. Validation: 2201 ii. Door open: 2202 iii. Ink cartridge removed: 2203 iv. Magnetic reader/writer error: 2204 v. Ticket path jammed: 2205 vi. Intrusion alarm: 2206 b. The following item types are defined: <ul style="list-style-type: none"> i. One-way peak: 1 ii. One-way off-peak: 2 iii. Round-trip peak: 3 iv. Round-trip off-peak: 4 v. Ten-trip peak: 5 vi. Ten-trip off-peak: 6 vii. Monthly: 7 viii. Weekly: 8 ix. Handicap one-way: 9 x. Handicap round-trip: 10 xi. Handicap Ten-trip: 11 xii. Handicap Monthly: 12 xiii. Handicap Weekly: 13
Implement network	<p>To implement the network, do the following:</p> <ul style="list-style-type: none"> a. Install network adapters, b. Install 10BASE2 coax with BNC adapters.
Implement protocols	<p>To implement the Transport and Session layers, obtain or write software to do it.</p>
Implement dialogs	<p>In software inside the F-16 ticket validator, the following functionality is performed.</p> <ul style="list-style-type: none"> a. Whenever a ticket is validated or one of the conditions changes status, use the "Condition— Unsolicited Status" dialog, and log the data to the log file. b. When the network initiates any of the following, respond appropriately: <ul style="list-style-type: none"> i. Condition— Report Status ii. Condition— Read Attributes iii. Condition— Alter Attributes iv. Variable Access— Read v. Variable Access— Write vi. Log File— Report log status vii. Log File— Upload log viii. Log File— Initialize log
Create Documentation	<p>Documentation is as specified above in answers 1 - 6.</p>

6.4 Third example

Answer Questions	1	The device type is an Acme Registering Farebox, model ARF-201.
	2	The network will consist of TCP/IP over IrDA Infrared at 115.2 Kbps.
	3	The network functionality required is as follows: <ol style="list-style-type: none"> a. Log events that occur to the log file. Periodically, log counters to the log file as a comment (in a compressed form). Make the log file available to the network. b. On demand from the MHC, receive new operating data and programs (in .zip format). c. Report the current contents of the vault on demand. d. Open the vault door on command.
	4	The following dialogs are used to meet the functionality: <ol style="list-style-type: none"> a. For managing the log file, use the dialogs "Log File — Report log status," "Log File — Upload log," and "Log File — Initialize log." b. For checking file versions, use "File Management — Read Directory." For sending new files, use "File Management — Transfer request" and "File Management — Transfer block." c. To report the current contents of the vault, use the dialog "Variable Access — Read." d. For the command to open the door, use the dialog "Program Execution — Spawn."
	5	The following data items are required: <ol style="list-style-type: none"> a. For all events: <ol style="list-style-type: none"> i. _CAA002: PIN of operator ii. _CAA055: Route ID iii. _CAA059: Run number b. For comment data, selected fields are posted from <ol style="list-style-type: none"> i. acct.sale ii. acct.prod[j] iii. acct.cash iv. acct.oth c. For contents of the vault, use <ol style="list-style-type: none"> i. sysconf.sa[0].contents ii. acct.sub[0].ct[j].type, count, value

	<p>6 The following data values are required:</p> <ul style="list-style-type: none"> a. The following condition ID's apply: <ul style="list-style-type: none"> i. Operator log on: 7201 ii. Operator log off: 7202 iii. Select route and run: 7203 iv. Power loss: 7204 v. Power restored: 7205 vi. Enter bypass mode: 7206 vii. Clear bypass mode: 7207 viii. Unjammer activated: 7208 b. The following content types are defined: <ul style="list-style-type: none"> i. Penny: 1 ii. Nickel: 2 iii. Dime: 3 iv. Quarter: 4 v. Fifty cent piece: 5 vi. Silver dollar: 6 vii. Susan B. Anthony dollar: 7 viii. Subway Token: 8 ix. One dollar bill: 9 x. Two dollar bill: 10 xi. Five dollar bill: 11 xii. Ten dollar bill: 12 xiii. Twenty dollar bill: 13
Implement network	<p>To implement the network, do the following: Install IrDA compatible Infrared ports and obtain software for IrDA and TCP/IP protocols.</p>
Implement protocols	<p>To implement the Transport and Session layers, obtain or write software to do it.</p>
Implement dialogs	<p>In software inside the ARF-201 farebox, the following functionality is performed.</p> <ul style="list-style-type: none"> a. Whenever an event occurs, log it to the log file. b. Periodically, log the counter data to the log file. c. When the network initiates any of the following, respond appropriately: <ul style="list-style-type: none"> i. "Log File — Report log status" ii. "Log File — Upload log" iii. "Log File — Initialize log" iv. "File Management — Read Directory" v. "File Management — Transfer request" and "File Management — Transfer block" vi. "Variable Access — Read" vii. "Program Execution — Spawn"
Create Documentation	<p>Documentation is as specified above in answers 1 - 6.</p>

6.5 Fourth example.

In this example, a complete system is desired to support parking applications. This involves supporting five different device types:

- C Ticket issuers (for entry into the parking area),
- C Pay-on-foot terminals (for adding the correct value before exiting),
- C Exit ticket readers (for people who paid at the Pay-on-foot terminal),
- C Clerk booth fee computers (for people that do not use the Pay-on-foot terminal), and
- C Lot sign controllers (to indicate whether the lot is full).

For each device type, we go through the process as before, defining how the device will interact over the network. Note that in some cases, we have not been exhaustive in listing all possible applicable items.

Ticket Issuers:

Answer Questions	1	The device type is an Acme Ticket Issuer, model ATI-2000.
	2	The network consists of a dedicated asynchronous serial connection over short-haul modems using the ISO 1745 protocol.
	3	<p>The network functionality used is as follows</p> <ul style="list-style-type: none"> a. When a ticket is issued, an event message is sent up line. b. The following conditions are indicated: c. The internal clock is synchronized over the network. d. The device saves all ticket issuance events and condition status changes to a log file. This file can be uploaded and purged periodically.
	4	<p>The following dialogs are used to meet the functionality:</p> <ul style="list-style-type: none"> a. For ticket issuance events and change of condition statuses, use the dialog "Condition — Unsolicited Status." b. For reporting of status of conditions, use the dialog "Condition— Report Status." c. For synchronizing the internal clock, use the dialogs "Variable Access— Read" and "Variable Access— Write." d. For managing the log file, use the dialogs "Log File— Report log status," "Log File— Upload log," and "Log File— Initialize log."
	5	<p>The following data items are required:</p> <ul style="list-style-type: none"> a. For ticket issuance events: _CAA033: Serial number of item b. For all events: _CAA001: Record Sequence Number from log file. c. For synchronizing the clock, use the variable name: systemdatetime.

	<p>6 The following condition ID's apply:</p> <ul style="list-style-type: none"> a. Ticket issuance: 2001 b. Door open: 2002 c. Ticket stock low: 2003 d. Ticket stock empty: 2004 e. Printer error: 2005
Implement network	<p>To implement the network, do the following:</p> <ul style="list-style-type: none"> a. Purchase short haul modems, b. Run the cable lengths as needed, c. Obtain or write software to implement the ISO 1745 protocol.
Implement protocols	<p>To implement the Transport and Session layers, obtain or write software to do it.</p>
Implement dialogs	<p>In software inside the ticket issuer, the following functionality is performed.</p> <ul style="list-style-type: none"> a. Whenever a ticket is issued or one of the conditions changes status, use the "Condition— Unsolicited Status" dialog, and log the data to the log file. b. When the network initiates any of the following, respond appropriately: <ul style="list-style-type: none"> i. Condition— Report Status ii. Variable Access— Read iii. Variable Access— Write iv. Log File— Report log status v. Log File— Upload log vi. Log File— Initialize log
Create Documentation	<p>Documentation is as specified above in answers 1 - 6.</p>

Pay-on-foot Terminals:

Answer Questions	1	The device type is an Acme Pay-on-foot Terminal, model POF-3000.
	2	The network consists of a dedicated asynchronous serial connection over short-haul modems using the ISO 1745 protocol.
	3	<p>The network functionality used is as follows</p> <ul style="list-style-type: none"> a. When a ticket is paid, an event message is sent up line. b. The following conditions are indicated: <ul style="list-style-type: none"> i. Is the door open? ii. Is the device short on change? iii. Is the device out of change? iv. Is the bill vault full? v. Is the coin vault full? vi. Is there a ticket handling error? vii. Is there a coin system error? viii. Is there a bill system error? ix. Is there an intrusion alarm? x. Lots of other conditions... c. The internal clock is synchronized over the network. d. The device saves all ticket paying events and condition status changes to a log file. This file can be uploaded and purged periodically.
	4	<p>The following dialogs are used to meet the functionality:</p> <ul style="list-style-type: none"> a. For ticket paying events and change of condition statuses, use the dialog "Condition — Unsolicited Status." b. For reporting of status of conditions, use the dialog "Condition— Report Status." c. For synchronizing the internal clock, use the dialogs "Variable Access— Read" and "Variable Access— Write." d. For managing the log file, use the dialogs "Log File— Report log status," "Log File— Upload log," and "Log File— Initialize log."
	5	<p>The following data items are required:</p> <ul style="list-style-type: none"> a. For ticket paying events: <ul style="list-style-type: none"> i. _CAA033: Serial number of item ii. _CAA006: Total value iii. _CAA017 and following: Breakdown by denomination. iv. _CAA029: Entry time. b. For all events: _CAA001: Record Sequence Number from log file. c. For synchronizing the clock, use the variable name: systemdatetime.

	<p>6 The following condition ID's apply:</p> <ul style="list-style-type: none"> a. Ticket paid: 3001 b. Door open: 3002 c. Change low: 3003 d. Change empty: 3004 e. Bill vault full: 3005 f. Coin vault full: 3006 g. Ticket handling error: 3007 h. Coin system error: 3008 i. Bill system error: 3009 j. Intrusion alarm: 3010
Implement network	<p>To implement the network, do the following:</p> <ul style="list-style-type: none"> a. Purchase short haul modems, b. Run the cable lengths as needed, c. Obtain or write software to implement the ISO 1745 protocol.
Implement protocols	<p>To implement the Transport and Session layers, obtain or write software to do it.</p>
Implement dialogs	<p>In software inside the ticket issuer, the following functionality is performed.</p> <ul style="list-style-type: none"> a. Whenever a ticket is paid or one of the conditions changes status, use the "Condition— Unsolicited Status" dialog, and log the data to the log file. b. When the network initiates any of the following, respond appropriately: <ul style="list-style-type: none"> i. Condition— Report Status ii. Variable Access— Read iii. Variable Access— Write iv. Log File— Report log status v. Log File— Upload log vi. Log File— Initialize log
Create Documentation	<p>Documentation is as specified above in answers 1 - 6.</p>

Exit Ticket Readers:

Answer Questions	1	The device type is an Acme Exit Ticket Reader, model ETR-4000.
	2	The network consists of a dedicated asynchronous serial connection over short-haul modems using the ISO 1745 protocol.
	3	<p>The network functionality used is as follows</p> <ul style="list-style-type: none"> a. When a ticket is accepted, an event message is sent up line. b. The following conditions are indicated: <ul style="list-style-type: none"> i. Is the door open? ii. Is there a ticket handling error? c. The internal clock is synchronized over the network. d. The device saves all ticket acceptance events and condition status changes to a log file. This file can be uploaded and purged periodically.
	4	<p>The following dialogs are used to meet the functionality:</p> <ul style="list-style-type: none"> a. For ticket acceptance events and change of condition statuses, use the dialog "Condition — Unsolicited Status." b. For reporting of status of conditions, use the dialog "Condition— Report Status." c. For synchronizing the internal clock, use the dialogs "Variable Access— Read" and "Variable Access— Write." d. For managing the log file, use the dialogs "Log File— Report log status," "Log File— Upload log," and "Log File— Initialize log."
	5	<p>The following data items are required:</p> <ul style="list-style-type: none"> a. For ticket issuance events: _CAA033: Serial number of item b. For all events: _CAA001: Record Sequence Number from log file. c. For synchronizing the clock, use the variable name: systemdatetime.
	6	<p>The following condition ID's apply:</p> <ul style="list-style-type: none"> a. Ticket acceptance: 4001 b. Door open: 4002 c. Ticket handling error: 4003
Implement network	<p>To implement the network, do the following:</p> <ul style="list-style-type: none"> a. Purchase short haul modems, b. Run the cable lengths as needed, c. Obtain or write software to implement the ISO 1745 protocol. 	
Implement protocols	<p>To implement the Transport and Session layers, obtain or write software to do it.</p>	

Implement dialogs	<p>In software inside the ticket issuer, the following functionality is performed.</p> <ul style="list-style-type: none"> a. Whenever a ticket is accepted or one of the conditions changes status, use the "Condition— Unsolicited Status" dialog, and log the data to the log file. b. When the network initiates any of the following, respond appropriately: <ul style="list-style-type: none"> i. Condition— Report Status ii. Variable Access— Read iii. Variable Access— Write iv. Log File— Report log status v. Log File— Upload log vi. Log File— Initialize log
Create Documentation	Documentation is as specified above in answers 1 - 6.

Clerk Booth Fee Computers:

Answer Questions	1	The device type is an Acme Clerk Booth Fee Computer, model CBC-5000.
	2	The network consists of a dedicated asynchronous serial connection over short-haul modems using the ISO 1745 protocol.
	3	<p>The network functionality used is as follows</p> <ul style="list-style-type: none"> a. When a ticket is accepted for exit, an event message is sent up line. b. When a person pays to exit, an event is sent up line. c. The following conditions are indicated: <ul style="list-style-type: none"> i. Is the door open? ii. Is the device low on ticket stock? iii. Is the device out of ticket stock? iv. Is there a printer error? v. Is there a sales agent logged in? d. The internal clock is synchronized over the network. e. The device saves all ticket acceptance events, exit purchases, and condition status changes to a log file. This file can be uploaded and purged periodically. f. Keep track of who logs in.
	4	<p>The following dialogs are used to meet the functionality:</p> <ul style="list-style-type: none"> a. For ticket issuance events and change of condition statuses, use the dialog "Condition — Unsolicited Status." b. For reporting of status of conditions, use the dialog "Condition— Report Status." c. For synchronizing the internal clock, use the dialogs "Variable Access— Read" and "Variable Access— Write." d. For managing the log file, use the dialogs "Log File— Report log status," "Log File— Upload log," and "Log File— Initialize log."

	<p>5 The following data items are required:</p> <ul style="list-style-type: none"> a. For ticket acceptance events: _CAA033: Serial number of item b. For ticket paying events: <ul style="list-style-type: none"> i. _CAA033: Serial number of item ii. _CAA006: Total value iii. _CAA017 and following: Breakdown by denomination. iv. _CAA029: Entry time. c. For all events: <ul style="list-style-type: none"> i. _CAA001: Record Sequence Number from log file. ii. _CAA002: Personal ID Number of person logged in. d. For synchronizing the clock, use the variable name: systemdatetime.
	<p>6 The following data values are required:</p> <p>The following condition ID's apply:</p> <ul style="list-style-type: none"> a. Ticket acceptance: 5001 b. Ticket payment 5002 c. Door open: 5002 d. Ticket stock low: 5003 e. Ticket stock empty: 5004 f. Printer error: 5005
Implement network	<p>To implement the network, do the following:</p> <ul style="list-style-type: none"> a. Purchase short haul modems, b. Run the cable lengths as needed, c. Obtain or write software to implement the ISO 1745 protocol.
Implement protocols	<p>To implement the Transport and Session layers, obtain or write software to do it.</p>
Implement dialogs	<p>In software inside the ticket issuer, the following functionality is performed.</p> <ul style="list-style-type: none"> a. Whenever a ticket is accepted or paid or one of the conditions changes status, use the "Condition— Unsolicited Status" dialog, and log the data to the log file. b. When the network initiates any of the following, respond appropriately: <ul style="list-style-type: none"> i. Condition— Report Status ii. Variable Access— Read iii. Variable Access— Write iv. Log File— Report log status v. Log File— Upload log vi. Log File— Initialize log
Create Documentation	<p>Documentation is as specified above in answers 1 - 6.</p>

Lot Sign Controllers:

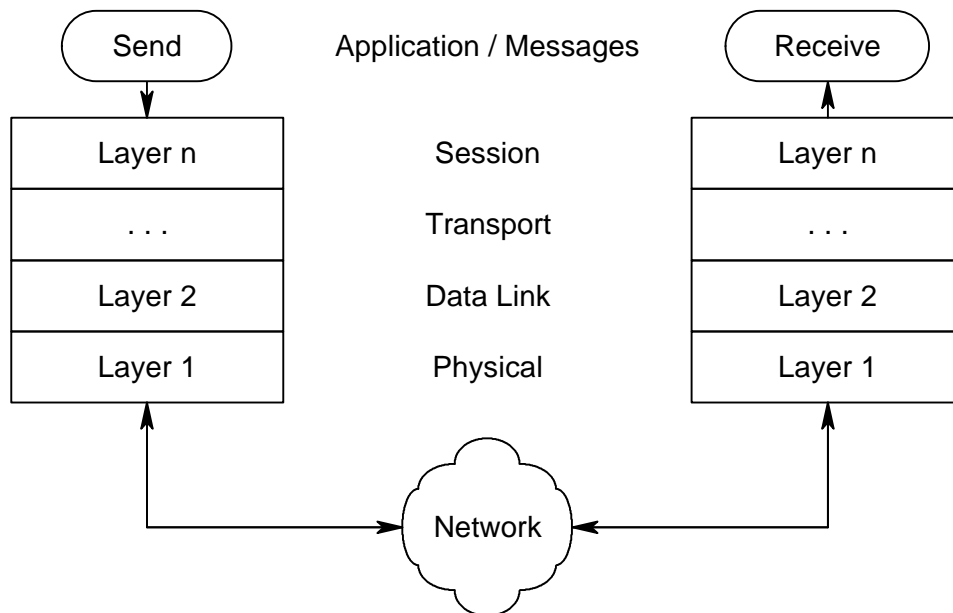
Answer Questions	1	The device type is an Acme Lot Sign Controller, model LSC-6000.
	2	The network consists of a dedicated asynchronous serial connection over short-haul modems using the ISO 1745 protocol.
	3	The network functionality used is as follows: The device is told how many spaces are free, from zero up to the capacity of the lot. This is based on the number of tickets issued but not yet accepted.
	4	The following dialogs are used to meet the functionality: For reading and setting the number of free spaces, use the dialogs "Variable Access—Read" and "Variable Access— Write."
	5	The following data items are required: For the number of free spaces, use the variable name: ACM_freespaces.
	6	There are no data values to specify.
Implement network	To implement the network, we do the following: <ul style="list-style-type: none"> a. Purchase short haul modems, b. Run the cable lengths as needed, c. Obtain or write software to implement the ISO 1745 protocol. 	
Implement protocols	To implement the Transport and Session layers, obtain or write software to do it.	
Implement dialogs	In software inside the lot sign controller, the following functionality is performed. <ul style="list-style-type: none"> a. Display the number of free spaces on the sign. b. Accept network usage of the dialogs "Variable Access—Read" and "Variable Access— Write". 	
Create Documentation	Documentation is as specified above in answers 1 - 6.	

Chapter 7

Protocol: Physical and Link Layers

7.1 Structure of this document

Modern communications protocols are layered, which means that the various functions performed by the protocol are partitioned from a low to a high level, being separated into layers. Each layer provides transparent services to the layers above, communicating directly with the equivalent layer of the connected peer device via the layers below.



The Vending Equipment Interface Specification defines the following layers:

- | | | |
|----|------------------------|----------------------|
| 1. | Physical and Data Link | See the next section |
| 2. | Transport and Session | See Chapter 8 |
| 3. | Dialogs | See Chapters 9 - 15 |
| 4. | Data Items | See Chapters 16 - 18 |

7.2 Overview.

The low layers of the protocol are specified here. This includes the physical layer, which is responsible for

1. link establishment and reporting,
2. data transfer, and
3. low level flow control.

The link layer sits above the physical layer and is responsible for

1. managing the physical layer,
2. timeout control,

3. transparency (encoding data for physical transfer),
4. framing of data (marking the beginning and ending of data blocks),
5. line control (rules of contention), and
6. error detection and correction (usually retransmission).

The following physical and link layers are supported:

1. Transmission Control Protocol (TCP) / Internet Protocol (IP) (see Internetworking with TCP/IP for further details).
2. X.25 (see CCITT Blue Book Volume VIII for further details).
3. The Agent Contention Link Layer over any of the following:
 - A. Leased line modems or direct RS-232C asynchronous serial connection.
 - B. Dial-up connection with asynchronous serial modems.
4. ISO 1745 protocol over any of the following:
 - A. Leased line modems or direct RS-232C asynchronous serial connection.
 - B. Dial-up connection with asynchronous serial modems.
5. Infrared Data Association (IrDA) Serial Infrared Physical Layer with Infrared Data Association Serial Infrared Link Access Protocol (IrLAP). See <http://www.irda.org>

Other standards-conforming physical and link layers can be supported. Preference is given to TCP/IP, since it has become the universal networking standard.

Chapter 8

Protocol: Transport and Session Layers

8.1 Overview.

The transport and session layers are specified here. The transport layer is implemented to provide the following functionality:

1. managing the lower layers,
2. sequence control, and
3. end-to-end data integrity.

The Agent Transport Layer (ATL) is specified in section 8.2.

The session layer is implemented here as an option to provide the following functionality:

1. managing the transport layer,
2. mutual identification and authentication, and optionally
3. message level encryption, and
4. message level data compression.

The Agent Session Layer (ASL) is specified in section 8.3. The ASL is not required if a lower layer provides authentication, encryption, and data compression (such as IPv6 or secure sockets), or if this functionality is not desired.

When the VEI is used in a broadcast scenario, the transport and session layers do not apply.

8.2 The Agent Transport Layer (ATL).

8.2.1 Transport layer overview.

The transport layer is entrusted with end-to-end data integrity, which means that the sender retransmits lost blocks and the receiver discards duplicated blocks. It also has the job of sequence control, which means that the higher levels of the protocol receive packets in the exact order in which they were sent. The ATL achieves this integrity and sequencing by assigning a sequence number to each block and acknowledging blocks by sequence number at the message level.

Protocols that require positive acknowledgment for each block before transmitting the next are inefficient, since much time is spent waiting for acknowledgments to arrive. The ATL, therefore, uses a sliding window concept, where a (tunable) window full of blocks can be sent before requiring an acknowledgment to send the next block. As acknowledgments arrive, the window slides allowing the transmitter to send at least as many blocks as were acknowledged (up to a full window).

8.2.2 Transport header.

The send and receive sides of the transport layer communicate with each other using a transport header. It is prefixed onto each buffer by the send side and stripped by the receive side. Its structure is as follows:

Magic Number	Binary	1 Byte	(ASCII 'K' = CBH)
Block Length	Binary	2 Bytes	(Whole block, including header)
Sequence Number	Binary	2 Bytes	(Of current block)
Acknowledgment Number	Binary	2 Bytes	(Next block sequence expected)
Control Bits	Binary	1 Byte	
Synchronize sequence numbers		Bit 7	(0x80)
Reset		Bit 6	(0x40)
Disconnecting		Bit 5	(0x20)
Need Acknowledgment		Bit 4	(0x10)
Following block missing		Bit 3	(0x08)
Acknowledgment Num Valid		Bit 2	(0x04)
Fresh Connection		Bit 1	(0x02)
Reserved		Bit 0	

Binary fields longer than a single byte are sent with the most significant byte (MSB) first.

The `Magic Number` is used to establish that the transport headers are aligned and initialized.

The `Block Length` field indicates the length of the entire block, including the transport header (data length + 8 bytes).

The `Sequence Number` field is the number identifying the data in the current block. If the block length is eight, this field has no meaning, since there is no data that would require an acknowledgment.

The `Acknowledgment Number` is the sequence number of the next block that the receiver wants to get. In other words, blocks in the window with sequence numbers below the acknowledgment number have been received successfully.

The `control bits` are used to indicate certain specialized functions.

The flag `Synchronize Sequence Numbers` is used during the synchronization process described in the next section. It indicates that the `Sequence Number` field contains the initial sequence number from this side.

The `Reset` flag indicates to the receiver that the sequence numbers are out of synchronization, and that both sides should go to the START state (see the next section).

The `Disconnecting` flag indicates to the receiver that the sender is about to terminate the connection.

The `Need Acknowledgment` flag indicates that the receiver expects an immediate acknowledgment message. However, the receiver is at liberty to send an acknowledgment whether or not this bit is set. This bit will usually be clear when the window is not full and

there are more blocks to be sent.

The `Fresh Connection` flag can be used during the synchronization process. When set, it indicates that a new connection is being established, that there are no unacknowledged received messages from previous connections. When this flag is clear and the sequence number falls within the previously active window and messages were still in transit, those messages below the acknowledgment number can be considered to have been received. Received messages may remain in a buffer when they were received but the acknowledgment had not yet been sent when the line disconnected.

The `Following block missing` flag indicates that the block with the `Sequence Number` set to the `Acknowledgment Number` needs to be retransmitted. This flag should not be set in absence of the `Acknowledgment Num Valid` flag.

The `Acknowledgment Num Valid` flag indicates that the field `Acknowledgment Number` is valid. If not set, the header is not an acknowledgment packet.

8.2.3 Transport synchronization.

Each block is assigned a `sequence number`. This `sequence number` prevents lost blocks since the receiver can detect non-sequential sequence numbers and request a retransmission. It prevents duplicated blocks since it can detect a block that contains the same sequence number and can discard them.

Upon establishing a new connection, the transport layer must determine the initial `sequence number` of the other side. It accomplishes this with a three-message handshake.

8.2.3.1 Sequence diagrams.

1. Normal synchronization.

<u>Initiator</u>		<u>Responder</u>
SYNC seq=x	----->	
		<----- SYNC seq=y , ACK x+1
ACK y+1	----->	
		Established synchronization.

2. Both sides want to initiate.

Communication line is silent.		
SYNC seq=x	<----->	SYNC seq=y
SYNC seq=x, ACK y+1	<----->	SYNC seq=y, ACK x+1
ACK y+1	<----->	ACK x+1
		Established synchronization.

3. Sender established, receiver not.

Block	----->
-------	--------

- <----- RESET

SYNC seq=x ----->

...
- 4. Timeout (usually 30 seconds) at start of synchronization sequence.
 - SYNC seq=x ----->

Not received, timeout. <----- SYNC seq=y, ACK x+1

SYNC seq=x ----->
- 5. Garbage at start of synchronization sequence.
 - SYNC seq=x ----->

<----- Not a SYNC or SYNC,ACK

RESET ----->

SYNC seq=x ----->

...
- 6. Timeout or garbage during synchronization sequence.
 - SYNC seq=x ----->

<----- SYNC seq=y, ACK x+1

ACK y+1 -----> Not received, timeout, or garbage.

<----- RESET

SYNC seq=x ----->

...

8.2.3.2 State machine.

The Agent Transport Layer can be implemented with a state machine. Generally, a state involves performing an action (i.e., sending a message) and then reading the communications line and processing the response. Possible responses that cause state transitions are

1. A meaningful message (i.e., SYNC, SYNC/ACK, ACK, RESET, ...) is received.
2. Nothing is received (timeout - TOUT).
3. Something unexpected is received (GARB).
4. No reading is done and the transition is automatic (AUTO).

This machine contains the following states:

1. START This state is the entry point. The communication line is read to see if a message is there. Responses:
 - A. (SYNC) go to SEND_SACK.
 - B. (TOUT) and initiator, go to SEND_SYNC. The initiator is the side that has something to be sent.
 - C. (TOUT) and responder, stay in START state, and read again.

- D. (RESET) stay in START state.
 - E. (GARB) go to SEND_RESET.
2. SEND_SYNC From this state, the initiator sends a SYNC message, which contains a header, with the *Synchronize Sequence Numbers* bit set, and the *Sequence Number* field set to the initial sequence value (randomly chosen). The only data included is the sender's window size, which is a two byte binary field. It then waits for a response. Responses:
- A. (SYNC) go to SEND_SACK.
 - B. (SYNC/ACK) and good ACK, go to SEND_ACK. The ACK is considered good when the *Acknowledgment Number* equals one greater than the *Sequence Number* just sent.
 - C. (SYNC/ACK) and bad ACK, go to RESET.
 - D. (TOUT) stay in SEND_SYNC, and send SYNC message again.
 - E. (RESET) go to START.
 - F. (GARB) go to SEND_RESET.
3. SEND_SACK From this state, the transmitter sends a SYNC/ACK message, which contains a header, with both the *Synchronize Sequence Numbers* and *Acknowledgment Num Valid* bits set, the *Sequence Number* field set to the newly chosen initial value, and the *Acknowledgment Number* set to value of one greater than the *Sequence Number* from the SYNC message just received. The only data included is the sender's window size.
- A. (SYNC) stay in SEND_SACK, and send SYNC/ACK again.
 - B. (SYNC/ACK) and good ACK, go to SEND_ACK.
 - C. (SYNC/ACK) and bad ACK, go to RESET.
 - D. (TOUT or RESET) go to START.
 - E. (good ACK) go to ESTABLISH.
 - F. (GARB or bad ACK) go to SEND_RESET.
4. SEND_ACK From this state, the machine sends an ACK message, which contains only a header, with the *Acknowledgment Num Valid* bit set, and the *Acknowledgment Number* set to the value of one greater than the *Sequence Number* from the SYNC/ACK message it just received. After sent, go to the ESTABLISH state.
5. SEND_RESET From this state, the machine sends a RESET message, which contains only a header, with the *Reset* bit set. After sent, go to the START state.

6. **ESTABLISH** From this state, the machine can send and receive packets, using the sequence numbers just negotiated as the basis. If a RESET is received or the communication line drops and must be reestablished, go to START state.

The state machine can be easily summarized in the following matrix.

From	To	START	SEND_SYNC	SEND_SACK	SEND_ACK	SEND_RESET	ESTABLISH
START	RESET, TOUT-responder		TOUT-initiator	SYNC		GARB	
SEND_SYNC	RESET		TOUT	SYNC	good SYNC/ACK	GARB, bad SYNC/ACK	
SEND_SACK	TOUT or RESET			SYNC	good SYNC/ACK	GARB, bad ACK, bad SYNC/ACK	good ACK
SEND_ACK							AUTO
SEND_RESET	AUTO						
ESTABLISH	RESET						

8.2.4 Receive side processing.

Message level acknowledgments can be sent by themselves (an 8-byte packet containing only a header) or with a data block. The `Acknowledgment Num Valid` bit is set and the `Acknowledgment Number` is set to the sequence number that the receiver expects to receive next. If the receiver detects a lost block (i.e., a sequence number was skipped), it acknowledges the last block it received, and it sets the `Following Block Missing` flag. Acknowledgments do not have to be sent after each block is received. They must be sent, though, in the following circumstances:

1. When the `More Data` bit is not set on a data block.
2. When the other side's window full of data has been received and not yet acknowledged.
3. When blocks have not been acknowledged, and there is a read timeout.
4. When the receiver has a block it can send. Piggyback the data onto the acknowledgment.

Acknowledgment of a particular sequence number indicates acknowledgment of any sequence number of lesser value within a window size. This means that a single acknowledgment message can acknowledge multiple blocks.

Discard duplicate blocks.

8.2.5 Send side processing.

Retransmissions. The retransmission timer is the time (in milliseconds, if possible) to wait on an acknowledgment of a block before retransmitting it. It is an adjustable value that is modified according to the speed of the communications medium. Faster connections cause the retransmission timer to be shorter (down to a minimum). Slower connections cause the retransmission timer to be longer (up to a maximum). As each acknowledgment of a non-retransmitted block is received, the retransmission timer is adjusted using the following formula:

```

elapsed = CURRENT_TIME - time_of_first_attempt_to_send_block;
delta = elapsed - (avg_time / 8);
avg_time += delta;
if (delta < 0)
    delta = -delta;
std_dev_time += delta - (std_dev_time / 4);
retransmission_timer = ((avg_time / 4) + std_dev_time) / 2;
if (retransmission_timer < MIN_RETRANS_TIMER)
    retransmission_timer = MIN_RETRANS_TIMER;
else if (retransmission_timer > MAX_RETRANS_TIMER)
    retransmission_timer = MAX_RETRANS_TIMER;

```

8.2.6 Tunable parameters:

1. Transmit window size. Number of blocks to buffer in the send window. For slow connections, use four to eight. For faster connections (i.e., fiber optic), use 20 to 24.
2. Maximum retransmissions per block. Usually three.
3. Maximum time to synchronize before resetting. Usually 30 seconds.
4. Maximum retransmission timer value. 30 seconds for fast connections, 60 seconds for slow ones.
5. Minimum retransmission timer value. Slow connections, 3000 ms; for fast ones, use 1000 ms.

8.3 The Agent Session Layer (ASL).

8.3.1 Session layer overview.

The ASL is patterned after the CCITT X.509:1988 (The Directory — Authentication framework) specification and ISO/IEC 9798-2:1994 (Information technology — Security Techniques — Entity authentication — Part 2: Mechanisms using symmetric encipherment algorithms). The data compression is performed using the methods specified in RFC 1950 and RFC 1951.

The session layer handles identification, authentication, and optional compression and encryption of messages. Any time a connection is established (or reestablished), it must successfully exchange the identification and authentication messages before normal communication can continue on the communication line. If messages are sent without the identification/authentication completion, the receiver must disconnect, and optionally attempt to reconnect.

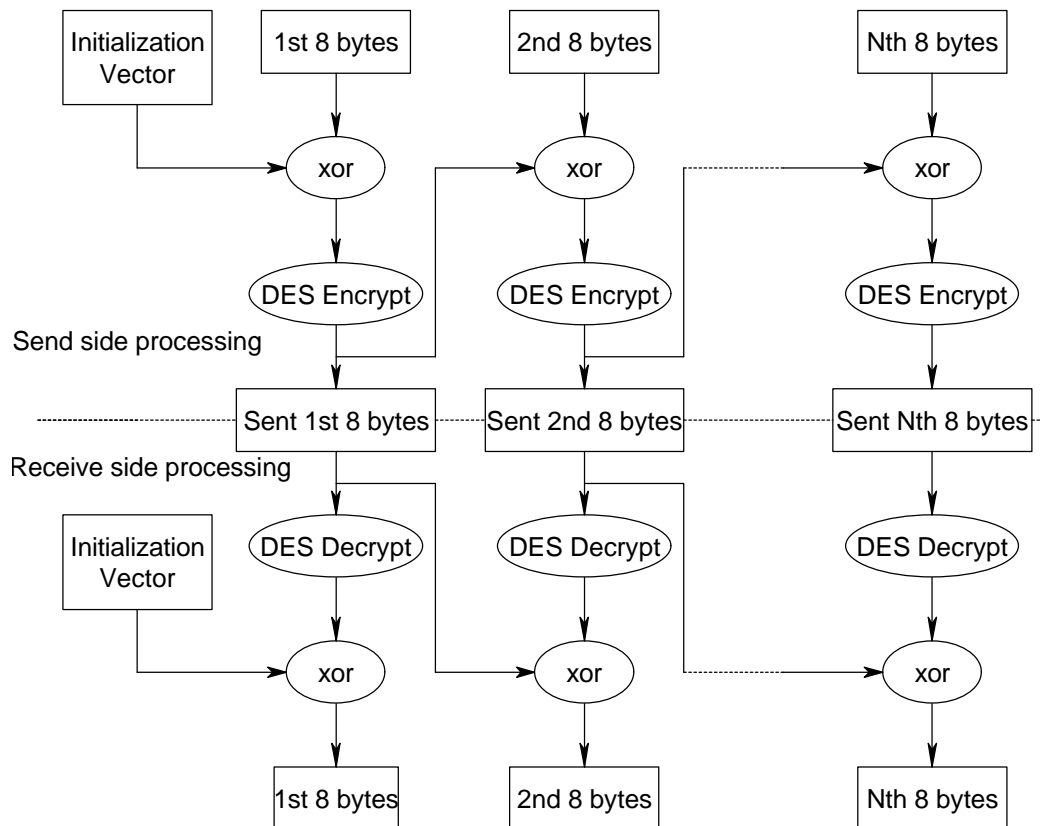
A capability byte is used to indicate the sender's ability to perform encryption of messages and compression of messages. Bit zero (LSB) set to one indicates the ability to perform message level encryption. Bit one set to one indicates the ability to perform message level compression.

Identity, as used in the ASL is a value that can be stored in 8 bytes or less. This document does not regulate or further specify the assignment of identity values.

Authentication is based on two entities knowing a secret password for one another. Such passwords can be stored in 8 bytes or less. This document does not regulate or further specify the assignment or distribution of passwords.

The one-way hash function used is the RSA Data Security, Inc. MD5a Message Digest Algorithm. It is always used on the entire unencrypted cleartext block.

Encryption is based on the Data Encryption Algorithm (DEA) Standard as specified in ANSI X3.92 or FIPS PUB 46. It is used in Cipher Block Chaining (CBC) mode as specified in ANSI X3.106 or FIPS PUB 81. The Initialization Vector will be the same as the DEA key, which will be the password. Padding shall be compliant with FIPS PUB 74, section 5.3.2 and FIPS PUB 81, Appendix C, paragraph 6.



8.3.2 Authentication request.

The client/slave sends the authentication request immediately on establishment of a connection. Its structure consists of the following:

Client identity	Binary	8 bytes	
Request sequence num	Binary	4 bytes	increasing
A Random Value	Binary	4 bytes	
Time of request	ASCII	6 bytes	(HHMMSS)
Date of request	ASCII	8 bytes	(YYYYMMDD)
Magic Number	Binary	1 bytes	5a
Client capability	Binary	1 byte	
Client identity	Binary	8 bytes	
Hash value	Binary	16 bytes	

The entire message after the initial Client identity field is encrypted using the DEA, with the key being the secret password for this device. The password itself is not actually transmitted.

8.3.3 Authentication response.

The server/master sends the authentication response after receiving the authentication request. If after decryption the Magic Number and Hash value are correct, the fields from the request match, and the client's capabilities are supported by and agreeable to the server, then the server/master responds by sending the authentication response message. Its structure is as follows:

Server identity	Binary	8 bytes	
Response sequence num	Binary	4 bytes	increasing
A random value	Binary	4 bytes	
Time of response	ASCII	6 bytes	(HHMMSS)
Date of response	ASCII	8 bytes	(YYYYMMDD)
Request sequence num	Binary	4 bytes	increasing
Time of request	ASCII	6 bytes	(HHMMSS)
Date of request	ASCII	8 bytes	(YYYYMMDD)
Server capability	Binary	1 byte	
Magic Number	Binary	7 bytes	a137990f8c52b7
Server's new DES key	Binary	32 bytes	
Server identity	Binary	8 bytes	
Hash value	Binary	16 bytes	

The entire message after the initial Server identity field is encrypted using the DEA, with the key being the secret password for the client. The server capability should never be set to exceed that of the client.

If the Magic Number and Hash value were not produced after decrypting the authentication request, the fields from the request did not match, or the client's capabilities are not supported by or agreeable to the server, then the server sends the following message:

Text	ASCII	6 bytes	"DENIED"
------	-------	---------	----------

8.3.4 Authentication acknowledgment.

The client/slave sends the authentication acknowledgment after receiving the authentication response. If after decryption the Magic Number and Hash value are correct, the fields from the response match, and the server's capabilities are supported by and agreeable to the client, then the client/slave responds by sending the following authentication acknowledgment message:

Response sequence num	Binary	4 bytes	increasing
A random value	Binary	2 bytes	
Time of response	ASCII	6 bytes	(HHMMSS)
Date of response	ASCII	8 bytes	(YYYYMMDD)
Client's new DES key	Binary	32 bytes	
Server identity	Binary	8 bytes	
Magic number	Binary	4 bytes	e4f06739
Hash value	Binary	16 bytes	

The entire message is encrypted using the DEA, with the key being the secret password for this device.

If the Magic Number and Hash value were not produced from decrypting the authentication response, the fields from the response did not match, or the server's capabilities are not supported by or agreeable to the client, then the client sends the following message:

Text	ASCII	6 bytes	"DENIED"
------	-------	---------	----------

If the server/master does not receive what it wants (i.e., a good authentication acknowledgment), it disconnects.

8.3.5 After authentication.

Messages to be sent from the application receive the following processing by the session layer.

1. The sender compresses the data using the deflate algorithm (with zlib), using the dictionary for the send direction.
2. The magic hex value 0xD3 is prefixed to the buffer.
3. Between one and eight characters are appended to the transmit buffer to make it a multiple of eight bytes in length. All but the last of these pad characters may be random. The final character is the pad count, which is the ASCII value '1' through '8', depending upon the actual number of bytes appended.
4. The sender encrypts the buffer using the sender's new randomly generated DES key in CBC mode with carry-over.

The receiver performs the inverse functions:

1. Decrypt the incoming buffer using the sender's DES key in CBC mode with carry-over.
2. Save the pad count value. A pad count value greater than eight or less than one indicates that both sides must re-authenticate. Strip the pad count and pad

characters from the buffer.

3. Strip the magic hex value 0xD3 from the start of the buffer.
4. Decompress the data using the inverse of the deflate algorithm (with zlib), using the dictionary for the receive direction.
5. Pass the buffer to the application.

Anytime after successful authentication the message "DENIED" is received, the receiver should go to the unauthenticated state and re-authenticate as necessary. At this time, both sides reinitialize their send and receive dictionaries for the zlib compression steps.

Chapter 9

Dialog Overview

9.1 Introduction

The Vending Equipment Interface Specification defines six dialog classes with several services provided by each. A dialog class is a grouping of functionally similar services. For example, the Variable Access dialog class consists of a set of services for using a variable on the Vending Equipment, such as to read the value of a variable.

Services are achieved by sending and receiving a set of messages called a dialog. A dialog consists of a request message, response message(s), and (sometimes) acknowledgment of the response(s). A request message, with or without data, may require a response, at the discretion of the requestor. In addition, a response message that contains data may require an acknowledgment message, at the discretion of the responder. However, some responses do not contain any data, and so they act as an acknowledgment. In this case, no separate acknowledgment message is required, as it would be an acknowledgment to an acknowledgment. See the following sequences.

<u>Sender</u>		<u>Receiver</u>
Request for data	---->	
	<-----	Response with data
Acknowledge receipt of response	---->	
<u>Sender</u>		<u>Receiver</u>
Request for service	---->	
	<-----	Response with success or failure
(No Acknowledgment needed.)		

Additionally, services are broken down into options. Following are their definitions.

USS	This involves the ability to send unsolicited condition status changes, such as alarms.
EVC	Event/Condition option. Any device that needs the network to disable alarms or request statuses would support this service.
VAR	Variable Control option. Any device that needs network control of local operating parameters would support this service. For example, what types of payment are accepted (dimes, quarters, dollar bills, etc.) or reading and setting the date/time. This entails allowing network access to certain variable data elements, addressed by name.
LOG	Logging option. Any device that supports journaling of all events, transactions, and variable data would support this service. For example, when a vault is removed, the event should log the date and time, the value of money contained, etc. At intervals, the MHC can upload the log file and

purge it.

- | | |
|------|---|
| FS | File System Control option. Any device that has a file system that should be accessed over the network would support this service. This service can be used to download new operating programs and data to a device, as an example. |
| FT | Funds Transfer option. Any device that needs to perform financial transactions such as account authorization would support this service. This option is used for devices that accept credit/debit cards as a form of payment. |
| MP | Multi Processing option. Any device that needs to have processes or local functions executed via network control would support this service. This option can be used to control the internal processes running at a device. |
| MP&T | Multi Processing with Terminal I/O option. Any device that has a process that acts as a slave terminal over the network would support this service. This option requires support of the MP optional services. This can be used as a generic monitor or shell by which a remote user can interact with programs running at a device. |

The options are implemented based on network functionality supported by the device. More complex devices usually provide all the functionality that simpler devices do.

9.2 Dialog Classes

Following is a breakdown of the dialog classes and the services they provide.

	<u>Dialog Class</u>	<u>Option Level</u>	<u>Services</u>
1.	Condition	USS Option	Unsolicited Status
		EVC Option	Report Status
		EVC Option	Read Attributes
		EVC Option	Alter Attributes
2.	Variable Access	VAR Option	Read
		VAR Option	Write
		VAR Option	List structure
3.	Log File	LOG Option	Create log
		LOG Option	Initialize log
		LOG Option	Report log status
		LOG Option	Delete log
		LOG Option	Upload log
		LOG Option	Write log
4.	File Management	FS Option	Transfer request
		FS Option	Transfer block
		FS Option	Terminate transfer
		FS Option	Read directory
		FS Option	Rename file
		FS Option	Delete file
5.	Transaction	FT Option	Authorization
		FT Option	Capture
		FT Option	Cancellation
		FT Option	Reversal
6.	Program Execution	MP Option	Spawn
		MP Option	Terminate
		MP Option	Restart
		MP Option	List program executions
		MP&T Options	Input
		MP&T Options	Output

9.3 Services

Following is a list of the services and their purpose. Note that each service is accomplished with a series of messages called a dialog.

9.3.1 Condition: A particular circumstance or situation at a device, together with the attributes that describe it. An intrusion alarm, a door opening, a component failure, and the sale of an article are all examples of condition status changes. The accompanying attributes might include for example the time of day at the device, the value of items sold, the denominations of coins and bills used to pay, or the serial number of a failing component.

9.3.1.1 Unsolicited status is a service initiated by a device at the time a condition status changes to alert the MHC that it happened. This is called an event.

9.3.1.2 Report status is a service initiated by the MHC to ask whether a condition is currently idle, active, or disabled. Inquiries can be made for a single condition, a list of conditions, all conditions, all active conditions, all disabled conditions, or all idle conditions.

9.3.1.3 Read attributes is a service initiated by the MHC to retrieve from the device the attributes of a particular condition. All conditions contain the following attributes:

- 9.3.1.3.1 Status: idle, active, or disabled. If the enabled attribute is true, then the status can be idle or active. If the enabled attribute is false, the status can be only disabled. Active means that the condition is currently true of the device. Idle means that the condition is not currently true of the device. For example, the door open condition is active when the door is open. It is idle when the door is closed. Usually, the idle status is preferred over the active to show proper operation.
- 9.3.1.3.2 Priority: Zero (lowest) to 10 (highest) This is an indication of the importance of the condition compared with other conditions.
- 9.3.1.3.3 Severity: Zero (best) to 10 (worst) This represents the effect the condition has on the device.
- 9.3.1.3.4 Enabled: True or False. If not enabled, status is always disabled.
- 9.3.1.3.5 Incident: True or False. False implies it is a remaining condition. For example, a door opening. True implies it is an occurrence that is only active for an instant. For example, the sale of an item.
- 9.3.1.3.6 Date/Time the condition last became active.
- 9.3.1.3.7 Date/Time the condition last became idle.
- 9.3.1.3.8 Contributes to overall device status: True or False. This might be true for certain component failures or other conditions that cause the device to be inoperable or partially operable (with reduced functionality).

Some conditions have additional attributes describing them.

9.3.1.4 Alter attributes is a service that allows modifiable attributes to be altered by the MHC. For example, the enabled attribute can be set to True or False for many conditions. Setting it to False causes the state of the condition to be set to disabled.

9.3.2 Variable access: The set of services related to accessing, by name, data items that are resident on the device. These data items may be simple types, compound types, arrays, or combinations of the same.

- 9.3.2.1 Read is a service initiated by the MHC to retrieve the value of a named variable or a list of named variables. Examples of variables that might be read are
- ! whether the device accepts 20 dollar bills,
 - ! the count of a certain type of transaction,
 - ! the name of the manufacturer of the device,
 - ! the device's overall status, or
 - ! certain financial counters.
- 9.3.2.2 Write is a service initiated by the MHC to modify the value of a named variable or a list of named variables. Examples of variables that might be written are
- ! to accept 20 dollar bills,
 - ! to force a device out of service,
 - ! to set the current date and time, or
 - ! to change the mode of operation.
- 9.3.2.3 List structure is a service initiated by the MHC to retrieve the specification of the structure of a data element down to its simple data types. For example, a date consists of three nonnegative integers, called year, month, and day.
- 9.3.3 A log file is a historic record of events and other archived information that can be used to determine a sequence of events at a device. Three types of data are stored in a log file: event data, variable data, and comment data.
- 9.3.3.1 Create log is a service initiated by the MHC to tell the device to begin logging event and variable data.
- 9.3.3.2 Initialize log is a service initiated by the MHC to tell the device to purge a certain range of records from the log file.
- 9.3.3.3 Report log status is a service initiated by the MHC to learn whether logging is taking place, and if so, the beginning and ending record sequence numbers.
- 9.3.3.4 Delete log is a service initiated by the MHC to tell the device to stop logging to the log file, and to remove it.
- 9.3.3.5 Upload log is a service initiated by the MHC to have the device transmit a range of records from the log file.
- 9.3.3.6 Write log is a service initiated by the MHC to have the device add a comment to the log file.

- 9.3.4 File management: For devices that have file storage capabilities, the file management services allow access to the file systems.
 - 9.3.4.1 Transfer request is a service initiated by the MHC to ask permission to send or receive a file. The response says whether this is possible, and if not, why not. If successful, the Transfer Block service begins immediately.
 - 9.3.4.2 Transfer block is a service sent after a successful transfer request to actually transfer a file. For uploading, the device initiates. For downloading, the MHC initiates.
 - 9.3.4.3 Terminate transfer is a service to interrupt the file transfer process.
 - 9.3.4.4 Read directory is a service initiated by the MHC to have the device list files present in a certain named directory, along with their attributes. Devices may have only a single directory (called "/") or may have a hierarchical tree structured file system. Files have the following attributes:
 - 9.3.4.4.1 Name
 - 9.3.4.4.2 Size in bytes
 - 9.3.4.4.3 Date and time of last modification
 - 9.3.4.5 Rename file is a service initiated by the MHC to change the name of a file stored on the device. This may include moving it to a different directory if the file system is hierarchical.
 - 9.3.4.6 Delete file is a service initiated by the MHC to remove a file from the file system, freeing its space.
- 9.3.5 Transaction is a set of messages used to do a specific function. The transactions defined here are for dealing with credit/debit card purchases.
 - 9.3.5.1 Authorization is a service initiated by the device, where a customer is attempting to purchase with a credit or debit card. The authorization response indicates approval or denial. A Capture or a Reversal should follow all approved authorizations.
 - 9.3.5.2 Capture is a service initiated by the device to say that a sale corresponding to an approved authorization was completed.
 - 9.3.5.3 Cancellation is a service initiated by the device to say that an authorization in progress should be halted. If funds were transferred, they should be returned. If a device receives authorization approval for a canceled request, it should send an immediate reversal.
 - 9.3.5.4 Reversal is a service initiated by the device to say that a sale corresponding to an approved authorization could not be completed. If funds were transferred, they

should be returned.

9.3.6 Program execution is an action or set of actions to be carried out by the device.

9.3.6.1 Spawn is a service initiated by the MHC having the device execute a program from the file system, or a named procedure. Parameters may be included. The ID of the program execution is returned in the response.

9.3.6.2 Terminate is a service initiated by the MHC having the device stop the execution of a program that is currently running.

9.3.6.3 Restart is a service initiated by the MHC having the device stop and start again a program that is currently running. Parameters may be included.

9.3.6.4 List program executions is a service initiated by the MHC to retrieve a list of programs that are currently executing, along with the ID of each.

9.3.6.5 Input is a service initiated by the MHC to send the included data to the input of an executing program. This can only be done for programs executed via the Spawn service.

9.3.6.6 Output is a service initiated by the device to send output from an executing program (started via the Spawn service) to the MHC.

9.4 Data formats

All data is organized into fields that are separated by the Field Separator (FS) character (Hex 1C). Fields are ASCII data of varying length.

See Chapter 16 for detailed descriptions of the data types.

9.5 Message header

Each message is introduced with a message header. It contains the following fields.

Field #	Field Name	Field Type
1	Message Identifier	ValMessageIdentifier
2	Message Type	ValMessageType
3	Message Sequence Number	ValMessageSequence
4	Dialog Sequence Number	ValMessageSequence
5	Message length	ValMessageLength
6	Sender's Device ID	ValDeviceID
7	Receiver's Device ID	ValDeviceID
8	Routing Information	Unspecified
9	Version	Decimal
10	Response Needed	Boolean

Note that Chapter 16 contains detailed data type definitions. The fields have the following descriptions.

Message Identifier	This field specifies uniquely the dialog class and the service.
Message Type	This field specifies the part of the dialog that this message constitutes. For example, request, response, or acknowledgment.
Message Seq Number	For the first request message of a dialog, its value is always one. For the first response to a request, its value is echoed back. For acknowledgments, its value is always echoed back. For subsequent requests or responses, its value increments on each message.
Dialog Seq Number	A number that the requester increments after each dialog (set of messages), but the responder and acknowledger only echo back.
Message length	Number of bytes in the message, including the header, records, and field separators.
Sender's Device ID	A unique number within a system that identifies the device that is sending the message. This may be used for return routing through a gateway.
Receiver's Device ID	A unique number within a system that identifies the device that is

the intended recipient of the message. This may be used for routing through a gateway. When messages are used in a broadcast (instead of a device-to-device connection), this field is empty.

Routing Information	The type of this field is unspecified. It shall be used to further direct intermediate sites how to deliver the message to its final destination. An example usage would be the channel number of a device in a multi-drop connection.
Version	The Vending Equipment Interface Specification Version. The value shall be "1.1".
Response Needed	True if the sender expects a response or acknowledgment. False if not. Always false for broadcasts.

9.6 Response Dialog Common Fields

All response dialogs consist of a dialog header followed by a field called "Request Accepted". This field can be True or False. If it is False, a field follows it called "Failure Reason," which indicates why the requested action cannot be done.

Sometimes, the request is for information that is too large to fit into a single response message. Here, the "Request Accepted" (True) field is followed by a field called "Dialog Final Block." This field can be True or False. When it is True, the next acknowledgment is the last message of the dialog, and all data requested has been sent. When it is False, another response block is expected after the acknowledgment is received. Data that is split in this way is considered a byte stream—that is, a list is not split into multiple lists, but instead is treated as a single list.

Chapter 10

Condition Dialogs

10.1 Definition

A condition is a particular circumstance or situation at a device. An intrusion alarm, a door opening, a component failure, and the sale of an article are all examples of condition status changes. The dialog that reports a condition includes the attributes that describe it. They might include for example the time of day at the device, the value of items sold, the denominations of coins and bills used to pay, or the serial number of a failing component.

10.2 Condition — Unsolicited Status

Purpose: To alert the network that a condition's status has changed.

Option: USS.

When used: At the time the condition's status changes.

Structure:

Request:

From: Device

To: Controller

Possible response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Condition Attributes	RecConditionAttributes

Response:

From: Device

To: Controller

Expected response: None

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader

10.3 Condition — Report Status

Purpose: To allow the MHC to ask whether a condition is currently idle, active, or disabled. It can ask about a single condition, a list of conditions, all conditions, or all conditions of a particular status.

Option: Event Control

When used: When the MHC needs information about a condition's status.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

Single or list of conditions:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Report Status Category	ValStatusCategory
3	List of Conditions	RecConditionList

All in a category:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Report Status Category	ValStatusCategory

Response:

From: Device

To: Controller

Possible response: Acknowledgment message

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True
3	Dialog Final Block	Boolean
4	Status Report List	RecStatusReportList

Request not accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Report Status Failure Reason	ValFailureReason

Acknowledgment:

From: Controller

To: Device

Expected response: If more blocks, response message, else none.

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader

10.4 Condition — Read Attributes

Purpose: To allow the MHC to determine the attributes of a particular condition.

Option: Event Control

When used: When the MHC needs more detailed information about a condition.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Condition ID	RecConditionID

Response:

From: Device

To: Controller

Possible response: Acknowledgment message

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True
3	Condition Attributes	RecConditionAttributes

Request not accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Read Attributes Failure Reason	ValFailureReason

Acknowledgment:

From: Controller

To: Device

Expected response: None

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader

10.5 Condition — Alter Attributes

Purpose: To allow the MHC the ability to modify the attributes of a particular condition.

Option: Event Control

When used: When conditions need to be disabled or enabled.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Attribute Modification List	RecAttributeModificationList

Response:

From: Device

To: Controller

Expected response: None

Request accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

Request not accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Alter Attributes Failure Reason	ValFailureReason

Chapter 11

Variable Access Dialogs

11.1 Definition

Variable Access dialogs are related to accessing, by name, data items that are resident on the device. A variable is a data item that can assume any one of a set of values. They may be simple data types, compound types, arrays, or combinations of the same.

Simple data types are named with an identifier. For example, "devnum" is a named variable that contains the identifying number of the device.

An array is an indexed collection of variables of the same type. An array entry is named with an identifier and an index contained in brackets. For example, "Message[2]" is the third entry in the named array variable "Message". Index values begin with zero.

A compound data type is a collection of variables of different types, the individual parts of which are given a name. A compound data member is named with the main variable separated from the member name by a period. For example, "OptionalMessage.Name" is the member called "Name" of the compound data variable called "OptionalMessage".

11.2 Variable Access — Read

Purpose: To allow the MHC the ability to retrieve the value of a named data item located at the device.

Option: Variable Control.

When used: When the MHC needs to know the value of a named data item located at the device.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Read Variable Mode	ValReadVariableMode
3	List of Variable Names	RecVariableNameList

Response:

From: Device

To: Controller

Possible response: Acknowledgment message

Request not accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Read Failure Reason	ValFailureReason

Request accepted, basic mode (from Read Variable Mode field):

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True
3	Dialog Final Block	Boolean
4	Variable Value List	RecVariableValueList

Request accepted, structure interspersed mode:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True
3	Dialog Final Block	Boolean
4	Variable Interspersed List	RecVariableInterspersedList

Acknowledgment:

From: Controller

To: Device

Expected response: If more blocks, response message, else none.

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader

Note: Record structures such as RecDateTime when sent in structure interspersed mode are named VariableName.FieldName. For example, "systemdatetime.Year", "systemdatetime.Month", etc.

11.3 Variable Access — Write

Purpose: To allow the MHC the ability to set the value of a named data item located at the device.

Option: Variable Control.

When used: When the MHC needs to change the value of a named data item located at the device.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	List of Variable Names	RecVariableNameList
3	Variable Value List	RecVariableValueList

Response:

From: Device

To: Controller

Expected response: None.

Request not accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Write Failure Reason	ValFailureReason

Request accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

11.4 Variable Access — List Structure

Purpose: To allow the MHC to access the low level structure of variables down to the simple data types. For example, a date consists of three non-negative integers, called year, month, and day.

Option: Variable Control.

When used: When the MHC needs to know the structure of a variable.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	List of Variable Names	RecVariableNameList

Response:

From: Device

To: Controller

Possible response: Acknowledgment message

Request not accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	List Structure Failure Reason	ValFailureReason

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True
3	Dialog Final Block	Boolean
4	Variable Structure List	RecVariableStructureList

Acknowledgment:

From: Controller

To: Device

Expected response: If more blocks, response message, else none.

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader

Chapter 12

Log File Dialogs

12.1 Definition

A log file is a historic record of events and other archived information that can be used to determine a sequence of events at a device. Three types of data are stored in a log file:

1. event data,
2. variable data, and
3. comment data.

Event data is a record containing the condition attributes associated with a change of condition status. It is all the information that would be found in a Condition — Read Attributes response, with the Log Record Header (see chapter 16) substituted for the Message Header. It is inserted at the time the condition status changes.

Variable data is a record containing a Log Record Header (see chapter 16) and a list of variable values at the time the record is added to the Log. It is inserted periodically (usually daily), and when the information is required (i.e., to allow balancing financial data with money). Generally, device information and system configuration data are written to the log file any time they change. Accounting data is usually written to the log file at midnight, at the close of a shift, when any money container is removed (i.e., when revenue servicing takes place), when the log file is uploaded, and on request.

Comment data is a record containing a Log Record Header (see chapter 16) and the non-header data from a Log File — Write Log request. It is inserted at the time the request is received and accepted.

12.2 Log File — Create Log.

Purpose: To tell the device to begin logging event, variable, and comment data.

Option: Logging.

When used: On first communication with a device, after Report Log status shows no logging.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader

Response:

From: Device

To: Controller

Expected response: None

Request not accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Create Log Failure Reason	ValFailureReason

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

12.3 Log File — Initialize Log.

Purpose: To purge specified entries from the Log File.

Option: Logging.

When used: After entries from the Log File have been successfully uploaded to the MHC and stored there.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	First Sequence Number to purge	ValRecordSequence
3	Last Sequence Number to purge	ValRecordSequence

Response:

From: Device

To: Controller

Expected response: None

Request not accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Initialize Log Failure Reason	ValFailureReason

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

12.4 Log File — Report Log Status.

Purpose: To determine whether logging is enabled, and if so, what are the beginning and ending record sequence numbers.

Option: Logging.

When used: At startup, to insure logging is enabled.
Before uploading Log File entries, to determine what is there.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader

Response:

From: Device

To: Controller

Possible response: Acknowledgment message

Request not accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Log Status Failure Reason	ValFailureReason

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True
3	Logging Enabled	Boolean
4	Oldest Record Sequence Number	ValRecordSequence
5	Newest Record Sequence Number	ValRecordSequence

Acknowledgment:

From: Controller

To: Device

Expected response: None

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader

12.5 Log File — Delete Log.

Purpose: To stop logging to the Log File and remove it.

Option: Logging.

When used: As needed.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader

Response:

From: Device

To: Controller

Expected response: None

Request not accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Delete Log Failure Reason	ValFailureReason

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

12.6 Log File — Upload Log.

Purpose: To retrieve a range of records from the log file.

Option: Logging.

When used: Periodically.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	First sequence number to send	ValRecordSequence
3	Last sequence number to send	ValRecordSequence

Response:

From: Device

To: Controller

Possible response: Acknowledgment message

Request not accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Upload Log Failure Reason	ValFailureReason

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True
3	Dialog Final Block	Boolean
4	Log file data	RecLogFileData

Acknowledgment:

From: Controller

To: Device

Expected response: If more blocks, response message, else none.

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader

12.7 Log File — Write Log.

Purpose: To add a comment to the Log File.

Option: Logging.

When used: When the MHC wants to insert some information into the log file.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Comment data	Unspecified

Response:

From: Device

To: Controller

Expected response: None

Request not accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Write Log Failure Reason	ValFailureReason

Request accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

Chapter 13 File Management Dialogs

13.1 Definition

File Management dialogs allow remote access to file systems.

13.2 File Management — Transfer Request

Purpose: Request permission to begin sending or receiving a file.

Option: File System Control.

When used: When a file needs to be transferred.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	File Transfer Direction	ValFileTransferDirection
3	File Name	ValFileName
4	File Attributes	RecAdditionalAttributeList

Response:

From: Device

To: Controller

Expected response: If uploading and successful, Transfer Block.
If downloading and successful, send Transfer Block.

Request not accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Transfer Request Failure Reason	ValFailureReason

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

13.3 File Management — Transfer Block

Purpose: To send a file to the receiver.
 Option: File System Control.
 When used: After a successful Transfer Request dialog.
 Structure:

Request:
 From: Sender
 To: Receiver
 Possible response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Dialog Final Block	Boolean
3	Data from file	Unspecified

Response:
 From: Receiver
 To: Sender
 Expected response: If more data, next block, else none.

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader

13.4 File Management — Terminate Transfer

Purpose: If the receiver, to stop a file transfer that is in progress.
 If the sender, to tell the other side that a file transfer is stopping prematurely.

Option: File System Control.

When used: Generally, an error situation.

Structure:

Request:

From: Either side.
 To: Other side.
 Possible response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Termination Reason	ValTerminationReason

Response:

From: Other side.
 To: Requesting side.
 Expected response: None

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader

13.5 File Management — Read Directory

Purpose: To allow the MHC to know what files are present in a directory.

Option: File System Control.

When used: When the directory information is needed.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Directory Name or File Name	ValFileName

Response:

From: Device

To: Controller

Possible response: Acknowledgment message

Request not accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Read Directory Failure Reason	ValFailureReason

Request accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True
3	Dialog Final Block	Boolean
4	File Attribute List	RecFileAttributeList

Acknowledgment:

From: Controller

To: Device

Expected response: If more blocks, response message, else none

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader

13.6 File Management — Rename File

Purpose: To give a file a new name.
 Option: File System Control.
 When used: When a file needs to be called by a different name.
 Structure:
 Request:
 From: Controller
 To: Device
 Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Current File Name	ValFileName
3	New File Name	ValFileName

Response:
 From: Device
 To: Controller
 Expected response: None

Request not accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Rename File Failure Reason	ValFailureReason

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

13.7 File Management — Delete File

Purpose: To allow the MHC to remove a file from the device's file system.

Option: File System Control.

When used: When a file needs to be deleted remotely.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	File name to delete	ValFileName

Response:

From: Device

To: Controller

Expected response: None

Request not accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Delete File Failure Reason	ValFailureReason

Request accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

Chapter 14 Transaction Dialogs

14.1 Definition

Financial Transaction dialogs are used for supporting the use of credit/debit cards as a payment medium at the Vending Equipment.

14.2 Transaction — Authorization

Purpose: To request permission from a financial authority to make a sale, transferring money from a specified (credit/debit card) account.

Option: Funds Transfer.

When used: When a customer uses a credit/debit card to make a purchase, before approving the sale.

Structure:

Request:

From: Device

To: Controller

Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Date and Time of Transaction	RecDateTime
3	Device's Transaction ID	ValTransactionID
4	Card Track List	RecCardTrackList
5	Encrypted PIN	ValEncryptedPIN
6	Pinpad Serial number	Alphanumeric[16]
7	Key Serial Number	ValKeySerialNumber
8	Amount of transaction	ValMoney
9	List of sales items	RecSalesItemList
10	Payment Type	ValPaymentType
11	Additional Attributes	RecAdditionalAttributeList

Response:

From: Controller

To: Device

Possible response: Acknowledgment message

Request not approved:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Approved	Boolean: False
3	Authorization Denial Reason	ValFailureReason
4	Additional Attributes	RecAdditionalAttributeList

Request approved:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Approved	Boolean: True
3	Date and Time of transaction	RecDateTime
4	Device's Transaction ID	ValTransactionID
5	Card Account Number	ValCardAccountNumber
6	Card Expiration Date	RecDate
7	Customer Name	ValCustomerName
8	Controller Authorization Number	ValTransactionID
9	Available Credit Limit	ValMoney
10	Card Type	ValCardType
11	Bank Authorization Reference Number	ValTransactionID
12	Bank Authorization Number	ValTransactionID
13	Additional Attributes	RecAdditionalAttributeList

Acknowledgment:

From: Device

To: Controller

Expected response: None

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader

14.3 Transaction — Capture

Purpose: To indicate that an approved authorization resulted in a successful sale of goods.

Option: Funds Transfer.

When used: After an approved authorization and successful delivery of vended items.

Structure:

Request:

From: Device

To: Controller

Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Controller Authorization Number	ValTransactionID

Response:

From: Controller

To: Device

Expected response: None

Request not accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Capture Failure Reason	ValFailureReason

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

14.4 Transaction — Cancellation

Purpose: To interrupt a pending authorization request.

Option: Funds Transfer.

When used: When an authorization request has been sent, no response is yet received, and the transaction cannot be completed.

Structure:

Request:

From: Device

To: Controller

Expected response: Response message

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Device's Transaction ID	ValTransactionID

Response:

From: Controller

To: Device

Expected response: None

Request not accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Cancellation Failure Reason	ValFailureReason

Request accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

14.5 Transaction — Reversal

Purpose: To indicate that an approved authorization did not result in a completely successful delivery of goods to the customer. If money was transferred, it must be returned.

Option: Funds Transfer.

When used: When an authorization approval arrives for which no pending request remains. When after authorization approval, vended items cannot be delivered.

Structure:

Request:

From: Device

To: Controller

Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Controller Authorization Number	ValTransactionID
3	Amount to reverse	ValMoney
4	Reversal Reason	ValReversalReason

Response:

From: Controller

To: Device

Expected response: None

Request not accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Reversal Failure Reason	ValFailureReason

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

Chapter 15

Program Execution Dialogs

15.1 Definition

A Program Execution is an action or set of actions to be carried out by the device.

15.2 Program Execution — Spawn

Purpose: To have the device execute a named function or program.

Option: Multi-processing.

When used: When a function or program should be executed.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Program needing input	Boolean
3	Program producing output	Boolean
4	Program name	ValFileName
5	Parameter list	RecParameterList

Response:

From: Device

To: Controller

Possible response: Acknowledgment message

Request not accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Spawn Failure Reason	ValFailureReason

Request accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True
3	Process ID	ValProcessID

Acknowledgment:

From: Controller

To: Device

Expected response: None

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader

15.3 Program Execution — Terminate

Purpose: To stop a particular instance of a program (process) from running.

Option: Multi-processing.

When used: When a process should not continue running.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Process ID	ValProcessID

Response:

From: Device

To: Controller

Expected response: None

Request not accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Terminate Failure Reason	ValFailureReason

Request accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

15.4 Program Execution — Restart

Purpose: To cause a process on a device to terminate and spawn again.

Option: Multi-processing.

When used: When a process should be started over.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Process ID	ValProcessID

Response:

From: Device

To: Controller

Possible response: Acknowledgment message

Request not accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Restart Failure Reason	ValFailureReason

Request accepted:

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True
3	New Process ID	ValProcessID

Acknowledgment:

From: Controller

To: Device

Expected response: None

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader

15.5 Program Execution — List Program Executions

Purpose: To provide to the MHC a list of program executions (or processes) that are currently running at a device.

Option: Multi-processing.

When used: When the MHC needs to see what processes are running at a device.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader

Response:

From: Device

To: Controller

Possible response: Acknowledgment message

Request not accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	List Processes Failure Reason	ValFailureReason

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True
3	Dialog Final Block	Boolean
4	Process Status List	RecProcessStatusList

Acknowledgment:

From: Controller

To: Device

Expected response: If more blocks, response message, else none.

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Message Header	RecMessageHeader

15.6 Program Execution — Input

Purpose: To allow the MHC to send input to a specific process.

Option: Multi-processing with Terminal I/O.

When used: When a process Spawned by the MHC uses input.

Structure:

Request:

From: Controller

To: Device

Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Process ID	ValProcessID
3	Input Data	Unspecified

Response:

From: Device

To: Controller

Expected response: None

Request not accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Input Failure Reason	ValFailureReason

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

15.7 Program Execution — Output

Purpose: To allow a process to send information to the MHC.
 Option: Multi-processing with Terminal I/O.
 When used: When a process Spawned by the MHC produces output.
 Structure:

Request:

From: Device
 To: Controller
 Expected response: Response message

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Process ID	ValProcessID
3	Output data	Unspecified

Response:

From: Controller
 To: Device
 Expected response: None

Request not accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: False
3	Output Failure Reason	ValFailureReason

Request accepted:

Field #	Field Name	Field Type
1	Message Header	RecMessageHeader
2	Request Accepted	Boolean: True

Chapter 16 Data Type Definitions

16.1 Data formats

All data is organized into fields that are separated by the Field Separator (FS) character (Hex 1C). Fields are ASCII data of varying length. Following are the basic data types.

<u>Type</u>	<u>Valid Characters</u>	<u>Default Range</u>
Integer	— 0-9	- 2147483648 to 2147483647
Hexadecimal	0-9 A-F	0 to FFFFFFFF
Decimal	— 0-9 .	1 to 16 digits
Alphanumeric	— 0-9 A-Z a-z _	1 to 80 characters
ASCII Text	Hex 20 - Hex 7E inclusive.	1 to 80 characters
Unspecified	Any binary values.	Unspecified

The following qualifiers can be used for Integer and Decimal:

Positive:	This means the negative sign and a zero value are not allowed.
Non-negative	This means the negative sign is not allowed.

Some types will be defined using the formalism known as the Backus-Naur Form. The following symbols are meta-symbols, not belonging to the type being defined:

::=	Means "is defined as".
	Means "or".
{ }	Encloses items which may be repeated zero or more times.

All other symbols are part of the definition. Syntactic constructs are printed in italics

Simple types are named beginning with the characters "Val". Compound types are named beginning with the characters "Rec".

Note that the remaining sections of this chapter are arranged alphabetically (omitting the first three characters Val and Rec).

16.2 ValAdditionalAttribute Names defined in Section 17.1

16.3 RecAdditionalAttributeList

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Count of items in list (N)	Non-Negative Integer
2 to N+1	Additional Attribute	RecAdditionalAttribute

16.4 RecAdditionalAttribute

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Attribute Name	ValAdditionalAttribute
2	Assigned Value	Dependent on Attribute

16.5 RecAttributeModificationList

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Count of items in list (N)	Non-Negative Integer
2 to N+1	Attribute Modification	RecAttributeModification

16.6 RecAttributeModification

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Attribute ID	ValAttributeID
2	Assigned Value	Dependent on Attribute

16.7 ValAttributeID

<u>Value</u>	<u>Attribute Name</u>	<u>Attribute's Data Type</u>
1	State	ValConditionStatus
2	Priority	ValConditionPriority
3	Severity	ValConditionSeverity
4	Enabled	Boolean
5	Incident	Boolean
6	Date/Time condition last became active	RecDateTime
7	Date/Time condition last became idle	RecDateTime
8	Contributes to overall device status	Boolean
9	File name	ValFileName
10	Size in bytes	Non-negative Integer
11	Date and time of last modification	RecDateTime
12	Check value	Hexadecimal

Additional or vendor specific attributes are given a unique alphanumeric name by the rules given in chapter 18. This name is used as the Attribute ID.

16.8 Boolean

Alphanumeric:

F	False
T	True

16.9 ValCardAccountNumber
Integer: 12 to 21 digits.

16.10 ValCardType
Alphanumeric Text: (1 to 20 characters)
American Express
Debit card
MasterCard
Discover
Visa
Carte Blanche
Diners Club
Unknown card
Test

16.11 RecCardTrackList

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Count of items in list (N)	Non-Negative Integer
2 to N+1	Card Track	RecCardTrack

16.12 RecCardTrack

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Track ID	ValTrackID
2	Track Data	ValTrackData

16.13 RecConditionAttributes

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Date and Time	RecDateTime
2	Condition ID	RecConditionID
3	Condition Status	ValConditionStatus
4	Condition Priority	ValConditionPriority
5	Condition Severity	ValConditionSeverity
6	Condition Enabled	Boolean
7	Condition Incident	Boolean
8	Date and Time last became active	RecDateTime
9	Date and Time last became idle	RecDateTime
10	Overall device status is dependent on this Condition's status	Boolean
11	Overall device status	ValDeviceStatus
12	Additional Attributes List	RecAdditionalAttributeList

16.14 RecConditionID

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Manufacturer ID	ValManufacturerID
2	Model ID	ASCII Text
3	Subsystem ID	Integer
4	Condition Code	Integer

16.15 RecConditionList

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Count of items in list (N)	Non-Negative Integer
2 to N+1	Condition ID	RecConditionID

16.16 ValConditionPriority
Integer: 0 (lowest) to 99 (highest)

16.17 ValConditionSeverity
Integer: 0 (best) to 99 (worst)

16.18 ValConditionStatus
Alphanumeric:
A Active
D Disabled
I Idle

16.19 ValCounter
Positive Integer: 1 to 99,999,999

16.20 ValCustomerName
ASCII Text. From card. (1 to 30 characters)

16.21 RecDateTime

Field #	Field Name	Field Type
1	Year	Integer: -4713 to 9999
2	Month	Integer: 1 to 12
3	Day of month	Integer: 1 to 31
4	Hour of day	Integer: 0 to 23
5	Minute	Integer: 0 to 59
6	Second	Decimal: 0.000 to 59.999
7	Seconds behind GMT	Integer: -43200 to 43200

An unknown date and time has empty fields.

Not all devices have a clock available to post date and time. Such devices report unknown date and time.

16.22 RecDate

Field #	Field Name	Field Type
1	Year	Integer: -4713 to 9999
2	Month	Integer: 1 to 12
3	Day of month	Integer: 1 to 31

- 16.23 ValDeviceID
Integer: 0 to 4,294,967,296
- 16.24 ValDeviceStatus
Alphanumeric:
F Fully Operable
I Inoperable
P Partially Operable
- 16.25 ValDirection
Integer:
1 Entry
2 Exit
3 Entry and Exit
- 16.26 ValEncryptedPIN
Hexadecimal: 16 digits.
- 16.27 ValFailureReason
Integer:
1 Service unavailable
2 Condition ID unknown
3 Invalid report status category
4 Invalid attribute
5 Unchangeable attribute
6 Invalid attribute value
7 Unknown Variable
8 Internal Error
9 Variable is unmodifiable
10 Invalid value for variable
11 Log File already exists
12 Sequence number not in Log File
13 Bad first sequence number
14 Bad last sequence number
15 Logging is (already) disabled
16 Permission denied
17 File does not exist
18 Invalid file name
19 Invalid transfer direction
20 No space available
21 File already exists
22 Invalid controller authorization number
23 No such pending authorization
24 Invalid amount to reverse
25 Invalid reversal reason
26 File is not executable
27 Invalid arguments
28 No such process
29 Invalid process ID

1000	Authorization limits reached
1001	Bad Debt
1002	Bad Name on card
1003	Bad PIN
1004	Bad basis
1005	Bad expiration date
1006	Bad track 1 cannot sell without name
1007	Bank not supported
1008	Cannot Read Card
1009	Card in negative file
1010	Card is expired
1011	Card number error (bad check digit)
1012	Card number is not all digits
1013	Card number is too long
1014	Card type not taken
1015	Credit Transaction attempted on Debit Card
1016	Debit Transaction attempted on Credit Card
1017	Decline
1018	Excessive PIN Tries
1019	Excessive activity for POS
1020	General Denial
1021	Institution Not Found on File
1022	Insufficient funds
1023	Invalid Account or Driver's license number
1024	Invalid Card Format
1025	Lost/Stolen Card
1026	No Account on File
1027	No expiration date on card
1028	No name on card
1029	No network response
1030	Not initialized for encryption
1031	Off line referral
1032	Permanent Restraint
1033	Request Would Exceed Daily Limit Try Lesser Amount
1034	Unspecified Error

16.28 ValFieldType

Integer:

1	Integer
2	Hexadecimal
3	Decimal
4	Alphanumeric
5	ASCII Text
6	Boolean
7	ValAdditionalAttribute
8	ValAttributeID
9	ValCardAccountNumber
10	ValCardType
11	ValConditionPriority

12	ValConditionSeverity
13	ValConditionStatus
14	ValCounter
15	ValCustomerName
16	ValDeviceID
17	ValDeviceStatus
18	ValDirection
19	ValEncryptedPIN
20	ValFailureReason
21	ValFieldType
22	ValFileName
23	ValFileTransferDirection
24	ValFreewheel
25	ValKeySerialNumber
26	ValLogRecordType
27	ValManufacturerID
28	ValMessageIdentifier
29	ValMessageLength
30	ValMessageSequence
31	ValMessageType
32	ValMoney
33	ValPaymentType
34	ValProcessID
35	ValReadVariableMode
36	ValRecordSequence
37	ValSecurityMode
38	ValStatusCategory
39	ValReversalReason
40	ValTerminationReason
41	ValTrackData
42	ValTrackID
43	ValTransactionID
44	ValVariableName

16.29 RecFileAttributeList

Field #	Field Name	Field Type
1	Count of items in list (N)	Non-Negative Integer
2 to N+1	File Attributes	RecFileAttribute

16.30 RecFileAttribute

Field #	Field Name	Field Type
1	File Name	ValFileName
2	Size in bytes	Non-Negative Integer
3	Date and Time of last Modification	RecDateTime

- 16.31 ValFileName
ASCII Text as described below (up to 80 characters).

```

FileName ::= / { DirectoryName } UnqualifiedFileName
DirectoryName ::= UnqualifiedFileName /
UnqualifiedFileName ::= LetterOrDigit { PeriodOrLetterOrDigit }
LetterOrDigit ::= Letter | Digit
Letter ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | S | T | U | V | W | X | Y | Z |
          a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | s | t | u | v | w | x | y | z | _
Digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
PeriodOrLetterOrDigit ::= Letter | Digit | .

```

- 16.32 ValFileTransferDirection
Alphanumeric:

D Download from MHC to Device.
U Upload from Device to MHC

- 16.33 ValFreewheel
Integer:

0 Normal operation (No freewheeling)
1 Freewheel exit
2 Freewheel entry
3 Freewheel entry and exit

- 16.34 ValKeySerialNumber
Hexadecimal: 20 digits

- 16.35 RecLogFileData
A Log File is a collection of Log File Records.

16.36 RecLogFileRecord

Field #	Field Name	Field Type
1	Log Record Type	ValLogRecordType
2	Record Sequence Number	ValRecordSequence
3	Number of fields in record	Non-Negative Integer
4	Free Text	Event: RecConditionAttributes Variable: RecVariableLogFile Comment: Unspecified

16.37 ValLogRecordType

Alphanumeric:

E	Event data
V	Variable data
C	Comment data

16.38 ValManufacturerID

A short (usually 3-character) alphanumeric abbreviation of the name of the manufacturer.

16.39 RecMessageHeader

Field #	Field Name	Field Type
1	Message Identifier	ValMessageIdentifier
2	Message Type	ValMessageType
3	Message Sequence Number	ValMessageSequence
4	Dialog Sequence Number	ValMessageSequence
5	Message length	ValMessageLength
6	Sender's Device ID	ValDeviceID
7	Receiver's Device ID	ValDeviceID
8	Routing Information	Unspecified
9	Version	Decimal
10	Response Needed	Boolean

16.40 ValMessageIdentifier

Alphanumeric: 4 characters

EVNT Condition — Unsolicited status
STAT Condition — Report status
RATR Condition — Read attributes
AATR Condition — Alter attributes
READ Variable Access — Read
WRIT Variable Access — Write
STRU Variable Access — List structure
CLOG Log File — Create log
ILOG Log File — Initialize log
RLOG Log File — Report log status
DLOG Log File — Delete log
ULOG Log File — Upload log
WLOG Log File — Write log
TREQ File Management — Transfer request
TRAN File Management — Transfer block
STOP File Management — Terminate transfer
RDIR File Management — Read directory
RENF File Management — Rename file
DELF File Management — Delete file
AUTH Transaction — Authorization
CAPT Transaction — Capture
CANC Transaction — Cancellation
REVE Transaction — Reversal
SPAW Program Execution — Spawn
TERM Program Execution — Terminate
REST Program Execution — Restart
INPU Program Execution — Input
OUTP Program Execution — Output
LSTP Program Execution — List program executions

16.41 ValMessageLength

Integer: 1 to 999,999

16.42 ValMessageSequence

Integer: 1 to 999,999

16.43 ValMessageType

Alphanumeric:

Q Request
R Response
A Acknowledgment

16.44 ValMoney

Decimal with two (2) digits past decimal point.

16.45 RecParameterList

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Count of items in list (N)	Non-Negative Integer
2 to N+1	Parameter	Unspecified ASCII Text

16.46 ValPaymentType
Alphanumeric:

A	Coins
B	Bills
AB	Cash
C	Credit Card
D	Debit Card
I	ID Card
L	Billable Local Account

16.47 ValPermission

Integer of binary flags:

0	No permission
1	Execute (or search in directory) permission
2	Write permission
4	Read permission
8	Hold executable in memory (Unix sticky bit)
16	Set group ID on execution
32	Set user ID on execution

16.48 ValProcessID

Non-Negative Integer

16.49 RecProcessStatusList

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Count of items in list (N)	Non-Negative Integer
2 to N+1	Process Status	RecProcessStatus

16.50 RecProcessStatus

Field #	Field Name	Field Type
1	Process ID	ValProcessID
2	Program Name	ValFileName
3	Parameter List	RecParameterList
4	Date and Time when process started	RecDateTime
5	Seconds of CPU Time used.	Non-negative Integer

16.51 ValReadVariableMode

B Basic — values only
S Structure interspersed alternately

16.52 ValRecordSequence

Integer: 1 to 99,999,999

16.53 ValReversalReason

Integer:
1 Cannot deliver product
2 No authorization pending
3 Communications failure

16.54 RecSalesItemList

Field #	Field Name	Field Type
1	Count of items in list (N)	Non-Negative Integer
2 to N+1	Sales Item	Unspecified ASCII Text

16.55 ValSecurityMode

Integer:
1 Normal mode
2 Security mode (no entry allowed without alarm)
3 Alarms disabled

16.56 ValStatusCategory

Alphanumeric:
L List of conditions
1 All active conditions
2 All idle conditions
3 All idle and active conditions
4 All disabled conditions
5 All disabled and active conditions

- 6 All disabled and idle conditions
- 7 All conditions

16.57 RecStatusReportList

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Count of items in list (N)	Non-Negative Integer
2 to N+1	Status Report	RecStatusReport

16.58 RecStatusReport

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Condition ID	RecConditionID
2	Condition Status	ValConditionStatus
3	Condition Severity	ValConditionSeverity

16.59 RecSubsystemID

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Manufacturer ID	ValManufacturerID
2	Model ID	ASCII Text
3	Subsystem ID	Integer

16.60 ValTerminationReason

Integer:

- 1 Out of space on file system
- 2 Other size-based constraint has been reached.
- 3 Terminating communication.
- 4 Unspecified.

16.61 ValTrackData

TrackData ::= *StartSentinal DataFields EndSentinal*

StartSentinal ::= ;

DataFields ::= *Text { Separator Text }*

EndSentinal ::= ?

Separator ::= =

Text ::= *TextChar { TextChar }*

TextChar ::= *Letter | Digit | OtherText*

Letter ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | S | T | U | V | W | X | Y | Z |

a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | s | t | u | v | w | x | y | z | _

Digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

OtherText ::= , | . | - | + | \ | | / | ! | @ | # | \$ | % | ^ | & | * | : | ' | ` | ~ | < | >

16.62 ValTrackID

Integer:

- | | |
|---|---------|
| 1 | Track 1 |
| 2 | Track 2 |
| 3 | Track 3 |

16.63 ValTransactionID

Integer: 0 to 999,999,999,999

16.64 RecVariableInterspersedList

Field #	Field Name	Field Type
1	Count of items in list (N)	Non-Negative Integer
2 to N+1	Variable Interspersed	RecVariableInterspersed

16.65 RecVariableInterspersed

Field #	Field Name	Field Type
1	Field Name	ValVariableName
2	Field Type	ValFieldType
3	Field Value	Field Type (See 2 above).

16.66 RecVariableLogFile

Field #	Field Name	Field Type
1	Variable Name List	RecVariableNameList
2	Variable Value List	RecVariableValueList

16.67 RecVariableNameList

Field #	Field Name	Field Type
1	Count of items in list (N)	Non-Negative Integer
2 to N+1	Variable Name	ValVariableName

16.68 ValVariableName

ASCII Text as described below (up to 80 characters).

VariableName ::= CompleteVariable | ComponentVariable
CompleteVariable ::= Identifier
Identifier ::= Letter { LetterOrDigit }
Letter ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | S | T | U | V | W | X | Y | Z |
a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | s | t | u | v | w | x | y | z | _
LetterOrDigit ::= Letter | Digit
Digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
ComponentVariable ::= IndexedVariable | MemberDesignator
IndexedVariable ::= ArrayVariable [Expression] { [Expression] }
ArrayVariable ::= VariableName
Expression ::= FactorItem { , FactorItem }
FactorItem ::= Factor | FactorRange
FactorRange ::= Factor - Factor
Factor ::= Digit { Digit }
MemberDesignator ::= StructureVariable . MemberIdentifier
StructureVariable ::= VariableName
MemberIdentifier ::= Identifier

16.69 RecVariableStructureList

Field #	Field Name	Field Type
1	Count of items in list (N)	Non-Negative Integer
2 to N+1	Variable Structure	RecVariableStructure

16.70 RecVariableStructure

Field #	Field Name	Field Type
1	Field Name	ValVariableName
2	Field Type	ValFieldType

16.71 RecVariableValueList

<u>Field #</u>	<u>Field Name</u>	<u>Field Type</u>
1	Count of items in list (N)	Non-Negative Integer
2 to N+1	Variable Value	Varies based on what was specified.

Chapter 17 Additional Data Items

17.1 Additional Attributes

The names in the following table appear in fields of type ValAdditionalAttribute.

The data types are described in Chapter 16.

The “Req” field specifies that for applicable additional attributes, it is

R Required, or
O Optional.

<u>Name</u>	<u>Description</u>	<u>Data Type</u>	<u>Req</u>
_CAA001	Record sequence number from Log File	ValRecordSequence	R
_CAA002	Personal ID Number (PIN) of service person logged in	Positive Integer	R
_CAA003	Subsystem ID	RecSubsystemID	R
_CAA004	Sub-component serial number	ASCII Text— Unique	R
_CAA005	Cycle counter for sub-component	ValCounter	O
_CAA006	Total value of all items	ValMoney	R
_CAA007	Total count of all items	Non-Negative Integer	R
_CAA008	Number of item types	Non-Negative Integer	O
_CAA009	Item type	ASCII Text— defined	O
_CAA010	Item value transacted (added / subtracted)	ValMoney	O
_CAB010	Item final value after transaction	ValMoney	O
_CAA011	Item count	Non-Negative Integer	O
_CAA012	First serial number of continuous stock	ASCII Text (Usually numeric)	R
_CAA013	Last serial number of continuous stock	ASCII Text (Usually numeric)	O
_CAA014	Payment type	ValPaymentType	R
_CAA015	Credit/Debit Authorization Reference #	ValTransactionID	R
_CAA016	Credit/Debit Account Number	ValCardAccountNumber	R

<u>Name</u>	<u>Description</u>	<u>Data Type</u>	<u>Req</u>
_CAA017	Number of denominations used to pay	Non-Negative Integer	O
_CAA018	Denomination value	ValMoney	O
_CAA019	Number accepted	Non-Negative Integer	O
_CAA020	Number given as change	Non-Negative Integer	O
_CAA021	Number rejected	Non-Negative Integer	O
_CAA022	Price list (or table) ID	ASCII Text (Usually numeric)	R
_CAA023	Discount given off full price	ValMoney	R
_CAA024	Serial number of part that was replaced	ASCII Text— Unique	R
_CAA025	Original issuing machine number	ValDeviceID	O
_CAA026	Location of origin (entry / boarding)	ASCII Text— defined	R
_CAB026	Name of origin location	ASCII Text	O
_CAC026	Origin zone	Non-Negative Integer	O
_CAA027	Location of destination	ASCII Text— defined	R
_CAB027	Name of destination location	ASCII Text	O
_CAC027	Destination Zone	Non-Negative Integer	O
_CAA028	Location of transfer	ASCII Text— defined	R
_CAA029	Start date and time of validity	RecDateTime	R
_CAA030	End date and time of validity	RecDateTime	R
_CAA031	Number of items sold	Non-Negative Integer	R
_CAA032	Item type	ASCII Text— defined	R
_CAA033	Serial number of item	ASCII Text (Usually numeric)-- Unique	R
_CAA034	Number of adults on group item	Non-Negative Integer	O
_CAA035	Number of children on group item	Non-Negative Integer	O
_CAA036	Ticket check value (for detecting fraud)	Hexadecimal	O
_CAA037	Job ID number from MHC	Positive Integer	R
_CAA038	Account number to void and credit	ValCardAccountNumber	R

<u>Name</u>	<u>Description</u>	<u>Data Type</u>	<u>Req</u>
_CAA039	(Cumulative) number of times event occurred	ValCounter	O
_CAA040	ID of device that uploaded a file or a log	ValDeviceID	O
_CAA041	Agency ID	Positive Integer — defined	O
_CAA042	Agency Name	ASCII Text	O
_CAA043	Issuing agency ID	Positive Integer	O
_CAA044	Owner agency ID	Positive Integer	O
_CAA045	Rider category ID	Positive Integer — defined	O
_CAA046	Rider category name	ASCII Text	O
_CAA047	Denial reason	ValFailureReason	O
_CAA048	Usage record	ASCII Text— unspecified	O
_CAA049	Number of operations on this item	ValCounter	O
_CAA050	Print position on this item	Non-Negative Integer	O
_CAA051	Trip number	Positive Integer	O
_CAA052	Ride number	Positive Integer	O
_CAA053	Origin date and time	RecDateTime	O
_CAA054	Exit for alighting date and time	RecDateTime	O
_CAA055	Route ID	Positive Integer — defined	O
_CAA056	Route name	ASCII Text	O
_CAA057	Route destination	ASCII Text	O
_CAA058	Route direction	ASCII Text	O
_CAA059	Run number	Positive Integer	O
_CAA060	Transfer type used on ride	Positive Integer — defined	O
_CAA061	Transfer type name used on ride	ASCII Text	O
_CAA062	Vehicle number	Positive Integer	O

<u>Name</u>	<u>Description</u>	<u>Data Type</u>	<u>Req</u>
_CAA063	Transaction ID	ValTransactionID	R
_CAA064	Owner name	ASCII Text	O
_CAA065	Owner permission	ValPermission	O
_CAA066	Group name	ASCII Text	O
_CAA067	Group permission	ValPermission	O
_CAA068	User name	ASCII Text	O
_CAA069	User permission	ValPermission	O
_CAA070	File size	Non-negative Integer	O

17.2 Variables

The names in the following table appear in fields of type ValVariableName.

The data types are described in Chapter 16.

The “Req” field specifies that where a variable applies, it is

R	Required, or
O	Optional, and
M	Modifiable, or
U	Unmodifiable.

17.2.1 Variables dealing with the device that change infrequently.

<u>Name</u>	<u>Description</u>	<u>Data Type</u>	<u>Req</u>
devinfo	Device information structure	Record	RM
devinfo.agencyID	Numeric ID of agency that owns device	Integer — defined	OU
devinfo.agencyname	Name of agency that owns device	ASCII Text	OU
devinfo.manufact	Device manufacturer	ValManufacturerID	RU
devinfo.model	Device model ID	ASCII Text	RU
devinfo.devnum	Device ID number	Integer	RM
devinfo.locnum	Device location number	Integer	OM
devinfo.vehiclenum	Vehicle number	Integer	OM
devinfo.locname	Device location name	ASCII Text	OM
devinfo.zonenum	Zone number where device is located	Integer	OM
devinfo.zonename	Zone name where device is located	ASCII Text	OM
devinfo.itime	Date and time device was initialized	RecDateTime	OU
devinfo.devaddr	Network address of device	ASCII Text	RM
devinfo.mhcaddr	Network address of MHC	ASCII Text	RM

17.2.2 Variables dealing with the configuration of the device.

sysconf	System configuration structure— information concerning the configuration of the device.	Record	RM
sysconf.sanum	Number of sub-assemblies within device	Non-Negative Integer	RU
sysconf.sa[i]	Sub-assemblies structure	Record	OU
sysconf.sa[i].subID	Identification number of the sub-assembly	Integer — defined	OU
sysconf.sa[i].desc	Description of the i-th sub-assembly	ASCII Text	OU
sysconf.sa[i].manufact	Manufacturer of i-th sub-assembly	ValManufacturerID	OU
sysconf.sa[i].model	Model number of i-th sub-assembly	ASCII Text	OU
sysconf.sa[i].serial	Serial number of i-th sub-assembly	ASCII Text	OU
sysconf.sa[i].elecID	Electronic ID number of i-th sub-assembly	ASCII Text (Usually numeric or hexadecimal)	OU
sysconf.sa[i].sversion	Software version of i-th sub-assembly	ASCII Text	OU
sysconf.sa[i].contents	Contents of i-th sub-assembly	Integer— defined	OU
sysconf.sa[i].cdesc	I-th sub-assembly contents description	ASCII Text	OU
sysconf.sa[i].required	True if i-th sub-assembly is required for device to stay in service	Boolean	OM
sysconf.sa[i].maxcap	Maximum capacity of i-th sub-assembly	Non-Negative Integer	OM
sysconf.sa[i].warncap	Warning capacity of i-th sub-assembly	Non-Negative Integer	OM
sysconf.prnum	Number of product locations in device	Non-Negative Integer	RU
sysconf.pr[i]	Product location structure	Record	OU
sysconf.pr[i].location	Product location (i.e., slot #)	ASCII Text	OU
sysconf.pr[i].product	Product type	ASCII Text	OU
sysconf.pr[i].capacity	Maximum number of items that can be stored in a location within the device	Non-Negative Integer	OU
sysconf.pr[i].full	Normal filling level	Non-Negative Integer	OU
sysconf.pr[i].unit	Standard dispensed quantity	Non-Negative Integer	OU
sysconf.pr[i].begin	Beginning serial number	Non-Negative Integer	OU
sysconf.pr[i].end	Ending serial number	Non-Negative Integer	OU

sysconf.pr[i].price	Price each (-1 implies variable price)	ValMoney	OU
sysconf.numsd	Number of software and operating data modules	Non-Negative Integer	RU
sysconf.sd[i]	Software and Operating data structure	Record	OU
sysconf.sd[i].module	Name of module	ASCII Text	OU
sysconf.sd[i].version	Version number	Non-Negative Integer	OU
sysconf.sd[i].timeload	Date and Time loaded	RecDateTime	OU
sysconf.sd[i].userID	ID of who loaded the data	Non-Negative Integer	OU
sysconf.intl	International structure	Record	RU
sysconf.intl.decpos	Decimal point position for money type	Integer (usually 2)	RU
sysconf.intl.telcode	Telephone country code	Integer	RM
sysconf.intl.currency	Name of currency	ASCII Text	RU
sysconf.numbl	Number of items in the blacklist	Non-negative Integer	RU
sysconf.blcrc	CRC of blacklist	Hexadecimal	OU
sysconf.bl[i]	Blacklist structure	Record	OU
sysconf.bl[i].start	Start of range to blacklist	Integer	OU
sysconf.bl[i].end	End of range to blacklist	Integer	OU
sysconf.bns	Bank Note System structure	Record	OM
sysconf.bns.numtype	Number of bank note types	Non-negative integer	OU
sysconf.bns.bt[i]	Bank Note Type structure	Record	OU
sysconf.bns.bt[i].type	Bank Note Type ID (Denomination)	Integer— defined	OU
sysconf.bns.bt[i].desc	Description of i-th bank note type	ASCII Text	OU
sysconf.bns.bt[i].accept	Should this type be accepted	Boolean	OM
sysconf.gt	Gate parameters	Record	OM
sysconf.gt.direction	Direction of passing	ValDirection	RM
sysconf.gt.freewheel	Freewheel mode	ValFreewheel	RM
sysconf.gt.entryexit	Override entry/exit check	Boolean	OM
sysconf.gt.datetime	Override date/time check	Boolean	OM

sysconf.gt.sensitive	Sensitive mode	Boolean	OM
sysconf.gt.daycode	Day code	Integer	OM
sysconf.gt.yearcode	Year code	Integer	OM
sysconf.gt.agcode	Agency code	Integer	OM
sysconf.gt.passback	Anti-passback timeout	Non-negative Integer	OM
sysconf.gt.testmode	Test mode alarm timeout	Non-negative Integer	OM
sysconf.gt.numtp	Number of pass types	Non-negative Integer	RM
sysconf.gt.tp[i]	Gate pass type structure	Record	OU
sysconf.gt.tp[i].type	Pass type	ASCII Text— defined	OU
sysconf.gt.tp[i].desc	Description of i-th pass type	ASCII Text	OU
sysconf.gt.tp[i].accept	Accept pass type	Boolean	OM
sysconf.numpt	Number of payment types	Non-negative Integer	RU
sysconf.pt[i]	Payment type structure	Record	OU
sysconf.pt[i].type	Payment type	ValPaymentType	OU
sysconf.pt[i].accept	Accept payment type	Boolean	OM
sysconf.curprices	Current price table in use	Integer	OM
sysconf.curperiod	Current time period ID in use	Integer	OM
sysconf.secmode	Security mode	ValSecurityMode	OM
sysconf.setinoperable	Set the device out of service	Boolean	OM
sysconf.job	Current shift information structure	Record	OU
sysconf.job.route	Route ID	ASCII Text	OM
sysconf.job.run	Run ID	ASCII Text	OM
sysconf.job.direction	Direction	ASCII Text	OM
sysconf.job.destination	Destination of the route	ASCII Text	OM
sysconf.job.user	User / Driver ID	ASCII Text	OU
sysconf.job.seclvl	User / Driver security level	ASCII Text	OU

17.2.3 Variables dealing with dates and times.

systemdatetime	Date and Time of system clock	RecDateTime	OM
----------------	-------------------------------	-------------	----

17.2.4 Variables dealing with counting.

acct	Accounting information— internal counters maintained by the device	Record	RU
acct.resetdatetime	Date and Time counters were reset	RecDateTime	RU

17.2.4.1 Variables dealing with sales counts.

acct.sale	Sales information structure	Record	OU
acct.sale.vps0	Value of paid sales since initialization	ValMoney	OU
acct.sale.vps1	Value of paid sales since last reset	ValMoney	OU
acct.sale.nps0	Number of paid sales since initialization	ValCounter	OU
acct.sale.nps1	Number of paid sales since last reset	ValCounter	OU
acct.sale.nvi0	Number of vended items since initialization	ValCounter	OU
acct.sale.nvi1	Number of vended items since last reset	ValCounter	OU
acct.sale.vts0	Value of test sales since initialization	ValMoney	OU
acct.sale.vts1	Value of test sales since last reset	ValMoney	OU
acct.sale.nts0	Number of test sales since initialization	ValCounter	OU
acct.sale.nts1	Number of test sales since last reset	ValCounter	OU
acct.sale.nti0	Number of test items since initialization	ValCounter	OU
acct.sale.nti1	Number of test items since last reset	ValCounter	OU
acct.sale.vfv0	Value of free vends since initialization	ValMoney	OU
acct.sale.vfv1	Value of free vends since last reset	ValMoney	OU
acct.sale.nfv0	Number of free vends since initialization	ValCounter	OU
acct.sale.nfv1	Number of free vends since last reset	ValCounter	OU
acct.sale.nfi0	Number of free items since initialization	ValCounter	OU
acct.sale.nfi1	Number of free items since last reset	ValCounter	OU
acct.sale.ecm0	Value of sales in exact change mode since initialization	ValMoney	OU
acct.sale.ecm1	Value of sales in exact change mode since last reset	ValMoney	OU
acct.sale.anc0	Value of non-cash sales since initialization	ValMoney	OU
acct.sale.anc1	Value of non-cash sales since last reset	ValMoney	OU

17.2.4.2 Variables dealing with product counts.

acct.numprod	Number of product types	Non-negative Integer	RU
acct.prod[i]	Product type information	Record	OU
acct.prod[i].product	Product type	ASCII Text	OU
acct.prod[i].vps0	Value of paid sales of i-th product since initialization	ValMoney	OU
acct.prod[i].vps1	Value of paid sales of i-th product since last reset	ValMoney	OU
acct.prod[i].nps0	Number of paid sales of i-th product since initialization	ValCounter	OU
acct.prod[i].nps1	Number of paid sales of i-th product since last reset.	ValCounter	OU
acct.prod[i].nvi0	Number of vended items of i-th product since initialization	ValCounter	OU
acct.prod[i].nvi1	Number of vended items of i-th product since last reset	ValCounter	OU
acct.prod[i].vts0	Value of test sales of i-th product since initialization	ValMoney	OU
acct.prod[i].vts1	Value of test sales of i-th product since last reset	ValMoney	OU
acct.prod[i].nts0	Number of test sales of i-th product since initialization	ValCounter	OU
acct.prod[i].nts1	Number of test sales of i-th product since last reset.	ValCounter	OU
acct.prod[i].nti0	Number of test items of i-th product since initialization	ValCounter	OU
acct.prod[i].nti1	Number of test items of i-th product since last reset.	ValCounter	OU
acct.prod[i].vfv0	Value of free vends of i-th product since initialization	ValMoney	OU
acct.prod[i].vfv1	Value of free vends of i-th product since last reset.	ValMoney	OU
acct.prod[i].nfv0	Number of free vends of i-th product since initialization	ValCounter	OU

acct.prod[i].nfv1	Number of free vends of i-th product since last reset.	ValCounter	OU
acct.prod[i].nvi0	Number of free vended items of i-th product since initialization	ValCounter	OU
acct.prod[i].nvi1	Number of free vended items of i-th product since last reset	ValCounter	OU
acct.prod[i].dtmlso	Date and Time i-th product was last sold out	RecDateTime	OU
acct.prod[i].saso0	Number of times i-th product was selected after it sold out since initialization	ValCounter	OU
acct.prod[i].saso1	Number of times i-th product was selected after it sold out since last reset	ValCounter	OU
acct.prod[i].qam0	Number of i-th product added to machine since initialization	ValCounter	OU
acct.prod[i].qam1	Number of i-th product added to machine since last reset	ValCounter	OU
acct.prod[i].qrm0	Number of i-th product removed from machine since initialization	ValCounter	OU
acct.prod[i].qrm1	Number of i-th product removed from machine since last reset	ValCounter	OU

17.2.4.3 Variables dealing with cash counts.

acct.cash.tti0	Cash taken for test items since initialization	ValMoney	OU
acct.cash.tti1	Cash take for test items since last reset	ValMoney	OU
acct.cash.vcs0	Value of cash sales since initialization	ValMoney	OU
acct.cash.vcs1	Value of cash sales since last reset	ValMoney	OU
acct.cash.ncs0	Number of cash sales since initialization	ValCounter	OU
acct.cash.ncs1	Number of cash sales since last reset	ValCounter	OU
acct.cash.cvi0	Number of cash vended items since initialization	ValCounter	OU
acct.cash.cvi1	Number of cash vended items since last reset	ValCounter	OU
acct.cash.ccb0	Value of cash to cash box since initialization	ValMoney	OU
acct.cash.ccb1	Value of cash to cash box since last reset	ValMoney	OU
acct.cash.ctt0	Value of cash to tubes since initialization	ValMoney	OU
acct.cash.ctt1	Value of cash to tubes since last reset	ValMoney	OU
acct.cash.aca0	Value of all cash accepted since initialization	ValMoney	OU
acct.cash.aca1	Value of all cash accepted since last reset	ValMoney	OU
acct.cash.vcd0	Value of all cash dispensed since initialization	ValMoney	OU
acct.cash.vcd1	Value of all cash dispensed since last reset	ValMoney	OU
acct.cash.cdc0	Value of cash dispensed as change since initialization	ValMoney	OU
acct.cash.cdc1	Value of cash dispensed as change since last reset	ValMoney	OU
acct.cash.cdg0	Value of cash discounts given (vend price minus paid price) since initialization	ValMoney	OU
acct.cash.cdg1	Value of cash discounts given since last reset	ValMoney	OU
acct.cash.vco0	Value of cash overpayment since	ValMoney	OU

	initialization		
acct.cash.vco1	Value of cash overpayment since last reset	ValMoney	OU
acct.cash.cco0	Count of cash overpayments since initialization	ValCounter	OU
acct.cash.cco1	Count of cash overpayments since last reset	ValCounter	OU
acct.cash.vcu0	Value of cash underpayment since initialization	ValMoney	OU
acct.cash.vcu1	Value of cash underpayment since last reset	ValMoney	OU
acct.cash.ccu0	Count of cash underpayments since initialization	ValCounter	OU
acct.cash.ccu1	Count of cash underpayments since last reset	ValCounter	OU
acct.cash.cam0	Value of cash added manually since initialization	ValMoney	OU
acct.cash.cam1	Value of cash added manually since last reset	ValMoney	OU
acct.cash.iic0	Value of cash inserted by inserting a container since initialization	ValMoney	OU
acct.cash.iic1	Value of cash inserted by inserting a container since last reset	ValMoney	OU
acct.cash.rrc0	Value of cash removed by removing a container since initialization	ValMoney	OU
acct.cash.rrc1	Value of cash removed by removing a container since last reset	ValMoney	OU
acct.coin.vac0	Value of accepted coins since initialization	ValMoney	OU

17.2.4.4 Variables dealing with coin counts.

acct.coin.vac1	Value of accepted coins since last reset	ValMoney	OU
acct.coin.nac0	Number of accepted coins since initialization	ValCounter	OU
acct.coin.nac1	Number of accepted coins since last reset	ValCounter	OU
acct.coin.nrc0	Number of rejected coins since initialization	ValCounter	OU
acct.coin.nrc1	Number of rejected coins since last reset	ValCounter	OU
acct.coin.ccb0	Number of coins to cash box since initialization	ValCounter	OU
acct.coin.ccb1	Number of coins to cash box since last reset	ValCounter	OU
acct.coin.ctt0	Number of coins to tubes since initialization	ValCounter	OU
acct.coin.ctt1	Number of coins to tubes since last reset	ValCounter	OU
acct.coin.ncd0	Number of coins dispensed since initialization	ValCounter	OU
acct.coin.ncd1	Number of coins dispensed since last reset	ValCounter	OU
acct.coin.cdc0	Number of coins dispensed as change since initialization	ValCounter	OU
acct.coin.cdc1	Number of coins dispensed as change since last reset	ValCounter	OU
acct.coin.cam0	Number of coins added manually since initialization	ValCounter	OU
acct.coin.cam1	Number of coins added manually since last reset	ValCounter	OU
acct.coin.vct0	Value of coins in all tubes since initialization	ValMoney	OU
acct.coin.vct1	Value of coins in all tubes since last reset	ValMoney	OU
acct.coin.tca	Total change available	ValMoney	OU
acct.coin.numva	Number of types of coins	Non-Negative Integer	OU
acct.coin.va	Coin breakdown by denomination	Record	OU

acct.coin.va[i].type	Type of coin	Integer— defined	OU
acct.coin.va[i].desc	Description of i-th type of coin	ASCII Text	OU
acct.coin.va[i].vac0	Value accepted i-th coin type since initialization	ValMoney	OU
acct.coin.va[i].vac1	Value accepted i-th coin type since last reset	ValMoney	OU
acct.coin.va[i].nac0	Number of accepted i-th coin type since initialization	ValCounter	OU
acct.coin.va[i].nac1	Number of accepted i-th coin type since last reset	ValCounter	OU
acct.coin.va[i].nrc0	Number of rejected coins of i-th type since initialization	ValCounterOU	
acct.coin.va[i].nrc1	Number of rejected coins of i-th type since last reset	ValCounter	OU
acct.coin.va[i].ccb0	Number of coins of i-th type to cash box since initialization	ValCounter	OU
acct.coin.va[i].ccb1	Number of coins of i-th type to cash box since last reset	ValCounter	OU
acct.coin.va[i].ctt0	Number of coins of i-th type to tubes since initialization	ValCounter	OU
acct.coin.va[i].ctt1	Number of coins of i-th type to tubes since last reset	ValCounter	OU
acct.coin.va[i].ncd0	Number of coins of i-th type dispensed since initialization	ValCounter	OU
acct.coin.va[i].ncd1	Number of coins of i-th type dispensed since last reset	ValCounter	OU
acct.coin.va[i].cdc0	Number of coins of i-th type dispensed as change since initialization	ValCounter	OU
acct.coin.va[i].cdc1	Number of coins of i-th type dispensed as change since last reset	ValCounter	OU
acct.coin.va[i].cam0	Number of coins of i-th type added manually since initialization	ValCounter	OU
acct.coin.va[i].cam1	Number of coins of i-th type added manually since last reset	ValCounter	OU
acct.coin.va[i].vct0	Value of coins of i-th type in all tubes	ValMoney	OU

	since initialization		
acct.coin.va[i].vct1	Value of coins of i-th type in all tubes since last reset	ValMoney	OU
acct.coin.va[i].tca	Total change available of i-th coin type	ValMoney	OU

17.2.4.5 Variables dealing with counts of bills.

acct.bill	Information on bank notes	Record	OU
acct.bill.vba0	Value of bills accepted since initialization	ValMoney	OU
acct.bill.vba1	Value of bills accepted since last reset	ValMoney	OU
acct.bill.nba0	Number of bills accepted since initialization	ValCounter	OU
acct.bill.nba1	Number of bills accepted since last reset	ValCounter	OU
acct.bill.nbr0	Number of bills rejected since initialization	ValCounter	OU
acct.bill.nbr1	Number of bills rejected since last reset	ValCounter	OU
acct.bill.vbs0	Value of bills to stacker since initialization	ValMoney	OU
acct.bill.vbs1	Value of bills to stacker since last reset	ValMoney	OU
acct.bill.nbs0	Number of bills to stacker since initialization	ValCounter	OU
acct.bill.nbs1	Number of bills to stacker since last reset	ValCounter	OU
acct.bill.epf0	Value of bills taken on error or power failure since initialization	ValMoney	OU
acct.bill.epf1	Value of bills taken on error or power failure since last reset	ValMoney	OU
acct.bill.numba	Number of bill types	Non-negative Integer	RU
acct.bill.ba[i]	Structure of counts by bank note type	Record	OU
acct.bill.ba[i].type	Bank Note Type	Integer— defined	OU
acct.bill.ba[i].desc	Description of i-th bank note type	ASCII Text	OU
acct.bill.ba[i].vba0	Value of i-th type bills accepted since initialization	ValMoney	OU
acct.bill.ba[i].vba1	Value of i-th type bills accepted since last reset	ValMoney	OU
acct.bill.ba[i].nba0	Number of i-th type bills accepted since initialization	ValCounter	OU
acct.bill.ba[i].nba1	Number of i-th type bills accepted since last reset	ValCounter	OU
acct.bill.ba[i].nbr0	Number of i-th type bills rejected since initialization	ValCounter	OU

acct.bill.ba[i].nbr1	Number of i-th type bills rejected since last reset	ValCounter	OU
acct.bill.ba[i].vbs0	Value of i-th type bills to stacker since initialization	ValMoney	OU
acct.bill.ba[i].vbs1	Value of i-th type bills to stacker since last reset	ValMoney	OU
acct.bill.ba[i].nbs0	Number of i-th type bills to stacker since initialization	ValCounter	OU
acct.bill.ba[i].nbs1	Number of i-th type bills to stacker since last reset	ValCounter	OU
acct.bill.ba[i].epf0	Value of i-th type bills taken on error or power failure since initialization	ValMoney	OU
acct.bill.ba[i].epf1	Value of i-th type bills taken on error or power failure since last reset	ValMoney	OU
acct.bill.ba[i].nbdc0	Number of i-th type of bills dispensed as change since initialization	ValCount	OU
acct.bill.ba[i].nbdc1	Number of i-th type of bills dispensed as change since last reset	ValCount	OU
acct.bill.ba[i].vbdc0	Value of i-th type of bills dispensed as change since initialization	ValMoney	OU
acct.bill.ba[i].vbdc1	Value of i-th type of bills dispensed as change since last reset	ValMoney	OU

17.2.4.6 Variables dealing with counts of tokens.

acct.tok	Information related to tokens	Record	OU
acct.tok.numtoks	Number of token types	Non-Negative Integer	RU
acct.tok.tk[i]	Record for each token type	Record	OU
acct.tok.tk[i].type	Token type ID	Integer — defined	OU
acct.tok.tk[i].desc	Description of i-th token type	ASCII Text	OU
acct.tok.tk[i].ntd0	Number of i-th type tokens dispensed since initialization	ValCounter	OU
acct.tok.tk[i].ntd1	Number of i-th type tokens dispensed since last reset	ValCounter	OU
acct.tok.tk[i].vtd0	Value of i-th type tokens dispensed since initialization	ValMoney	OU
acct.tok.tk[i].vtd1	Value of i-th type tokens dispensed since last reset	ValMoney	OU
acct.tok.tk[i].vto0	Value of i-th type token overpayment since initialization	ValMoney	OU
acct.tok.tk[i].vto1	Value of i-th type token overpayment since last reset	ValMoney	OU
acct.tok.tk[i].vtf0	Value of i-th type tokens filled since initialization	ValMoney	OU
acct.tok.tk[i].vtf1	Value of i-th type tokens filled since last reset	ValMoney	OU
acct.tok.tk[i].ntf0	Number of i-th type tokens filled since initialization	ValCounter	OU
acct.tok.tk[i].ntf1	Number of i-th type tokens filled since last reset	ValCounter	OU
acct.tok.tk[i].nta0	Number of i-th type tokens accepted since initialization	ValCounter	OU
acct.tok.tk[i].nta1	Number of i-th type tokens accepted since last reset	ValCounter	OU
acct.tok.tk[i].vta0	Value of i-th type tokens accepted since initialization	ValMoney	OU
acct.tok.tk[i].vta1	Value of i-th type tokens accepted since last reset	ValMoney	OU

acct.tok.tk[i].ntc0	Number of i-th type tokens to cash box since initialization	ValCounter	OU
acct.tok.tk[i].ntc1	Number of i-th type tokens to cash box since last reset	ValCounter	OU
acct.tok.tk[i].vtc0	Value of i-th type tokens to cash box since initialization	ValMoney	OU
acct.tok.tk[i].vtc1	Value of i-th type tokens to cash box since last reset	ValMoney	OU
acct.tok.tk[i].ntt0	Number of i-th type tokens to tubes since initialization	ValCounter	OU
acct.tok.tk[i].ntt1	Number of i-th type tokens to tubes since last reset	ValCounter	OU
acct.tok.tk[i].vtt0	Value of i-th type tokens to tubes since initialization	ValMoney	OU
acct.tok.tk[i].vtt1	Value of i-th type tokens to tubes since last reset	ValMoney	OU

17.2.4.7 Variables dealing with counts of credit/debit card usage.

acct.cdc	Information on credit/debit cards	Record	OU
acct.cdc.ncp0	Number of credit/debit card payments since initialization	ValCounter	OU
acct.cdc.ncp1	Number of credit/debit card payments since last reset	ValCounter	OU
acct.cdc.vcp0	Value of credit/debit card payments since initialization	ValMoney	OU
acct.cdc.vcp1	Value of credit/debit card payments since last reset	ValMoney	OU
acct.cdc.vda0	Value debited from credit/debit card accounts since initialization	ValMoney	OU
acct.cdc.vda1	Value debited from credit/debit card accounts since last reset	ValMoney	OU
acct.cdc.vca0	Value credited to credit/debit card accounts since initialization	ValMoney	OU
acct.cdc.vca1	Value credited to credit/debit card accounts since last reset	ValMoney	OU
acct.cdc.vdc0	Value of discounts on credit/debit card sales since initialization	ValMoney	OU
acct.cdc.vdc1	Value of discounts on credit/debit card sales since last reset	ValMoney	OU
acct.cdc.ndc0	Number of discounts on credit/debit card sales since initialization	ValCounter	OU
acct.cdc.ndc1	Number of discounts on credit/debit card sales since last reset	ValCounter	OU
acct.cdc.nna0	Number of non-accepted credit/debit cards since initialization	ValCounter	OU
acct.cdc.nna1	Number of non-accepted credit/debit cards since last reset	ValCounter	OU

17.2.4.8 Other counts.

acct.rev	Revaluation information	Record	OU
acct.rev.vac0	Value added to cards as an incentive to reuse since initialization	ValMoney	OU
acct.rev.vac1	Value added to cards as an incentive to reuse since last reset	ValMoney	OU
acct.rev.ugd0	User group discounts since initialization	ValMoney	OU
acct.rev.ugd1	User group discounts since last reset	ValMoney	OU
acct.oth	Other information record	Record	OU
acct.oth.npo0	Number of power outages since initialization	ValCounter	OU
acct.oth.npo1	Number of power outages since last reset	ValCounter	OU
acct.oth.nrl0	Number of reads of the logged data since initialization	ValCounter	OU
acct.oth.nrl1	Number of reads of the logged data since last reset	ValCounter	OU
acct.oth.ndo0	Number of door openings since initialization	ValCounter	OU
acct.oth.ndo1	Number of door openings since last reset	ValCounter	OU
acct.oth.npc0	Number of price changes since initialization	ValCounter	OU
acct.oth.npc1	Number of price changes since last reset	ValCounter	OU
acct.oth.nsc0	Number of software/operating data changes since initialization	ValCounter	OU
acct.oth.nsc1	Number of software/operating data changes since last reset	ValCounter	OU
acct.oth.dtls	Date and Time last serviced	RecDateTime	OU
acct.oth.svid	User ID of last service access	Integer	OU
acct.oth.numerr	Number of errors in following structure	Non-negative Integer	RU
acct.oth.er[i]	Error record array	Record	OU
acct.oth.er[i].type	ID of i-th error	Integer — defined	OU
acct.oth.er[i].desc	Description of i-th error	ASCII Text	OU

acct.oth.er[i].subID	Associated sub-assembly ID	Integer — defined	OU
acct.oth.er[i].count0	Number of i-th error since initialization	ValCounter	OU
acct.oth.er[i].count1	Number of i-th error since last reset	ValCounter	OU
acct.numsub	Number of sub-assemblies	Non-Negative Integer	RU
acct.sub[i]	Sub-assembly accounting information	Record	OU
acct.sub[i].type	Type ID of i-th sub-assembly	Integer — defined	OU
acct.sub[i].desc	Description of i-th sub-assembly	ASCII Text	OU
acct.sub[i].manufact	Manufacturer of i-th sub-assembly	ValManufacturerID	OU
acct.sub[i].model	Model number of i-th sub-assembly	ASCII Text	OU
acct.sub[i].serial	Serial number of i-th sub-assembly	ASCII Text	OU
acct.sub[i].cycles	Number of cycles (as in MCBF) of i-th sub-assembly since its insertion	ValCounter	OU
acct.sub[i].numct	Number of types of contents in a sub-assembly (usually zero or one)	Non-Negative Integer	RU
acct.sub[i].ct[j]	Record of each type of contents	Record	OU
acct.sub[i].ct[j].type	Type of contents	Integer— defined	OU
acct.sub[i].ct[j].desc	Description of j-th content type in i-th sub-assembly	ASCII Text	OU
acct.sub[i].ct[j].count	Number of j-th content type in i-th sub-assembly	ValCounter	OU
acct.sub[i].ct[j].value	Value of j-th content type in i-th sub-assembly	ValMoney	OU
acct.pri	Information on printers	Record	OU
acct.pri.noj0	Number of stock jams since initialization	ValCounter	OU
acct.pri.noj1	Number of stock jams since last reset	ValCounter	OU

17.2.4.9 Variables dealing with counts related to gates.

acct.gat	Information on faregates	Record	OU
acct.gat.nop0	Number of passages since initialization	ValCounter	OU
acct.gat.nop1	Number of passages since last reset	ValCounter	OU
acct.gat.nox0	Number of exits since initialization	ValCounter	OU
acct.gat.nox1	Number of exits since last reset	ValCounter	OU
acct.gat.non0	Number of entries since initialization	ValCounter	OU
acct.gat.non1	Number of entries since last reset	ValCounter	OU
acct.gat.numerr	Number of types of errors	Non-negative Integer	RU
acct.gat.er[i]	Information on each type of error	Record	OU
acct.gat.er[i].type	Type of error	Integer— defined	OU
acct.gat.er[i].desc	Description of i-th type of error	ASCII Text	OU
acct.gat.er[i].nen0	Number of i-th type of error on entry since initialization	ValCounter	OU
acct.gat.er[i].nen1	Number of i-th type of error on entry since last reset	ValCounter	OU
acct.gat.er[i].nex0	Number of i-th type of error on exit since initialization	ValCounter	OU
acct.gat.er[i].nex1	Number of i-th type of error on exit since last reset	ValCounter	OU
acct.gat.numttp	Number of ticket types	Non-negative Integer	RU
acct.gat.tt[i]	Counts by ticket type	Record	OU
acct.gat.tt[i].type	Ticket type	Integer— defined	OU
acct.gat.tt[i].desc	Description of i-th ticket type	ASCII Text	OU
acct.gat.tt[i].nen0	Number of i-th ticket type taken at entry since initialization	ValCounter	OU
acct.gat.tt[i].nen1	Number of i-th ticket type taken at entry since last reset	ValCounter	OU
acct.gat.tt[i].nex0	Number of i-th ticket type taken at exit since initialization	ValCounter	OU
acct.gat.tt[i].nex1	Number of i-th ticket type taken at exit	ValCounter	OU

	since last reset		
acct.gat.tt[i].capt0	Number of i-th ticket type captured since initialization	ValCounter	OU
acct.gat.tt[i].capt1	Number of i-th ticket type captured since last reset	ValCounter	OU

Chapter 18

Creating Extended Data Items

Three types of data item modifications can be made by users of this standard: additional attributes of conditions, named variables, and new values for existing types. To guarantee that all data items do not have the same name as any other data item, we list the following rules:

1. All additional attributes created as part of this standard have the underscore character (_) as a prefix. No additional attributes created apart from this standard shall have the underscore as a prefix.
2. Each manufacturer / implementor of this standard shall have a unique 3-character ID, generally derived from their name. The VEI controlling body (currently Agent Systems, Inc.) will approve/assign these ID's. Implementors who wish to have an ID value should use the form on the following page.
3. The manufacturer ID shall be used as a prefix to all additional attributes and named variables (including record members) created by that manufacturer. Beyond this, it is the responsibility of the manufacturer / implementor to guarantee uniqueness.
4. Implementors desiring new data values to be defined for existing VEI data types, or who want to have new generic data types or named variables shall also submit the form on the following page to the VEI controlling body. The address to send it for this version of VEI is

VEI Data Modification Request
Agent Systems, Inc.
14802 Venture Drive
Farmers Branch, Texas 75234-2426
FAX: (972) 392-7301
E-mail: vei@agentsystems.com

VEI Data Modification Request

Requestor Name _____

Requestor Company _____

Requestor Address _____

Request Type (circle one): New ID New Variable New Value

Description of new ID, variable, or value: (Note: For an ID, include a primary and alternate suggestion. For a variable, include a complete definition of the data type, range of values, and value definition.)

Appendix A

Glossary

ACLL	Agent Contention Link Layer.
ACK	A single ASCII character passed back to the sending party that acknowledges the successful receipt of a sent packet.
Acknowledgment	The message of a dialog given to communicate that a response was received.
Application Layer	The highest protocol layer that contains messages specific to the application.
ASCII	American Standard Code for Information Interchange.
ASL	Agent Session Layer.
Asynchronous	A data transmission method that sends one character at a time, contrasted with synchronous methods, that send a packet of data and then re-synchronize their clocks.
ATL	Agent Transport Layer.
Attribute	A data element, having a defined meaning, with a statement of the set of possible values it may take.
Authentication	The process of verifying the true origin or nature of the sender and/or the integrity of the text of a message. With public key cryptography, it involves digital signatures. With secret key cryptography, it involves proving knowledge of a secret token.
Authorization	The process by which permission is granted by a properly appointed person or persons to do some action for the organization.
Baud	The rate of the modulation of an analog signal. A 300 baud signal modulates at 300 times per second. One baud is approximately equal to one bit per second.
BCD	Binary Coded Decimal. An encoding technique, also known as packed decimal, where strings of ASCII digits 0 through 9 are packed into consecutive 4-bit nibbles.
BEL	A single ASCII control character sent to indicate that a dialog failed.
Binary	The lowest level representation of data on a computer system or network. Every character is made up of 8 binary digits or bits. There are 8 bits in one byte.

Bit	A binary digit with a value of zero or one.
Blacklist	A list of card numbers that are automatically rejected.
BPS	Bits Per Second. The preferred term for measuring data transmission speed on data communications devices.
Broadcast	A system of delivering data where there is a single sender and potentially multiple receivers. Broadcast messages are not acknowledged by the receivers, so there is no assurance of integrity of the data stream. Broadcasts on IP are only within subnets (i.e. not to the entire network).
Capture	A transaction sent after the merchant has shipped the goods. This transaction will trigger the movement of funds.
CCITT	International Telephone and Telegraph Consultative Committee. An international standards making organization consisting of national telecommunication authorities. Also called ITU.
Cleartext	A data format that is not protected by encryption.
Client	The peer communicating entity that initiates a connection or uses services provided by a server.
Condition	A particular circumstance or situation at a device. It has a number of attributes associated with it. The most important is the status, which can be active, idle, or disabled. When the status of a condition changes, it is called an event. For example, the "door open" condition changes status from idle to active when the door opens. It changes back to idle when the door closes.
CRC	Cyclic Redundancy Check. A mathematical value calculated on a data block that will differ with high probability if the block is modified. It can be used to guarantee data integrity over a communications medium.
Cryptography	The enciphering and deciphering of messages in secret code or cipher.
Data	Any representation to which meaning is or might be assigned.
Data block	The unit of data transferred between link layers. Also called packet. It is introduced with STX and concluded with ETX.
DEA	Data Encryption Algorithm.
DES	Data Encryption Standard (See DEA).
Dialog	A sequence of one or more related messages exchanged between peers to provide a service.

Dialog Class	A group of functionally related services.
Digital Signature	Information encrypted with an entity's private key, which is appended to a message to assure the recipient of the authenticity and integrity of the message.
Download	The process of transferring data to a remote entity.
Encryption	The process of applying an algorithm to a piece of data (cleartext) to create a new piece of data (ciphertext) that is unrecognizable to a third party observer. A key is used to decrypt the data back to its readable form. As long as the key is kept private among the parties involved in the transmission of the data, third parties cannot easily determine the cleartext from the ciphertext.
Endian	The order of bytes within a word. Big Endian is the most significant byte first. Little Endian is the least significant byte first.
ENQ	A single ASCII character sent when requesting permission to send a data block.
EOT	End of transmission. A single ASCII character sent to indicate a dialog sequence is complete.
ETX	A single ASCII character sent to indicate the end of a data block.
Event	A condition status change. They can be spontaneously reported to the MHC using the "Condition — Unsolicited Status" dialog.
Execution	The state of a program where its instructions are being carried out by the computer's processor.
Farebox	A device that stores fares received from mass transit patrons at the time of entry to a vehicle.
Faregate	A device that controls access to a service area based on presentation by the patron of a ticket or token.
Field	A meaningful piece of data carried as a data element or data item within a record or message. Fields are delimited by as ASCII Field Separator (FS) character.
File	An unambiguously named collection of information having a common set of attributes.
FIPS PUB	Federal Information Processing Standards Publication.
FS	A single ASCII character used to separate fields within a record.

Hexadecimal	A data representation that uses a base 16 numbering scheme to represent a single alphanumeric value. One byte is represented by two hexadecimal characters (0 through 9 and A through F).
IEC	International Electrotechnical Commission.
IETF	Internet Engineering Task Force. The body that reviews and approves standards for use on the internet.
Incident	A condition that has no status. That is, at any given point in time, an incident condition is neither active nor idle. It is a potentially recurring event.
Integrity	The assurance that the data received is in fact the data sent.
Interface	A system (hardware and software) that links two devices together for communication purposes.
IP	Internet Protocol.
ISO	International Organization for Standardization.
ITU	International Telecommunication Union (See CCITT).
Journal	A set of recorded, time-tagged condition status changes, variable data, and/or comments, which may be logically ordered during retrieval.
Journaling	The process of recording a journal.
Layered Protocol	A method of designing communications protocols such that layer N at the destination receives exactly the same object sent by layer N at the source. Each layer provides a set of services that are transparent to the layers above.
Link Layer	The protocol layer that describes the basic, generic communication of packets between devices. The contents of packets from higher layers are not relevant to the Link Layer.
Log	A log is a historic record of events and other archived information that can be used to determine a sequence of events at a device. Three types of data are stored in a log file: events, variables, and comments.
LSB	Least Significant Bit. The bit within a binary number that can contribute the least to the magnitude of the number.
Message	A unit of data transferred between application layers, consisting of a set of fields with an initial message header.
Message Header	A set of fields indicating the message type, sequence, etc. used to

introduce a message.

MHC	Managing Host Computer. The computer that controls the vending device over the network using the Vending Equipment Interface Standard.
MSB	Most Significant Bit. The bit within a binary number that can contribute the most to the magnitude of the number.
Network	An interconnected or intersecting configuration of systems or components.
NIST	National Institute of Standards and Technology.
NSA	National Security Agency.
Open specification	A specification that is published and freely available for use by anyone.
PAN	Cardholder's Primary Account Number; the number on the card.
PIN	Personal Identification Number.
Pinpad	A numeric keypad used for entering a PIN. They generally use some form of encryption to protect the private information entered.
Priority	The importance of the condition relative to other conditions. High priority conditions may preempt lower priority conditions in the order of processing.
Private Key	A mathematical key (kept secret by the owner) used to create digital signatures, or to decrypt messages or files.
Process	A particular instance of a program currently in execution.
Program	A function or a set of functions that can be executed at a computer.
Proprietary specification	A specification that is not published, not available for use, or for which trade secret status is asserted by the owner.
Protocol	A set of rules to which two or more entities agree specifying how they communicate with one another.
Public Key	A mathematical key that is available publicly. It is used to verify signatures created with the matched private key. Public keys are also used to encrypt messages or files that can only be decrypted using the matched private key.
Record	A general term for data structure and data layout used to describe or contain data fields.

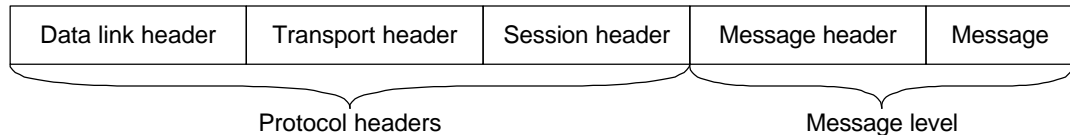
Request	The initial message of a dialog which asks for a specific action to take place. It expects a response.
Response	The second message of a dialog that acknowledges receipt of the request, conveys success or failure, and may include information requested. It may expect an acknowledgment.
Reversal	A transaction sent when a previous authorization needs to be canceled or decreased.
RFC	Request For Comments. A technical document submitted to the IETF to become an internet standard.
Secret Key	A mathematical key known only by authorized entities. It is used to encrypt and decrypt blocks of data or to authenticate one another.
Sequence Control	Guaranteeing that data are received in the order in which they were sent.
Server	The peer communicating entity that receives a connection and provides services to clients over the network.
Service	A single network function achieved by sending and receiving messages in a dialog.
Session Layer	The layer of the protocol that identifies and authenticates the sender and receiver to each other. It may also provide message level encryption.
Severity	The representation of the effect the condition has on the controlled device. A higher severity indicates a worse effect, whereas a lower severity indicates a better effect.
Standard specification	An open specification that has been accepted for use by multiple vendors.
STX	A single ASCII control character used to introduce a data block.
TCP	Transmission Control Protocol.
Timeout	A condition that occurs after a certain time has elapsed without a specified intervening event occurring.
Transaction	A sequence of related messages consisting of a request, a response, and an acknowledgment. See also dialog.
Transport Layer	The layer responsible for guaranteeing data sequencing and integrity.
Type	An abstract description of the class of data that may be conveyed by the value of a variable.
Upload	The process of transferring data from a remote entity.

Variable	One or more data elements referred to together by a single name or description.
VEI	Vending Equipment Interface.
X.25	The CCITT standard for packet-switched networks.
XOR	Exclusive OR bitwise operation.
zlib	A data compression/decompression library that implements the algorithms specified in RFC 1950 and RFC 1951.

Appendix B Dialog Tutorial

B.1 Overview

This chapter provides sample dialogs between a device and the MHC. It describes various dialogs in each dialog class. Generally, messages would have the following form as monitored on the communication line:

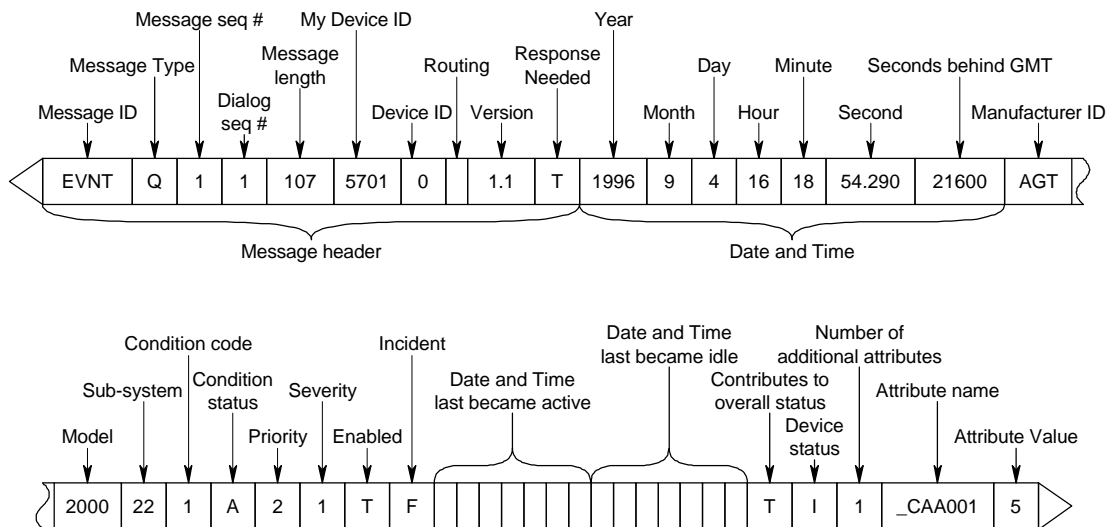


In the examples given in this chapter, however, only the Message level sections are shown, since the protocol headers do not affect the meaning or use of the messages. Each field is in a separate box, and the field-separator characters are represented by the lines separating the boxes. Triangles show the beginning and ending of each message.

B.2 Condition — Unsolicited Status

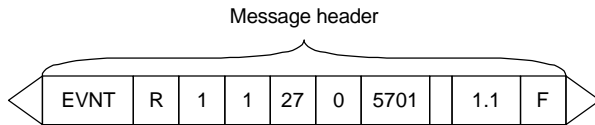
A device uses this dialog to tell the MHC that the status of a condition has changed. For example, when someone opens the door of the device, the “door open” condition changes status from idle to active. The dialog is called “unsolicited” because the MHC did not explicitly request this information.

Following is the request message sent by the device to the MHC.



Note that three sections contain no data: “routing” in the message header, the “date and time last became active”, and the “date and time last became idle.” These are empty because there is no routing necessary, and the condition had never occurred before.

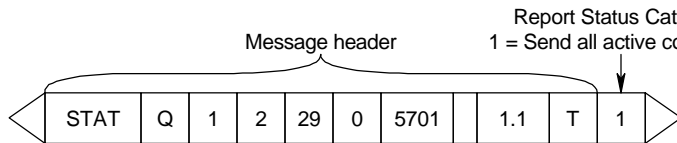
Following is the response from the MHC. It indicates only that the request message was received. Note that the dialog sequence number is the same as in the request.



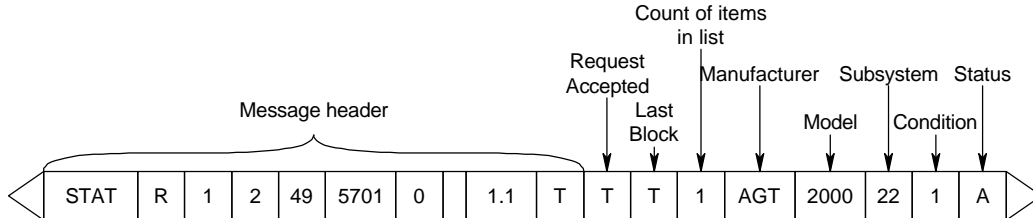
B.3 Condition — Report Status

With this dialog, the MHC asks for the status of a condition or list of conditions from a device. The MHC might instead ask for all active conditions, as the following dialog shows.

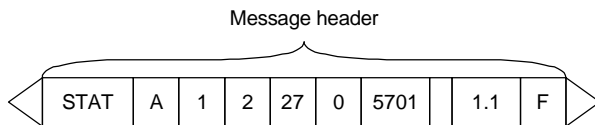
The MHC initiates this dialog with the following request message:



The device responds with the status report:



And, the MHC acknowledges receipt of the status report:

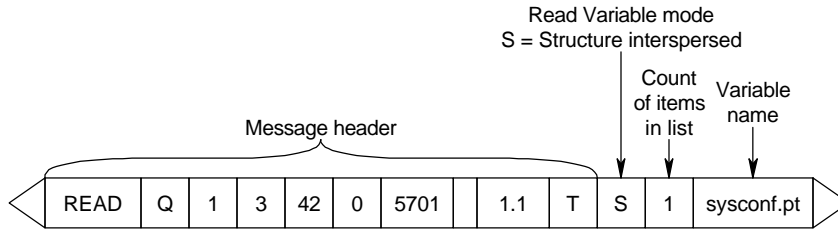


B.4 Variable Access — Read

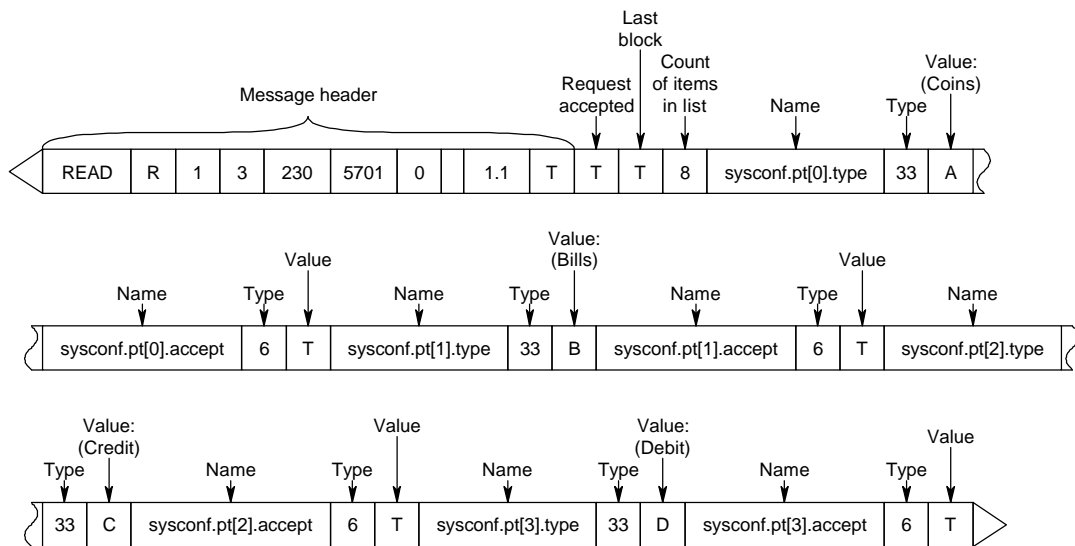
The MHC uses this dialog to retrieve the value of a variable data item stored within a device. There are two modes for the request: basic mode and structure-interspersed mode. In basic mode, the response contains only the value of the requested variables. In structure-interspersed mode, the response contains the name of each field and the data type before its value.

The MHC sends the following request to the device, asking for the payment type array of

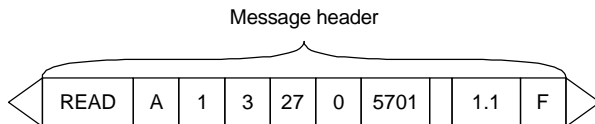
structures:



The device answers the request with the following response message, which indicates that all four payment types (coins, bills, credit, debit) are currently accepted:

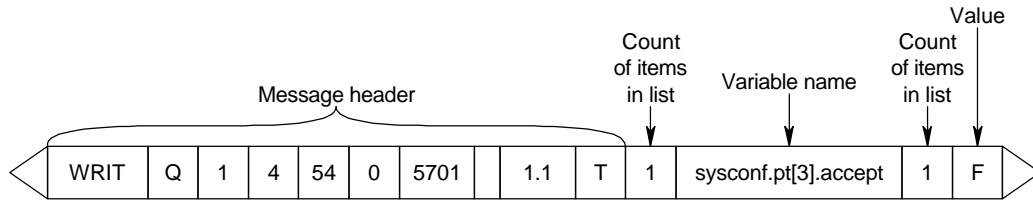


And, the MHC acknowledges receipt of the answer:

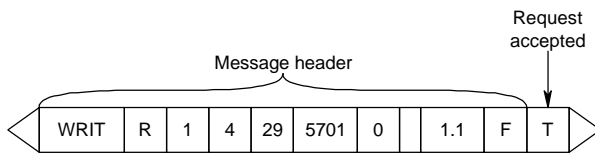


B.5 Variable Access — Write

In this dialog, the MHC requests a change to the value of a variable data item stored at a device. The following example shows the MHC telling the device not to accept debit cards as a payment type.

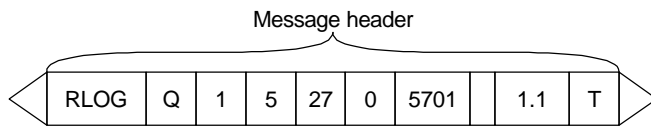


And, the device responds indicating receipt of the request:

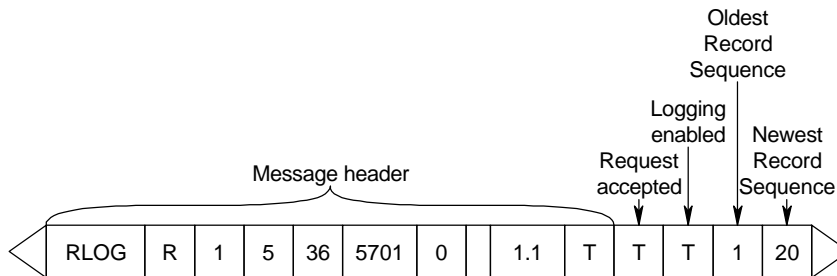


B.6 Log File — Report Log Status

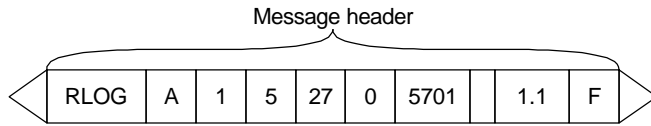
The MHC uses this dialog to determine the status of logging events, variables, and comments at a device. The response contains the beginning and ending record sequence numbers in the log file.



And the device responds with the following

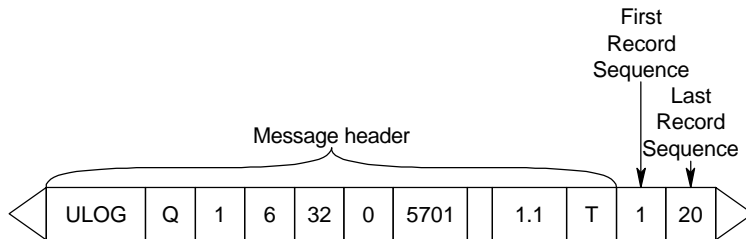


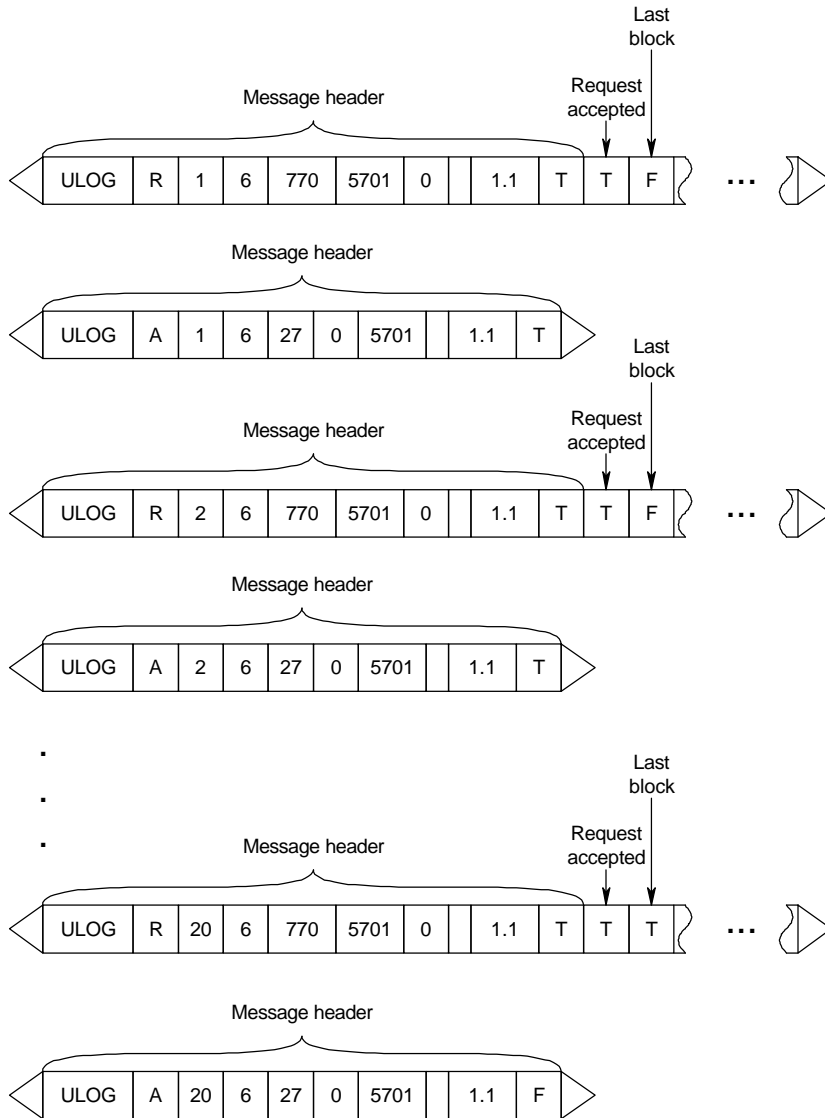
The MHC acknowledges receipt of the response with the following:



B.7 Log File — Upload Log

The MHC uses this dialog to retrieve a specified portion of the log file stored at a device. The device sends the requested portion as a data stream to the MHC, which acknowledges each block as it is received. Here is the request for the device to begin sending the log file.

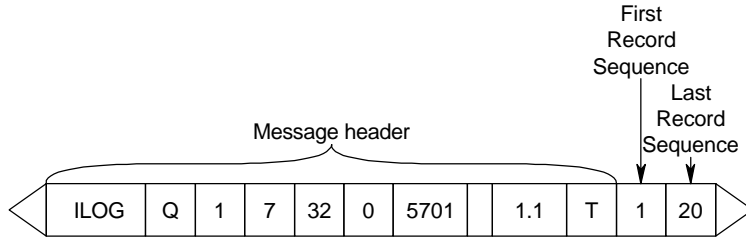




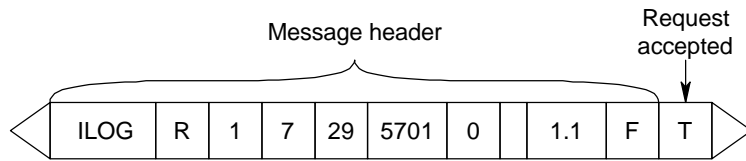
B.8 Log File — Initialize Log

With this dialog, the MHC purges a specified portion of the log file stored at a device. This is typically used after successfully uploading part of the log file to the MHC.

This is the request from MHC to device:



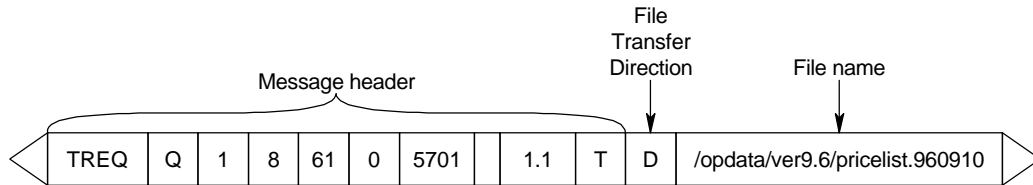
The device sends the following response:



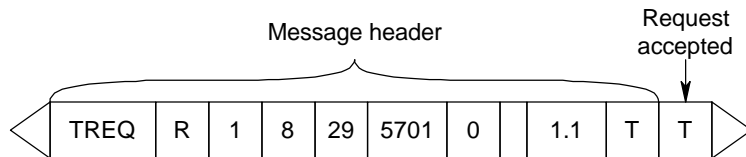
B.9 File Management — Transfer Request / Transfer Block

In this dialog sequence, the MHC transfers a file to the device's file system. This is typically used to send programs and operating data to a device.

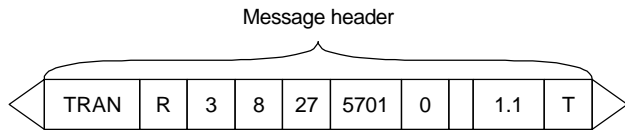
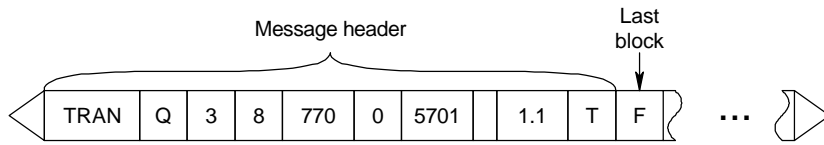
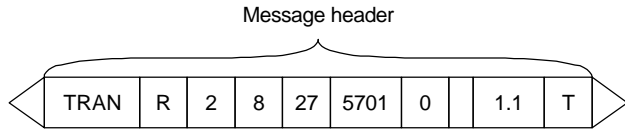
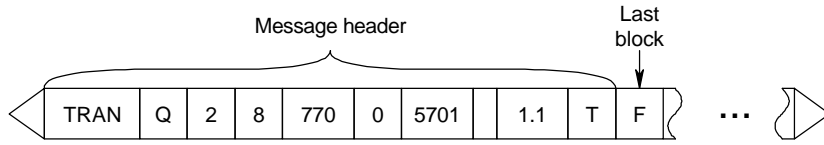
The MHC requests the file transfer, by name.



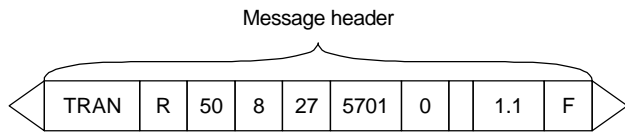
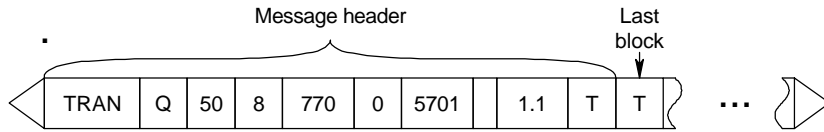
The device accepts the transfer request.



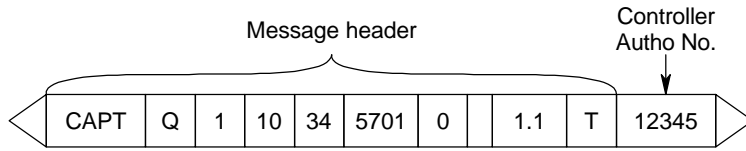
The Transfer Block sequence occurs immediately after the transfer request is accepted, with the dialog sequence number unchanged.



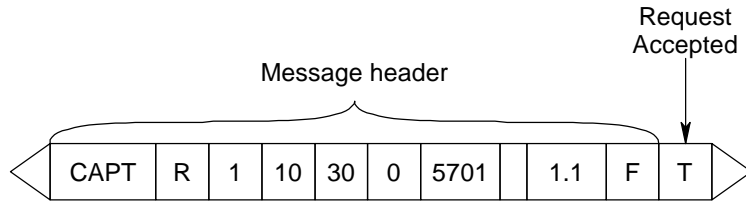
•
•
•



The device initiates this dialog with the following request message:



The MHC responds with the following, indicating receipt of the request.



Appendix C

C language routines

Following is C language code to implement the CRC-16 calculation.

```
void calccrc(
    unsigned short *crc,      /* 2 byte crc being calculated */
    unsigned char ch         /* current byte to add to crc */
)
{
    short i;

    for (i = 0; i < 8; i++)
    {
        if ((*crc ^ (unsigned short)ch) & 0x0001)
            *crc = (*crc >> 1) ^ 0xA001;
        else
            *crc >>= 1;
        ch >>= 1;
    }
} /* calccrc */
```

Additional C language implementations are available to be licensed from Agent Systems, Inc. for

1. The Agent Link Layer.
2. ISO 1745 link layer.
3. The Agent Transport Layer.
4. The Agent Session Layer.

Contact Agent Systems for further details.

Appendix D

Synchronizing Remote Clocks

Synchronizing the system clock on each device in a network is important, since those devices have to communicate information related to the current time, especially items related to establishing a sequence of events. We provide here a procedure to set the system clock at a remote device from the MHC, using the clock at the MHC as the standard. (The MHC's clock can be similarly controlled from some other standard.) We seek here to keep the clocks synchronized with as little network traffic as possible.

Periodically using a single dialog, Variable Access — Read (systemdatetime), the MHC can obtain the following information:

RTT	(Round-Trip Time) The time difference between sending the request and receiving the response.
AVR	(Average Round-Trip Time) As the dialog is communicated periodically, a running average of the round-trip time can be maintained.
OFS	(Offset of time between the two clocks) This is a measure of the time difference between two clocks. An offset greater than zero means the remote clock is ahead; an offset less than zero means the remote clock is behind.
SKW	(Time skew of the remote clock) This is a measure of the frequency difference of the remote clock (that is, the first derivative of offset with respect to time). Skew greater than zero means the remote clock is fast; skew less than zero means the remote clock is slow.
TNR	(Time of next read) This is the time that the next Variable Access — Read (systemdatetime) dialog should be done.

The following items are available at the time the response to the Variable Access — Read (systemdatetime) is received:

abs()	The absolute value function.
CURRENT_TIME	The value of time as given by the local system clock.
MAX_OFFSET	The maximum allowable offset of the remote device's clock. If the current offset is greater than this value, then the remote time should be set immediately.
MAX_PERIOD	The maximum amount of time between requests to read the systemdatetime at a remote device. Usually, a value of several days is appropriate.
MIN_PERIOD	The minimum amount of time between requests to read the systemdatetime at a remote device. Usually, a value of a few hours is appropriate.
REQUEST_TIME	The time the request was sent.
Response.systemdatetime	The value of the systemdatetime variable from the remote device.

The following method can be used to determine when the clock should be set using the dialog Variable Access — Write (systemdatetime). Note that the skew variable must be floating point. Other variables can

be integers. Note also that timings should be made using the highest granularity available on the clocks.

```

RTT = CURRENT_TIME - REQUEST_TIME           // Calculate the round trip time.

if (first)                                  // Initialize the variables.
    first = FALSE
    AVR = RTT * 8
    OFS = OFS_OLD = Response.systemdatetime - CURRENT_TIME - (RTT / 2)
    TIME_OLD = CURRENT_TIME
    SKW = (MAX_OFFSET - abs(OFS)) / MIN_PERIOD
else if (CURRENT_TIME == TIME_OLD)          // Nothing to do in this case.
    TNR = CURRENT_TIME + MIN_PERIOD
    return
else                                         // Update the variables.
    AVR = AVR + RTT - (AVR / 8)
    OFS = CURRENT_TIME - (AVR / 16) - Response.systemdatetime
    SKW = (OFS - OFS_OLD) / (CURRENT_TIME - TIME_OLD)

if (SKW == 0.0)                             // Prevent division by zero errors.
    SKW = MAX_OFFSET / MAX_PERIOD           // No skew ==> Wait a long time.

if (abs(OFS) > MAX_OFFSET)                  // Should we set the remote clock?
    Send the Variable Access - Write with systemdatetime = CURRENT_TIME + (AVR / 16)
    OFS_OLD = 0
    TIME_OLD = CURRENT_TIME
    TNR = CURRENT_TIME + (MAX_OFFSET / abs(SKW))
else
    TNR = CURRENT_TIME + ((MAX_OFFSET - abs(OFS)) / abs(SKW))

if (TNR > CURRENT_TIME + MAX_PERIOD)        // Don't wait too long to check again
    TNR = CURRENT_TIME + MAX_PERIOD
else if (TNR < CURRENT_TIME + MIN_PERIOD)   // Don't check again too soon.
    TNR = CURRENT_TIME + MIN_PERIOD

```

Appendix E Release Notes

E.1 Release 1.0

Released October 4, 1996.

E.2 Release 1.1

1. Added the field "Response Needed" in the message header.
2. Added zlib compression to the session layer as an option.
3. Added support for broadcasting in place of a connection-oriented protocol.
4. Added capability negotiation to the session layer.
5. Added the procedure for adding data values to existing data types.
6. Added the additional attributes field to the Authorization dialog, for supporting other types of transactions in addition to credit and debit card.
7. Added data items related to bus farebox and other mass transit related items.
8. Created definitions for the following: broadcast, cleartext, farebox, faregate, IETF, incident, journaling, open specification, pinpad, proprietary specification, RFC, standard specification, and zlib
9. Updated drawings and examples to reflect the above modifications.
10. Updated the language in certain places to be more precise and/or clear.
11. Removed the concept of implementation levels. It had implied more complexity than is intended. Fully compliant implementations can pick and choose which dialogs to use based on their intrinsic functionality needs.
12. Restructured some of the inconsistent and confusing paragraph numbering.
13. Put the example documentation in Chapter 6 into a tabular format.
14. Updated the examples in Chapter 4 to illustrate several device types using a variety of dialogs to implement their functionality.
15. Added third example, section 6.4.
16. Split the table in Chapter 17 into several smaller tables, and add section headings.

Submitted for partner review: November 17, 1997.

Released February 2, 1998.

E.3 Release 1.2

1. Changed transport header magic number value to CBH. Backward compatibility can be maintained by looking at the seven low-order bits.
2. Renamed transport header "More Data" field to "Need Acknowledgment".
3. Added the transport header "Fresh Connection" flag.
4. Reordered the steps for session layer packet processing.
5. Utilized standard packet padding algorithm in session layer processing.
6. Added session layer magic number prefix.
7. Specified session layer to use DES CBC mode with carry-over.
8. Added note to Variable Access — Read dialog for structure interspersed mode.
9. Added options for size, permissions, ownership, attributes to File Management — Transfer Request, via the additional attribute list.
10. Changed type of Transaction — Authorization request field Pinpad serial number to be Alphanumeric, 16 characters.
11. Changed type of all counts to be Non-Negative Integer rather than Positive Integer.
12. Added a few card types to ValCardType.
13. Added severity field to RecStatusReport.
14. Added variables associated with bills dispensed as change in 17.2.4.5.
15. Added definition of severity and priority to glossary.
16. In section B.11, length of messages changed to 34 and 30 bytes respectively.

Submitted for partner review July 17, 2001.

Released July 31, 2002.