TOSHIBA

Toshiba Global Commerce Solutions
SurePOS ACE

Application Client/Server Environment for 4690 OS Programming Reference

Version 8 Release 1.2

TC62-0046-02

Note:

Before using this information and the product it supports, be sure to read Safety Information-Read This First, Warranty Information, Uninterruptible Power Supply Information, and the information under "Notices" on page 733.

June 2019

This edition applies to Version 8 Release 1.2 of the Toshiba Global Commerce Solutions SurePOS Application Client/Server Environment for 4690 OS licensed program, program number 5697-N80, and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

About this guide

This guide describes the general functions provided by the Toshiba Global Commerce Solutions SurePOS Application Client/Server Environment for 4690 OS, which is commonly known as SurePOS ACE.

Who should read this guide

This guide is for SurePOS ACE programmers who need access to the data and initialization objects within SurePOS ACE.

Terms used in this guide assume that you are familiar with supermarket business concepts, the C++ programming language, object-oriented design and programming techniques, and Toshiba 4690 Operating System fundamentals.

Where to find more information

Current versions of Toshiba publications are available on the Toshiba Global Commerce Solutions website at http://www.toshibacommerce.com/support/publications. The publications listed under the General tab are available to the public.

Note: Access to the product publications require valid user credentials. For information on obtaining a user ID and password, click About us, and then FAQs.

To access a specific Toshiba product publication:

1. Go to http://www.toshibacommerce.com.
2. Click Support.
3. Click Publications.
4. Click the appropriate product category (for example, Hardware).
5. Scroll down and select the desired product.
6. Scroll down and select the appropriate manual listed under the Publications header, and the PDF will be downloaded to your computer.

Accessing the TGCS Knowledgebase site

Toshiba Global Commerce Solutions has developed a variety of Knowledgebase articles to assist you in using the Toshiba product set. To access the TGCS Knowledgebase articles:

1. Enter your user ID and Password.
2. Click Support.
3. Click Publications.
4. Click the appropriate product category (for example, Hardware).
5. Scroll down and select the desired product.
6. Click Restricted content (Security Alerts, BIOS & Publications).
7. Click the Knowledgebase link.

Prerequisite publications

These guides are available on the www.toshibacommerce.com Web site:

- *SurePOS Application Client/Server Environment for 4690 OS: Planning and Installation Guide*, TC62-0044.
- *4690 OS Messages Guide*, G362-0573, which helps you work with system messages.
- *SurePOS Application Client/Server Environment for 4690 OS: EPS User's Guide*, TC62-0047.

- *SurePOS Application Client/Server Environment for 4690 OS: Guide to Operations*, TC62-0045.
- *SurePOS Architecture and EMV Guide for EPS Payment*, TC62-0048.

Related publications

Data Integration Facility Documentation

- *Data Integration Facility: User's Guide*, TC62-0025.

Programming References

- *Borland TurboVision for DOS Programmer's Guide*, SC30-3980
- *Rogue Wave Introduction and Reference Manual Tools.h++*, SC30-3997

Non-Toshiba Publications about C++ and Object-Oriented Programming

Many books have been written about the C++ language and related programming topics, with varying approaches and emphasis. These non-Toshiba C++ publications are generally available but do not represent the entire list of available resources.

Toshiba does not specifically recommend any of these books. Other C++ books might be available in your locality.

When there are differences between this book and any of those listed below, this book takes precedence.

- *Design Patterns: Elements of Reusable Object-Oriented Software* by Gamma, Helm, Johnson, Vlissides, Addison-Wesley Publishing Company, ISBN Number: 0201633612
- *C++ Programming Style* by Tom Cargill, Addison-Wesley Publishing Company, ISBN Number: 0201563657
- *Taligent's Guide to Designing Programs: Well-Mannered Object-Oriented Design in C++* by Inc. Taligent, Addison-Wesley Publishing Company, ISBN Number: 0201408880
- *The C++ Programming Language* by Bjarne Stroustrup, Addison-Wesley Publishing Company, ISBN Number: 0201889544
- *The Annotated C++ Reference Manual* by Margaret A. Ellis and Bjarne Stroustrup, Addison-Wesley Publishing Company, ISBN Number: 0201514591
- *C++ Primer* by Stanley B. Lippman, Addison-Wesley Publishing Company, ISBN Number: 0201824701
- *Object-Oriented Analysis and Design With Applications (Addison-Wesley Object Technology Series)* by Grady Booch, Benjamin/Cummings, ISBN Number: 0805353402 Updated August 9, 2011 630 SurePOS

Notice statements

Notices in this guide are defined as follows:

Note

These notices provide important tips, guidance, or advice.

Important

These notices provide information or advice that might help you avoid inconvenient or problem situations.

Attention

These notices indicate potential damage to programs, devices, or data. An attention notice is placed just before the instruction or situation in which damage could occur.

CAUTION

These statements indicate situations that can be potentially hazardous to you. A caution statement is placed just before the description of a potentially hazardous procedure step or situation.

DANGER

These statements indicate situations that can be potentially lethal or extremely hazardous to you. A danger statement is placed just before the description of a potentially lethal or extremely hazardous procedure step or situation.

Summary of Changes

June 2019

This edition of the Toshiba Global Commerce Solutions SurePOS Application Client/Server Environment for 4690 OS documents the following functions for Version 8 Release 1.2:

- Support for EMV Contactless Cashback.
- Support for EMV Contactless PIN Bypass.
- Support for EMV Optimized Quick Chip.
- Support for Certegy.
- Support for Worldpay host SSL communication.
- Support for WIC Forgiven subtype.
- Support for Free Item Coupon Taxability.

September 2018

This edition of the Toshiba Global Commerce Solutions SurePOS Application Client/Server Environment for 4690 OS documents the following functions for Version 8 Release 1.1:

- Support for Cashier EMV Overtender.
- Support for the EMVAUTHMODE franking format variable.

February 2018

This edition of the Toshiba Global Commerce Solutions SurePOS Application Client/Server Environment for 4690 OS documents the following functions for Version 8 Release 1:

- Support for VeriFone Mx915 and Mx925 PCI 4.0 PIN Pads. (Ported back to V7R5 level 252).
- Enhance EMV Performance by modifying default EMV settings.
- Support for EMV Contactless Early Tap. This allows Contactless cards to be tapped prior to tender selection. (Ported back to V7R5 Level 252).
- Support for Large Transaction Amount (B030 message).
- Support for Reboot Notification and Reboot Time. (Ported back to V7R5 Level 252).
- Support WIC processing of department keyed coupons.
- Support for Alternate ID entry at the PIN pad.
- Support for EMV Fraud/Security Prevention.
- Support for Operator Override Logging.
- Support for Multiple TGZ File Loads.
- Support for changing the printer code page.
- Support for allowing/disabling storing scanned data from driver's licenses.
- Support for Datalogic Scale Sentry (license required).
- Support for Whitelisting.
- Remove Amount OK prompt on PIN pad for MSD Contactless transactions (full tenders).
- Support for storing Sales Transaction Receipts up to 7 days and searching/reprinting the receipts (Store Integrator GUI V4R2 Service Pack 1 is required).
- Support for disallowing SAFs of refund tenders based on a tender-specific option.
- Support for rechecking DOB after retrieved or recovered transaction.

March 2017

This edition of the Toshiba Global Commerce Solutions SurePOS Application Client/Server Environment for 4690 OS Programming Reference documents the additional updates for Version 7 Release 5.1.

April 2016

This edition of the Toshiba Global Commerce Solutions SurePOS Application Client/Server Environment for 4690 OS Programming Reference documents the additional updates for Version 7 Release 5.

February 2015

This edition of the Toshiba Global Commerce Solutions SurePOS Application Client/Server Environment for 4690 OS Programming Reference documents the new features for Version 7 Release 5.

Chapter 1. The SurePOS ACE Database

Summary of SurePOS ACE Files

This chapter is an overview of data files that Chapter 2, Descriptions of Files and Records on page 41 describes in detail. `ini` files are described in Chapter 3, .ini Files on page 445.

File Characteristics

Abbreviations used in these data file descriptions include:

C	Current period file copy
C. Archive	Compressed archive
CL.Archive	Compressed archive of long period file
CL	Current period long file
C/P	Current and previous period file copies
CPO/S&L	Current period, previous period, and old period file copies for each short and long file
CS	Current period short file
CS.Archive	Compressed archive of short period file
Direct	Direct file type
File Copies	Number of different file copies kept on a store controller, i.e., current and old copies of a file
Keyed	Keyed file type
LAN Dist	Various LAN file distribution attributes: C/C - Compound at close C/U - Compound per update Local - Local MC - Mirrored at close M/U - Mirrored per update
Max Lth	Maximum record length
O	Old period file copy
OL	Old period long file
OS	Old period short file
P	Previous period file copy
PL	Previous period long file
PS	Previous period short file
Random	Random file type
Seq	Sequential file type
U. Archive	Uncompressed archive
UL.Archive	Uncompressed archive of long period file

US.Archive	Uncompressed archive of short period file
User Data	Whether you are responsible for creating the file or its contents
Var	Variable length

File distribution attributes are for the 4690 OS. They are described in the discussion of LAN file distribution for the Multiple Controller Feature (MCF) in the *4690 System User's Guide*. The Transaction Summary Log (TLog) is distributed per update to reduce the risk of data loss to data from one transaction.

Note: You must configure an alternate controller to receive compound file copies to enable that controller to support terminals on a loop.

You can find other information about file sizes, file types, and distribution types in the *4690 Operating System: Programming Guide*.

Table 1 describes each major data file and its characteristics in alphabetical order.

Application File Directories

Application files are in these directories:

ADX_IPGM	Code files such as descriptors, screens, and options
ADX_IMNT	Maintenance versions of code files
ADX_IDT1	Input data files
ROOT	Work files

Files in ADX_IPGM are of a fixed size that you can change with personalization options and system file utilities. They are not susceptible to constant change. Files in ADX_IDT1 are generally user input files. These files are susceptible to change and might need to be backed up daily. Files in ADX_IDT4 are output files that SurePOS ACE manages. Application work files reside in the root.

Table 1 identifies the directory in which each file resides.

Application Logical File Names

All application files are accessed from SurePOS ACE or the host system using logical names. Logical names allow files to move from disk to disk with a change to the logical name definition and without requiring alterations to SurePOS ACE or the host interface. Logical names overcome the ADCS restriction of six-character names from the host and provide an interface to both maintenance and active copies.

Logical names assigned to application files appear in Table 1 with their host names. The first character of host names appears as *x*, which stands for either *%* or +. A host name that begins with *%* means the file is in bit string format. A host name that begins with + means the file is in standard format.

Files that begin with ACE (or APS) do not have a host name. You can assign host names by assigning a user file name. For example, for the Accounting Scratch Pad file acearspd.dat, you can define eamarspd to have the same definition as the logical name acearspd. Then, you can use %ARSPD to access the acearspd.dat file.

SurePOS ACE Files, Characteristics, and Logical Names

The following table describes characteristics for each SurePOS ACE data file. The description includes physical and logical files names, the number or categories of file copies kept, host names, directory, file access method, file length, LAN distribution characteristics, whether you are responsible for creating or maintaining data in the file, and the number of records in the file.

Table 1. SurePOS ACE Files, Characteristics, and Logical Names

File Name	File Description	File Copies or Categories	Directory	Logical Name	Host Name	File Type	Max Lth	LAN Dist	User Data	Number of Records
ACEARSPD.DAT	"ACEARSPD (Accounting Scratch Pad File)" on page 42	1	ADX_IDT4	ACEARSPD		Direct	47	Local	No	1
ACEAUTHZ.DAT	Contains operator options authorization values for base SurePOS ACE.	1	ADX_IPGM	ACEAUTHZ		Ran-dom	512	Local	Yes	1 per base option
ACEBINFL.DAT	"ACEBINFL (BIN File)" on page 43	1	ADX_IDT1	ACEBINFL		Keyed	Var	C/U	No	1
ACEBINLG.DAT	"ACEBINLG (ACEBIRBL Log File)" on page 50	1	ADX_IDT1	ACEBINLG		Seq	Var	M/U	No	Variable
ACECATE5.DAT	"ACECATE5 (Coupon Translation Options File)" on page 51	1	ADX_IDT1	ACECATE5		Ran-dom	512	M/C	Yes	Approx 20 + 1 per option
ACECFBAS.DAT	"ACECFBAS (Coupon Fraud Basic File)" on page 52	1	ADX_IPGM	ACECFBAS		Keyed	46	C/U	No	1 per basic coupon
ACECFGS1.DAT	"ACECFGS1 (Coupon Fraud GS1 File)" on page 52	1	ADX_IDT1	ACECFGS1		Keyed	72	C/U	No	1 per GS1 coupon
ACECLSCP.DAT	"ACECLSCP (Close Period Checkpoint File)" on page 52	1	ADX_IDT1	ACECLSCP		Direct	12	M/U	No	Variable
ACECLSLG.DAT ACECLSLP.DAT	"ACECLSLG (Close Log)" on page 54	Current Previous	ADX_IDT1 ADX_IDT1	ACECLSLG		Ran-dom		M/C	No	32000 bytes by default
ACECLSPG.DAT	"ACECLSPG (Close Progress Control File)" on page 54	1	ADX_IDT4	ACECLSPG		Ran-dom	18	M/U	No	Approx 16, which is 1 per ledger

File Name	File Description	File Copies or Categories	Directory	Logical Name	Host Name	File Type	Max Lth	LAN Dist	User Data	Number of Records
ACEDDESC.DAT	"ACEDDESC (Department Descriptors File)" on page 74	1	ADX_IPGM	ACEDDESC		Ran-dom	49	C/U	Yes	1 per department descriptor
ACEDPTLD.DAT	"ACEDPTLD (Department Subtotal Descriptors File)" on page 74	1	ADX_IDT1	ACEDPTLD		Direct	22	M/C	Yes	1 per subtotal group
ACEDPTLS.DAT	"ACEDPTLS (Department Lists File)" on page 75	1	ADX_IDT1	ACEDPTLS		Direct	8	C/U	Yes	1 per department in each of 20 subtotal groups
ACEDPTVC.DAT ACEDPTVP.DAT ACEDPTVO.DAT ACEDPVWK.DAT	"ACEDPTV* (Department Variance File)" on page 75	Current Previous Old Weekly	ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4	ACEDPTVC ACEDPTVP ACEDPTVO ACEDPVWK		Keyed	58	M/C	No	2 per department subtotal group
ACEFBPNT.DAT	"ACEFBPNT (Period Points Close Summary File)" on page 78	1	ADX_IDT4	ACEFBPNT		Direct	226	M/U	No	1
ACEFIRLG.DAT	"ACEFIRLG (ACEFIRBL Log File)" on page 79	1	ADX_IDT1	ACEFIRLG		Seq	Var	M/U	No	Variable
ACEIMLST.DAT	"ACEIMLST (Item Movement List Descriptors File)" on page 80	1	ADX_IDT1	ACEIMLST		Keyed	31	M/C	Yes	1 per item movement list
ACEITMLS.DAT	"ACEITMLS (Item Movement Lists File)" on page 81	1	ADX_IDT1	ACEITMLS		Direct	16	M/C	Yes	1 per item code in each item movement list
ACEMSCTL.DAT	"ACEMSCTL (Miscellaneous Transaction Lists File)" on page 81	1	ADX_IDT1	ACEMSCT		Direct	8	C/C	Yes	1 per account
ACEMSTLD.DAT	"ACEMSTLD (Miscellaneous Transaction Subtotal Descriptors File)" on page 82	1	ADX_IDT1	ACEMSTLD		Direct	22	C/C	Yes	1 per subtotal group
ACEMTRKB.DAT	"ACEMTRKB (Matrix Keyboard Options File)" on page 82	1	ADX_IDT1	ACEMTRKB		Direct	9	M/U	No	1 per key stem definition
ACENGSLC.DAT ACENGSLP.DAT ACENGSLO.DAT	"ACENGSL* (Negative Sales File)" on page 83	Current Previous Old	ADX_IDT4 ADX_IDT4 ADX_IDT4	ACENGSLC ACENGSLP ACENGSLO		Direct	64	M/C	No	1 per operator
ACERDESC.DAT ACERDESC.DAT	"ACERDESC (Report Descriptors File)" on page 84	Active Maint.	ADX_IPGM ADX_IMNT	ACERDESC ACERDESM		Ran-dom	49	C/U	Yes	1 per report descriptor

File Name	File Description	File Copies or Categories	Directory	Logical Name	Host Name	File Type	Max Lth	LAN Dist	User Data	Number of Records
ACERXFIL.DAT	“ACERXFIL (Pharmacy File)” on page 84	1	ADX_IDT4	ACERXFIL		Keyed	254	C/U	No	1 per prescription
ACESDEF.DAT	“ACESDEF (Manager's Terminal Defaults File)” on page 86	Active	ADX_IPGM	ACESDEF		N/A	N/A	Local	No	Variable
ACESDESC.DAT ACESDESC.DAT	“ACESDESC (Sales Descriptors File)” on page 86	Active Maint.	ADX_IPGM ADX_IMNT	ACESDESC ACESDESM		Ran-dom	49	C/U	Yes	1 per sales descriptor
ACESRCPC.DAT	“ACESRCPC (Suspend/Retrieve Checkpoint File)” on page 86	1	ADX_IDT4	ACESRCPC		Direct	5	M/C	No	2
ACESRCPC.0*	“ACESRCPC (Suspend/Retrieve Checkpoint File)” on page 86	1	ADX_IDT4	LCLSRCPC		Direct	5	Local	No	2
ACETIRLG.DAT	“ACETIRLG (ACETIRBL Log File)” on page 87	1	ADX_IDT1	ACETIRLG		Seq	Var	M/U	No	Variable
ACETOFSF.DAT	“ACETOFSF (Terminal Offline Status File)” on page 88	1	ADX_IDT4	ACETOFSF		Keyed	20	M/C	No	1 per terminal + 1
ACETSVER.DAT	“ACETSVER (ACE Terminal Sales Version File)” on page 89	1	ADX_IDT1	ACETSVER		Keyed	72	C/U	Yes	1 per terminal
ACEUCCTL.DAT	“ACEUCCTL (Unattended Close Control File)” on page 90	1	ADX_IDT1	ACEUCCTL		Keyed	48	M/C	No	1 per unattended close procedure
ACEVCODE.DAT	“ACEVCODE (Full-Screen Velocity Code Help File)” on page 91	1	ADX_IDT1	ACEVCODE		Seq	81	M/U	No	2 per velocity code definition
ACEWERCL.DAT	“ACEWERCL.DAT (WIC EBT Claim/Reconciliation Status File)” on page 93	1	ADX_IDT4	ACEWERCL		Keyed	35	M/C	No	1 per WIC EBT Claim File
ACEWE??.APL	“ACEWE??.APL (WIC EBT Approved Product List File)” on page 95	1 per WIC EBT agency	ADX_IDT1	None		Keyed	39	C/C	No	1 per UPC
ACEWE??.DES	“ACEWE??.DES (WIC EBT Description File)” on page 96	1 per WIC EBT agency	ADX_IDT1	None		Keyed	46	C/C	No	1 per UPC
ACEWE??.HCL	“ACEWE??.HCL (WIC EBT Hot Card List File)” on page 97	1 per WIC EBT agency	ADX_IDT1	None		Keyed	22	C/C	No	1 per hot card
APSAUTHZ.DAT	Contains options authorizations for SurePOS ACE EPS.	1	ADX_IPGM	APSAUTHZ		Ran-dom	512	Local	Yes	1 per SurePOS ACE EPS option

File Name	File Description	File Copies or Categories	Directory	Logical Name	Host Name	File Type	Max Lth	LAN Dist	User Data	Number of Records
APSOPTNS.DAT	"APSOPTNS (SurePOS ACE EPS Personalization Options File)" on page 93	1	ADX_IPGM	APSOPTNS		Ran-dom	512	M/C	Yes	Approx 20 + 1 per option
CPNAUTHZ.DAT	Contains options authorizations for coupon translation.	1	ADX_IPGM	CPNAUTHZ		Ran-dom	512	Local	Yes	1 per coupon translation option
EAMACCTC.DAT EAMACCTP.DAT EAMACCTO.DAT EAMACCCL.DAT EAMACCPL.DAT EAMACCOL.DAT	"EAMACC* (Accounting Totals File)" on page 98	CS PS OS CL PL OL	ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4	EAMACCTC EAMACCTP EAMACCTO EAMACCCL EAMACCPL EAMACCOL	xACCTC xACCTP xACCTO xACCCL xACCPL xACCOL	Keyed	512	M/C	No	5 per operator using operator accountability + 5 per terminal using terminal accountability + 5 per store
EAMBATCH.DAT	"EAMBATCH (Batch Maintenance File)" on page 119	2	ADX_IDT1	EAMBATCH	xBATCH	Seq	Var	M/C	Yes	Var
EAMCOUPC.DAT EAMCOUPO.DAT	"EAMCOUP* (Preferred Customer Coupon Tracking File)" on page 126	Current Old	ADX_IDT4 ADX_IDT4	EAMCOUPC EAMCOUPO	xCOUPC xCOUPO	Seq	Var	M/U	No	1 per tracked coupon
EAMCPIMG.DAT	"EAMCPIMG (Check Image Checkpoint File for Close Processing)" on page 130	1	ADX_IDT4	EAMCPIMG		Direct	12	M/U	No	1
EAMCSCF1	"EAMCSCF1 (Checkout Support Control File)" on page 130	1	ROOT	EAMCSCF1	xCSCF1	Keyed	28	M/C	No	1 per terminal + 2
EAMCUSTA.DAT EAMCUSTO.DAT	"EAMCUST* (Customer Account Status File)" on page 133	Current Old	ADX_IDT4 ADX_IDT4	EAMCUSTA EAMCUSTO	xCUSTA xCUSTO	Keyed	28	M/C	No	1 per used customer account
EAMCXLAT	"EAMCXLAT (Customer Translation File)" on page 135	1	ADX_IDT1	EAMCXLAT	xCXLAT	Keyed	12	M/U	No	1 per customer ID per customer account
EAMDDnnn	"EAMDD* (Delayed Data Maintenance Data File)" on page 136	Up to 999	ADX_IDT1	EAMD:nnn	xD:nnn	Seq	var	M/C	Yes	1 per record changed
EAMDEPTC.DAT EAMDEPTP.DAT	"EAMDEP* (Department Totals File)" on page 137	CS PS	ADX_IDT4 ADX_IDT4	EAMDEPTC EAMDEPTP	xDEPTC	Direct	512	M/C	No	1 per 12 departments

File Name	File Description	File Copies or Categories	Directory	Logical Name	Host Name	File Type	Max Lth	LAN Dist	User Data	Number of Records
EAMDEPTO.DAT EAMDEPCL.DAT EAMDEPPL.DAT EAMDEPOL.DAT		OS CL PL OL	ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4	EAMDEPTO EAMDEPCL EAMDEPPL EAMDEPOL	xDEPTP xDEPTO xDEPCL xDEPPL xDEPOL					
EAMDMCTL.DAT	"EAMDMCTL (Delayed Data Maintenance Control File)" on page 145	1	ADX_IDT1	EAMDMCTL	xDMCTL	Keyed	48	M/U	Yes	1 per delayed maint. batch
EAMDMERR.DAT	"EAMDMERR (Delayed Data Maintenance Error File)" on page 151	1	ADX_IDT4	EAMDMERR	xDMERR	Seq	var	M/U	No	1 per delayed maint error
EAMDRnnn	Delayed Data Maintenance Report	1	ADX_IDT4	EAMR:nnn	xR:nnn	Seq	var	M/C	No	Varies per report
EAMEXCPT.DAT EAMEXCPO.DAT EAME*yymd*.DB*n* EAME*yymd*.DC*n*	"EAMEXCP* (Exception Log File)" on page 154	Current Old U. Archive C. Archive	ADX_IDT4 ADX_IDT4 optional optional	EAMEXCPT EAMEXCPO	xEXCPT xEXCPO	Seq	var	M/U	No	1 per exception entry
EAMFBACT.DAT	"EAMFBACT (Preferred Customer Activity File)" on page 206	1	ADX_IDT1	EAMFBACT	xFBACT	Keyed	254	C/U	Yes	1 per preferred customer
EAMFBAPP.DAT	"EAMFBAPP (Preferred Customer Previous Period Activity File)" on page 212	1	ADX_IDT4	EAMFBAPP	xFBAPP	Keyed	36	M/C	No	1 per preferred customer per (monthly) activity period
EAMFBAUD.DAT EAMFBAUO.DAT	"EAMFBAU* (Preferred Customer Audit Log)" on page 213	1	ADX_IDT4 ADX_IDT4	EAMFBAUD EAMFBAUO	xFBAUD xFBAUO	Seq	40	M/C	No	1 per active customer
EAMFBCHG.DAT	"EAMFBCHG (Preferred Customer Enrollment Change Log)" on page 215	1	ADX_IDT4	EAMFBCHG	xFBCHG	Seq	254	M/C	No	1 per changed enrollment
EAMFBCUS.DAT	"EAMFBCUS (Preferred Customer Enrollment File)" on page 217	1	ADX_IDT1	EAMFBCUS	xFBCUS	Keyed	254	M/C	Yes	1 per customer

File Name	File Description	File Copies or Categories	Directory	Logical Name	Host Name	File Type	Max Lth	LAN Dist	User Data	Number of Records
EAMFBFLT.DAT EAMFBFL1.DAT EAMFBFL2.DAT EAMFBFL3.DAT EAMFBFL4.DAT EAMFBFL5.DAT EAMFBFL6.DAT EAMFBFL7.DAT EAMFBFL8.DAT EAMFBFL9.DAT	"EAMFBFLT (Preferred Customer Panel Filter File)" on page 219	1 1 1 1 1 1 1 1 1 1	ADX_IDT1 ADX_IDT1 ADX_IDT1 ADX_IDT1 ADX_IDT1 ADX_IDT1 ADX_IDT1 ADX_IDT1 ADX_IDT1 ADX_IDT1	EAMFBFLT EAMFBFL1 EAMFBFL2 EAMFBFL3 EAMFBFL4 EAMFBFL5 EAMFBFL6 EAMFBFL7 EAMFBFL8 EAMFBFL9	xFBFLT xFBFL1 xFBFL2 xFBFL3 xFBFL4 xFBFL5 xFBFL6 xFBFL7 xFBFL8 xFBFL9	Seq	var	C/U	Yes	Varies per records filtered
EAMFBTCM.DAT	"EAMFBTCM (Preferred Customer Targeted Coupon Messages File)" on page 225	1	ADX_IDT1	EAMFBTCM	xFBTCM	Keyed	126	C/U	Yes	1 per targeted coupon
EAMFBXFR.DAT	"EAMFBXFR (Preferred Customer Transfer File)" on page 226	1	ADX_IDT1	EAMFBXFR	xFBXFR	Keyed	36	C/U	No	1 per transfer
EAMF@nnn EAMF@nnn EAMF$nnn EAMF$nnn	Terminal Input Format	Data Data Maint. Symbols Sym. Maint.	ADX_IPGM ADX_IMNT ADX_IPGM ADX_IMNT	EAMF: EAMG: EAMH: EAMI:	xF:nnn xG:nnn xH:nnn xI:nnn					Controlled by operating system utility
EAMHDPTC.DAT EAMHDPTP.DAT	"EAMHDPT* (Hourly Department Totals File)" on page 227	Current Previous	ADX_IDT4 ADX_IDT4	EAMHDPTC EAMHDPTP	xHDPTC xHDPTP	Seq	var	M/C	No	1 per time interval
EAMIMGC.DAT EAMIMGH.DAT	"EAMIMG* (Check Image File)" on page 228	Current Previous (Host)	ADX_IDT4	EAMIMGC EAMIMGH		Seq	var	M/U M/C	Yes	Var
EAMIMOVE.DAT EAMIMOVO.DAT EAMIMOLC.DAT EAMIMOLP.DAT EAMK*yymd*.WU*n* EAMK*yymd*.DU*n* EAMK*yymd*.WC*n* EAMK*yymd*.DC*n*	"EAMIMO* (Item Movement Totals File)" on page 230	CS PS CL PL UL.Archive US.Archive CS.Archive CL.Archive	ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4 Optional Optional Optional Optional	EAMIMOVE EAMIMOVO EAMIMOLC EAMIMOLP	xIMOVE xIMOVO xIMOLC xIMOLP	Keyed	22	M/C	No	1 per item tracked if sold

File Name	File Description	File Copies or Categories	Directory	Logical Name	Host Name	File Type	Max Lth	LAN Dist	User Data	Number of Records
EAMIRCHG.DAT	"EAMIRCHG (Item Record Change File)" on page 233	1	ADX_IDT4	EAMIRCHG	xIRCHG	Seq	var	M/U	No	1 per changed item record
EAMITEMR.DAT TRMITEMR.DAT TRMITEMB.DAT TRMITEMP.DAT WRKITEMR.DAT WRKITEMZ.ZIP	"Basic Item Record File Layouts" on page 237	1 Act. Term. BU Term. Pend Term. Work Term. Work Term. ZIP	ADX_IDT1	EAMITEMR TRMITEMR TRMITEMB TRMITEMP WRKITEMR WRKITEMZ	xITEMR	Keyed	46 - 512	C/U C/U C/U C/U Local	Yes	1 per item on file
EAMM@nnn EAMM@nnn EAMM$nnn EAMM$nnn	Terminal Input Modulo Table	Data Data Maint. Symbols Sym. Maint.	ADX_IPGM ADX_IMNT ADX_IPGM ADX_IMNT	EAMM: EAMN: EAMY: EAMZ:	xQ:nnn xA:nnn xU:nnn xV:nnn					Controlled by operating system utility
EAMMAINT.DAT	"EAMMAINT (ADDMI Maintenance File)" on page 300	1	ADX_IDT1	EAMMAINT	xMAINT	Seq	Var	M/C	Yes	1 per record changed + 1 per batch
EAMMTRNC.DAT EAMMTRNP.DAT EAMMTRNO.DAT EAMMTRCL.DAT EAMMTRPL.DAT EAMMTROL.DAT	"EAMMTR* (Miscellaneous Transactions File)" on page 310	CS PS OS CL PL OL	ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4	EAMMTRNC EAMMTRNP EAMMTRNO EAMMTRCL EAMMTRPL EAMMTROL	xMTRNC xMTRNP xMTRNO xMTRCL xMTRPL xMTROL	Keyed	36	M/C	Yes	1 per misc account defined
EAMOPnnn	"EAMOPnnn (Terminal Options File)" on page 316	1 per terminal	ADX_IPGM	EAMO:nnn	xP:nnn	Ran-dom	512	C/U	Yes	Approx 20 + 1 per unique option
EAMOPGnn	"EAMOPGnn (Terminal Group Options File)" on page 316)	0-99 per store	ADX_IPGM	EAMOG:nn		Ran-dom	512	C/U	Yes	Approx 20 + 1 per unique option

File Name	File Description	File Copies or Categories	Directory	Logical Name	Host Name	File Type	Max Lth	LAN Dist	User Data	Number of Records
EAMOPERA.DAT	"EAMOPERA (Operator Authorization File)" on page 311	1	ADX_IDT1	EAMOPERA	xOPERA	Keyed	84	C/U	Yes	1 per operator
EAMOPTNS.DAT	"EAMOPTNS (Base Personalization Options File)" on page 317	1	ADX_IPGM	EAMOPTNS	xOPTNS	Ran-dom	512	C/U	Yes	1 per option
EAMOSTAT.DAT	"EAMOSTAT (Operator Status File)" on page 319	1	ADX_IDT4	EAMOSTAT		Keyed	56	M/U	No	2 per operator (set in options)
EAMPANEL.DAT EAMPANL1.DAT EAMPANL2.DAT EAMPANL3.DAT EAMPANL4.DAT EAMPANL5.DAT EAMPANL6.DAT EAMPANL7.DAT EAMPANL8.DAT EAMPANL9.DAT	"EAMPAN* (Preferred Customer Panel Diary File)" on page 320	1 1 1 1 1 1 1 1 1 1	ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4	EAMPANEL EAMPANL1 EAMPANL2 EAMPANL3 EAMPANL4 EAMPANL5 EAMPANL6 EAMPANL7 EAMPANL8 EAMPANL9	xPANEL xPANL1 xPANL2 xPANL3 xPANL4 xPANL5 xPANL6 xPANL7 xPANL8 xPANL9	Seq	var	M/C	No	Varies per TLog strings filtered
EAMPERFC.DAT EAMPERFP.DAT EAMPERFO.DAT EAMPERCL.DAT EAMPERPL.DAT EAMPEROL.DAT	"EAMPER* (Operator Performance File)" on page 333	CS PS OS CL PL OL	ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4	EAMPERFC EAMPERFP EAMPERFO EAMPERCL EAMPERPL EAMPEROL	xPERFC xPERFP xPERFO xPERCL xPERPL xPEROL	Keyed	512	M/C	No	2 per active operator
EAMS@nnn EAMS@nnn EAMS$nnn EAMS$nnn	Terminal Input Sequence Table	Data Data Maint. Symbols	ADX_IPGM ADX_IMNT ADX_IPGM	EAMQ:nnn EAMA:nnn EAMU:nnn EAMV:nnn	xQ:nnn xA:nnn xU:nnn xV:nnn					Controlled by operating system utility

File Name	File Description	File Copies or Categories	Directory	Logical Name	Host Name	File Type	Max Lth	LAN Dist	User Data	Number of Records
		Sym. Maint.	ADX_IMNT							
EAMSRK*.0*	"EAMSRKY* (Suspend/Retrieve Keyed Index File)" on page 343	CS PS CL PL	ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4	LCLSRKYC LCLSRKYP LCLSRKLC LCLSRKLP	xSRKYC xSRKYP xSRKLC xSRKLP	Keyed	45	Local	No	1 for store suspend totals + 1 for store retrieve totals + 1 per terminal for terminal suspend totals + 1 per terminal for terminal retrieve totals + 1 per operator for operator suspend totals + 1 per operator for operator retrieve totals + 1 per suspended transaction
EAMSRTR*	"EAMSRTR* (Suspended Transactions File)" on page 345	1	ADX_IDT4	LCLSRTRX LCLSRTRP	xSRTRX	Seq	var	Local	No	1 per suspended transaction
EAMSRKYC.DAT EAMSRKYP.DAT EAMSRKLC.DAT EAMSRKLP.DAT	"EAMSRKY* (Suspend/Retrieve Keyed Index File)" on page 343	CS PS CL PL	ADX_IDT4 ADX_IDT4 ADX_IDT4 ADX_IDT4	EAMSRKYC EAMSRKYP EAMSRKLC EAMSRKLP	xSRKYC xSRKYP xSRKLC xSRKLP	Keyed	45	M/C	No	1 for store suspend totals + 1 for store retrieve totals + 1 per terminal for terminal suspend totals + 1 per terminal for terminal retrieve totals + 1 per operator for operator suspend totals + 1 per operator for operator retrieve totals + 1 per suspended transaction
EAMSRTRX.DAT	"EAMSRTR* (Suspended Transactions File)" on page 345	1	ADX_IDT4	EAMSRTRX	xSRTRX	Seq	var	M/C	No	1 per suspended transaction
EAMTAXTn.DAT	"EAMTAXT* (Tax Table File)" on page 346	Active Maint.	ADX_IPGM	EAMTAX:n	xTAX:n	Seq	var	C/U	Yes	1 per tax bracket
EAMTENDV.DAT	"EAMTENDV (Tender Verification File)" on page 348	1	ADX_IDT1	EAMTENDV	xTENDV	Keyed	14/26	C/U	Yes	1 per tender type per account
EAMTERMS.DAT	"EAMTERMS (Terminal Status File)" on page 350	1	ADX_IDT4	EAMTERMS	xTERMS	Keyed	72	M/C	No	2 per terminal + 1
EAMTLIST.DAT EAMTLISP.DAT EAMTLISO.DAT	"EAMTLIS* (Tender Listing File)" on page 356	Current Previou	ADX_IDT4 ADX_IDT4 ADX_IDT4	EAMTLIST EAMTLISP EAMTLISO	xTLIST xTLISP xTLISO	Seq	var	M/C	No	1 per tender + 1 per group + 2 (set in options)

File Name	File Description	File Copies or Categories	Directory	Logical Name	Host Name	File Type	Max Lth	LAN Dist	User Data	Number of Records
		s Old								
EAMTMnnn.DAT	"EAMTMnnn (Terminal Monitor File)" on page 359	1	ROOT	EAMW:nnn	xW:nnn	Ran-dom	60	Local	No	2 per monitored item sale; 40 records maximum
EAMTPROD.DAT EAMTPROO.DAT	"EAMTPRO* (Terminal Productivity File)" on page 359	Current Old	ADX_IDT4 ADX_IDT4	EAMTPROD EAMTPROO	xTPROD xTPROO	Seq	var	M/C	No	1 per terminal per hour
EAMTRAK.DAT EAMTRAKO.DAT	"EAMTRAK* (Transaction Tracking File)" on page 361	Current Old	ADX_IDT4 ADX_IDT4	EAMTRAK EAMTRAKO		Keyed	5	M/U	No	1000 per terminal (set in options)
EAMTRANA.DAT EAMTRANB.DAT EAMTRANC.DAT EAMJ*yymd*.DB*n* EAMJ*yymd*.DC*n*	"EAMTRAN* (Transaction Summary Log)" on page 361	C/P C/P C/P U. Archive C. Archive	ADX_IDT4 ADX_IDT4 ADX_IDT4 optional optional	EAMTRANA EAMTRANB EAMTRANC	xTRANA xTRANB xTRANC	Seq	var	M/U	No	1 per transaction
EAMTRANA.IDX EAMTRANB.IDX EAMTRANC.IDX EAMJ*yymd*.DD*n* EAMJ*yymd*.DF*n*	Electronic Journal Index file	1 1 1 U. Archive C. Archive	ADX_IDT4 ADX_IDT4 ADX_IDT4 optional optional	EAMMDXA EAMMDXB EAMMDXC	xMDXA xMDXB xMDXC	Direct	64	M/U	No	1 per indexed transaction
EAMTRANA.MDX EAMTRANB.MDX EAMTRANC.MDX EAMJ*yymd*.DE*n* EAMJ*yymd*.DG*n*	Manager's Journal Index file	1 1 1 U. Archive C. Archive	ADX_IDT4 ADX_IDT4 ADX_IDT4 optional optional	EAMMIDXA EAMMIDXB EAMMIDXC	xIDXA xIDXB xIDXC	Direct	64	M/U	No	1 per indexed transaction
EAMTSnnn.DAT	"EAMTSnnn (Totals Save File)" on page 435	1	ROOT	EAMX:nnn	xX:nnn	Seq	var	Local	No	variable
EAMXssss.DAT	"EAMX* (Preferred Customer Activity Transfer File)" on page 436	1	ADX_IDT4	EAM:	xEAMX	Seq	32	M/C	No	1 per transfer of customer's activity

File Name	File Description	File Copies or Categories	Directory	Logical Name	Host Name	File Type	Max Lth	LAN Dist	User Data	Number of Records
NLSAUTHZ.DAT	Options authorizations for NLS options	1	ADX_IPGM	NLSAUTHZ		Ran-dom	512	C/U	Yes	1 per NLS option
NLSOPTNS.DAT	“NLSOPTNS (NLS Personalization Options File)” on page 438	1	ADX_IPGM	NLSOPTNS		Ran-dom	512	C/U	Yes	Approx 20 + 1 per NLS option
SAPATH.DDF	Data dictionary file	1	ADX_IPGM	N/A		Ran-dom	var	C/C	No	variable
RmddIDX	Reprint Sales Receipt Index file	1	EJ\R			Direct	64	M/U	No	1 per sales transaction for all terminals
Rmddttt	Reprint Sales Receipt file	1	EJ/R			Seq	var	M/U	No	1 per sales transaction for this terminal
SAPATH.RVT	Source file for `SAPATH.DDF`	1	ADX_IPGM	N/A		Ran-dom	var	C/C	No	variable
SGNATURE.DAT SGNATURP.DAT SGNATURT.DAT	“SGNATURE (Signature Data File)” on page 441	1 1 1	ADX_IDT4 ADX_IDT4 ADX_IDT4	SGNATURE SGNATURP SGNATURT		Seq	var	M/C	No	variable

User File Responsibilities

You are responsible for maintaining data in several SurePOS ACE files. Some of those files are shipped with default data. Other files do not have default data, and you must create the data for them through one of the SurePOS ACE user interfaces or command interfaces.

Data You Create

Data does not exist in these files until you create it:

acecate5.dat

You can create data for the Coupon Translation file through the Translation pull-down of the Personalization menu.

acedptld.dat

You can create data for the Department Subtotal Descriptors file through the Reporting List Editor in Misc personalization.

acedptls.dat

You can create data for the Department Lists file through the Reporting List Editor in Misc personalization.

aceimlst.dat

You can create data for the Item Movement Lists Descriptors file through the Reporting List Editor or through the Selective Item Report (Item Reporting and Maintenance Workbench) in Manager Procedures. Once created, you can maintain the data through the Reporting List Editor

aceitmls.dat

You can create data for the Item Movement Lists file through the Reporting List Editor or through the Selective Item Report (Item Reporting and Maintenance Workbench) in Manager Procedures. Once created, you can maintain the data through the Reporting List Editor

acemsctl.dat

You can create data for the Miscellaneous Transaction Lists file through the Reporting List Editor.

acemstld.dat

You can create data for the Miscellaneous Transaction Subtotal Descriptors file through the Reporting List Editor.

acewe??.apl

You create the WIC EBT Approved Product List file that SurePOS ACE uses by running ACEWERBL with a UPC/PLU List (Approved Product List) from a WIC EBT agency as the input file.

acewe??.hcl

You create the WIC EBT Hot Card List file that SurePOS ACE uses by running ACEWERBL with a Hot Card List from a WIC EBT agency as the input file.

ASCII BIN file

You are responsible for entering the ASCII data, which is input to ACEBIRBL, in the format specified in "Input File" on page 601

eamfbcus.dat

You can create data for the Preferred Customer Enrollment file through Customer Data Maintenance or the Selective Item Report (Customer Reporting and Maintenance Workbench) in Manager Procedures.

eamfbtcm.dat

You can create data for the Preferred Customer Targeted Coupon Messages file through Targeted Coupon Messages Data Maintenance.

eammtr*.dat

You can create data for the Miscellaneous Transactions Accounts file through the Accounting interface.

eamopnnn

You can create Personalization Options files for specific terminals through Personalization.

eamtendv.dat

You can create data for the Tender Verification file through the `Tender Verification Data Maintenance` or `Delayed Data Maintenance`.

Files with Default Data

You can change default data in these files:

- aceauthz.dat, the base Operator Options Authorization file. You can maintain data in this file through the Personalization user interface with the F9 key or through the Personalization command line interface.
- aceddesc.dat, the Department Descriptors file. You can maintain this file through the Descriptors Editor.
- apsoptns.dat, eamoptns.dat, and nlsoptns.dat, the Personalization Options files. You can maintain this data through Personalization.
- apsauthz.dat, the SurePOS ACE EPS Operator Options Authorization file. You can maintain this data through the Personalization user interface with the F9 key or through the Personalization command line interface.
- cpnauthz.dat, the Coupon Translation Operator Options Authorization file. You can maintain data in this file through the Personalization user interface with the F9 key or through the Personalization command line interface.
- eamfbact.dat, the Preferred Customer Activity file, which SurePOS ACE creates when an enrolled customer is involved in a sales transaction. You can maintain this file through Customer Data Maintenance or the Selective Customer Report (Customer Reporting and Maintenance Workbench).
- eamfbflt.dat, the Preferred Customer Panel Filter file. You create filter files to record only selected data in Preferred Customer Panel Diary files. You can modify this file or create other files.
- eamitemr.dat, the Item Record file. You can maintain this file through the Selective Item Report (Item Reporting and Maintenance Workbench), Item Data Maintenance, or Delayed Data Maintenance.
- eamopera.dat, the Operator Authorization file. You can maintain this file through Operator Authorization Records Data Maintenance or Delayed Data Maintenance.
- eamtaxtn.dat, the Tax Tables. You can maintain this data through the Tax Plans entry of the Miscellaneous pull-down in Personalization.
- nlsauthz.dat, the NLS Operator Options Authorization file. You can maintain data in this file through the Personalization user interface with the F9 key or through the Personalization command line interface.

Defaults for each of the following files are shipped with the product. You can change data in these files although it is possible that their default values might meet most of your needs.

- acerdesc.dat, the Report Descriptor file. You can maintain data in this file through the Descriptors Editor in personalization or with the ACECNVDL module.
- acesdesc.dat, the Sales Descriptor file. You can maintain data in this file through the Descriptors Editor in personalization or with the ACECNVDL module.
- eamf@nnn, the Terminal Input Label Format file. You can change data in this file through selection 3 of the 4690 OS Installation and Update Aids menu.
- eamm@nnn, the Terminal Input Modulo Check file. You can change data in this file through selection 3 of the Installation and Update Aids menu.

- eams@nnn, the Terminal Input Sequence Table. You can change data in this file through selection 3 of the Installation and Update Aids menu.

Other files are created and maintained as needed by SurePOS ACE. However, you might consider the creation and maintenance of the eamddnnn Delayed Data Maintenance files and the eammaint.dat ADDMI file as a host system responsibility if your delayed data maintenance is generated at the host. It is also possible to create these files through the Delayed Data Maintenance interface or the Selective Item Report (Item Reporting and Maintenance Workbench) at the store.

You can also add or delete records in eamdmctl.dat, the Delayed Data Maintenance control file, from the host.

Managing the Close of the Reporting Period

SurePOS ACE allows ten closing procedures, four of which are defined for you by default. You can define new procedures or change defined procedures in Miscellaneous -> Close Period personalization.

The four defined closing procedures are described in the following table:

Table 2. Default Close Procedures

		Files	
ID	Name	Closed by Default	Optionally Closed
1	Daily Close (Short Period)	• Exception Log • Reporting Period Short files: • Accounting Totals Short • Department Totals Short • Miscellaneous Transaction Short • Operator Performance Short • Item Movement Short • Check Image Capture file • WIC EBT files for active agencies	• Tender Listing • Item Movement Long • Terminal Productivity • Customer Account Status • Reserved 1-2 • User 1-2
2	Weekly Close (Long Period)	• Exception Log • Tender Listing • Terminal Productivity • Customer Account Status • Reporting Period files:	• Reserved 1-2 • User 1-2

ID	Name	Files	
		Closed by Default	Optionally Closed
		<ul><li><ul><li>Accounting Totals Long/ Short</li><li>Department Totals Long/ Short</li><li>Miscellaneous Transaction Long/Short</li><li>Operator Performance Long/Short</li><li>Item Movement Short and Long</li></ul></li><li>Check Image Capture file</li><li>WIC EBT files for active agencies</li></ul>	
3	Department Short	Department Totals Short file	• Exception Log • Tender Listing • Item Movement Short • Terminal Productivity • Customer Account Status • Item Movement Long/Short • Reserved 1-2 • User 1-2
4	Single File	Item Movement Short	• Exception Log • Tender Listing • Terminal Productivity • Customer Account Status • Item Movement Long/Short • Reserved 1-2 • User 1-2

A close of the reporting period or a close of the Department Totals Short file causes a close of the Transaction Summary Log (TLog). The ID of the close procedure used to close the period is recorded in TLog string X'21'.

A close of the reporting period or a close of the Department Totals Short file causes a close of the Signature Data File.

A close of the reporting period causes a close of the till status totals used in the Cash Drawer Position Report.

A close of the reporting period causes a close of the Transaction Tracking file.

A close of the reporting period or a close of the Accounting Totals file causes a close of the Suspend Retrieve Keyed file.

The processing to support a close of the TLog or till status totals occurs in the Close Reporting Totals function of the Accounting application. The processing to support the close of all other files occurs in Checkout Support.

Chapter 2. Descriptions of Files and Records

This chapter describes the various files and records used by SurePOS ACE. Field types are identified by the following abbreviations:

Array	A series of fields of the same data type
ASCII	ASCII data
Bit	The value of one flag bit
Byte	Usually bit flag values
Charr	Single character byte
INT	Integer
Long	Long integer
Mix	Field content is dependent on a field from a record in another file and can contain data of various types. That file is identified.
PD	Packed decimal data
PI	Packed integer
PL	Packed long integer
PS	Packed string
String	Array of data type char
Structure	A combination of fields that can be various data types. See description below.
Union	A specific length at a specific offset that is used alternatively by one or more fields. See description below.

Certain files contain variable length records. The variable length fields in those records are identified in the following ways:

vXX	Gives the maximum size of the field. A length value is substituted for XX.
v?	Means that the length of the field is variable or user-defined.
vrbl	Means that field size is dependent on a field from a record in another file. That file is identified.

Unions and Structures

SurePOS ACE provides an SQL interface to the database, for either viewing the data or changing it. The 32-bit Windows® version, which is named `gconf32.exe`, uses the `sapath.rvt` file to map the database into an SQL data dictionary format. Using `sapath.rvt` as source, it creates an `sapath.ddf` data mapping.

Some of the fields that `sapath.rvt` maps are used in more than one way, depending on such things as pricing methods, personalization options, and whether you have enabled the Loyalty program. When a field has more than one possible use, it is mapped as a *union*. Unions do not define any data mapping themselves, but rather group one or more data definitions that use the same offset and length. Each of the multiple definitions in a union is a single type that maps the whole length of the union.

A union might be composed of a *structure*, which can be subdivided into any number of types and lengths.

ACEARSPD (Accounting Scratch Pad File)

The Accounting Scratch Pad file contains running totals for pickups and tender counts.

Logical name	`<ACEARSPD>`
Data object reference	*ISSAScratchPadData* <`SASPDDAT.CPP`>
Organization	Direct
Distribution class	Local
File copies	1
Record length	47 bytes

Field Name	Type	Length	Decimal Offset	Description
LastCleared	ASCII	14	0	Date that this file was last cleared
TotalOperators	ASCII	1	14	Total number of operators counted.
PadTotals()	Int	32	15	An array of eight 4-byte work areas for storing pickup or tender count totals: **PadTotals(1)** Cash **PadTotals(2)** Checks **PadTotals(3)** Food stamps **PadTotals(4)** Miscellaneous tender 1 **PadTotals(5)** Miscellaneous tender 2 **PadTotals(6)** Miscellaneous tender 3 **PadTotals(7)** Manufacturer coupons **PadTotals(8)** Store coupons

ACEBINFL (BIN File)

The BIN file, which is generated by the ACEBIRBL application, contains these types of records:

Data	Contains BIN data taken from the input file for the ACEBIRBL application.
Reserved	Created in the file to improve performance by reducing the number of file reads

Logical name	`<ACEBINFL>`
Data object reference	*ISEPSOfflineBINStorage* <`EPSBNSTR.CPP`>
Organization	Keyed
Distribution class	Compound on update
File copies	Maximum number of files specified by LOGNAMES variable in `acebinlg.ini` file (default files are current and previous)
Record length	Variable
Key length	21 bytes (BIN number and PAN length)

Data Record

Note: BIN File format version 02 or later must be used for FSA support. EMV tenders support FSA and do NOT support PIN encouragement or on-line WIC.

Table 3. Data Record for ACEBINFL Version 01

Field Name	Type	Length	Decimal Offset	Description
BIN	ASCII	20	0	Decimal BIN number, which is left-justified and padded with spaces
PAN Length	Int	1	20	Length of Personal Account Number (PAN)
Card ID	ASCII	2	21	Card ID for the record. These are the valid values: **1 - 20** Business Partner **21 - 40** Customer All other values are reserved.
User Reserved	ASCII	v200	23	Reserved for user data

Table 4. Data Record for ACEBINFL Version 02

Field Name	Type	Length	Decimal Offset	Description
BIN	ASCII	20	0	Decimal BIN number, which is left-justified and padded with spaces
PAN Length	Int	1	20	Length of Personal Account Number (PAN)

Field Name	Type	Length	Decimal Offset	Description
Processing Flag	Int	2	21	**X'0001'** Partial authorization allowed **X'0002'** PIN Encouragement allowed **X'0010'** FSA allowed
User Flag	Int	2	23	Field for user flag.
Toshiba Reserved	ASCII	9	25	Reserved for Toshiba use.
User Reserved	ASCII	v200	34	Reserved for user data

Table 5. Data Record for ACEBINFL Version 03

Field Name	Type	Length	Decimal Offset	Description
BIN	ASCII	20	0	Decimal BIN number, which is left-justified and padded with spaces
PAN Length	Int	1	20	Length of Personal Account Number (PAN)
Processing Flag	Int	2	21	**X'0001'** Partial authorization allowed **X'0002'** PIN Encouragement allowed **X'0004'** Online WIC **X'0010'** FSA allowed
User Flag	Int	2	23	Field for user flag
WIC Agency ID	ASCII	2	25	WIC Agency ID
Toshiba Reserved	ASCII	7	27	Reserved for Toshiba use
User Reserved	ASCII	v200	34	Reserved for user data

Table 6. Data Record for ACEBINFL Version 04

Field Name	Type	Length	Decimal Offset	Description
BIN	ASCII	20	0	Decimal BIN number, which is left-justified and padded with spaces
PAN Length	Int	1	20	Length of Personal Account Number (PAN)

Field Name	Type	Length	Decimal Offset	Description
Processing Flag	Int	2	21	**X'0001'** Partial authorization allowed **X'0002'** PIN Encouragement allowed **X'0004'** Online WIC **X'0010'** FSA allowed
User Flag	Int	2	23	Field for user flag
WIC Agency ID/Card Plan ID	ASCII	2	25	WIC Agency ID when the Processing Plan is set to X'0004'; otherwise, it is the Card Plan ID.
Network ID	ASCII	2	27	Network
Toshiba Reserved	ASCII	5	29	Reserved for Toshiba use
User Reserved	ASCII	v200	34	Reserved for user data

Reserved Record

Table 7. Reserved Record for ACEBINFL

Field Name	Type	Length	Decimal Offset	Description
BIN	ASCII	20	0	Always `99999999999999999999`
PAN Length	Int	1	20	Decimal 99 (X'63')
Min	Int	1	21	Number of digits in the smallest BIN number in the file
Max	Int	1	22	Number of digits in the largest BIN number in the file
Create Date/Time	PD	7	23	Date and time created. Format is `YYYYMMDDhhmmss`.
File Format Version	ASCII	2	30	Toshiba file format version
User Format Version	ASCII	2	32	User file format version

If this record is not found in the BIN file, the default value for the Min field is 3 and the default value for the Max field is 15.

ACEBNINP (ACEBIRBL Input File)

The ACEBNINP.DAT file contains records used as input for the ACEBIRBL program. The input file read by the BIN application as an ASCII file with no packed data. Each field is fixed-length except for the last field in the detail record. The end of a record will have a carriage return and line feed (0x0D0A). The input file has three types of records: header, trailer, and detail records.

Full-line comments are allowed in the input file. Any line that begins with a semicolon is considered to be a comment line.

Logical name	<ACEBNINP>
Data object reference	*ISBINInputFileData* <BIN_IDAT.CPP>
Organization	Sequential
Distribution class	Mirrored at close
File copies	Two: input file and saved input file with .ASV extension
Record length	Variable

After running the ACEBIRBL application, the BIN input file is renamed accordingly:

- If the ACE BIN file (ACEBINFL) was created successfully with no errors then the input file is renamed with an ASV file extension. For example, ACEBNINP.DAT would be renamed to ACEBNINP.ASV.
- If the ACE BIN file (ACEBINFL) *was not* created successfully due to errors then the input file is renamed with an ABD file extension. For example, ACEBNINP.DAT would be renamed to ACEBNINP.ABD.

Header Record

The first record in the input file is always the header record. Table 8 contains the layout of the header record.

Note: For FSA support, change the Toshiba file format version from 01 to 02.

Table 8. Layout of Header Record in Input File Version 04 for ACEBIRBL

Field Name	Type	Length	Decimal Offset	Description
Record ID	ASCII	2	0	Always H1
Sequence Number	ASCII	6	2	Always 000001
File Name	ASCII	8	8	Always BIN FILE
Create Date/Time	ASCII	14	16	Date and time in the format CCYYMMDDhhmmss
Action Code*	ASCII	1	30	R (replace) or U (update). This is a case-sensitive field.
File Format Version	ASCII	2	32	Toshiba file format version (01–04). Indicates the detail record format in the ACEBNINP file and the subsequent record format in the ACEBINFL file.
User File Format Version	ASCII	2	34	Always 00. Allows the user to define versions of user data in the Detail Records.

Field Name	Type	Length	Decima l Offset	Description
Record Count	ASCII	7	36	Number of detail records in the file

Note: * Version 1–3 support Action Code `R` (replace). Version 4 supports Action Codes `R` (replace) and `U` (update.

The BIN file rebuild application validates that the Record ID is `H1`, the Sequence Number is `000001`, the File Name is `BIN FILE`, the Create Date/Time is valid, the Action Code is `R`, and the File Format Version is `01`. If any of these validations fail, the BIN file application will not process the file and an error record is written to the application event log and BIN File Report Log.

Detail Record

The detail records in the file follow the header record. Table 9 contains the layout of the detail record for BIN file version 01.

Table 9. Layout of Detail Record in Input File Version 01 for ACEBIRBL

Field Name	Type	Length	Decima l Offset	Description
Record ID	ASCII	2	0	Always `D1`
Sequence Number	ASCII	6	2	Sequence number of the record in the file. This is a sequential number.
BIN Number	ASCII	20	8	BIN number, which is padded on the right with spaces as needed.
PAN Length	ASCII	2	28	Length of the Personal Account Number (PAN). Valid range is 01-99.
Action Code*	ASCII	1	30	Always `A` (add)
Card ID	ASCII	2	31	Card ID for this record. `99` indicates a dual-purpose card.
User Reserved	ASCII	v200	33	Reserved for user data

Note: * Currently, only `A` (add) is supported.

If the Record ID is not `D1`, the BIN or sequence number is invalid, the PAN length is not in the 01-99 range, the action code field is not `A`, or the User Reserved field exceeds 200 length, then an error record is written to the application event log and BIN File Report Log. If any of these validations fail, the BIN file application will not process the file.

Table 10 contains the layout of the detail record for BIN file version 02.

Table 10. Layout of Detail Record in Input File Version 02 for ACEBIRBL

Field Name	Type	Length	Decima l Offset	Description
Record ID	ASCII	2	0	Always `D1`
Sequence Number	ASCII	6	2	Sequence number of the record in the file. This is a sequential number.

Field Name	Type	Length	Decima l Offset	Description
BIN Number	ASCII	20	8	BIN number, which is padded on the right with spaces as needed.
PAN Length	ASCII	2	28	Length of the Personal Account Number (PAN). Valid range is 01 - 99.
Action Code	ASCII	1	30	A (add)
Processing Flag	ASCII	4	31	**X'0001'** Partial authorization allowed **X'0002'** PIN Encouragement allowed **X'0010'** FSA allowed
User Flag	ASCII	4	35	User integer for flag usage.
User Reserved	ASCII	v200	39	Reserved for user data

Note: When partial authorization is enabled and FSA is allowed for the Health Benefit Account credit tender, the balance due is set in the transaction amount field of the message that is sent to the host; otherwise, the QHP total amount is set in this field. Because partial authorization is enabled, the host provider may approve all or partial amounts associated with the transaction, QHP, and Rx fields. If partial authorization is disabled and the amount tendered is greater than the QHP balance on the HBA card, then the tender is declined.

Table 11. Layout of Detail Record in Input File Version 03 for ACEBIRBL

Field Name	Type	Length	Decima l Offset	Description
Record ID	ASCII	2	0	Always `D1`
Sequence Number	ASCII	6	2	Sequence number of the record in the file. This is a sequential number.
BIN Number	ASCII	20	8	BIN number, which is padded on the right with spaces as needed.
PAN Length	ASCII	2	28	Length of the Personal Account Number (PAN). Valid range is 01 - 99.
Action Code	ASCII	1	30	A (add)
Processing Flag	ASCII	4	31	**X'0001'** Partial authorization allowed **X'0002'** PIN Encouragement allowed **X'0010'** FSA allowed **X'0004'** Online WIC allowed

Field Name	Type	Length	Decima l Offset	Description
User Flag	ASCII	4	35	User integer for flag usage.
WIC State ID	ASCII	2	39	WIC Agency ID
User Reserved	ASCII	v200	41	Reserved for user data.

Table 12. Layout of Detail Record in Input File Version 04 for ACEBIRBL

Field Name	Type	Length	Decima l Offset	Description
Record ID	ASCII	2	0	Always `D1`
Sequence Number	ASCII	6	2	Sequence number of the record in the file. This is a sequential number.
BIN Number	ASCII	20	8	BIN number, which is padded on the right with spaces as needed.
PAN Length	ASCII	2	28	Length of the Personal Account Number (PAN). Valid range is 01 - 99.
Action Code	ASCII	1	30	`A` (add) or `D` (delete).
Processing Flag	ASCII	4	31	**X'0001'** Partial authorization allowed **X'0002'** PIN Encouragement allowed **X'0010'** FSA allowed **X'0004'** Online WIC allowed **X'0000'** No Flags Set
User Flag	ASCII	4	35	User integer for flag usage.
WIC Agency ID/Card Plan ID	ASCII	2	39	WIC Agency ID when the Processing Plan is set to X'0004'; otherwise it is the Card Plan ID.
Network ID	ASCII	2	41	Network ID
User Reserved	ASCII	v200	43	Reserved for user data

Note: ACEBIRBL application uses the BIN number with PAN length (not entire record) as a criteria match for the delete function. The detail record length for delete record is 43 bytes, but fields after detail action code (such as Processing flag, User flag, WIC Agency ID/Card Plan ID, or Network ID) will be ignored.

Trailer Record

The last record in the file is always the trailer record. Table 13 contains the layout of the trailer record.

Table 13. Layout of Trailer Record in Input File for ACEBIRBL

Field Name	Type	Length	Decimal Offset	Description
Record ID	ASCII	2	0	Always `T1`
Sequence Number	ASCII	6	2	Next sequential sequence number after the sequence number of the last detail record in the file

Note: * Currently, only `A` (add) is supported.

The BIN file rebuild application validates that the Trailer ID is `T1`, the next sequence number is correct, and the record count matches what the BIN file rebuild application read from the file. If validation fails, a new BIN file is not created and an error record is written to the application event log and BIN File Report Log.

Sample BIN Input File (<ACEBNINP>)

This is an example of the possible contents of a Version 01 ACEBNINP file:

```
;Header Record
H1000001BIN FILE20040928101540R01000000002
;Detail Records
;Valid for account numbers 4321000000000000 to 4321999999999999
D10000024321                16A99
;Valid for account numbers 3210000000000000000 to 3219999999999999999
D1000003321                 19A99
;Trailer Record
T1000004
```

This is an example of the possible contents of a Version 04 ACEBNINP file:

```
;Header Record
H1000001BIN FILE20100520101540R04000000002
;Detail Records
;Valid for account numbers 4008230000000000 to 4008239999999999
;Processing Flag 0000 = No Flag and Card Plan ID "VI"
D1000002400823              16A00000000VIXX
;Valid for account numbers 5103650000000000 to 5103659999999999
;Processing Flag 0004 = Online WIC and WIC Agency ID "MI"
D1000003510365              16A00040000MIXX
;Trailer Record
T1000004
```

ACEBINLG (ACEBIRBL Log File)

The ACEBIRBL Log file contains records for each execution of the ACEBIRBL program. Each record contains ASCII data that includes the start and end time of the program execution, the number of input records that were processed, and information about any errors that occurred.

Logical names	`<ACEBINLG>`
Data object reference	<`LOGFILE.CPP`>
Organization	Sequential
Distribution class	Mirrored per update
File copies	Maximum number of files specified by LOGNAMES variable in `acebinlg.ini` file (default files are current and previous)
Record length	Variable

This is an example of the possible file contents:

```
ACE BIN Report Log
Started processing: <acebninp>: 09/16/04;17:55:24;Replace;
Warning: Invalid Action Code in Record #2
Processed 100 records
Completed processing: <acebinfl>: 09/16/04;17:55:24
```

ACECATE5 (Coupon Translation Options File)

Operators use the SurePOS ACE Personalization interface (available from the SurePOS ACE Main Menu) to change option values in personalization DAT files.

Table 14. Companion Options INI and DAT Files

Options INI File	Options DAT File	Logical DAT File Name	Description
`optnparm.ini`	eamoptns.dat	<EAMOPTNS>	Base personalization options
`acecpntr.ini`	acecate5.dat	<ACECATE5>	Coupon translation options
`nlsparms.ini`	nlsoptns.dat	<NLSOPTNS>	National language support (NLS) personalization options
`apsparms.ini`	apsoptns.dat	<APSOPTNS>	SurePOS ACE EPS personalization options

Each personalization options `dat` file is a random access file that contains values for all options declared in its companion `ini` file. SurePOS ACE creates each `dat` file from its companion `ini` file if the `dat` file does not exist.

For a description of options `ini` files, see "OPTNPARM (Options)" on page 534.

For a description of how to use DIF along with ACEPERSL command, see "Using ACEPERSL to Maintain Personalization Options" on page 572.

For a description of how to enter user data in `ini` files, see "Tiered Overlay Files" on page 460.

Logical names	`<ACECATE5>`
Organization	Random access
Distribution class	Compound on Close
File copies	1
Record length	512 bytes
User data	Yes, in ini files

ACECFBAS (Coupon Fraud Basic File)

The coupon fraud basic file, which is generated by the ACEFIRBL program, contains records of UPC5/EAN99 coupon lookup keys.

Logical names	`<ACECFBAS>`
Data object reference	*`ISCouponFraudBasicStorage`* `<ACECFBAS.CPP>`
Organization	Keyed
Distribution class	Compound on Update
File copies	Maximum number of files specified by `LOGNAMES` variable in the `acefirlg.ini` file (default files are current and previous)
Record length	46 bytes
Key length	7 bytes (Coupon Number)

Table 15. File layout - ACECFBAS

Field Name	Type	Length	Decimal Offset	Description
UPC5/EAN99 lookup key	PD	7	0	UPC5/EAN99 (record key).
Reserved	ASCII	39	7	Reserved for future use.

ACECFGS1 (Coupon Fraud GS1 File)

The coupon fraud GS1file, which is generated by the ACEFIRBL program, contains records of GS1 DataBar coupon lookup keys.

Logical names	`<ACECFGS1>`
Data object reference	*`ISCouponFraudGS1Storage`* `<ACECFGS1.CPP>`
Organization	Keyed
Distribution class	Compound on Update
File copies	Maximum number of files specified by `LOGNAMES` variable in the `acefirlg.ini` file (default files are current and previous)
Record length	72 bytes
Key length	35 bytes (Coupon Number)

Table 16. File layout - ACECFGS1

Field Name	Type	Length	Decimal Offset	Description
GS1 DataBar lookup key	PD	35	0	GS1 DataBar (record key).
Reserved	ASCII	37	35	Reserved for future use.

ACECLSCP (Close Period Checkpoint File)

The Close Period Checkpoint file keeps checkpoint information of the closing process.

Logical names	`<ACECLSCP>`
Organization	Direct
Distribution class	Mirrored per update
File copies	1
Record length	12 bytes

Field Name	Type	Length	Decimal Offset	Description
Checkpoint ID	Long	4	0	One of the following checkpoint values that represent the close stage: 0 - CheckpointNone 1 - CheckpointRestartingClose 2 - CheckpointAboutToStartClose 3 - CheckpointFlaggedStoreCloseAsStarted 4 - CheckpointTerminalsPrepared 5 - CheckpointOperatorsPrepared 6 - CheckpointTransactionStoragePrepared 7 - CheckpointOperatorsClosed 8 - CheckpointTerminalsClosed 9 - CheckPointSuspendRetrieveClosed 10 - CheckPointLocalSuspendRetrieveClosedNode1 11 - CheckPointLocalSuspendRetrieveClosedNode2 12 - CheckPointLocalSuspendRetrieveClosedNode3 13 - CheckPointLocalSuspendRetrieveClosedNode4 14 - CheckPointLocalSuspendRetrieveClosedNode5 15 - CheckPointLocalSuspendRetrieveClosedNode6 16 - CheckPointLocalSuspendRetrieveClosedNode7 17 - CheckPointLocalSuspendRetrieveClosedNode8 18 - CheckPointLocalSuspendRetrieveClosed 19 - CheckpointTransactionStorageClosed 20 - CheckpointForceSignOffTerminal 21 - CheckpointTransactionStorageClosedNoSuspend 22 - CheckpointTerminalNotResponding 23 - CheckpointLast
Restart count	Long	4	4	Number of times that the checkpoint history has been recovered.
DateTime	Long	4	8	The number of seconds since 00:00:00 January 1, 1901 GMT.
Extra	Long	4	12	Integer representing the close ID.

ACECLSLG (Close Log)

The adx_idt1\aceclslg.dat file is a log of ledger processing entries that SurePOS ACE makes as it processes a close procedure. For problem determination purposes, SurePOS ACE logs an entry for each ledger substage that it completes. See "ACECLSPG (Close Progress Control File)" on page 54 for a description of the meaning of each status code.

The maximum size of the aceclslg.dat file is controlled by the `SIZE` parameter in `adx_idt1\aceclslg.ini`. (See "ACECLSLG (Close Log Rollover Settings)" on page 473.) The default setting is 32000 bytes. You can change the value to suit your own preferences. When aceclslg.dat reaches the threshold size, SurePOS ACE rolls it to become the previous period file and allocates a new current file. The name of the previous period file is specified by the `LOGNAME` parameter of the INI file. By default, the previous period file name is aceclslp.dat. You can add any number of file names to the `LOGNAME` list to retain old copies of the log, which SurePOS ACE rolls whenever aceclslg.dat reaches the threshold size.

Logical name	`<ACECLSLG>`
Data object reference	`<LOGFILE.CPP>`
Organization	Sequential
Distribution class	Mirrored per update
File copies	1
Record length	

ACECLSPG (Close Progress Control File)

The Close Progress Control file is a random access file that contains records representing each ledger defined in SurePOS ACE. It is a work file that helps control Checkout Support application processing. The Checkout Support application creates and manages this file and its records. Only Checkout Support updates the file.

The Close Progress Control file contains checkpoint information for the various database ledgers. This information is required to restart Checkout Support and protect against loss or duplication of data in accounting and report files. Because the file tracks the last successful stage and substage completed for each ledger, upon restart or recovery SurePOS ACE can selectively complete necessary processing on a ledger by ledger basis. Reports against this file indicate the close status of the ledgers.

The prime version of the Close Progress Control File resides on the file server controller. An image version exists on the alternate file server but is accessed only when the alternate is made the acting file server controller. The file is distributed every time the file is updated or closed.

Logical name	`<ACECLSPG>`
Data object reference	*ISCloseProgressData* `<SATPCSTR.CPP>`
Organization	Random
Distribution class	Mirrored per update
File copies	1
Record length	16 bytes (plus 2 for EOL)

Ledger Record

Field Name	Type	Length	Decimal Offset	Description
LedgerID	ASCII	2	0	An ID that identifies the ledger for this record: 0 - Accounting Database Ledger 1 - Miscellaneous Transaction Ledger Files 2 - Operator Performance Database Ledger 3 - Tender Listing Database Ledger 4 - Terminal Productivity Database Ledger 5 - Exception Log Database Ledger 6 - Transaction Tracking Database Ledger 7 - Item Movement Database Ledger 8 - Customer Account Status Database Ledger 9 - Department Totals Database Ledger 10 - TLog Processor Control Database Ledger 11 - EPS Message Database Ledger 12 - Coupon Tracking Database Ledger 13 - Customer Audit Database Ledger 14 - Customer Activity Database Ledger 15 - User Database Ledger 16 - Negative Sales Database Ledger 17 - Operator Status Ledger 18 - EPS Accounting Ledger 19 - EPS Tender Listing Database Ledger 20 - Signature File Database Ledger 21 - WIC EBT Files Database Ledger
CloseStage	Int	2	2	Defines major milestones for close period processing.
CloseSubStage	Int	2	4	Defines intermediate milestones for close period processing. The value stored in this field indicates the substage that Checkout Support has completed.
CloseTime	String	10	6	The time that the current close began. (Format = MMDDYYHHMM)

Note: The combination of the values stored in `CloseStage` and `CloseSubStage` indicates the last step that Checkout Support *completed* processing for the ledger.

Accounting Database Ledger

Ledger ID	0
Ledger abbreviation	AC
Ledger database class	*ISAccountDatabaseLedger* `<ACCNTDBL.CPP>`
Ledger storage class	*ISSAManagedAccountStorage* `<SAMNASTR.CPP>`

Table 17. Close Stages for Accounting Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
1	0	Updated the store record with the current close date in the current short accounting file (<EAMACCTC>).
2	0	Initial state for this stage, which switches to a new short reporting period.
	1	Deleted the existing old short period file (<EAMACCTO>).
	2	Renamed the previous short period file (<EAMACCTP>) to old.
	3	Renamed the current accounting file (<EAMACCTC>) to previous.
	4	Created a new current accounting file.
	0xFF	All substages are complete for this stage of the close.
3	0	Merged short period totals into a temporary copy of the long totals accounting file.
4	0	Deleted the current long totals accounting file (<EAMACCCL>) and renamed the temporary long totals file to the new current long totals file.
5	0	Updated the store record with the current close date in the current long accounting file (<EAMACCCL>).
6	0	Initial state for this stage, which switches to a new long reporting period.
	1	Deleted the old long accounting file (<EAMACCOL>).
	2	Renamed the previous long accounting file (<EAMACCPL>) to old long.
	3	Renamed the current long accounting file (<EAMACCCL>) to previous long.
	4	Created a new current long accounting file.
	0xFF	All substages are complete for this stage of the close.
0xFE	0	Close has completed for this ledger.

Miscellaneous Transaction Ledger

Ledger ID	1
Ledger abbreviation	MT
Ledger database class	*ISSAMiscTransactionDatabaseLedger* <SAMSCTDL.CPP>
Ledger storage class	*ISSAMiscTransactionAccountStorage* <SAMTASTR.CPP>

Table 18. Close Stages for Miscellaneous Transaction Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
1	0	Updated the store record with the current close date in the current short miscellaneous transactions file (<EAMMTRNC>).
2	0	Initial state for this stage, which switches to a new short reporting period.
	1	Deleted the old short period miscellaneous transactions file (<EAMMTRNO>).
	2	Renamed the previous short period file (<EAMMTRNP>) to old.
	3	Renamed the current short period file (<EAMMTRNC>) to previous.
	4	Created a new current short period miscellaneous transactions file.
	0xFF	All substages are complete for this stage of the close.
3	0	Merged short period totals into a temporary copy of the current long miscellaneous transactions file (<EAMMTRWK>).
4	0	Deleted the current long miscellaneous transactions file (<EAMMTRCL>) and renamed the temporary long totals file to the new current long totals file.
5	0	Updated the store record with the current close date in the current long miscellaneous transactions file (<EAMMTRCL>).
6	0	Initial state for this stage, which switches to a new long reporting period.
	1	Deleted the old long miscellaneous transactions file (<EAMMTROL>).
	2	Renamed the previous long miscellaneous transactions file (<EAMMTRPL>) to old long.
	3	Renamed the current long period file (<EAMMTRCL>) to previous long.
	4	Created a new current long miscellaneous transactions file.
	0xFF	All substages are complete for this stage of the close.

Stage	SubStage	Stage Description
0xFE	0	Close has completed for this ledger.

Operator Performance Database Ledger

Ledger ID	2
Ledger abbreviation	OP
Ledger database class	*ISOperatorPerformanceDatabaseLedger* <OPRTRPDL.CPP>
Ledger storage class	*ISSAPerformanceStorage* <SAPRFSTR.CPP>

Table 19. Close Stages for Operator Performance Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
1	0	Updated the store record with the current close date and time in the current operator performance file (<EAMPERFC>).
2	0	Initial state for this stage, which switches to a new short reporting period.
	1	Deleted the old short period operator performance file (<EAMPERFO>).
	2	Renamed the previous short period file (<EAMPERFP>) to old.
	3	Renamed the current short period file (<EAMPERFC>) to previous.
	4	Created a new current short period operator performance file.
	0xFF	All substages are complete for this stage of the close.
3	0	Merged short period totals into a temporary copy of the current long operator performance file (<EAMPERWK>).
4	0	Deleted the current long operator performance file (<EAMPERCL>) and renamed the temporary long totals file to the new current long totals file.
5	0	Updated the store record with the current close date in the current long operator performance file (<EAMPERCL>).
6	0	Initial state for this stage, which switches to a new long reporting period.
	1	Deleted the old long operator performance file (<EAMPEROL>).
	2	Renamed the previous long operator performance file (<EAMPERPL>) to old long.

Stage	SubStage	Stage Description
	3	Renamed the current long period file (<EAMPERCL>) to previous long.
	4	Created a new current long operator performance file.
	0xFF	All substages are complete for this stage of the close.
0xFE	0	Close has completed for this ledger.

Tender Listing Database Ledger

Ledger ID	3
Ledger abbreviation	TL
Ledger database class	*ISTenderListingDatabaseLedger* `<TNDRLSDL.CPP>`
Ledger storage class	*ISSACSTenderListingStorage* `<SACTLSTR.CPP>`

Table 20. Close Stages for Tender Listing Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
1	0	Updated the store record with the current close date and time in the current tender listing file (<EAMTLIST>).
2	0	Initial state for this stage, which switches to a new reporting period.
	1	Deleted the old tender listing file (<EAMTLISO>).
	2	Renamed the previous tender listing file (<EAMTLISP>) to old.
	3	Renamed the current tender listing file (<EAMTLIST>) to previous.
	4	Created a new current short tender listing file.
	0xFF	All substages are complete for this stage of the close.
0xFE	0	Close has completed for this ledger.

Terminal Productivity Database Ledger

Ledger ID	4
Ledger abbreviation	TP
Ledger database class	*ISTerminalProductivityDatabaseLedger* `<TPDBLDGR.CPP>`
Ledger storage class	*ISSATPStatisticsStorage* `<SATPSSTR.CPP>`

Ledger interval class	*ISTerminalProductivityInterval* `<TPINTRVL.CPP>`

Table 21. Close Stages for Terminal Productivity Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
1	0	Stored the current statistics from memory into the current period file (<EAMTPROD>).
2	0	Updated the store record with the current close date in the current period terminal productivity file (<EAMTPROD>).
3	0	Initial state for this stage, which switches to a new reporting period.
	1	Deleted the old terminal productivity file (<EAMTPROO>).
	2	Renamed the current terminal productivity file (<EAMTPROD>) to old.
	3	Created a new current period terminal productivity file.
	4	Wrote the begin period record to the new current period terminal productivity file (<EAMTPROD>)
	0xFF	All substages are complete for this stage of the close.
0xFE	0	Close has completed for this ledger.

Exception Log Database Ledger

Ledger ID	5
Ledger abbreviation	EX
Ledger database class	*ISSAExceptionLogDatabaseLedger* `<SAEXCLDL.CPP>`
Ledger storage class	*ISCSExceptionStorage* `<CSEXCSTR.CPP>`

Table 22. Close Stages for Exception Log Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
1	0	Initial state for this stage, which switches to a new reporting period.
	1	Deleted the old exception log file (<EAMEXCPO>).
	2	Renamed the current exception log file (<EAMEXCPT>) to old.
	3	Created a new current period exception log file.

Stage	SubStage	Stage Description
	0xFF	All substages are complete for this stage of the close.
0xFE	0	Close has completed for this ledger.

Transaction Tracking Database Ledger

Ledger ID	6
Ledger abbreviation	TK
Ledger database class	*ISTransactionTrackingDatabaseLedger* `<TRNSCTDL.CPP>`
Ledger storage class	*ISSATransactionTrackingStorage* `<SATRTSTR.CPP>`

Table 23. Close Stages for Transaction Tracking Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
1	0	Initial state for this stage, which switches to a new reporting period.
	1	Deleted the old transaction tracking file (<EAMTRAKO>).
	2	Renamed the current transaction tracking file (<EAMTRAK>) to old.
	3	Created a new current period transaction tracking file.
	0xFF	All substages are complete for this stage of the close.
0xFE	0	Close has completed for this ledger.

Item Movement Database Ledger

Ledger ID	7
Ledger abbreviation	IM
Ledger database class	*ISItemMovementDatabaseLedger* `<ITMMVMDL.CPP>`
Ledger storage class	*ISSAItemMovementStatisticsStorage* `<SAIMSSTR.CPP>`
Ledger interval class	*ISItemMovementInterval* `<ITMMVMNI.CPP>`

Table 24. Close Stages for Item Movement Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.

Stage	SubStage	Stage Description
0	0	Ledger has been initialized and is ready to process close request.
1	0	Flushed the item movement statistics to the current long and short period item movement files, and updated the date of the last sale.
2	0	Updated the store record with the current close date in the current short item movement file (<EAMIMOVE>)
3	0	Initial state for this stage, which switches to a new reporting period.
	1	Deleted the previous short item movement file (<EAMIMOVO>).
	2	Renamed the current short item movement file (<EAMIMOVE>) to previous short.
	3	Created a new current short item movement file.
	0xFF	All substages are complete for this stage of the close.
4	0	Updated the store record with the current close date in the current long item movement file (<EAMIMOCL>)
5	0	Initial state for this stage, which switches to a new reporting period.
	1	Deleted the previous long item movement file (<EAMIMOPL>).
	2	Renamed the current long item movement file (<EAMIMOCL>) to previous long.
	3	Created a new current long item movement file.
	0xFF	All substages are complete for this stage of the close.
0xFE	0	Close has completed for this ledger.

Customer Account Status Database Ledger

Ledger ID	8
Ledger abbreviation	CA
Ledger database class	*ISCustomerAccountStatusDatabaseLedger* `<CSTMASDL.CPP>`
Ledger storage class	*ISSACustomerAccountStatusStorage* `<SACASSTR.CPP>`

Table 25. Close Stages for Customer Account Status Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.

Stage	SubStage	Stage Description
1	0	Updated the store record with the current close date in the current period customer account status file (<EAMCUSTA>).
2	0	Initial state for this stage, which switches to a new reporting period.
	1	Closed the current period customer account status file (<EAMCUSTA>).
	2	Deleted the old period customer account status file (<EAMCUSTO>).
	3	Renamed the current period customer account status file (<EAMCUSTA>) to old.
	4	Destroyed the current period file object, because it is no longer valid.
	5	Created a new current period customer account status file.
	0xFF	All substages are complete for this stage of the close.
0xFE	0	Close has completed for this ledger.

Department Totals Database Ledger

Ledger ID	9
Ledger abbreviation	DT
Ledger database class	*ISDepartmentTotalsDatabaseLedger* `<DPRTMTDL.CPP>`
Ledger storage class	*ISSADepartmentStatisticsStorage* `<SADPSSTW.CPP>`, *ISSADepartmentVarianceStatisticsStorage* `<SADVSSTR.CPP>`, *ISSAIntervalDepartmentStatisticsStorage* `<SAIDSSTR.CPP>`
Ledger interval class	*ISDepartmentTotalsInterval* `<DPRTMNTI.CPP>`, *ISIntervalDepartmentStatistics* `<INTRVLDS.CPP>`

Table 26. Close Stages for Department Totals Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
1	0	Stored the department totals statistics currently in memory. Wrote the date and time of the close in the store record of the current short period department totals file (<EAMDEPTC>).
2	0	Initial state for this stage, which switches to a new short reporting period.
	1	Copied the previous short period department totals file (<EAMDEPTP>) to the old short period file (<EAMDEPTO>).

Stage	SubStage	Stage Description
	2	Copied the current short period department totals file (<EAMDEPTC>) to the previous short period file.
	3	Deleted the current period department totals file (<EAMDEPTC>) to old.
	4	Created a new current period department totals file.
	0xFF	All substages are complete for this stage of the close.
3	0	Merged short period totals into a temporary copy of the long department totals file (<EAMDEPWK>). Updated department variance statistics for the same period.
4	0	Completed applying the short period department totals to the long period department totals: 1. Deleted the current long department totals file (<EAMDEPCL>). 2. Renamed the temporary department totals file (<EAMDEPWK>) to the current long department totals file. Completed applying the short period department totals to the department variance totals: 1. Deleted the current department variance file (<ACEDPTVC>). 2. Renamed the temporary department variance totals file (<ACEDPVWK>) to the current department variance totals file.
5	0	Updated the close date in the store record in the current long department totals file.
6	0	Initial state for this stage, which switches to a new long reporting period.
	1	Copied the previous long period department totals file (<EAMDEPPL>) to the old long period file (<EAMDEPOL>).
	2	Copied the current long period department totals file (<EAMDEPCL>) to the previous period long file.
	3	Deleted the current long period department totals file (<EAMDEPCL>).
	4	Deleted both the short and long current period department totals files. (This is done to account for a possible change in the option determining the use of an extended department totals record.)
	5	Created new short and long current period department totals files.
	0xFF	All substages are complete for this stage of the close.
7	0	Initial state for this stage, which switches to a new reporting period for the department variance statistics.
	1	Deleted the old department variance statistics file (<ACEDPTVO>).

Stage	SubStage	Stage Description
	2	Renamed the previous department variance statistics file (<ACEDPTVP>) to the old department variance statistics file.
	3	Renamed the current department variance statistics file (<ACEDPTVC>) to the previous department variance statistics file.
	4	Created a new current department variance statistics file.
	0xFF	All substages are complete for this stage of the close.
8	0	Deleted the previous interval department totals file (<EAMHDPTP>).
9	0	Renamed the current interval department totals file (<EAMHDPTC>) to the previous interval department totals file.
10	0	Completed close period processing for interval department totals by creating a new current interval department totals file.
0xFE	0	Close has completed for this ledger.

TLog Processor Control Database Ledger

Ledger ID	10
Ledger abbreviation	TC
Ledger database class	*ISSATLogProcessorControlDatabaseLedger* `<SATPCNDL.CPP>`
Ledger storage class	*ISSATLogProcessorControlStorage* `<SATPCSTR.CPP>`

Table 27. Stages for Closes of Reporting Periods and Department Totals

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
1	0	Updated the close control record with the new TLog name, location, and time and reset the user files-rolled flag. Stored record back in <EAMCSCF1>.
2	0	Reset the terminal productivity statistics.
3	0	Reset the control storage to allow for accurate processing of a new TLog.
4	0	Saved the reset control storage data in <EAMCSCF1> and distributed <EAMCSCF1> to all controllers.
5	0	Created the new TLog and index files for the Electronic Journal and Manager's Journal.
7	0	Reset the store closing data and new TLog in <EAMTERMS>.

Stage	SubStage	Stage Description
8	0	Archived the old TLog, Electronic Journal Index, and Manager's Journal Index files.
9	0	Archived the old Exception Log.
10	0	Archived the previous short Item Movement file.
11	0	Archived the previous long Item Movement file.
12	0	Verify whether the suspend files have been rolled. Compare the DateTime field in the TLog Close string (0x21) and the date/time field in the ACESRCPC file.
0xFE	0	Close has completed for this ledger.

Table 28. Stages for File Closes

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
206	0	Entry point for file closes.
207	0	Reset the store closing data.
208	0	No action performed.
209	0	Archived the old Exception Log, if closing Exception Log.
210	0	Archived the previous short Item Movement file, if closing short Item Movement file.
211	0	Archived the previous long Item Movement file, if closing long Item Movement file.
0xFE	0	Close has completed for this ledger.

EPS Message Database Ledger

Ledger ID	11
Ledger abbreviation	AM
Ledger database class	*ISAPSDatabaseLedger* `<APSDTBSL.CPP>`
Ledger storage class	*ISCSAPSStorage* `<CSAPSSTR.CPP>`

Table 29. Close Stages for EPS Message Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.

Stage	SubStage	Stage Description
1	0	Created and sent a close store complete (StoreCAPComplete) EPS message to the EPS server via a GIPC pipe.
0xFE	0	Close has completed for this ledger.

Coupon Tracking Database Ledger

Ledger ID	12
Ledger abbreviation	CP
Ledger database class	*ISEMCouponTrackingDatabaseLedger* `<EMCPNTDL.CPP>`
Ledger storage class	*ISSAEMCouponTrackingStorage* `<SAEMCSTR.CPP>`

Table 30. Close Stages for Coupon Tracking Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
1	1	The old period file (<EAMCOUPO>) exists, no action taken at this substage. Processing continues with the next substage.
1	2	If the work file (<EAMCOUPW>) existed, it has been deleted in preparation for append processing.
1	3	The old period file (<EAMCOUPO>) has been copied to the work file (<EAMCOUPW>), followed by appending the data from the current period file (<EAMCOUPC>) to the work file (<EAMCOUPW>).
1	4	The old period file (<EAMCOUPO>) has been deleted.
1	5	Rename of the work file (<EAMCOUPW>) to the old period file (<EAMCOUPO>) has been completed.
1	6	The current period file (<EAMCOUPC>) has been deleted.
1	0xFF	If the old period file (<EAMCOUPO>) file existed prior to closing, then the append and closing process for this stage has been completed. If the old period file (<EAMCOUPO>) did not exist at closing, a rename of the current period file (<EAMCOUPC>) to the old period file name (<EAMCOUPO>) has been performed. This is the only substage for this condition.
2	0	Created a new current period coupon tracking file (<EAMCOUPC>).
0xFE	0	Close has completed for this ledger.

Customer Audit Database Ledger

Ledger ID	13
Ledger abbreviation	AU
Ledger database class	*ISCustomerAuditDatabaseLedger* `<CSTMADDL.CPP>`
Ledger storage class	*ISSACSCustomerAuditStorage* `<SACSADST.CPP>`

Table 31. Close Stages for Customer Audit Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
1	0	Appended the data stored in the current customer audit file (<EAMFBAUD>) to the old period file (<EAMFBAUO>), then deleted the current period file.
2	0	Created a new current period customer audit file (<EAMFBAUD>).
0xFE	0	Close has completed for this ledger.

Customer Activity Database Ledger

Ledger ID	14
Ledger abbreviation	CT
Ledger database class	*ISCustomerActivityDatabaseLedger* `<CSTMRADL.CPP>`
Ledger storage class	*ISSACustomerStorage* `<SACSTSTR.CPP>`

Table 32. Close Stages for Customer Activity Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
1	0	Backed up the current period customer activity file (<EAMFBACT>) by copying it to <EAMFBACX>.
2	0	Transferred customer points from the current period activity file (<EAMFBACT>) to the previous period file (<EAMFBAPP>), then created new current period file and reset customer point values.
0xFE	0	Close has completed for this ledger.

User Database Ledger

Ledger ID	15
Ledger abbreviation	UD
Ledger database class	*ISUserDatabaseLedger* `<USERLDGR.CPP>`

To ensure that each user file is not rolled a second time if a failure occurs during close period processing for this ledger, each user file is internally assigned a unique ID called a *fileId*. Close processing for each user file requires 3 stages. To ensure that stage numbers do not accidentally overlap other enumerated close period stages for this ledger, a base value called RollUserFileStage is included in the calculation of the stage number. The formula for calculating the stage number is:

```
n = RollUserFileStage + ( 3 &mult; ( fileId - 1 ))
```

Table 33. Close Stages for User Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
n	0	Raised the RollingUserDatabaseStarted event to allow each file to append trailer information and then close the file.
n+1	0	Initial state for this stage, which switches to a new reporting period for this user file.
	1	Deleted the old period version of this user file.
	2	Renamed the previous period version of this user file to the old period name.
	3	Ensured that the previous period version no longer exists.
	4	Renamed the current period version to the previous period name.
	0xFF	All substages are complete for this stage of the close.
n+2	0	Raised the RollingUserDatabaseCompleted event to allow the new current period user file to be created, its header to be written, and, if necessary, the file to be initialized with data.
0xFE	0	Close has completed for this ledger.

Negative Sales Database Ledger

Ledger ID	16
Ledger abbreviation	NS
Ledger database class	*ISNegativeSalesDatabaseLedger* `<NGTVSLDL.CPP>`

Table 34. Close Stages for Negative Sales Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status not initialized for this ledger.
0	0	Ledger has been initialized and is ready to process close request.
1	0	Initial state for this stage, which switches to a new reporting period.
	1	Deleted the old period negative sales file (<ACENGSLO>).
	2	Renamed the previous period negative sales file (<ACENGSLP>) to the old period file.
	3	Renamed the current period negative sales file (<ACENGSLC>) to the previous period file.
	4	Created a new current period negative sales file.
	0xFF	All substages are complete for this stage of the close.
0xFE	0	Close has completed for this ledger.

Operator Status Ledger

Ledger ID	17
Ledger abbreviation	OS
Ledger database class	*ISOperatorStatusDatabaseLedger* `<OPRTRSDL.CPP>`
Ledger storage class	*ISSAOperatorStorage* `<SAOPRSTR.CPP>`

This ledger does not utilize any unique close stages.

EPS Accounting Ledger

Ledger ID	18
Ledger abbreviation	AA
Ledger database class	*ISAPSAccountingLedger* `<APSACCTL.CPP>`
Ledger storage class	*ISCSAPSStorage* `<CSAPSSTR.CPP>`

This ledger does not utilize any unique close stages.

EPS Tender Listing Ledger

Ledger ID	19
Ledger abbreviation	AT

Ledger database class	*EPSTenderListingDatabaseLedger* <EPSTLSDL.CPP>
Ledger storage class	*EPSTenderListingStorage* <EPSTLSTR.CPP>

Table 35. Close Stages for EPS Tender Listing Ledger

Stage	SubStage	Stage Description
1	0	Write end period record.
2	1	Delete the old tender listing file.
2	2	Rename previous tender listing file to old (epstlisp.dat -> epstliso.dat)
2	3	Rename current tender listing file to previous (epstlist.dat -> epstlisp.dat)
2	4	Create a new current tender listing file (epstlist.dat). Set recovery value to 0 and write to recovery value file.
2	0xFF	All EPS tender listing files have been successfully renamed.
0xFE	0	EPS Tender Listing ledger has completed.

Signature File Ledger

Ledger ID	20
Ledger abbreviation	SF
Ledger database class	*SignatureFileDatabaseLedger* <SGNFLDBL.CPP>
Ledger storage class	*SignatureFileStorage* <SGNFLSTR.CPP>
Ledger data class	*SignatureFileData* <SGNFLDAT.CPP>

Table 36. Close Stages for Signature Files Database Ledger

Stage	SubStage	Stage Description
0	0	Close status not initialized for this ledger.
1	0	Append the Close Record to the Signature File.
2	0	This stage is used if the Options -> Database -> E-Sign -> Replace Previous Signature File at Store Close option is off. Initial state for this stage which appends the current Signature File (<SGNATURE>) to the previous Signature File (<SGNATURP>).
2	1	Copy the previous Signature File (<SGNATURP>) to the temporary Signature File (<SGNATURT>).
2	2	Append the current Signature File (<SGNATURE>) to the temporary Signature File (<SGNATURT>).
2	3	Delete the current Signature File (<SGNATURE>).
2	0xFF	All substages are complete for this stage of the close.

Stage	SubStage	Stage Description
3	0	Delete the previous Signature File (<SGNATURP>).
4	0	If Replace Previous Signature File at Store Close option is on, then rename the current Signature File (<SGNATURE>) to the previous Signature File (<SGNATURP>). If the Replace Previous Signature File at Store Close option is off, then rename the temporary Signature File (<SGNATURT>) to the previous Signature File (<SGNATURP>).
5	0	Create a new current Signature File (<SGNATURE>).
0xFE	0	Close has completed for Signature File ledger.

WIC EBT Files Database Ledger

Ledger ID	21
Ledger abbreviation	WE
Ledger database class	*ISWICEBTFilesDatabaseLedger* `<WEFLSDBL.CPP>`
Ledger storage class	*ISWICEBTFilesStorage* `<WEFLSTR.CPP>`

Each WIC state that is defined in Options -> WIC Transaction -> Agency personalization has an ID field (*n*). This ledger uses the ID field as the stage value. Each stage iterates through the substages shown in Table 37.

Table 37. Close Stages for WIC EBT Files Database Ledger

Stage	SubStage	Stage Description
0xFF	0xFF	Close status is not initialized for this ledger.
0	0	The ledger has been initialized and is ready to process the close request.
n	1	Append the Trailer Record to the Claim File <ACEWIC4:>*xx*c.dat (*xx* is the two-character State for ID *n*).
n	2	Delete the previous Claim File <ACEWIC4:>*xx*p.dat (*xx* is the two-character State for ID *n*).
n	3	Copy the State's Claim File to the Temporary Transit File (<TEMPWIC4>).
n	4	Rename the Temporary Transit File (<TEMPWIC4>) to the State's Transmit File <WIC4:>*xxyjjj*.s?? (*xx* is the two-character State for ID *n*, *y* is the last digit of the current year, *jjj* is the Julian date, and *??* is a sequence number).
n	5	Write a Reconciliation Record to the WIC Reconcile File (<ACEWERCL>).
n	6	Rename the State's current Claim File (<ACEWIC4:>*xx*c.dat) to the previous Claim File (<ACEWIC4:>*xx*p.dat).

Stage	SubStage	Stage Description
n	7	If the `Agency is Active` option is enabled, create a new current Claim File (<ACEWIC4:>*xx*c.dat).
n	0xFF	All substages are complete for this stage of the close.
0xFE	0	Close has completed for this ledger.

ACECPNFR (ACEFIRBL Input File)

The `ADX_IDT1:ACECPNFR.DAT` file contains records used as input for the ACEFIRBL program. The input file read by the fraudulent coupon rebuild application is an ASCII file containing a combination of UPC5/EAN99 and GS1 DataBar coupons. This file is derived from the Fraudulent Coupon File received from the Coupon Information Center. This input file only contains the coupon lookup key column from the Fraudulent Coupon File.

The format of this file is a single column of UPC5/EAN99 and GS1 DataBar coupons where the only delimiter is a carriage return and line feed (0x0D0A) after each coupon.

Logical names	`<ACECPNFR>`
Organization	Sequential
Distribution class	Local
File copies	Two: input file and saved input file with a .ASV extension
Record length	Variable

After running the ACEFIRBL application, the coupon fraud input file is renamed accordingly:

- If the appropriate number of coupon fraud files (ACECFBAS and/or ACECFGS1) are created successfully with no errors, then the input file is renamed with an `.ASV` file extension. For example, `ACECPNFR.INI` would be renamed to `ACECPNFR.ASV`. Note, output files for both basic UPC5/EAN99 and GS1 DataBar coupons will only be generated if both types of coupons are present in the input file. When only UPC5/EAN99 coupons are present in the input file, then only a basic output file will be generated (`ACECFBAS`). Similarly, when the input file only contains GS1 DataBar coupons, then only a GS1 output file will be generated (ACECFGS1).
- If either file (ACECFBAS and/or ACECFGS1) was not created successfully due to errors, then the input file is renamed with an ABD file extension. For example, `ACECPNFR.INI` would be renamed to `ACECPNFR.ABD`. To avoid errors, UPC5/EAN99 coupons must be 11 or 12 digits and GS1 DataBar coupons must be 25 to 70 digits. ACEFIRBL will report length violations in the report log (`ACEFIRLG.DAT`).

Sample Coupon Fraud Input File (<ACECPNFR>)

The following is an example of the possible contents of an ACECPNFR file that contains both basic and GS1 DataBar coupons:

```
549000000936
51901410001
539800000935
553569000009
544700992833
581421992742
8110106141410012472750110760950000
54400000078
521000000765
54470000093
```

```
544000000825
81101061414165132131001107603101231950000
```

ACEDDESC (Department Descriptors File)

The Department Descriptors File (`adx_ipgm\aceddesc.dat`) contains descriptors for each department defined through personalization.

An image version of this file is kept on the alternate file server. Distribution of the file takes place only when the file is closed to minimize the performance impact.

Logical name	`<ACEDDESC>`
Data object reference	*ISDepartmentDescriptorStorage* `<DSCRPSTR.CPP>`
Organization	Fixed direct
Distribution class	Compound per Update
File copies	1
Record length	64 bytes

Field Name	Type	Length	Decimal Offset	Description
DepartmentName	ASCII	40	0	Department descriptor
Reserved	ASCII	24	40	Reserved

You can examine this file through the Descriptor Editor, which is available from the Miscellaneous pull-down menu in personalization.

ACEDPTLD (Department Subtotal Descriptors File)

The Department Subtotal Descriptors file saves department subtotal group IDs and descriptors from the Department Totals Report that an operator edits in Miscellaneous -> Reporting List Editor -> Department Totals Report personalization.

Logical name	`<ACEDPTLD>`
Data object reference	*ISDepartmentReportingListStorage* `<DPRRLSTR.CPP>`
Organization	Fixed direct
Distribution class	Compound per Update
File copies	1
Record length	22 bytes

Field Name	Type	Length	Decimal Offset	Description
ListId	ASCII	4	0	Subtotal group ID assigned by SurePOS ACE as an operator uses the Reporting List Editor in Personalization to add a subtotal group to the Department Totals Report.

Field Name	Type	Length	Decimal Offset	Description
Description	ASCII	18	4	Name of subtotal group that an operator assigns in personalization.

ACEDPTLS (Department Lists File)

The Department Lists file saves the list of departments in each subtotals group in the Department Totals Report that an operator edits in Miscellaneous -> Reporting List Editor -> Department Totals Report personalization.

The file contains one record for every department that appears in each subtotal group in the Department Totals report. Each record contains the subtotal group ID and a department ID. If there are 3 subtotal groups in the report, there are 3 sets of records in this file. If there are 5 departments in one of the subtotal groups in the report, there are 5 records in this file for that subtotal group.

Logical name	`<ACEDPTLS>`
Data object reference	*ISDepartmentReportingListStorage* `<DPRRLSTR.CPP>`
Organization	Fixed direct
Distribution class	Compound per Update
File copies	1
Record length	8 bytes

Field Name	Type	Length	Decimal Offset	Description
ListID	ASCII	4	0	The ID of the subtotal group where the department is reported in the Department Totals report.
DepartmentNumber	ASCII	4	4	The department number of one department that belongs to the subtotal group.

ACEDPTV* (Department Variance File)

The Department Variance file (`adx_idt4\acedptv?.dat`) is keyed on the two-byte Group ID field, which identifies department groups defined through Miscellaneous -> Reporting List Editor -> Department Totals personalization. The report uses the same subtotal groups as those defined for the Department Totals report.

The file maintains data for the Department Variance Report, which lists sales variances for department subgroups when comparing current and previous, or current and old, accounting periods.

The report also shows customer count variances and total department sales variances.

Sales amounts in the report are in whole dollars.

The size of the file is limited to a maximum of 1200 records.

Logical name	`<ACEDPTVC>`, `<ACEDPTVP>`, `<ACEDPTVO>`, `<ACEDPTWK>`
Data object reference	*ISSADepartmentVarianceStatisticsData* `<SADVSDAT.CPP>`
Organization	Keyed

Distribution class	Mirrored on Close
File copies	Current, Previous, Old, Weekly
Record length	58 bytes
Key length	2 bytes: GroupId

Store Record

Field Name	Type	Length	Decimal Offset	Description
GroupId	Int	2	0	'10001' defines this as the store record.
Period	Int	2	2	Identifier for period being compared to the current period: **1** Previous **2** Old
TimeStamp	PD	5	4	Date and time.
UserExit	ASCII	49	9	For use by user extensions.

Amount Record

Field Name	Type	Length	Decimal Offset	Description
GroupId	Int	2	0	Valid values for the amount record are 1 to 9999, for identifying which of the subtotal group IDs is being compared.
GrossTotal01	Int	4	2	Based on a weekly close before the first day of the week, this is the gross positive sales total for items sold in the subtotal group on Sunday, the first day of the week.
Delta01	Int	4	6	This is the difference between the gross positive sales total for Sunday and the current day of the week.
GrossTotal02	Int	4	10	The gross positive sales total for Monday.
Delta02	Int	4	14	Difference between the Monday amount and the current amount.
GrossTotal03	Int	4	18	The gross positive sales total for Tuesday.
Delta03	Int	4	22	Difference between the Tuesday amount and the current amount.

Field Name	Type	Length	Decimal Offset	Description
GrossTotal04	Int	4	26	The gross positive sales total for Wednesday.
Delta04	Int	4	30	Difference between the Wednesday amount and the current amount.
GrossTotal05	Int	4	34	The gross positive sales total for Thursday.
Delta05	Int	4	38	Difference between the Thursday amount and the current amount.
GrossTotal06	Int	4	42	The gross positive sales total for Friday.
Delta06	Int	4	46	Difference between the Friday amount and the current amount.
GrossTotal07	Int	4	50	The gross positive sales total for Saturday.
Delta07	Int	4	54	Difference between the Saturday amount and the current amount.

Count Record

Field Name	Type	Length	Decimal Offset	Description
GroupId	Int	2	0	'10000', which identifies this record as a department variance count record.
GrossCount01	Int	4	2	Based on a weekly close before Sunday (the first day of the week), this is the item count for items sold in the subtotal group on Sunday.
Delta01	Int	4	6	Difference between current and Sunday counts.
GrossCount02	Int	4	10	Number of items sold in the subtotal group on Monday.
Delta02	Int	4	14	Difference between current and Monday counts.
GrossCount03	Int	4	18	Number of items sold in the subtotal group on Tuesday.
Delta03	Int	4	22	Difference between current and Tuesday counts.
GrossCount04	Int	4	26	Number of items sold in the subtotal group on Wednesday.
Delta04	Int	4	30	Difference between current and Wednesday counts.
GrossCount05	Int	4	34	Number of items sold in the subtotal group on Thursday.

Field Name	Type	Length	Decimal Offset	Description
Delta05	Int	4	38	Difference between current and Thursday counts.
GrossCount06	Int	4	42	Number of items sold in the subtotal group on Friday.
Delta06	Int	4	46	Difference between current and Friday counts.
GrossCount07	Int	4	50	Number of items sold in the subtotal group on Saturday.
Delta07	Int	4	54	Difference between current and Saturday counts.

ACEFBPNT (Period Points Close Summary File)

The Period Points Close Summary file contains a summary of the statistics that SurePOS ACE collects while closing a points period. The data is used to create the Period Points Close Summary Report.

Logical names	`<ACEFBPNT>`
Data object reference	*ISPeriodPointsStatisticsData* `<PPSTATDT.CPP>`
Organization	Direct
Distribution class	Mirrored per update
File copies	1
Record length	226 bytes

Field Name	Type	Length	Decimal Offset	Description
EMProgramName	ASCII	20	0	EM program name.
StoreNumber	ASCII	4	20	Store number.
EndDate	ASCII	8	24	End/close date for points period. The format is YY/MM/DD.
CarryForwardLimit	ULong	4	32	Points carry forward limit.
OnlyPeriodPointsRedeeme d	Byte	1	36	Only period points can be redeemed indicator.
AliasRecord	Long	4	37	Number of records already processed for the period.
DoneRecord	Long	4	41	Number of records processed.
InactiveRecord	Long	4	45	Number of records inactive for the period.
LostDueToCarry	Long	4	49	Number of records that lost points due to carry over limit.

Field Name	Type	Length	Decimal Offset	Description
NegativePointsCarry	Long	4	53	Number of records with negative carry over limits.
NoPoints	Long	4	57	Number of records with no period points.
NoCarry	Long	4	61	Number of records with no points carried over.
PositivePointsCarry	Long	4	65	Number of records with positive points carried over.
TotalPointsCarry	Long	4	69	Total points carried over.
TotalPointsEarned	Long	4	73	Total points earned in period.
TotalNegativeCarry	Long	4	77	Total negative points carried over.
TotalPointsDiscarded	Long	4	81	Total points lost due to carry over limit
TotalPointsRedeemed	Long	4	85	Total points redeemed in the period.
TotalTransactions	Long	4	89	Total transactions in the period.
ActiveRecordsWithPoints	Long	4	93	Number of active records with period points.
TotalPoints	Long	4	97	Total points available for redemption before rollover.
Records	Long	4	101	Number of records found in the activity file.
BadRead	Long	4	105	Number of read errors encountered while processing the activity file.
ErrorsLogged	Long	4	109	Number of errors logged during processing.
LastCategory	Byte	1	113	Last error category from ISError encountered during processing of the points close
LastCode	Int	2	114	Last error code from ISError encountered during processing of the points close
PointRanges	ULong	44	116	Point ranges, as defined in the *Points ranges for period close report* option in Loyalty -> Activity -> Close Period personalization (11-element array of the ranges).
PointRangeStatistics	ULong	44	160	Count of customers in each range, as defined by the PointRanges field (11-element array of the range statistics).
DiscountGroups	Int	22	204	Discount Group Assignment per options for associated range, as defined in the *Automatic discounts for associated points ranges* option in Loyalty -> Activity -> Close Period personalization (11-element array of the discount groups).

ACEFIRLG (ACEFIRBL Log File)

The ACEFIRBL Log file contains records for each execution of the ACEFIRBL program. Each record contains ASCII data that includes the start and end time of the program execution, the number of input records that were processed, and information about any errors that occurred.

Logical name	`<ACEFIRLG>`
Data object reference	`<LOGFILE.CPP>`
Organization	Sequential
Distribution class	Mirrored per Update
File copies	Maximum number of files specified by LOGNAMES variable in the `acefirlg.ini` file (default files are current and previous)
Record length	Variable

This is an example of the possible file contents:

```
[1:ACEFIRBL]Fri Nov 08, 2013 09:13:28
Started processing:   11/08/13  09:13:28

[2:ACEFIRBL]Fri Nov 08, 2013 09:13:28
Renaming file: <ACECPNFR> to <ACECPNFT>

[3:ACEFIRBL]Fri Nov 08, 2013 09:13:28 Processing Basic:   101 : 09:13:28

[4:ACEFIRBL]Fri Nov 08, 2013 09:13:28 Processing GS1:       0 : 09:13:28

[5:ACEFIRBL]Fri Nov 08, 2013 09:13:28 Renaming file: <ACECBAST> to <ACECFBAS>

[6:ACEFIRBL]Fri Nov 08, 2013 09:13:28 Initiating reload:    11/08/13  09:13:28

[7:ACEFIRBL]Fri Nov 08, 2013 09:13:28 Renaming file: <ACECPNFT> to
 c:\ADX_IDT1\acecpnfr.ASV

[8:ACEFIRBL]Fri Nov 08, 2013 09:13:28 Completed processing: 11/08/13  09:13:28
```

ACEIMLST (Item Movement List Descriptors File)

The Item Movement List Descriptors file contains names and IDs for lists used to report item movement in the store. All records contain a list ID, a list description, and the number of item codes in the list. Any all-zero item code ends the list.

The prime version of this file is on the file server, and the image version is on the alternate file server. Distribution is performed at close.

Logical names	`<ACEIMLST>`
Data object reference	*ISItemReportingListStorage* `<ITMRLSTR.CPP>`
Organization	Fixed direct
Distribution class	Compound per Update
File copies	1
Record length	31 bytes

Field Name	Type	Length	Decimal Offset	Description
ListId	ASCII	4	0	The ID of the item movement list as defined in the Selective Item Report or in the Reporting List Editor.
Description	ASCII	18	4	The description of the item movement list as defined in the Selective Item Report or in the Reporting List Editor.

Field Name	Type	Length	Decimal Offset	Description
TimeStamp	PD	5	22	The time and date that this record was added to the file.
ItemCount	Int	4	27	The number of items in this list.

ACEITMLS (Item Movement Lists File)

The Item Movement Lists file contains lists of items for reporting item movement in the store. The file is keyed with records accessed by list ID.

The file contains one record for every item in every item movement list. Each record contains the item movement list ID and an item code. If there are three item movement lists, there are three sets of records in this file. If there are five item codes in one list, there are five records in this file for that item movement list.

The prime version of this file is on the file server, and the image version is on the alternate file server. Distribution is performed at close.

Logical names	`<ACEITMLS>`
Data object reference	*ISItemReportingListStorage* `<ITMRLSTR.CPP>`
Organization	Fixed direct
Distribution class	Compound per Update
File copies	1
Record length	18 bytes

Field Name	Type	Length	Decimal Offset	Description
ListId	ASCII	4	0	The ID of the item movement list as defined in the Selective Item Report or in the Reporting List Editor.
ItemCodes	ASCII	14	4	The item code of one item in the list identified by the ListId field.

ACEMSCTL (Miscellaneous Transaction Lists File)

The Miscellaneous Transaction Lists file saves the lists of miscellaneous transaction accounts in each subtotals group in the Miscellaneous Transactions Report that an operator edits in Miscellaneous -> Reporting List Editor -> Miscellaneous Transactions Report personalization.

The file contains one record for every miscellaneous transaction account that appears in the Miscellaneous Transactions report. Each record contains the subtotal group ID and an account ID. If there are 3 subtotal groups in the report, there are 3 sets of records in this file. If there are 5 accounts in one of the subtotal groups, there are 5 records in this file for that subtotal group.

Logical names	`<ACEMSCTL>`
Data object reference	*ISMiscellaneousTransactionReportingListStorage* `<MSTRLSTR.CPP>`
Organization	Fixed direct
Distribution class	Compound per Update

File copies	1
Record length	8 bytes

Field Name	Type	Length	Decimal Offset	Description
ListId	ASCII	4	0	The ID of the subtotal group where the account is reported in the Miscellaneous Transactions report.
AccountId	Int	4	4	The ID of one miscellaneous account that belongs to the subtotal group.

ACEMSTLD (Miscellaneous Transaction Subtotal Descriptors File)

The Miscellaneous Transaction Subtotal Descriptors file saves department subtotal group IDs and names from the Department Totals Report that an operator edits in Miscellaneous -> Reporting List Editor -> Miscellaneous Transactions Reportpersonalization.

Logical names	`<ACEMSTLD>`
Data object reference	*ISMiscellaneousTransactionReportingListStorage* `<MSTRLSTR.CPP>`
Organization	Fixed direct
Distribution class	Compound per Update
File copies	1
Record length	22 bytes

Field Name	Type	Length	Decimal Offset	Description
ListId	ASCII	4	0	Subtotal group ID assigned by SurePOS ACE as an operator uses the List Editor in Personalization to add a subtotal group to the Miscellaneous Transactions Report.
Description	ASCII	18	4	Name of subtotal group that an operator assigns in personalization.

ACEMTRKB (Matrix Keyboard Options File)

Operators use the SurePOS ACE Personalization interface (available from the SurePOS ACE Main Menu) to change option values for Matrix keyboards. SurePOS ACE stores the values in this personalization `dat` file.

For a description of options INI files, see "OPTNPARM (Options)" on page 534.

Logical names	`<ACEMTRKB>`
Organization	Random access
Distribution class	Compound per Update
File copies	1
Record length	9 bytes
User data	No

Field Name	Type	Length	Decimal Offset	Description
KeyCode	PI	2	0	Key code for a key that you want to define for this keyboard (65-250).
ItemCode	PS	7	2	The item code that corresponds to the key code.

Note: Each 1134 (46E) bytes holds the information for one keyboard. For example, keyboard 1 begins in position 0, and the next keyboard (number 2) begins in position 1134 or X'46E'. This table lists the positions at which each keyboard begins:

Keyboard Number	Position (Decimal)	Position (Hex)
1	0	0
2	1134	X'46E'
3	2268	X'8DC'
4	3402	X'D4A'
5	4536	X'11B8'
6	5670	X'1626'
7	6804	X'1A94'
8	7938	X'1F02'
9	9072	X'2370'

ACENGSL* (Negative Sales File)

The Negative Sales file is a direct file that SurePOS ACE uses to record negative sales amounts. There is one record per operator.

Logical name	`<ACENGSLC>`, `<ACENGSLP>`, `<ACENGSLO>`
Organization	Fixed direct
Distribution class	Mirrored on Close
File copies	Current, Previous, Old
Record length	64 bytes

Field Name	Type	Length	Decimal Offset	Description
Terminal	PD	2	0	Terminal
TransNum	PD	2	2	TransNum
Operator	PD	5	4	Operator
TimeStamp	PD	5	9	TimeStamp
GrossPositive	Int	4	14	GrossPositive
GrossNegative	Int	4	18	GrossNegative
TotalCoupons	Int	4	22	TotalCoupons
ItemCount	Int	4	26	ItemCount

Field Name	Type	Length	Decimal Offset	Description
Reserved	ASCII	34	30	Reserved

ACEOPTLG (Personalization Options Changed Log File)

The ACEOPTLG Log file contains records of changes made to Personalization options by using the GUI, the ACEPERSL /U parameter from the command line or via the DIF XML Update Options Request.

The maximum size of the `aceoptlg.dat` file is controlled by the SIZE parameter in the `aceoptlg.ini` file. The LOGNAME parameter of the `aceoptlg.ini` file lists the file names to be used when saving the log file if the size exceeds the maximum size specified on the SIZE option. (See "ACEOPTLG (Personalization Options Changed Log)" on page 500 for more information about both parameters.)

Logical name	`<ACEOPTLG>, <ACEOPTLP>`
Data object reference	`<OPCHLGFL.CPP>`
Organization	Sequential
Distribution class	Mirrored per Update
File copies	Maximum number of files specified by the LOGNAME variable in the `aceoptlg.ini` file (default files are current and previous).
Record length	Variable

ACERDESC (Report Descriptors File)

The Report Descriptors file contains descriptors for each default report.

An image version of this file is kept on the alternate file server. This distribution takes place only when the file is closed to minimize the performance impact.

Logical name	`<EAMRDESC>`
Data object reference	*ISReportDescriptorStorage* `<DSCRPSTR.CPP>`
Organization	Variable sequential
Distribution class	Compound per Update
File copies	1
Record length	Variable

You can examine this file through the Descriptor Editor, which is available from the Miscellaneous pull-down of the Personalization menu.

ACERXFIL (Pharmacy File)

The Pharmacy File contains information about prescriptions from the pharmacy for which payment has not been received. When a prescription is filled or refilled on the pharmacy system, an entry is written to this file.

Logical name	`<ACERXFIL>`
Organization	Keyed

Distribution class	Compound on update
File copies	1
Record length	254 bytes

Table 38. ACERXFIL.DAT Layout

Field Name	Type	Length	Decimal Offset	Description
RXTRANSNUM	PD	6	0	Prescription transaction number
FLAG	Int	2	6	Flag bits X'8000' - Reserved X'4000' - Reserved X'2000' - Reserved X'1000' - Reserved X'0800' - Reserved X'0400' - Item sold on 4690 system X'0200' - Reserved X'0100' - Reserved X'0080' - Line item discountable X'0040' - Reserved X'0020' - Reserved X'0010' - Reserved X'0008' - Reserved X'0004' - Reserved X'0002' - Reserved X'0001' - Reserved
DEPARTMENT	Int	2	8	Department number
Reserved		1	10	Reserved
PRICE	PD	3	11	Prescription selling price (00 - 9999.99)
TOTALPRI	PD	3	14	Total prescription price
Reserved		49	17	Reserved
TRANSNO	PD	2	66	POS transaction number
Reserved		73	68	Reserved
REGISNO	PD	2	141	POS terminal number
OPERNO	PD	5	143	Operator number
Reserved	ASCII	8	148	Reserved
POSTIME	PD	2	156	Time in format `HHMM` that prescription was sold
POSDATE	PD	3	158	Date in format `YYMMDD` that prescription was sold
Reserved		17	161	Reserved
SALE.CTR	Int	1	178	Sales counter
CHECKPOINT	Int	4	179	Checkpoint restart value
Reserved		71	183	Reserved for future use

A Java back-office application is responsible for reading from or writing to the acerxfil.dat file. Fields that are specified in Table 38 and that are not described as reserved may be updated by the application. See “Developing a Pharmacy Interface” on page 631 for more information.

ACESDEF (Manager's Terminal Defaults File)

The Manager's Terminal Defaults file does not exist when you install SurePOS ACE. The file is created the first time that a report is requested using ACETVREL.

The `acesdef.dat` file is organized like an INI file. For each report that you can request through the Reports function (which invokes ACETVREL), this file contains a section that has the values that were specified the last time that the report was requested. The values are used as the default values the next time that the report is requested. Any changes that are made while requesting a report will be saved and used as the default values the next time that the report is requested.

Logical name	`<ACESDEF>`
Distribution class	Local
File copies	1

ACESDESC (Sales Descriptors File)

The Sales Descriptors file contains descriptors that are used by the Terminal Sales application.

An image version of this file is kept on the alternate file server. Distribution of the file takes place only when the file is closed to minimize the performance impact.

Logical name	`<ACESDESC>`
Data object reference	*ISSalesDescriptorStorage* `<DSCRPSTR.CPP>`
Organization	Variable sequential
Distribution class	Compound per Update
File copies	1
Record length	Variable

You can examine this file through the Descriptor Editor, which is available from the Miscellaneous pull-down of the Personalization menu.

ACESRCPC (Suspend/Retrieve Checkpoint File)

The Suspend/Retrieve Checkpoint file keeps a pointer to the Suspended Transactions file (eamsrtrx.dat) and also contains checkpoint information about suspend/retrieve processing.

Logical names	<ACESRCPC>, <LCLSRCPC:>
Data object reference	*ISSuspendedTransactionCheckPointStorage* <`SSTCPSTR.CPP`>
Organization	Direct
Distribution class	Prime Copy: Mirrored per update Local: Local
File copies	1
Record length	5 bytes

Field Name	Type	Length	Decimal Offset	Description
SuspendTransPtr	Long	4	0	This is a pointer to the last record read from eamstrx.dat, the Suspended Transactions file.
Stage	Byte	1	4	One of the following checkpoint values that represent the suspend/retrieve stage: • For Suspend Transaction: 0 - BeginNewTransaction 1 - StoreUpdated 2 - TerminalUpdated 3 - OperatorUpdated 4 - TransactionComplete • For Close Suspend Processing: 5 - BeginClose 6 - DeleteOldSeqStage 7 - DeleteOldKeyedStage 8 - RenameKeyedStage 9 - CreateNewKeyedStage 10 - MergeStoreStage 11 - MergeTerminalAndOperatorsStage 12 - DeleteLongStage 13 - RenameTempStage 14 - DeleteOldLongStage 15 - RenameLongStage 16 - CreateNewLongStage 17 - MovePreviousIndexStage 18 - RenameSeqStage 19 - CreateNewSeqStage 20 - MoveSuspendsStage 21 - CloseComplete
Merge Sub-Stage	Int	2	5	Substage value that is used for store close recovery when processing the MergeTerminalsAndOperatorsStage.
DateTime	PI	5	7	The date and time of the close, in the format `YYMMDDHHmm`.

ACETIRLG (ACETIRBL Log File)

The ACETIRBL Log File contains records of significant program execution steps of the ACETIRBL program.

The maximum size of the acetirlg.dat file is controlled by the SIZE parameter in the `acetirlg.ini` file. The LOGNAME parameter of the `acetirlg.ini` file lists the file names to be used when saving the log file if the size exceeds the maximum size specified on the SIZE option. (See "ACETIRLG (ACETIRBL Log Configuration)" on page 526for more information about both parameters.)

Logical names	<ACETIRLG>, <ACETIRLP>
Data object reference	<`LOGFILE.CPP`>
Organization	Sequential
Distribution class	Mirrored per update
File copies	Maximum number of files specified by LOGNAMES variable in `acetirlg.ini` file (default files are current and previous)
Record length	Variable

ACETOFSF (Terminal Offline Status File)

The Terminal Offline Status file is used to determine which terminals have the latest level of the terminal Item Record file and to track the despool status of each terminal.

The file is created by ACECSMLL at system IPL if the file does not already exist. The file creation occurs whether or not the TOF Enabled flag is turned on in personalization.

Note: In order to keep the acetofsf.dat file up-to-date, the associated Terminal record in acetofsf.dat is deleted when a current period record in eamterms.dat is deleted during close processing.

The file contains one Store record and a Terminal record for each terminal in the system. The allocated size of the file is based on 1200 records.

Logical names	<ACETOFSF>
Data object reference - Store record	*ISStoreOfflineStatusData* <`SOFSTDAT.CPP`>
Data object reference - Terminal record	*ISTerminalOfflineStatusData* <`TOFSTDAT.CPP`>
Organization	Keyed
Distribution class	Mirrored at close
File copies	1
Record length	20 bytes

Store Record

If theBuildDateTime field in the Store record contains X'000000000000', the terminal Item Record file is being rebuilt and no terminal can load the file. The *ExitInhibit* flag, which is controlled by the Close Reporting Totals application, is used to inhibit terminals from exiting standalone while a store close is being initiated.

Field Name	Type	Length	Decimal Offset	Description
TerminalID	PD	2	0	X'9999' = Store record
BuildDateTime	PD	6	2	Date of the last build in the format `YYMMDDHHmmss`

Field Name	Type	Length	Decimal Offset	Description
ExitInhibit	Int	1	8	Flag to inhibit exiting standalone: 0 - Terminals may exit standalone -1 - Terminals may not exit standalone
Reserved	ASCII	11	9	Reserved

Terminal Record

When loading of a terminal Item Record file is complete, the terminal updates the Terminal record to contain the build level of the terminal Item Record file. If the BuildDateTime field in the Terminal record contains X'000000000000', the terminal does not have a terminal Item Record file.

The *DespoolTrxActive* flag is set on while the terminal is uploading saved transaction data to the controller. If the flag is on and the *Despool of TOF transaction* option is set to 0 in personalization, an unattended request to close reporting totals waits until all data has been despooled.

Field Name	Type	Length	Decimal Offset	Description
Terminal	PD	2	0	Terminal ID
BuildDateTime	PD	6	2	Date of the current terminal Item Record file in the format `YYMMDDHHmmss`
DespoolTrxActive	Int	1	8	Flag indicator for despooling of transaction data: 0 - Despooling is not active -1 - Despooling is active
DespoolEJActive	Int	1	9	Flag indicator for despooling of electronic journal (EJ) data: 0 - Despooling is not active -1 - Despooling is active
Reserved	ASCII	10	10	Reserved

ACETSVER (ACE Terminal Sales Version File)

The ACETSVER file contains a record for each terminal in the store. The record contains the version of the terminal that is actively running along with the time that the terminal last initialized or exited Terminal Offline (TOF) mode.

The size of the file is calculated based on the Maximum number of terminal fields in Personalization (Options->Store->Accounting). Significant room is added to allow for expansion to more terminals than stated in the field in Personalization.

Logical name	<ACETSVER>

Data object reference	*ISTerminalSalesVersionData* <`TSVERDAT.CPP/HPP`>
Organization	Keyed
Distribution class	Compound per Update
File copies	1
Record length	72 bytes
Key length	2 bytes; terminal number

Field Name	Type	Length	Decimal Offset	Description
Terminal	PD	2	0	Terminal number.
DateTime	PD	5	2	Date/Time of the last terminal initialization or exit from TOF.
Terminal Version	ASCII	20	7	Version of terminal sales currently loaded.
Toshiba Reserved		30	27	Reserved for future use.
User Reserved		15	57	Reserved for customer use.

ACEUCCTL (Unattended Close Control File)

The Unattended Close Control file contains the parameters that control each unattended control procedure.

Logical name	<ACEUCCTL>
Data object reference	*ISSAUCTriggerData* <`SAUCTDAT.CPP`>
Data storage reference	*ISSAUCTriggerStorage* <`SAACTSTR.CPP`>
Organization	Keyed
Distribution class	Mirrored at close
File copies	1
Record length	48 bytes

Field Name	Type	Length	Decimal Offset	Description
Id	PD	3	0	MM+DD+HH; For a "Day" format ID, the value of MM will be x'00', and the value of DD will be a zero plus the day of the week (with 1=Sunday & 7=Saturday). For a "Date" format ID, MM will be the month of the year and DD will be the day of the month. HH is the hour of the day for both formats.
HostSequenceNumber	PD	1	3	Numbered occurrence of a unique trigger ID; Starts at x'00'.
RequestType	PD	1	4	x'00'
TriggerType	Int	1	5	x'A8': • x'80' = Operator cannot delete

Field Name	Type	Length	Decimal Offset	Description
				• x'20' = Scheduler controlled maintenance • x'08' = Start program
UC_Flags	Int	1	6	Flags bits: x'80' - Reserved x'40' - Reserved x'20' - Reserved x'10' - Reserved x'08' - Reserved x'04' - Reserved x'02' - Reserved x'01' - Cancel unattended close if sales are below minimum.
ExecutionTime	PD	3	7	MMDDhh - the month (MM), date(DD), and hour (hh) of the last execution of the program
Close Year	PD	2	10	YYYY; For a "Day" format ID, the value will be a '0000'. For a "Date" format ID, the value will be the year of the close.
TriggerFile	ASCII	8	12	Blank
TargetFile	ASCII	8	20	"ACEACSCL" - name of the unattended close program
Output File	ASCII	8	28	"ACEUCRDP" - name of the read pipe for unattended close
Close Number	PD	2	36	Close number selection
Close Minute	PD	2	38	Minute (mm) within an hour at which to execute the unattended close
Minimum Sales Amount	PD	4	40	Range: $0.00 - $999,999.99
Status	PD	1	44	x'00'
Reserved	ASCII	2	45	Reserved

ACEVCODE (Full-Screen Velocity Code Help File)

The adx_idt1\acevcode.dat file defines default velocity codes. It is a text file that you can edit. When using SurePOS ACE in full-screen mode, you can display the Velocity Code Chart. These are the keys that let you display the chart and navigate through it:

245	Displays the Velocity Code Chart on a full-screen terminal
246	Pages up through the chart 13 lines at a time
247	Pages down through the chart 13 lines at a time
248	Scrolls up through the chart one section at a time

249	Scrolls down through the chart one section at a time

Note: A section is the set of lines between asterisks in the ACEVCODE.DAT file. In the example file that ships with SurePOS ACE, a section is one line. You can customize the file to put multiple lines in each section.

Logical names	<ACEVCODE>
Organization	Sequential
Distribution class	Mirrored per update
File copies	1
Record length	81 bytes

SurePOS ACE provides an example acevcode.dat file that you can customize to satisfy your requirements. Figure 1 shows the beginning of the file that ships with SurePOS ACE.

```
*
      ===================================================================
      ==                                                               ==
      ==                        PRODUCE CODES                          ==
      ==                                                               ==
      ===================================================================

      ===================================================================
      ==                      NUMERICAL SEQUENCE                       ==
      ===================================================================
*
      4011 BANANA DOLE         4247 STRAWBERRIES        4551 BRUSSEL SPRT
*
      4012 ORANGES             4248 STRAWBERRIES        4552 NAPPA
*
      4013 NAVEL ORANGE        4251 STEM STRBRY         4553 PEAR TYL GLD
*
      4015 RED DEL APPLE       4252 FLAT STRBRYS        4554 RED CABBAGE
*
      4016 RED DEL APPLE       4253 STRAWBERRIES        4555 SAV CABBAGE
*
      4017 GRANNY SMITH        4254 BBRYS QT            4562 CARROTS
*
      4019 MCINTOSH APPLE      4256 CARAMBOLA           4566 CAUL NUGGTS
```

```
*
      4020 GLD DEL APPLE       4257 FRI CHERIMOY        4567 BROCCOFLOWER
*
      4021 GOLD DEL APPLE      4258 RAINIER CHER        4575 CELERY HEART
*
      4022 THOMPSON GRP        4261 COCONUT             4585 CELERY ROOT
*
      4023 GRAPES FLAME        4265 FEIJOAS             4590 CORN
*
      4026 PEARS               4267 FIGS MISSION        4593 CUCUMBERS
*
      4027 GRPFRT RED          4270 GRP BLK SEED        4596 CUCUMBERS
*
      4028 STRAWS PINT         4272 CONCRD GRAPE        4598 FRIEDA DAIKO
*
      4030 KIWI                4273 GRP RD GLOBE        4604 ENDIVE LTCE
*
      4031 WATERMELON          4275 ITALIA GRAPE        4605 ESCAROLE
*
      4032 WATERMLN SDL        4281 GRFRT RIOSTA        4608 GARLIC JBO
*
      4033 SM LEMONS           4284 GRPFRT RED          4612 GINGER ROOT
*
      4034 HONEYDEWS           4287 GRFRT RIOSTA        4614 COLLARD GRN
```

Figure 1. Example ACEVCODE.DAT File - Partial

APSOPTNS (SurePOS ACE EPS Personalization Options File)

Operators use the SurePOS ACE Personalization interface (available from the SurePOS ACE Main Menu) to change option values in personalization DAT files.

Operators use the SurePOS ACE Personalization interface (available from the SurePOS ACE Main Menu) to change option values in personalization DAT files:

Table 39. Companion Options INI and DAT Files

Options INI File	Options DAT File	Logical DAT File Name	Description
`optnparm.ini`	eamoptns.dat	<EAMOPTNS>	Base personalization options
`acecpntr.ini`	acecate5.dat	<ACECATE5>	Coupon translation options
`nlsparms.ini`	nlsoptns.dat	<NLSOPTNS>	National language support (NLS) personalization options
`apsparms.ini`	apsoptns.dat	<APSOPTNS>	SurePOS ACE EPS personalization options

Each personalization options dat file is a random access file that contains values for all options declared in its companion `ini` file. SurePOS ACE creates each dat file from its companion `ini` file if the dat file does not exist.

For a description of options `ini` files, see "OPTNPARM (Options)" on page 534.

For a description of how to use DIF along with the ACEPERSL command, see "Using ACEPERSL to Maintain Personalization Options" on page 572.

Logical name	<APSOPTNS>
Data object reference	*ISOptions* <OPTIONS.CPP/HPP>
Organization	Random access
Distribution class	Compound at close
File copies	1
Record length	512 bytes
User data	In the `INI` files

ACEWERCL.DAT (WIC EBT Claim/Reconciliation Status File)

acewercl.dat is a SurePOS ACE file that contains the status of these WIC EBT files:

- Claim files that have been submitted by the store
- Reconciliation responses that have been received from the WIC EBT agency

This file contains entries for all WIC EBT agencies for which claim files are built in the store.

A record is added to this file each time that Checkout Support builds a claim file. The record is automatically updated when WIC EBT Inbound Processing detects an agency response file for that claim. The response file can be either an Auto Reconciliation File or an Error File.

During a daily close, any records over ninety days old are removed from the file.

Logical name	<ACEWERCL>

Data object reference	*ISWICEBTReconcileData*
Organization	Keyed
Distribution class	Mirror on close
File copies	1
Record length	35 bytes
Key length	8 bytes

Table 40. Reconciliation Tracking Record

Field Name	Type	Length	Decimal Offset	Description
Tracking ID	ASCII	8	0	An identifier created from the file name of the files associated with this entry. The format of the identifier is `AAYJJJnn` where `AA` is the agency ID, `Y` is the last digit of the year, `JJJ` is the Julian day of the year, and `nn` is a sequence number that is reset to 00 at the beginning of each day.
Create Date	PD	4	8	Date that the claim file was created. The format of the date is `YYYYMMDD` where `YYYY` is the year, `MM` is the month, and `DD` is the day of the month.
Create Time	PD	3	12	Time that the claim file was created. The format of the time is `HHMMSS` where `HH` is the hour, `MM` is the minutes, and `SS` is the seconds.
Claim Total	PD	6	15	Total amount that was claimed by this file. Two decimal places are implied.
Settlement Total	PD	6	21	Total settlement amount specified in the Auto Reconciliation file from the WIC EBT agency. Two decimal places are implied.
Settlement Date	PD	4	27	Date that the settlement amount will be paid. The format of the date is `YYYYMMDD` where `YYYY` is the year, `MM` is the month, and `DD` is the day of the month.
Response Date	PD	3	31	Date that the most recent file was received. The format of the date is `YYMMDD` where `YY` represents the year, `MM` is the month, and `DD` is the day of the month.
Status Flag	ASCII	1	34	Flag that indicates the current status of this claim. A - Reconciliation file was received with a matching settlement total C - Reconciliation file was corrupted E - Operator Performance Database Ledger P - Claim has been prepared X - Reconciliation file was received with a different settlement total

Field Name	Type	Length	Decimal Offset	Description
				E - Operator Performance Database Ledger

ACEWE??.APL (WIC EBT Approved Product List File)

acewe??.apl is the SurePOS ACE version of the Approved Product List file that is provided by a WIC EBT agency. ACEWE??.APL contains only the fields from the Approved Product List file that SurePOS ACE requires to process WIC EBT transactions.

This file is created when WIC EBT Inbound Processing detects a new file from a WIC EBT agency.

The file name has these components:

- adx_idt1:acewe, which derives from the logical name <WIC1:>
- The two-character agency ID, which is represented by ??
- A file extension of .apl, which indicates an Approved Product List

Logical name	None
Data object reference	*ISWICEBTUPCListData*
Organization	Keyed
Distribution class	Compound on close
File copies	1 per WIC EBT agency supported by the store
Record length	39 bytes
Key length	7 bytes (length of UPC)

Field Name	Type	Length	Decimal Offset	Description
UPC	PD	7	0	UPC, right justified with leading zeroes, no check digit
UPC Check Digit	ASCII	1	7	Check digit for the UPC
Category	PD	1	8	Food category assigned by the WIC EBT agency
SubCategory	PD	2	9	Food subcategory assigned by the WIC EBT agency
BenefitQty	PD	3	11	Quantity of benefit units that are required to redeem this item
BenefitUnit	ASCII	10	14	Short description of the unit of measure used for benefit
ItemPrice	PD	3	24	Reference price for the item
PriceType	PD	1	27	Type of price in the ItemPrice field: **0** No price **1** Maximum price

Field Name	Type	Length	Decimal Offset	Description
				2 Contractual price **3** Not subject to maximum price controls (used for all CVV items)
DateEffective	PD	3	28	Date that the product authorization is effective. The format of the date is `YYMMDD` where `YY` represents the year, `MM` is the month, and `DD` is the day of the month.
DateEnd	PD	3	31	Date that the product authorization expires. The format of the date is `YYMMDD` where `YY` represents the year, `MM` is the month, and `DD` is the day of the month.
ZeroSubCatFlag	PD	1	34	Flag that indicates the subcategories against which the item can be redeemed. **0** Item can only be used for the subcategory, either zero or nonzero, that is specified for it. **1** Item can be used for the nonzero subcategory that is specified for it and also for the zero subcategory.
UPC/PLU Indicator	PD	1	35	First two digits of the UPC/PLU field from the input APL file. The first digit indicates whether the item code is a UPC (`0`) or a PLU (`1`).
Reserved	PD	3	36	Reserved for future use

ACEWE??.DES (WIC EBT Description File)

acewe??.des contains the descriptors that SurePOS ACE requires to process WIC EBT transactions.

SurePOS ACE builds the acewe??.des file from the sequential WIC EBT file when WIC EBT Inbound Processing detects a new file from a WIC EBT agency.

The file name has these components:

- adx_idt1:acewe, which derives from the logical name <WIC1:>
- The two-character agency ID, which is represented by `??`
- A file extension of .des, which indicates a Description file

Logical name	None
Data object reference	*ISWICEBTDescriptionListData*
Organization	Keyed
Distribution class	Compound on close

File copies	1 per WIC EBT agency supported by the store
Record length	46 bytes
Key length	3 bytes (length of Category/Subcategory field)

Field Name	Type	Length	Decimal Offset	Description
Category / Subcategory	PD	3	0	Concatenation of the Category and Subcategory with an added leading zero. (If the category is 04 and the subcategory is 003, the packed-decimal value in this field is 004003.)
BenefitUnit	ASCII	10	3	Short description of the measure of benefit issuance. Possible values are lb, ounce, dozen, and can.
Subcategory Description	ASCII	33	13	Product description of this category / subcategory item.

ACEWE??.HCL (WIC EBT Hot Card List File)

acewe??.hcl is the SurePOS ACE version of the Hot Card List file that is provided by a WIC EBT agency. acewe??.hcl contains only the fields from the Hot Card List file that SurePOS ACE requires to process WIC EBT transactions.

This file is created when WIC EBT Inbound Processing detects a new file from a WIC EBT agency.

The file name has these components:

- adx_idt1:acewe, which derives from the logical name <WIC1:>
- The two-character agency ID, which is represented by `??`
- A file extension of .hcl, which indicates a Hot Card List

Logical name	None
Data object reference	*ISWICEBTHotCardListData*
Organization	Keyed
Distribution class	Compound on close
File copies	1 per WIC EBT agency supported by the store
Record length	22 bytes
Key length	10 bytes (length of PAN)

Field Name	Type	Length	Decimal Offset	Description
PAN	PD	10	0	Participant account number (PAN), right justified with leading zeroes
PANLength	PD	1	10	Length of the PAN, excluding leading zeroes
DateTimeBad	PD	7	11	Date and time that the PAN is declared not valid. Format of the date and time is `YYYYMMDDhhmmss` where `YYYY` is the year, `MM` is the month, `DD` is the day of the month, `hh` is

Field Name	Type	Length	Decimal Offset	Description
				the hour, `mm` is the minutes, and `ss` is the seconds.
Reason Code	ASCII	4	18	Reason code for adding the PAN to the hot list. Note: Some agencies (for example, New Mexico) may choose not to use this field, and it will then be all spaces.

EAMACC* (Accounting Totals File)

The keyed Accounting Totals file contains totals data for operators or terminals or both. A separate file exists for the current period, the previous period, and one old period.

When Foreign Currency is not enabled:

- Five records exist for the store account (record types 0, 4-7).
- Five records exist for each operator who worked on an operator-accountability terminal (record types 2, 4-7).
- Five records exist for each terminal using terminal accountability (record types 3, 8-B).

When Foreign Currency is enabled, the record count for each account type (store, operator, terminal) increases to nine. Additional records are present when Foreign Currency is enabled, to track the foreign currency amounts in the original value of the entered currency:

- Nine records exist for the store account (record types 0, 4-7, C-F).
- Nine records exist for each operator who worked on an operator-accountability terminal (record types 2, 4-7, C-F).
- Nine records exist for each terminal using terminal accountability (record types 3, 8-B, G-J).

The file is sized based on the number of terminals and operators defined in personalization and whether or not Foreign Currency is enabled.

Accounting Totals can be closed *short* or *long*, where a long close includes the totals from several short closes. This allows for daily as well as weekly totals. A separate file exists for the current period, previous period, and one old period, which is a total of six Accounting Totals files.

All accounting files and records are created and maintained by SurePOS ACE. An image version of each Accounting file is on the alternate file server. Distribution takes place at file close to minimize the performance impact.

Note for SA Users

See Table 1 for a description of how the SurePOS ACE file differs from the 4680-4690 Supermarket Application file.

Logical name	<EAMACCTC>, <EAMACCTP>, <EAMACCTO>, <EAMACCCL>, <EAMACCPL>, <EAMACCOL>, <EAMACCWK>
Data object reference	*ISManagedAccount* <`MNGDASTR.CPP`>
Organization	Keyed
Distribution class	Mirrored at close
File copies	Current, Previous, Old, Current short, Previous short, Weekly
Record length	512 bytes
Key length	6 bytes: Type/Store, Type/Terminal, or Type/Operator

Store Record

Field Name	Type	Length	Decimal Offset	Description
RecordType	ASCII	1	0	Record type = 1 for the store record.
StoreId	PD	5	1	0 for the store record; Operator/terminal ID field in Operator/Terminal record; AccountID in Tender Totals by Variety record.
TimeStamp	PD	5	6	Date and Time when current period started or previous period ended (format: `YYMMDDHHmm`).
INDICAT0	Int	1	11	**Bit 0 X'80'** IsReconciled: bit flag indicating reconciliation status. Indicates that store office totals have been reconciled for the period. Based on whether carry forward office tender has run. **Bit 1 X'40'** IsPrinted: bit flag indicating store office accounting totals have printed since the last update to the record. **Bit 2 X'20'** Reserved **Bit 3 X'10'** Reserved **Bit 4 X'08'** Reserved **Bit 5 X'04'** AutoPickupDone: bit flag indicating that Auto Pickup has completed. **Bit 6 X'02'** AutoCouponsCounted: bit flag indicating that Auto Coupons have been counted. **Bit 7 X'01'** AutoReportsActive: bit flag indicating that Auto Reports are

Field Name	Type	Length	Decimal Offset	Description
				running in the background and have not yet completed.
LastTerminal	PD	2	12	Terminal which last updated this record.
Restart	Int	4	14	This field is used as a control field on a system restart. It is used to determine if this record has been updated by a particular transaction.
GrossPlus	Int	4	18	Gross positive. Accumulation of positive sales entries for all terminals in store. Updated when an operator signs off but does not include totals from voided or training transactions. *(sales + deposits + taxes + tender fees)* This total is not reset by the close of a reporting period but accumulate continuously.
GrossMinus	Int	4	22	Gross negative. Accumulation of negative sales entries for all terminals in the store. Updated when an operator signs off but does not include totals from voided or training transactions. *This is a positive amount. (deposit returns + item refunds + tax refunds + discounts + cancels of sales, deposits, item refunds, deposit returns, and tender fees)* This total is not reset by the close of a reporting period but accumulate continuously.
SalesTransactionCount	Int	4	26	Number of sales transactions
LoanTenderAmount	Array	8x4	30	This represents the following eight 4-byte tender type totals for loans made during the period:
AMTCASH	Int	4	30	Sum of loans - cash
AMTCHECK	Int	4	34	Sum of loans - checks
AMTFOODS	Int	4	38	Sum of loans - food stamps
AMTMISC1	Int	4	42	Sum of loans - miscellaneous tender 1
AMTMISC2	Int	4	46	Sum of loans - miscellaneous tender 2
AMTMISC3	Int	4	50	Sum of loans - miscellaneous tender 3
AMTMANUF	Int	4	54	Sum of loans - manufacturer coupons
AMTSTORE	Int	4	58	Sum of loans - store coupons

Field Name	Type	Lengt h	Decimal Offset	Description
PickupTenderAmount	Array	8x4	62	This represents the following eight 4-byte tender type totals for pickups made during the period:
AMTCASH	Int	4	62	Sum of pickups - cash
AMTCHECK	Int	4	66	Sum of pickups - checks
AMTFOODS	Int	4	70	Sum of pickups - food stamps
AMTMISC1	Int	4	74	Sum of pickups - miscellaneous tender 1
AMTMISC2	Int	4	78	Sum of pickups - miscellaneous tender 2
AMTMISC3	Int	4	82	Sum of pickups - miscellaneous tender 3
AMTMANUF	Int	4	86	Sum of pickups - manufacturer coupons
AMTSTORE	Int	4	90	Sum of pickups - store coupons
CountedTenderAmount	Array	8x4	94	This represents the following eight 4-byte tender type totals for tender counts made during the period. These totals represent the tender count when the store safe is *counted down*. Each of the eight total fields follows:
AMTCASH	Int	4	94	Counted tender - cash
AMTCHECK	Int	4	98	Counted tender - checks
AMTFOODS	Int	4	102	Counted tender - food stamps
AMTMISC1	Int	4	106	Counted tender - miscellaneous tender 1
AMTMISC2	Int	4	110	Counted tender - miscellaneous tender 2
AMTMISC3	Int	4	114	Counted tender - miscellaneous tender 3
AMTMANUF	Int	4	118	Counted tender - manufacturer coupons
AMTSTORE	Int	4	122	Counted tender - store coupons
NetTenderAmount	Array	8x4	126	This represents the following eight 4-byte net tender totals for tenders made during the period. Totals represent the change in the net tender position as a result of miscellaneous transactions, loans, pickups, and tender counts. Totals reflect the *short/over* position of the safe. Each of the eight total fields follows:
AMTCASH	Int	4	126	Net tender - cash
AMTCHECK	Int	4	130	Net tender - checks
AMTFOODS	Int	4	134	Net tender - food stamps
AMTMISC1	Int	4	138	Net tender - miscellaneous tender 1

Field Name	Type	Length	Decimal Offset	Description
AMTMISC2	Int	4	142	Net tender - miscellaneous tender 2
AMTMISC3	Int	4	146	Net tender - miscellaneous tender 3
AMTMANUF	Int	4	150	Net tender - manufacturer coupons
AMTSTORE	Int	4	154	Net tender - store coupons
OpeningTenderAmount	Array	8x4	158	This represents the following eight 4-byte tender totals for tenders left in the store safe after the bank deposit was made for the last period:
AMTCASH	Int	4	158	Opening balance - cash
AMTCHECK	Int	4	162	Opening balance - checks
AMTFOODS	Int	4	166	Opening balance - food stamps
AMTMISC1	Int	4	170	Opening balance - miscellaneous tender 1
AMTMISC2	Int	4	174	Opening balance - miscellaneous tender 2
AMTMISC3	Int	4	178	Opening balance - miscellaneous tender 3
AMTMANUF	Int	4	182	Opening balance - manufacturer coupons
AMTSTORE	Int	4	186	Opening balance - store coupons
MiscTransactionAmount	Int	4	190	Miscellaneous transactions amount credited to the store by the Controller procedure.
EFTData	PD	100	194	Reserved for SurePOS ACE EPS use.
PeriodTimeStamp	PD	5	294	Date and time when current period started or previous period ended (format: `YYMMDDHHmm`) updated only by short close.
TaxableExemptAmount	Int	4	299	Taxable amount exempted
TaxExemptAmountA	Int	4	303	Amount of tax A exempted
TaxExemptAmountB	Int	4	307	Amount of tax B exempted
TaxExemptAmountC	Int	4	311	Amount of tax C exempted
TaxExemptAmountD	Int	4	315	Amount of tax D exempted
TaxableExemptAmountA	Int	4	319	Amount of taxable A exempted
TaxableExemptAmountB	Int	4	323	Amount of taxable B exempted
TaxableExemptAmountC	Int	4	327	Amount of taxable C exempted

Field Name	Type	Lengt h	Decimal Offset	Description
TaxableExemptAmountD	Int	4	331	Amount of taxable D exempted
TaxExemptAmountE	Int	4	335	Amount of tax E exempted
TaxExemptAmountF	Int	4	339	Amount of tax F exempted
TaxExemptAmountG	Int	4	343	Amount of tax G exempted
TaxExemptAmountH	Int	4	347	Amount of tax H exempted
TaxableExemptAmountE	Int	4	351	Amount of taxable E exempted
TaxableExemptAmountF	Int	4	355	Amount of taxable F exempted
TaxableExemptAmountG	Int	4	359	Amount of taxable G exempted
TaxableExemptAmountH	Int	4	363	Amount of taxable H exempted
Reserved	PD	17	367	Reserved
ManAutoCouponAmount	Int	4	384	Electronic manufacturer auto coupon amount total.
StoreAutoCouponAmount	Int	4	388	Electronic store auto coupon amount total.
DoubledCouponAmount	Int	4	392	Double coupon amount total.
PCTransactionAmount	Int	4	396	Preferred customer transaction amount total.
PCTransactionCount	Int	2	400	Preferred customer transaction count total.
PCAutoCouponCount	Int	2	402	Preferred customer auto coupon count total.
PCAutoCouponAmount	Int	4	404	Preferred customer auto coupon amount total.
UsrInteger	Int	60	408	Fifteen 4-byte integers for use with user extensions.
UsrReserved	ASCII	40	468	Reserved for data.

Operator/Terminal Record

Field Name	Type	Length	Decimal Offset	Description
RecordType	ASCII	1	0	**2** Operator record

Field Name	Type	Length	Decimal Offset	Description
				3 Terminal record
AccountId	PD	5	1	Operator or terminal ID
TimeStamp	PD	5	6	Date and Time of transaction that last updated this record (format: `YYMMDDHHmm`).
INDICAT0	Int	1	11	**Bit 0 X'80'** IsReconciled: bit flag indicating reconciliation status. Indicates that the positive terminal record has been reconciled for the period. Based on whether a transfer tender to store office has run. **Bit 1 X'40'** IsPrinted: bit flag indicating the operator or terminal totals have printed since the last update to the record. **Bit 2 X'20'** IsMissingTransactions: bit flag indicating that the system is missing TLog records for this operator or terminal. **Bit 3 X'10'** Reserved **Bit 4 X'08'** Reserved **Bit 5 X'04'** AutoPickupDone: bit flag indicating that Auto Pickup has completed. **Bit 6 X'02'** AutoCouponsCounted: bit flag indicating that Auto Coupons have been counted. **Bit 7 X'01'** Reserved
TransactionNumber	PD	2	12	Transaction number of the last update to this file.

Field Name	Type	Length	Decimal Offset	Description
Restart	Int	4	14	This is used as a control field on a system restart to determine if this record has been updated by a particular transaction.
GrossPlus	Int	4	18	Gross positive. Accumulation of the positive sales entries. Does not include totals from voided or training transactions. *(sales + deposits + taxes + tender fees)*
GrossMinus	Int	4	22	Gross negative. Accumulation of the negative sales entries. Does not include totals from voided or training transactions. This is a positive amount. *(deposit returns + item refunds + tax refunds + discounts + cancels of sales, deposits, item refunds, deposit returns, and tender fees)*
SalesTransactionCount	Int	4	26	Number of sales.
LoanTenderAmount	Array	4x8	30	This is an array of eight 4-byte tender type totals for loans made during the period:
AMTCASH	Int	4	30	Sum of loans - cash
AMTCHECK	Int	4	34	Sum of loans - checks
AMTFOODS	Int	4	38	Sum of loans - food stamps
AMTMISC1	Int	4	42	Sum of loans - miscellaneous tender 1
AMTMISC2	Int	4	46	Sum of loans - miscellaneous tender 2
AMTMISC3	Int	4	50	Sum of loans - miscellaneous tender 3
AMTMANUF	Int	4	54	Sum of loans - manufacturer coupons
AMTSTORE	Int	4	58	Sum of loans - store coupons
PickupTenderAmount	Array	4x8	62	This is an array of eight 4-byte tender type totals for pickups made during the period:
AMTCASH	Int	4	62	Sum of pickups - cash
AMTCHECK	Int	4	66	Sum of pickups - checks
AMTFOODS	Int	4	70	Sum of pickups - food stamps
AMTMISC1	Int	4	74	Sum of pickups - miscellaneous tender 1
AMTMISC2	Int	4	78	Sum of pickups - miscellaneous tender 2
AMTMISC3	Int	4	82	Sum of pickups - miscellaneous tender 3
AMTMANUF	Int	4	86	Sum of pickups - manufacturer coupons
AMTSTORE	Int	4	90	Sum of pickups - store coupons
CountedTenderAmount	Array	4x8	94	This is an array of eight 4-byte tender type totals for tender counts made during the period. These totals represent the tender

Field Name	Type	Length	Decimal Offset	Description
				count when the store safe is *counted down*. Each of the eight total fields follows:
AMTCASH	Int	4	94	Counted tender - cash
AMTCHECK	Int	4	98	Counted tender - checks
AMTFOODS	Int	4	102	Counted tender - food stamps
AMTMISC1	Int	4	106	Counted tender - miscellaneous tender 1
AMTMISC2	Int	4	110	Counted tender - miscellaneous tender 2
AMTMISC3	Int	4	114	Counted tender - miscellaneous tender 3
AMTMANUF	Int	4	118	Counted tender - manufacturer coupons
AMTSTORE	Int	4	122	Counted tender - store coupons
NetTenderAmount	Array	4x8	126	This is an array of eight 4-byte net tender totals for tenders made during the period. Totals represent the change in the net tender position as a result of miscellaneous transactions, loans, pickups, and tender counts. Totals reflect the *short/over* position of the safe. Each of the eight total fields follows:
AMTCASH	Int	4	126	Net tender - cash
AMTCHECK	Int	4	130	Net tender - checks
AMTFOODS	Int	4	134	Net tender - food stamps
AMTMISC1	Int	4	138	Net tender - miscellaneous tender 1
AMTMISC2	Int	4	142	Net tender - miscellaneous tender 2
AMTMISC3	Int	4	146	Net tender - miscellaneous tender 3
AMTMANUF	Int	4	150	Net tender - manufacturer coupons
AMTSTORE	Int	4	154	Net tender - store coupons
ItemSalesAmount	Int	4	158	Total item sales amount that does not include voided and training transactions.
DepositAmount	Int	4	162	Total deposit amount that does not include voided and training transactions.
RefundAmount	Int	4	166	Total refund amount that does not include voided and training transactions. Does include cancels.
DepositReturnAmount	Int	4	170	Total deposit return amount that does not include voided and training transactions. Does include cancels.

Field Name	Type	Length	Decimal Offset	Description
MiscReceiptAmount	Int	4	174	Total miscellaneous transaction receipt amount that does not include voided and training transactions. Does include cancels.
MiscPayoutAmount	Int	4	178	Total miscellaneous transaction payout amount that does not include voided and training transactions. Does include cancels.
DiscountAmount	Int	4	182	Total discount amount that does not include voided and training transactions. Does include cancels.
TaxableAmountExempt	Int	4	186	Total taxable amount to which tax exemptions apply. Does not include voided and training transactions. Does include cancels.
ItemCancelAmount	Int	4	190	Total item cancel amount that does not include voided and training transactions.
DepositCancelAmount	Int	4	194	Total deposit cancel amount that does not include voided and training transactions.
CreditTransactionAmount	Int	4	198	Total credit transaction amount that does not include voided and training transactions. Does include cancels.
TenderFeeAmount	Int	4	202	Total tender fees (including checks) paid. Does not include voided and training transactions. Does include cancels.
MiscTransactionAmount	Int	4	206	Total miscellaneous transaction amount credited to this individual operator or terminal account by a controller procedure.
TenderCashingAmount	Int	4	210	Total amount of tender cashed with no-sale procedure.
TenderExchangeAmount	Int	4	214	Total amount of tender exchanged with no-sale procedure.
TaxableAmountA	Int	4	218	Total tax plan A taxable sales. Does not include voids or training.
TaxAmountA	Int	4	222	Total plan A tax paid.
TaxableAmountB	Int	4	226	Total tax plan B taxable sales. Does not include voids or training.
TaxAmountB	Int	4	230	Total plan B tax paid.
TaxableAmountC	Int	4	234	Total tax plan C taxable sales. Does not include voids or training.
TaxAmountC	Int	4	238	Total plan C tax paid.

Field Name	Type	Length	Decimal Offset	Description
TaxableAmountD	Int	4	242	Total tax plan D taxable sales. Does not include voids or training.
TaxAmountD	Int	4	246	Total plan D tax paid.
StandaloneGrossPlus	Int	4	250	Standalone gross positive.
StandaloneGrossMinus	Int	4	254	Standalone gross negative.
VoidTransactionAmount	Int	4	258	Total amount of void transactions.
TrainingTransactionAmount	Int	4	262	Total amount of training transactions
ItemSalesCount	Int	4	266	Number of items sold (scanned or keyed) not including voided transactions or training transactions.
ItemSalesKeyedCount	Int	4	270	Number of items sold by keyed item code not including voided transactions or training transactions.
ItemSalesLookupKeysCount	Int	4	274	Number of items sold by item lookup keys not including voided transactions or training transactions.
TradingStamps	Int	4	278	Reserved.
DepositCount	Int	2	282	Number of deposits not including voided transactions or training transactions.
RefundCount	Int	2	284	Number of refunds not including voided transactions or training transactions.
DepositReturnCount	Int	2	286	Number of deposit returns not including voided transactions or training transactions.
MiscReceiptCount	Int	2	288	Number of miscellaneous item record receipts not including voided transactions or training transactions.
MiscPayoutCount	Int	2	290	Number of miscellaneous item record payouts not including voided transactions or training transactions.
DiscountCount	Int	2	292	Number of discounts not including voided transactions or training transactions.
TaxExemptionCount	Int	2	294	Number of tax exemptions not including voided transactions or training transactions.
ItemCancelCount	Int	2	296	Number of item sale cancels not including voided transactions or training transactions.

Field Name	Type	Length	Decimal Offset	Description
DepositCancelCount	Int	2	298	Number of deposit cancels not including voided transactions or training transactions.
CreditTransactionCount	Int	2	300	Number of credit transactions not including voided transactions or training transactions.
SpecialSignOffCount	Int	2	302	Number of special sign-ons.
NoSaleTransactionCount	Int	2	304	Number of transactions where the cash drawer was opened before or during the transaction by a no-sale or non-sales procedure not including training transactions.
NetTenderCount	Int	16	306	This represents the following eight 2-byte net tender count fields:
NUMCASH	Int	2	306	Number of cash transactions
NUMCHECK	Int	2	308	Number of checks cashed
NUMFOODS	Int	2	310	Number of food stamp transactions
NUMMISC1	Int	2	312	Number of miscellaneous tender 1
NUMMISC2	Int	2	314	Number of miscellaneous tender 2
NUMMISC3	Int	2	316	Number of miscellaneous tender 3
NUMMANUF	Int	2	318	Number of manufacturer coupons
NUMSTORE	Int	2	320	Number of store coupons
LoanCount	Int	2	322	Number of loans to this till.
PickupCount	Int	2	324	Number of pickups from this till.
StandaloneTransactionCount	Int	2	326	Number of standalone transactions.
VoidTransactionCount	Int	2	328	Number of voided transactions.
TrainingTransactionCount	Int	2	330	Number of training transactions.
StandaloneTaxAmount	Int	4	332	Standalone tax amount.
TaxableAmountE	Int	4	336	Total tax plan E taxable sales. Does not include voids or training.
TaxAmountE	Int	4	340	Total plan E tax paid.
TaxableAmountF	Int	4	344	Total tax plan F taxable sales. Does not include voids or training.
TaxAmountF	Int	4	348	Total plan F tax paid.
TaxableAmountG	Int	4	352	Total tax plan G taxable sales. Does not include voids or training.

Field Name	Type	Length	Decimal Offset	Description
TaxAmountG	Int	4	356	Total plan G tax paid.
TaxableAmountH	Int	4	360	Total tax plan H taxable sales. Does not include voids or training.
TaxAmountH	Int	4	364	Total plan H tax paid.
Reserved	Int	4	368	Reserved
SalesPoints	Int	4	372	Total primary points given for sales to preferred customers.
BonusPoints	Int	4	376	Total bonus points given to preferred customers.
RedeemedPoints	Int	4	380	Total points redeemed by preferred customers.
ManAutoCouponAmount	Int	4	384	Total amount of awarded electronic manufacturer automatic coupons.
StoreAutoCouponAmount	Int	4	388	Total amount of awarded electronic store automatic coupons.
DoubledCouponAmount	Int	4	392	Total amount of doubled coupons.
PCTransactionAmount	Int	4	396	Total sales amount in preferred customer transactions.
PCTransactionCount	Int	2	400	Number of preferred customer transactions.
PCAutoCouponCount	Int	2	402	Number of preferred customer automatic coupons awarded.
PCAutoCouponAmount	Int	4	404	Total amount of preferred customer automatic coupons awarded.
CouponAmount	Int	24	408	This represents six 4-byte coupon tier amount fields. Each is the total amount of coupons in a coupon multiplication tier level as reported in the Coupon Multiplication Limit Report.
CouponTier1Amount	Int	4	408	The total amount of coupons in coupon multiplication tier 1 as reported in the Coupon Multiplication Limit Report.
CouponTier2Amount	Int	4	412	The total amount of coupons in coupon multiplication tier 2 as reported in the Coupon Multiplication Limit Report.
CouponTier3Amount	Int	4	416	The total amount of coupons in coupon multiplication tier 3 as reported in the Coupon Multiplication Limit Report.

Field Name	Type	Length	Decimal Offset	Description
CouponTier4Amount	Int	4	420	The total amount of coupons in coupon multiplication tier 4 as reported in the Coupon Multiplication Limit Report.
CouponTier5Amount	Int	4	424	The total amount of coupons in coupon multiplication tier 5 as reported in the Coupon Multiplication Limit Report.
CouponTier6Amount	Int	4	428	The total amount of coupons in coupon multiplication tier 6 as reported in the Coupon Multiplication Limit Report.
CouponCount	PD	12	432	This represents six 2-byte coupon tier count fields. Each is the count of coupons in a coupon multiplication tier level as reported in the Coupon Multiplication Limit Report.
CouponTier1Count	Int	2	432	The count of coupons in coupon multiplication tier 1 as reported in the Coupon Multiplication Limit Report.
CouponTier2Count	Int	2	434	The count of coupons in coupon multiplication tier 2 as reported in the Coupon Multiplication Limit Report.
CouponTier3Count	Int	2	436	The count of coupons in coupon multiplication tier 3 as reported in the Coupon Multiplication Limit Report.
CouponTier4Count	Int	2	438	The count of coupons in coupon multiplication tier 4 as reported in the Coupon Multiplication Limit Report.
CouponTier5Count	Int	2	440	The count of coupons in coupon multiplication tier 5 as reported in the Coupon Multiplication Limit Report.
CouponTier6Count	Int	2	442	The count of coupons in coupon multiplication tier 6 as reported in the Coupon Multiplication Limit Report.
UsrInteger	Int	24	444	Six 4-byte UsrInteger fields.
UsrReserved	ASCII	40	468	User data area.

Tender Totals By Variety Record

Field Name	Type	Length	Decimal Offset	Description
RecordType	ASCII	1	0	Record type extension:

Field Name	Type	Length	Decimal Offset	Description
				Note: Amounts maintained in record types 0 through B are always kept in the equivalent value of the store's local currency. 4 - Operator/store loans 5 - Operator/store pickups 6 - Operator/store tender counts 7 - Operator/store net tenders 8 - Terminal loans 9 - Terminal pickups A - Terminal tender counts B - Terminal net tenders Note: When Foreign Currency is enabled in options, record types C through J will also be present in the Accounting Totals files. Amounts maintained in record types C through J are always kept in the original value of the entered currency. C - Operator/store loans currency D - Operator/store pickups currency E - Operator/store tender counts currency F - Operator/store net tenders currency G - Terminal loans currency H - Terminal pickups currency I - Terminal tender counts currency J - Terminal net tenders currency
AccountId	PD	5	1	X'0000000000' - Store Record (all other values) - Either the Operator or Terminal ID, depending on the record type.
Restart	PD	4	6	Control field on a system restart used to determine if this record has been updated by a particular transaction.
Tender01Amount	Array	6x4	10	This represents the following six 4-byte totals for six varieties of cash tender:
AMTCSHTND1	Int	4	10	Face value of cash tender variety 1
AMTCSHTND2	Int	4	14	Face value of cash tender variety 2
AMTCSHTND3	Int	4	18	Face value of cash tender variety 3

Field Name	Type	Length	Decimal Offset	Description
AMTCSHTND4	Int	4	22	Face value of cash tender variety 4
AMTCSHTND5	Int	4	26	Face value of cash tender variety 5
AMTCSHTND6	Int	4	30	Face value of cash tender variety 6
Tender02Amount	Array	6x4	34	This represents the following six 4-byte totals for six varieties of check tender:
AMTCHKTND1	Int	4	34	Face value of check tender variety 1
AMTCHKTND2	Int	4	38	Face value of check tender variety 2
AMTCHKTND3	Int	4	42	Face value of check tender variety 3
AMTCHKTND4	Int	4	46	Face value of check tender variety 4
AMTCHKTND5	Int	4	50	Face value of check tender variety 5
AMTCHKTND6	Int	4	54	Face value of check tender variety 6
Tender03Amount	Array	6x4	58	This represents the following six 4-byte totals for six varieties of food stamp tender:
AMTFSTND1	Int	4	58	Face value of food stamp tender variety 1
AMTFSTND2	Int	4	62	Face value of food stamp tender variety 2
AMTFSTND3	Int	4	66	Face value of food stamp tender variety 3
AMTFSTND4	Int	4	70	Face value of food stamp tender variety 4
AMTFSTND5	Int	4	74	Face value of food stamp tender variety 5
AMTFSTND6	Int	4	78	Face value of food stamp tender variety 6
Tender04Amount	Array	6x4	82	This represents the following six 4-byte totals for six varieties of miscellaneous 1 tender (which are various credit card tenders by default):
AMTMISC1TND1	Int	4	82	Face value of miscellaneous 1 tender variety 1
AMTMISC1TND2	Int	4	86	Face value of miscellaneous 1 tender variety 2
AMTMISC1TND3	Int	4	90	Face value of miscellaneous 1 tender variety 3
AMTMISC1TND4	Int	4	94	Face value of miscellaneous 1 tender variety 4
AMTMISC1TND5	Int	4	98	Face value of miscellaneous 1 tender variety 5
AMTMISC1TND6	Int	4	102	Face value of miscellaneous 1 tender variety 6
Tender05Amount	Array	6x4	106	This represents the following six 4-byte totals for six varieties of miscellaneous 2 tender (which is a debit card tender by default):
AMTMISC2TND1	Int	4	106	Face value of miscellaneous 2 tender variety 1
AMTMISC2TND2	Int	4	110	Face value of miscellaneous 2 tender variety 2
AMTMISC2TND3	Int	4	114	Face value of miscellaneous 2 tender variety 3

Field Name	Type	Length	Decimal Offset	Description
AMTMISC2TND4	Int	4	118	Face value of miscellaneous 2 tender variety 4
AMTMISC2TND5	Int	4	122	Face value of miscellaneous 2 tender variety 5
AMTMISC2TND6	Int	4	126	Face value of miscellaneous 2 tender variety 6
Tender06Amount	Array	6x4	130	This represents the following six 4-byte totals for six varieties of miscellaneous 3 tender (which is an undefined tender type by default):
AMTMISC3TND1	Int	4	130	Face value of miscellaneous 3 tender variety 1
AMTMISC3TND2	Int	4	134	Face value of miscellaneous 3 tender variety 2
AMTMISC3TND3	Int	4	138	Face value of miscellaneous 3 tender variety 3
AMTMISC3TND4	Int	4	142	Face value of miscellaneous 3 tender variety 4
AMTMISC3TND5	Int	4	146	Face value of miscellaneous 3 tender variety 5
AMTMISC3TND6	Int	4	150	Face value of miscellaneous 3 tender variety 6
Tender07Amount	Array	6x4	154	This represents six 4-byte totals for six varieties of manufacturers' coupon tender:
AMTMCTND1	Int	4	154	Face value of manufacturer coupon tender variety 1
AMTMCTND2	Int	4	158	Face value of manufacturer coupon tender variety 2
AMTMCTND3	Int	4	162	Face value of manufacturer coupon tender variety 3
AMTMCTND4	Int	4	166	Face value of manufacturer coupon tender variety 4
AMTMCTND5	Int	4	170	Face value of manufacturer coupon tender variety 5
AMTMCTND6	Int	4	174	Face value of manufacturer coupon tender variety 6
Tender08Amount	Array	6x4	178	This represents the following six 4-byte totals for six varieties of store coupon tender:
AMTSCTND1	Int	4	178	Face value of store coupon tender variety 1
AMTSCTND2	Int	4	182	Face value of store coupon tender variety 2
AMTSCTND3	Int	4	186	Face value of store coupon tender variety 3
AMTSCTND4	Int	4	190	Face value of store coupon tender variety 4
AMTSCTND5	Int	4	194	Face value of store coupon tender variety 5
AMTSCTND6	Int	4	198	Face value of store coupon tender variety 6
Tender01Count	Array	6x2	202	This represents the following six 2-byte counts for six varieties of cash tender:

Field Name	Type	Length	Decimal Offset	Description
CNTCSHTND1	Int	2	202	Cash tender variety 1
CNTCSHTND2	Int	2	204	Cash tender variety 2
CNTCSHTND3	Int	2	206	Cash tender variety 3
CNTCSHTND4	Int	2	208	Cash tender variety 4
CNTCSHTND5	Int	2	210	Cash tender variety 5
CNTCSHTND6	Int	2	212	Cash tender variety 6
Tender02Count	Array	6x2	214	This represents the following six 2-byte counts for six varieties of check tenders:
CNTCHKTND1	Int	2	214	Check tender variety 1
CNTCHKTND2	Int	2	216	Check tender variety 2
CNTCHKTND3	Int	2	218	Check tender variety 3
CNTCHKTND4	Int	2	220	Check tender variety 4
CNTCHKTND5	Int	2	222	Check tender variety 5
CNTCHKTND6	Int	2	224	Check tender variety 6
Tender03Count	Array	6x2	226	This represents the following six 2-byte counts for six varieties of food stamp tenders:
CNTFSTND1	Int	2	226	Food stamp tender variety 1
CNTFSTND2	Int	2	228	Food stamp tender variety 2
CNTFSTND3	Int	2	230	Food stamp tender variety 3
CNTFSTND4	Int	2	232	Food stamp tender variety 4
CNTFSTND5	Int	2	234	Food stamp tender variety 5
CNTFSTND6	Int	2	236	Food stamp tender variety 6
Tender04Count	Array	6x2	238	This represents the following six 2-byte counts for six varieties of miscellaneous 1 tenders (which are various credit card tenders by default):
CNTMISC1TND1	Int	2	238	Miscellaneous 1 tender variety 1
CNTMISC1TND2	Int	2	240	Miscellaneous 1 tender variety 2
CNTMISC1TND3	Int	2	242	Miscellaneous 1 tender variety 3
CNTMISC1TND4	Int	2	244	Miscellaneous 1 tender variety 4
CNTMISC1TND5	Int	2	246	Miscellaneous 1 tender variety 5
CNTMISC1TND6	Int	2	248	Miscellaneous 1 tender variety 6
Tender05Count	Array	6x2	250	This represents the following six 2-byte counts for six varieties of miscellaneous 2 tenders (which is a debit card tender by default):

Field Name	Type	Length	Decimal Offset	Description
CNTMISC2TND1	Int	2	250	Miscellaneous 2 tender variety 1
CNTMISC2TND2	Int	2	252	Miscellaneous 2 tender variety 2
CNTMISC2TND3	Int	2	254	Miscellaneous 2 tender variety 3
CNTMISC2TND4	Int	2	256	Miscellaneous 2 tender variety 4
CNTMISC2TND5	Int	2	258	Miscellaneous 2 tender variety 5
CNTMISC2TND6	Int	2	260	Miscellaneous 2 tender variety 6
Tender06Count	Array	6x2	262	This represents the following six 2-byte counts for six varieties of miscellaneous 3 tenders (which is undefined by default):
CNTMISC3TND1	Int	2	262	Miscellaneous 3 tender variety 1
CNTMISC3TND2	Int	2	264	Miscellaneous 3 tender variety 2
CNTMISC3TND3	Int	2	266	Miscellaneous 3 tender variety 3
CNTMISC3TND4	Int	2	268	Miscellaneous 3 tender variety 4
CNTMISC3TND5	Int	2	270	Miscellaneous 3 tender variety 5
CNTMISC3TND6	Int	2	272	Miscellaneous 3 tender variety 6
Tender07Count	Array	6x2	274	This represents the following six 2-byte counts for six varieties of manufacturers' coupons by default:
CNTMCTND1	Int	2	274	Manufacturer coupon tender variety 1
CNTMCTND2	Int	2	276	Manufacturer coupon tender variety 2
CNTMCTND3	Int	2	278	Manufacturer coupon tender variety 3
CNTMCTND4	Int	2	280	Manufacturer coupon tender variety 4
CNTMCTND5	Int	2	282	Manufacturer coupon tender variety 5
CNTMCTND6	Int	2	284	Manufacturer coupon tender variety 6
Tender08Count	Array	6x2	286	This represents the following six 2-byte counts for six varieties of store coupons by default:
CNTSCTND1	Int	2	286	Store coupon tender variety 1
CNTSCTND2	Int	2	288	Store coupon tender variety 2
CNTSCTND3	Int	2	290	Store coupon tender variety 3
CNTSCTND4	Int	2	292	Store coupon tender variety 4
CNTSCTND5	Int	2	294	Store coupon tender variety 5
CNTSCTND6	Int	2	296	Store coupon tender variety 6
Tender01Amount (continued)	Array	3x4	298	This represents the following three 4-byte totals for three varieties of cash tender:

Field Name	Type	Length	Decimal Offset	Description
AMTCSHTND7	Int	4	298	Face value of cash tender variety 7
AMTCSHTND8	Int	4	302	Face value of cash tender variety 8
AMTCSHTND9	Int	4	306	Face value of cash tender variety 9
Tender02Amount (continued)	Array	3x4	310	This represents the following three 4-byte totals for three varieties of check tender:
AMTCHKTND7	Int	4	310	Face value of check tender variety 7
AMTCHKTND8	Int	4	314	Face value of check tender variety 8
AMTCHKTND9	Int	4	318	Face value of check tender variety 9
Tender03Amount (continued)	Array	3x4	322	This represents the following three 4-byte totals for three varieties of food stamp tender:
AMTFSTND7	Int	4	322	Face value of food stamp tender variety 7
AMTFSTND8	Int	4	326	Face value of food stamp tender variety 8
AMTFSTND9	Int	4	330	Face value of food stamp tender variety 9
Tender04Amount (continued)	Array	3x4	334	This represents the following three 4-byte totals for three varieties of miscellaneous 1 tender (credit card tenders by default):
AMTMISC1TND7	Int	4	334	Face value of miscellaneous 1 tender variety 7
AMTMISC1TND8	Int	4	338	Face value of miscellaneous 1 tender variety 8
AMTMISC1TND9	Int	4	342	Face value of miscellaneous 1 tender variety 9
Tender05Amount (continued)	Array	3x4	346	This represents the following three 4-byte totals for three varieties of miscellaneous 2 tender (debit card tenders by default):
AMTMISC2TND7	Int	4	346	Face value of miscellaneous 2 tender variety 7
AMTMISC2TND8	Int	4	350	Face value of miscellaneous 2 tender variety 8
AMTMISC2TND9	Int	4	354	Face value of miscellaneous 2 tender variety 9
Tender06Amount (continued)	Array	3x4	358	This represents the following three 4-byte totals for three varieties of miscellaneous 3 tender (undefined tenders by default):
AMTMISC3TND7	Int	4	358	Face value of miscellaneous 3 tender variety 7
AMTMISC3TND8	Int	4	362	Face value of miscellaneous 3 tender variety 8
AMTMISC3TND9	Int	4	366	Face value of miscellaneous 3 tender variety 9
Tender07Amount (continued)	Array	3x4	370	This represents the following three 4-byte totals for three varieties of manufacturers' coupon:
AMTMCTND7	Int	4	370	Face value of manufacturer coupon tender variety 7

Field Name	Type	Length	Decimal Offset	Description
AMTMCTND8	Int	4	374	Face value of manufacturer coupon tender variety 8
AMTMCTND9	Int	4	378	Face value of manufacturer coupon tender variety 9.
Tender08Amount (continued)	Array	3x4	382	This represents the following three 4-byte totals for three varieties of store coupon tender:
AMTSCTND7	Int	4	382	Face value of manufacturer coupon tender variety 7.
AMTSCTND8	Int	4	386	Face value of manufacturer coupon tender variety 8.
AMTSCTND9	Int	4	390	Face value of manufacturer coupon tender variety 9.
Tender01Count (continued)	Array	3x2	394	This represents the following three 2-byte counts for three varieties of cash tender:
CNTCSHTND7	Int	2	394	Cash tender variety 7
CNTCSHTND8	Int	2	396	Cash tender variety 8
CNTCSHTND9	Int	2	398	Cash tender variety 9
Tender02Count (continued)	Array	3x2	400	This represents the following three 2-byte counts for three varieties of check tender:
CNTCHKTND7	Int	2	400	Check tender variety 7
CNTCHKTND8	Int	2	402	Check tender variety 8
CNTCHKTND9	Int	2	404	Check tender variety 9
Tender03Count (continued)	Array	3x2	406	This represents the following three 2-byte counts for three varieties of food stamp tender:
CNTFSTND7	Int	2	406	Food stamp tender variety 7
CNTFSTND8	Int	2	408	Food stamp tender variety 8
CNTFSTND9	Int	2	410	Food stamp tender variety 9
Tender04Count (continued)	Array	3x2	412	This represents the following three 2-byte counts for three varieties of miscellaneous 1 tender (credit card tenders by default):
CNTMISC1TND7	Int	2	412	Miscellaneous 1 tender variety 7
CNTMISC1TND8	Int	2	414	Miscellaneous 1 tender variety 8
CNTMISC1TND9	Int	2	416	Miscellaneous 1 tender variety 9
Tender05Count (continued)	Array	3x2	418	This represents the following three 2-byte counts for three varieties of miscellaneous 2 tender (debit card tenders by default):
CNTMISC2TND7	Int	2	418	Miscellaneous 2 tender variety 7

Field Name	Type	Length	Decimal Offset	Description
CNTMISC2TND8	Int	2	420	Miscellaneous 2 tender variety 8
CNTMISC2TND9	Int	2	422	Miscellaneous 2 tender variety 9
Tender06Count (continued)	Array	3x2	424	This represents the following three 2-byte counts for three varieties of miscellaneous 3 tender (undefined tenders by default):
CNTMISC3TND7	Int	2	424	Miscellaneous 3 tender variety 7
CNTMISC3TND8	Int	2	426	Miscellaneous 3 tender variety 8
CNTMISC3TND9	Int	2	428	Miscellaneous 3 tender variety 9
Tender07Count (continued)	Array	3x2	430	This represents the following three 2-byte counts for three varieties of manufacturers' coupon:
CNTMCTND7	Int	2	430	Manufacturer coupon tender variety 7
CNTMCTND8	Int	2	432	Manufacturer coupon tender variety 8
CNTMCTND9	Int	2	434	Manufacturer coupon tender variety 9
Tender08Count (continued)	Array	3x2	436	This represents the following three 2-byte counts for three varieties of store coupon:
CNTSCTND7	Int	2	436	Store coupon tender variety 7
CNTSCTND8	Int	2	438	Store coupon tender variety 8
CNTSCTND9	Int	2	440	Store coupon tender variety 9
Reserved	ASCII	66	442	Reserved

EAMBATCH (Batch Maintenance File)

The Batch Maintenance File contains the commands and associated data that is required for updating or reporting a keyed file. The file contains batch header records and data records. All data records are associated with the prior batch header record. The file must begin with a batch header record. The records contain no internal field delimiters. The records are delimited by a CR/LF X'0D0A'.

Logical name	<EAMBATCH>
Data object reference	*ISKFBISimpleData, ISKFBIRangeData, ISKFBIFlagData, ISKFBITargetedCouponData, ISKFBIIncludeAllData* <KYDFLBTC.CPP>
Organization	Sequential
Distribution class	Mirrored on close
File copies	2
Record length	Variable

Batch Header Record

Batch header records denote the beginning of a group of maintenance changes that are to be applied to the same keyed file. The header record provides the keyed file name and other identification data for a maintanence batch. Each header record contains 15 bytes of fixed data and a two-byte record delimiter.

The flags in the header record define how to process record adds or replaces when the record is or is not present. These flags also allow adding a record when only partial record data is supplied and deleting multiple file records with only a single input data record.

Table 41. Layout for Batch Header Record in EAMBATCH File

Field Name	Type	Length	Decimal Offset	Description
RecordType	ASCII	1	0	*H* (X'48') for header record.
FileName	ASCII	8	1	Logical file name to be processed.
Store	PD	2	9	Store number of store that is intended to receive the update.
Batch	PD	3	11	Batch number used to associate errors and status.
Flags	Int	1	14	Bit 0 X'80' - If delete is not found, log an error. Bit 1 X'40' - Full-length records are treated as replaces, not adds. Bit 2 X'20' - Replace can do an add. Bit 3 X'10' - Add can do a replace. Bit 4 X'08' - Pad and add on partial replace when record is not on file. Bit 5 X'04' - If replace does an add, log an error. Bit 6 X'02' - If add does a replace, log an error. Bit 7 X'01' - Multiple deletes are allowed per record.

Data Record

Data records contain the information required to update, add, delete or report a single record or a group of records in a keyed file. If the key field in the input data overlaps the entire key field of the file record (`KEY-OFFSET` is 1, `KEY-LENGTH` is greater than or equal to the key length defined for the file), only a single keyed record is processed. If the data key does not include the record key, any record on file that matches the key data at the correct offset will be updated. If the key data contains a range, the key data is split in halves (low limit followed by high limit) and all records that have data within or equal to the limits at the key offset are processed. If no record exactly matches the input data key, whether or not it includes the record key, then none of the keyed records will be processed as a result of the input record. A `"do not care"` character can be used in a simple key, not flags and not range, to cause the key to match when not all data in the key matches. The key data is in the same format as the file data being referenced as a key.

An update will not be allowed to alter the key to a record, but the update field can contain the key at your discretion. Any update field that contains the complete record data, with or without the key, is considered a full record add or replace based on the options for the batch. Any update field that does not contain complete record data is considered a partial record update that requires the record to be on file. The update data must be exactly in the format that is needed in the updated record.

Flag fields to be used to test or modify records can be provided in either the KEY-DATA or UPD-DATA fields. Flags to turn on or off or to test for on or off conditions can only be in the range of one to four bytes each. If both on and and off flags are provided (up to eight bytes), they must be of equal length. If both on and off flags are provided, the off flags must precede the on flags. On an update, the requested flags are turned off first, then the request flags are turned on. On the test for a key match, the flags specified to be off must all be off and the flags specified to be on must all be on in order for the key to match. If the options specify that test flags are provided, but the input flags are all zero, then all records will match. Flags in the key are mutually exclusive with a key range or with `"do not care"` characters in the key.

The KEY-OFFSET and KEY-LENGTH fields can be considered a single three-byte packed field to simplify processing. In that case, the offset is the first threee digits in the field and the length is the following three digits. Similarly, the UPD-OFFSET and UPD-LENGTH fields can also be considered a single three-byte packed field.

The offset fields KEY-OFFSET and UPD-OFFSET are unit-indexed so that the first byte of a record is considered at offset 1. All offsets and lengths must be less than or equal to the full keyed record length to be valid. If the `KEY-DATA` is not the length specified by the KEY-LENGTH field or if the UPD-DATA is not the length specified by the UPD-LENGTH field, then the input record will be discarded and an error will be logged. All fields defined in the record must be present except for the data fields, which can be defined with a length of zero. The minimum record length is eight bytes and the maximum record length is 1023 bytes.

A KEY-OFFSET and KEY-LENGTH equal to zero means report or modify all records in the keyed file. A request to delete or report a record must have UPD-OFFSET and UPD-LENGTH equal to zero. A request to modify a record or set of records must have a nonzero value for UPD-OFFSET and UPD-LENGTH unless the Pad and Add flag is on. If a valid record key is provided with a Pad and Add request and no update data is given, then an empty (all X'00') record is added for the given record key if the record is not already on file.

If a key field, data field, or record ends prematurely, the next record is read and appended. This allows a X'0D0A' to be embedded in a data record. If the appended data does not create a valid record, it will be treated as a new record after the original record is discarded as invalid data.

Table 42. Layout for Data Record in EAMBATCH File

Field Name	Type	Length	Decimal Offset	Description
RecordType	ASCII	1	0	*D* (X'44') for data record
Options	Int	1	1	Option flags for updates. **Bit 0 X'80'** X'FE' in key data = do not care. **Bit 1 X'40'** Key data is doubled to give range. **Bit 2 X'20'** Flags to test off provided. **Bit 3 X'10'** Flags to test on provided.

Field Name	Type	Length	Decimal Offset	Description
				Bit 4 X'08' Report record(s). **Bit 5 X'04'** Report and delete records. **Bit 6 X'02'** Flags to turn off provided. **Bit 7 X'01'** Flags to turn on provided.
Key-Offset	PD	1.5	2	Offset of the key within record(s) - value is 0-508.
Key-Length	PD	1.5	3.5	Length of key data within input - value is 0-508.
Key-Data	User-defined	v255	5	Data to be used to find records to update.
UPD-Offset	PD	1.5	var	Offset of Update field within record(s) - value is 0-508.
UPD-Length	PD	1.5	var	Length of Update data within input - value is 0-508.
UPD-Data	User-defined	v255	var	Data to be updated into record(s).

Targeted Coupon Data Record

The targeted coupons in the Customer Activity File form a list, so special batch processing allows targeted coupons to be added to or deleted from the list. The list processing also allows reporting or alteration of records that contain selected coupons anywhere in the list. Special list processing occurs only with data record that begin with a record type of `T`. These special data records are valid only when the file name in the batch header record is `EAMFBACT` and either the key offset or the update offset has a value of 34. Records that update (add, delete or replace) targeted coupons are valid only if the update offset is 34 and the update data has a length that is a multiple of 2. The offset of 34 allows all targeted coupons in the record to be referenced through the offset of the first targeted coupon. The update length as a multiple of 2 recognizes the fact that each targeted coupon requires two bytes.

Targeted coupon records are a special form of data record that allows list processing for finding records with selected targeted coupons and for performing partial record updates on targeted coupons. This list processing applies to the key or update field whenever the key or update offset is 34. Other than the unique list processing, the only other differences between targeted coupon data records and normal data records are defined by the options flags in the data record. When the update offset is 34, the flag that normally defines a record deletion instead defines the deletion of targeted coupons and therefore allows update data to define the coupons to be deleted. Also, on targeted coupon updates, a single flag defines whether a targeted coupon is added to a list if it is already present in the list. If both the key and update point to targeted coupons, then an option flag allows for the replacement of one targeted coupon by another. Without any of these option flags, a targeted coupon update is processed as an add to the list of coupons. Key flags are not allowed with an offset of 34. The `"do not care"` character is not valid in any targeted coupon record where a key offset of 34 denotes list processing on the key field. When the key offset is not 34, the record key processing is performed the same as with standard data records. When the update offset is not 34, the record update processing is performed the same as with standard data records.

All list processing for targeted coupons requires that the batch process read the Preferred Customer options and use the Customer Activity record length, the offset of the customer name, and the maximum number of targeted coupons to determine the location of all targeted coupons within an activity record. List processing for the key field in a targeted coupon record involves finding and potentially altering or deleting all targeted coupons that match the single key coupon, any of multiple key coupons, or the range of key coupons. List processing for the update field in a targeted coupon record involves deleting the updated coupon wherever it is found, replacing the found coupons with the new coupons, or adding the new coupon to an unused position, perhaps only if the coupon is not already rpesent. An empty position in the coupon list is represented by X'0000', so that value is used to look for an empty slot for an add and to replace a coupon value on a delete. An add is always placed in the first empty slot in the record. An error is logged whenever an add to an activity record is not processed due to a lack of free space in the coupon list. After any targeted coupon replace or delete, the empty coupon slots in the activity record are moved to the end f the targeted coupon area so that the most recent adds should always be at the end of the coupon list. You can assure that all targeted coupons are at the beginning of the lists by processing a request to replace all targeted coupons of 0 with a value of 0.

The key or update data for a targeted update can contain up to the maximum number of targeted coupons allowed per customer. If the list of coupons to be replaced is longer than an associated list of replacement coupons, then the last replacement coupon is used multiple times. This allows a list of coupons to be replaced by a single coupon. An error is logged if the list of replacement coupons is longer than the list of coupons to be replaced.

Table 43. Layout for Targeted Coupon Data Record in EAMBATCH File

Field Name	Type	Length	Decimal Offset	Description
RecordType	ASCII	1	0	*T* (X'54') for data record
Options	Int	1	1	Option flags for updates. **Bit 0 X'80'** X'FE' in key data = do not care. **Bit 1 X'40'** Key data is doubled to give range. **Bit 2 X'20'** Flags to test off provided. **Bit 3 X'10'** Flags to test on provided. **Bit 4 X'08'** Report record(s) containing coupon. **Bit 5 X'04'** Delete coupon if present. **Bit 6 X'02'** Add coupon if not present.

Field Name	Type	Length	Decimal Offset	Description
				Bit 7 X'01' Replace coupon if present.
Key-Offset	PD	1.5	2	Offset of the key within record(s) - value is 0-508.
Key-Length	PD	1.5	3.5	Length of key data within input - value is 0-508.
Key-Data	User-defined	v255	5	Data to be used to find records to update.
UPD-Offset	PD	1.5	var	Offset of Update field within record(s) - value is 0-508.
UPD-Length	PD	1.5	var	Length of Update data within input - value is 0-508.
UPD-Data	User-defined	v255	var	Data to be updated into record(s).

Examples of Targeted Coupon Records

Adding Targeted Coupons to Customers

These examples show the eambatch.dat records for adding targeted coupons to a customer to meet the following conditions:

```
1) Add targeted coupon 321 to customer 91234567890.
2) Add targeted coupon 321 to customer 91234567890 if not already present.
3) Add targeted coupons 321 and 333 to customer 91234567890 if not present.
4) Add targeted coupon 321 to all customers who have targeted coupons
   between 320 and 330 and do not already have 321.
5) Add targeted coupon 321 to all customers with more than 9999 points.
```

```
   Type Opt  Off/Len Data (Key)         Off/Len Data (Update)
1) X'54 00   001 009 000000091234567890 034 002 0321'
2) X'54 02   001 009 000000091234567890 034 002 0321'
3) X'54 02   001 009 000000091234567890 034 004 03210333'
4) X'54 42   034 004 03200330           034 002 0321
5) X'54 40   010 008 0010000 99999999   034 002 0321
```

Deleting Targeted Coupons from Customers

These examples show the eambatch.dat records for deleting targeted coupons from a customer to meet the following conditions:

```
1) Delete targeted coupon 321 from customer 91234567890.
2) Delete targeted coupons 321 and 333 from customer 91234567890.
3) Delete targeted coupon 321 from all customers.
4) Delete targeted coupons 300 to 400 from all customers.
5) Delete targeted coupon 333 from all customers that have targeted coupon 321.
```

```
6) Delete targeted coupon 321 from all customers who are not eligible
   for preferred coupons.
```

```
   Type Opt  Off/Len Data (Key)        Off/Len Data (Update)
1) X'54 04   001 009 000000091234567890 034 002 0321'
2) X'54 04   001 009 000000091234567890 034 004 03210333'
3) X'54 04   034 002 0321               000 000'
4) X'54 44   034 004 03000400           000 000'
5) X'54 04   034 002 0321               034 002 0333'
6) X'54 14   031 001 04                 034 002 0321'
```

Replacing Targeted Coupons

These examples show the eambatch.dat records for replacing targeted coupons for a customer to meet the following conditions:

```
1) Replace all occurrences of targeted coupon 321 with 333.
2) Replace all targeted coupons between 320 and 330 with coupon 321.
```

```
   Type Opt  Off/Len Data (Key)        Off/Len Data (Update)
1) X'54 01   034 002 0321              034 002 0333'
2) X'54 41   034 004 03200330          034 002 0321'
```

Reporting Targeted Coupons

These examples show the eambatch.dat records for reporting targeted coupons for a customer to meet the following conditions:

```
1) Report all customers with targeted coupon 321.
2) Report all customers with targeted coupons 321 or 333.
3) Report all customers with any targeted coupons between 300 and 400.
```

```
   Type Opt  Off/Len Data (Key)        Off/Len Data (Update)
1) X'54 08   034 002 0321              000 000'
2) X'54 08   034 004 03210330          000 000
3) X'54 48   034 004 03000400          000 000'
```

Making Miscellaneous Targeted Coupon Changes

This example shows the eambatch.dat record for meeting the following conditions:

```
1) Compress all targeted coupons to the beginning of the record.
2) Set status level to 3 for all customers who have targeted coupon 333.
```

```
   Type Opt  Off/Len Data (Key)        Off/Len Data (Update)
1) X'54 01   034 002 0000              034 002 0000'
2) X'54 00   034 002 0333              029 001 03'
```

Secondary Points Data Record

Table 44. Layout for Secondary Points Data Record in EAMBATCH File

Field Name	Type	Length	Decimal Offset	Description
RecordType	ASCII	1	0	*S* (X'53') for data record.
Options	PD	1	1	Always X'00'.
KeyOffset	PD	1.5	2	Offset of the key within the record - value is X'001'.
KeyLength	PD	1.5	3.5	Length of key data within input - value is X'009'.
Key	PD	9	5	Customer key in eamfbact file.
UPDOffset	PD	1.5	14	Offset of secondary points club data in the activity record.
UPDLength	PD	1.5	15.5	Length of Update data within input - value is X'008'.
ClubStart	PD	1	17	Start club.
ClubEnd	PD	1	18	End club.
SecPoints	PD	4	19	Secondary points.
PointFieldBlank	PD	1	23	If nonzero value, set points to the value in SecPoints.
ClubStatus	PD	1	24	Action for club status: 0 = Activate, 1 = Deactivate, 2 = Ignore.

EAMCOUP* (Preferred Customer Coupon Tracking File)

The Preferred Customer Coupon Tracking File is mirrored for data security. There is a current period and an old period copy of this file. The old period file contains the data from the most recently closed reporting period and the current period file contains data for the period in process. The records in this file contain information about coupons that have been accepted. The file can contain manufacturer coupons, store coupons, and doubled coupons based on the values of the *Log coupons in tracking file* options in Coupons -> Tracking personalization. You can also specify low and high limits of the *Range of coupons to log in tracking file* option in personalization. Only coupons with a real monetary value are tracked in this file. Therefore, point coupons and cross-promotional coupons are not tracked.

Logical name	<EAMCOUPC>, <EAMCOUPO>, <EAMCOUPW>
Data object reference	*ISSAEMCouponTrackingData* <SAEMCDAT.CPP>
Organization	Sequential
Distribution class	Mirrored per update
File copies	Current, Old

Record length	Variable

Note: <EAMCOUPW> is a work file used during close processing.

Header Record

SurePOS ACE creates a header record when the EAMCOUPx file is created. This record contains the following information:

Table 45. Layout for EAMCOUPx Header Record

Field Name	Type	Length	Description
Date	ASCII	6	YYMMDD
Store Number	ASCII	4	SSSS

Coupon Tracking Records

Field Name	Type	Length	Description
TerminalId	ASCII	v3	Terminal ID where coupon was taken.
TransNum	ASCII	v4	Transaction number in which coupon was taken.
DateTime	ASCII	10	Date and time of transaction in the format `YYMMDDHHMM`.
RecordType	ASCII	2	Record type `99` for exception log.
OperatorId	ASCII	v9	Operator ID accepting the coupon.
CouponMfgNum	ASCII	v12	Manufacturer number of coupon. For GS1 DataBar coupons, this is the Primary GS1 Company Prefix that is in the range of 6 to 12 digits in length.
CouponUPC	ASCII	v14	Item code of coupon.
CouponFamilyNum	ASCII	v3	Family number of coupon. For GS1 DataBar coupons, this is the Primary Purchase Family Code, which matches the funder information.
CouponValue	ASCII	v6	Value of coupon.
ItemUPC	ASCII	v14	Item code of matching item (see Note 1 at the end of the table).
ItemFamily	ASCII	v3	Family number of matching item (see Note 1 at the end of the table).
ItemPrice	ASCII	v8	Price of matching item (see Note 1 at the end of the table).
CampaignNum	ASCII	v6	Promotion code from coupon (see Note 2 at the end of the table).
EntryFlags	ASCII	v3	Decimal representation of one-byte integer containing coupon entry type flags: **X'80'** Coupon was linked to **X'40'** Coupon key pressed with item code

Field Name	Type	Length	Description
			X'20' Keyed promotion code (see note) **X'10'** Keyed coupon value required **X'08'** Keyed coupon value **X'04'** Keyed coupon item code (0 = scanned) **X'02'** Coupon created by doubling **X'01'** Store coupon (0 = MFR coupon)
ValidFlags	ASCII	v5	Decimal representation of two-byte integer containing coupon validation flags: **X'8000'** Coupon required multiple items in order to be issued. There is no override required to get this scenario. **X'4000'** Keyed value limit check (`B009 xx ITEM LIMIT CHECK`). Refer to Manufacturer Coupon Amount in Options -> Limits -> Transactions Personalization. **X'2000'** Coupon value exceeds item value (`B065 COUPON VALUE EXCEEDS ITEM VALUE`). **X'1000'** Quantity not satisfied for coupon (`B064 COUPON MUST MATCH PREVIOUS SALE-QTY NOT SATISFIED`. The override flag in Options -> Coupon -> Overrides Personalization is Too few matching items sold. **X'0800'** Too many coupons relative to sales (`B064 COUPON MUST MATCH PREVIOUS SALE-TOO MANY COUPONS`. The override flag in Options -> Coupon -> Overrides Personalization is More coupons than items taken in transaction. **X'0400'** Too many like coupons for transaction (`B610 LIMITED NUMBER OF COUPONS PER ORDER`). The override flag in Options -> Coupon -> Overrides Personalization is Too many coupons taken in transaction.

Field Name	Type	Length	Description
			X'0200' Coupon has expired (`B611 COUPON HAS EXPIRED`). **X'0100'** Minimum purchase not satisfied (`B612 MINIMUM SALE NOT SATISFIED`). The override flag in Options -> Coupon -> Overrides Personalization is Coupon minimum purchase not satisfied. **X'0080'** Coupon good for free item. There is no override required to get this scenario. **X'0040'** Operator override required. **X'0020'** Manager override required. **X'0010'** The coupon did not require validation (no manufacturer number or 992 validation, or it is an on-file coupon without validation). There is no override required to get this scenario. **X'0008'** Only family super group or family group is valid (`B064 COUPON MUST MATCH PREVIOUS SALE-FAMILY GROUP VALID`). **X'0004'** Only manufacturer is valid (`B064 COUPON MUST MATCH PREVIOUS SALE-MANUFACTURER GROUP VALID`). **X'0002'** Only department is valid (`B064 COUPON MUST MATCH PREVIOUS SALE-DEPARTMENT VALID`). **X'0001'** No match found (`B064 COUPON MUST MATCH PREVIOUS SALE-NO MATCH FOUND`).
CouponBarcode	ASCII	v70	For GS1 DataBar coupons, this field contains the bar code of the coupon.

Note:

1. For GS1 DataBar coupons, if the coupon had the purchase requirements specified with OR conditions, then the purchase requirement used to satisfy the coupon is used for these fields. For purchase requirements specified with AND conditions, the primary purchase requirement is used for these fields. For purchase requirements specified with AND/OR conditions (APRC 3), the first purchase requirement that has a quantity greater than zero is used for these fields.
2. Only keyed promotion codes can be captured at this point since the ability to scan the promotion code has not been demonstrated and might require additional code modifications. An option allows the customer number to be placed in this field in place of the coupon promotion code.

In the ASCII format defined above, the fields are delimited by the standard PC delimiters of "," (X'222C22'). This ASCII format is readable. It allows the file to be directly printed or typed. Also, the file can be printed after being dumped to the host employing the standard ASCII to EBCDIC translation available through the operating system.

The ASCII format creates a large volume of data. An optional PACKED format can also be selected. The PACKED format contains the same data in the same order, but the data is packed so that two numeric characters fit into every byte and the field delimiter is a single colon (X'3A'). This format does not offer the ease of interpretation provided by the ASCII format, but it uses only about half the space of the ASCII format and substantially reduces the time required to transmit this data.

A record similar in format to that defined here can be logged in the exception log file if the coupon type is not selected for collection in the coupon tracking file and the item record associated with the coupon is flagged to *log all sales*. The first five fields above are formatted in the exception log exactly as is done for all other exception log records. The remaining portion of the record is an exact match to the ASCII format.

EAMCPIMG (Check Image Checkpoint File for Close Processing)

The Check Image Checkpoint for Close Processing file keeps track of the close stages for the Check Image files.

Logical name	<EAMCPIMG>
Organization	Direct
Distribution class	Mirrored on update
File copies	Current
Record length	12 bytes

Field Name	Type	Length	Decimal Offset	Description
Recovery count	ASCII	10	0	Checkout support's recovery count at the time that checkout support sent the close message to ACECKIMG.
Reserved	ASCII	1	10	A dash character.
Last close stage	ASCII	1	11	The last close stage that was started. 0 - Copying EAMIMGH to EAMIMGT 1 - Appending EAMIMGC to EAMIMGT 2 - Deleting EAMIMGH 3 - Renaming EAMIMGT to EAMIMGH 4 - Deleting EAMIMGC 5 - Close has completed

EAMCSCF1 (Checkout Support Control File)

The Checkout Support Control file contains a store record, a close record, and a record for each terminal. It is a keyed work file that helps control Checkout Support application processing.

The Checkout Support application creates and manages this file and its records. Only Checkout Support can update the file.

The Checkout Support Control file contains some of the checkpoint information required to restart the Checkout Support application and protect against loss or duplication of data in accounting and report files. The remainder of the required information is stored in the aceclspg.dat file. eamcscf1.dat also contains some of the information required to control the store closing process. Reports against this file detail how well the background process is keeping up with terminal sales.

The prime version of the Checkout Support Control file resides on the file server controller. An image version exists on the alternate file server but is accessed only when the alternate is made the acting file server controller. To minimize the performance impact, this file is distributed only when the file is closed.

Note for SA Users

See Table 1 for a description of how the SurePOS ACE file differs from the 4680-4690 Supermarket Application file.

Logical name	<EAMCSCF1>
Data object reference	*ISSATlogProcessorControlStorage* `<SATPCSTR.CPP>`, *ISSATlogProcessorStoreControlData* `<SATPCDAT.CPP>`, *ISSATlogProcessorCloseControlData* `<SATPCDAT.CPP>`, *ISSATlogProcessorTerminalControlData* `<SATPCDAT.CPP>`
Organization	Keyed
Distribution class	Mirrored at close
File copies	1
Record length	36 bytes
Key length	2 bytes: TerminalID

Store Record

Field Name	Type	Length	Decimal Offset	Description
TerminalID	PD	2	0	Terminal ID. This value is always X'9999' for a Store record.
LogPoint	Int	4	2	Pointer to the record in process or next to be processed by checkout support (restart point)
Restart	Int	4	6	Number of transactions processed by Checkout Support. This field is never reset and is used to avoid a duplicate update on a restart.
IMUpdate	Int	4	10	Pointer to the record to start processing to recover item movement data in memory
IMRestart	Int	4	14	Value of the restart field when item movement data was last written from memory to disk
NumXlog	Int	2	18	Number of entries written to the exception log for the transaction in process.
NumTList	Int	2	20	Number of entries written to the tender listing log for the transaction in process.

Field Name	Type	Length	Decimal Offset	Description
DateTime	PD	5	22	Date and time (Format: hhmm) from last record processed by checkout support
Reserved	ASCII	1	27	Reserved
ExRestart	Int	4	28	Pointer to the record to start processing to recover exception log data in memory.
TLRestart	Int	4	32	Pointer to the record to start processing to recover tender listing data in memory.

Close Record

Field Name	Type	Length	Decimal Offset	Description
Terminal	PD	2	0	Terminal ID. This value is always X'9998' for a Close record.
SLogName	ASCII	8	2	Name of the Transaction Log currently being processed by checkout support.
OSLogName	ASCII	8	10	Name of the Transaction Log previously in use by the terminals.
ClosePnt	Int	4	18	Pointer to the Close record in the previously used summary log.
DateTime	PD	5	22	Date and time of the most recent close or data sync of the summary log, in the format.`YYMMDDHHmm`).
Backup	ASCII	1	27	File backup time interval (hours).
UserFiles	PD	1	28	A binary value that indicates whether user files have been rolled for this Close record.
Reserved	ASCII	7	29	Reserved.

Terminal Record

Field Name	Type	Length	Decimal Offset	Description
Terminal	PD	2	0	Terminal ID (1-999).
TransNum	PD	2	2	Transaction number for the record most recently processed by checkout support for this terminal.

Field Name	Type	Length	Decimal Offset	Description
LogPoint	Int	4	4	Pointer to the transaction record most recently processed by checkout support for this terminal.
NetSales	Int	4	8	Net sales amount for the items in NUMITEMS.
NumItems	Int	2	12	Net number of item sales. Includes sales, deposits, refunds, deposit returns, cancels, and miscellaneous items all summed together. It does not include items from training or voided transactions.
NumCoups	Int	2	14	Net number of store and manufacturer coupons. This does not include coupons from training or voided transactions.
NumTrans	Int	2	16	Number of transactions. This does not include training or voided transactions.
RingTime	Int	2	18	Total ring time in seconds. Includes the time from the first item entry until balance due. Also includes the time from the previous total or tender to a subsequent item sale.
TendeTime	Int	2	20	Total tender time in seconds. Includes the time between a total and a tender, the time between tenders, and the time between the final tender and the close of the cash drawer.
Special	Int	2	22	Total special sign-off time in seconds. Includes the time between a special sign-off and the subsequent special sign-on.
InActive	Int	2	24	Time between transactions in seconds.
NonSales	Int	2	26	Time in seconds when non-sales procedures are being run. Voided and training transactions are in this time.
Restart	Int	4	28	Value used during restart and recovery.
TenderListingRestart	Int	4	32	Value used during tender listing restart and recovery.

EAMCUST* (Customer Account Status File)

The keyed Customer Account Status file accumulates account status information for each active customer account and tender type. Separate files exist for the current and old period.

Image versions of Customer Account Status files are on the alternate file server. Distribution takes place at file close to minimize performance impact.

Logical name	<EAMCUSTA>, <EAMCUSTO>

Data object reference	*ISSACustomerAccountStatusData* <`SACASDAT.CPP`>
Organization	Keyed
Distribution class	Mirrored at close
File copies	Current, Old
Record length	28 bytes
Key length	13 bytes: TenderID / AccountNumber

Store Record (Customer Account Status File)

Field Name	Type	Length	Decimal Offset	Description
TenderID	PD	1	0	TenderID is always equal to X'99' for a Store record.
AccountNumber	PD	12	1	AccountNumber is always equal to X'0' for a Store record.
Restart	Int	4	13	This field is used as a control field on a system restart. It is used to determine if this record has been updated by a particular transaction.
DateTime	PD	5	17	Date and Time to show whether period started or old period ended, in the format `YYMMDDHHmm`.
Reserved	ASCII	6	22	Reserved

Customer Record

Field Name	Type	Length	Decimal Offset	Description
TenderID	PD	1	0	The TenderID is composed of TenderType and TenderVariety: **X'1V'** Cash tender, variety V **X'2V'** Check tender, variety V **X'3V'** Food stamp tender, variety V **X'4V'** Misc. tender type 1, variety V

Field Name	Type	Length	Decimal Offset	Description
				X'5V' Misc. tender type 2, variety V **X'6V'** Misc. tender type 3, variety V V ranges from 1 to 9.
AccountNumber	PD	12	1	Customer account number.
Restart	Int	4	13	This field is used as a control field on a system restart. It is used to determine if this record has been updated by a particular transaction.
AmountTenders	Int	4	17	Amount of tenders against this account.
CountTenders	Int	2	21	Number of tenders against this account.
Usr	Int	4	23	Reserved for user extensions.
Reserved	ASCII	1	27	Reserved.

EAMCXLAT (Customer Translation File)

The keyed Customer Translation file contains associations between preferred customer IDs and a customer identifier that can be entered at the terminal during a transaction. The customer identifier could be a driver's license number or a Social Security ID.

This file is created by Checkout Support at initialization time. There is no SurePOS ACE interface that updates the file. The file is intended to be maintained at a central host site.

Logical name	<EAMCXLAT>
Data object reference	*ISSAAlternateCustomerNumberStorage* <SAACNSTR.CPP>, *ISSAAlternateCustomerNumberData* <SAACNDAT.CPP>
Organization	Keyed
Distribution class	Mirrored per update
File copies	1
Record length	12 bytes
Key length	6 bytes: Customer ID

Field Name	Type	Length	Decimal Offset	Description
CustomerID	UPD	6	0	Customer ID as keyed at the terminal.
AccountNumber	UPD	6	6	Preferred customer account number.

EAMDD* (Delayed Data Maintenance Data File)

The Delayed Maintenance Data file contains commands and associated data for executing a batch.

For price change-only item record maintenance, this file contains keys and prices. For other maintenance, it contains whole records for files being updated. For reports, it contains keys of records to be reported.

To create unique source file names for maintenance batches created in-store, a counter value is appended to the file name.

- If the *Use Expanded Batch File Limit* option is turned off in Personalization, the counter value is always three numeric characters. There can be a maximum of 999 unique files at the same time.
- If the option is turned on, there can be a maximum of 9999 unique files at the same time.
  - If the counter value is 0 - 999, then the actual counter value is appended to the file name as follows:

Table 46. Source File Naming Convention for Counter Values 0-999

Source File Number	Batch Source	Report File
001-999	EAMDD:xxx	EAMDR:xxx
Example: 123	EAMDD:123	EAMDR:123

  - If the counter value is 1000 - 9999, then logical names are used to define the file name as follows:

Table 47. Source File Naming Convention for Counter Values 1000-9999

Source File Number	Batch Source	Report File
1000-9999	DMBx:yyy	DMRx:yyy
Example: 4712	DMB4:712	DMR4:712

Refer to the description of the option in the *SurePOS ACE: Planning and Installation Guide* for more information.

Delayed Data Maintenance usually erases these source files when the batch is executed or erased. Report batch source data is deleted when the batch is cleared. If the source file for a non-report batch does not contain source data for multiple control records, it is deleted following its successful execution. An exception is for a source file involved in a regularly scheduled repeating batch. Source files referenced by multiple control records are never deleted by the application. This procedure reduces the number of potential files not in use in the system.

An image version of this file is on the alternate file server.

Logical name	<EAMDD:>*nnn* or <DMB*x*:>*nnn*
Data object reference	*ISDelayedDataMaintBatchInstructionData* <`DDMBTCID.CPP`>
Organization	Variable sequential
Distribution class	Mirrored per update
File copies	1
Record length	Variable

Field Name	Type	Length	Description
Key	PD	vrbl	Key of target record. The length is 13 for tender verification, 5 for operator authorization, and 6 or 7 for the item record
Command	ASCII	1	Command: **1** ADD a record **2** REPLACE a record **3** DELETE a record **4** REPORT a record **5** REPORT and DELETE a record **6** Host Message
Data	mix	vrbl	Data associated with the request. See appropriate database definition for target file record formats.

The key field is left justified. The data field contains the same information as a keyed record in the file to be updated. Delayed maintenance source data is also allowed in current PSS format or in host data format.

EAMDEP* (Department Totals File)

The Department Totals File is a direct file that contains totals data for each department in the store. From 1 to 999 departments can be defined through personalization. The Department Totals File also contains sufficient store totals data to allow for comparison of department and accounting data.

Department Totals can be closed *short* or *long,* where a long close includes the totals from several short closes. This allows for daily totals as well as weekly totals. A separate file exists for the current period, previous period, and one old period. Therefore, the applications maintain up to six Department Totals files.

Short and long Department Totals files contain eighty-four 512-byte sectors, enough to contain totals for 999 departments. The first sector contains data for 10 departments. Each remaining sector contains data for 12 departments.

Departments are positional within a 10- or 12-department sector. For example, department 10 occupies the tenth department slot in the first sector and department 14 occupies the fourth department slot in the second sector. Any unused department slots (because of undefined department numbers) contain all zeros.

SurePOS ACE performs all maintenance to this file. It should not be necessary for you to modify it.

An image version of this file is kept on the alternate file server. This distribution takes place only when the file is closed to minimize the performance impact.

Note for SA Users

See Table 1 for a description of how the SurePOS ACE file differs from the 4680-4690 Supermarket Application file.

Logical name	<EAMDEPTC>, <EAMDEPTP>, <EAMDEPTO>, <EAMDEPCL>, <EAMDEPPL>, <EAMDEPOL>, <EAMDEPWK>
Data object reference	*ISDepartmentStatisticsData* <`SADPSDAT.CPP`>
Organization	Fixed direct
Distribution class	Mirrored on Close
File copies	Short and long files for current, previous, and old periods
Record length	512 bytes

Record 1

Field Name	Type	Length	Decimal Offset	Description
PDTTime	PD	4	0	Date and time of the transaction that last updated this record. Format: `MMDDHHmm`
Restart	Int	4	4	Control field for a system restart. Checkout Support uses it to determine if this record has been updated by a particular transaction it is replaying.
The next part of the record maintains store totals on a store-wide basis. All counts and amounts include cancels but do not include totals from training or voided transactions.				
NumDepos	Int	4	8	Number of deposits.
AmtDepos	Int	4	12	Deposit amount.
NumDepcl	Int	4	16	Number of deposit cancels.
AmtDepcl	Int	4	20	Deposit cancel amount.
NumDeprt	Int	4	24	Number of deposit returns.
AmtDeprt	Int	4	28	Deposit return amount.
NumDisco	Int	4	32	Number of discounts.
AmtDisco	Int	4	36	Discount amount.
NumManuf	Int	4	40	Number of manufacturer coupons.
AmtManuf	Int	4	44	Manufacturer coupon amount.
GrossPos	Int	4	48	Standalone gross positive.
GrossNeg	Int	4	52	Standalone gross negative.
NumTrans	Int	4	56	Number of transactions. Does not include training or voided transactions.
DateTime	PD	5	60	Date and time when current period started or previous period ended, in the format `YYMMDDHHmm`.

Field Name	Type	Length	Decimal Offset	Description
AmtTaxsa	Int	4	65	Standalone tax amount
Reserved	ASCII	3	69	Reserved.
UsrExit	Int	20	72	Five 4-byte fields for use with extensions. The customer loyalty program can optionally use any or all of the five user exit fields for accumulating automatic coupon counts and amounts, depending on settings for *Department File Exit Fields for Auto Coupon Totals* options in Loyalty -> Activity -> Totals personalization. See the online help or the *SurePOS ACE Planning and Installation Guide* for information on configuring SurePOS ACE for this use.
Totals are maintained for each of ten departments, 1 to 10, in this first record. (Totals from training or voided transactions are not maintained.) In the following table, *n* in the Offset formula can have a value from 1 to 10, which is the department number.				
Department	PD	2	((n-1)*42)+ 92	Department ID (n).
NumSales	Int	4	((n-1)*42)+ 94	Number of item sales.
AmtSales	Int	4	((n-1)*42)+ 98	Amount of item sales.
NumCancl	Int	4	((n-1)*42)+ 102	Number of item sale cancels.
AmtCancl	Int	4	((n-1)*42)+ 106	Amount of item sale cancels.
NumRefnd	Int	4	((n-1)*42)+ 110	Number of refunds.
AmtRefnd	Int	4	((n-1)*42)+ 114	Amount of refunds.
NumStcpn	Int	4	((n-1)*42)+ 118	Number of store coupons.
AmtStcpn	Int	4	((n-1)*42)+ 122	Amount of store coupons.
NumTrans	Int	4	((n-1)*42)+ 126	Number of transactions that included an item from this department.
UEUnion	Union	4	((n-1)*42)+ 130	Defines these four bytes as one of two possible fields, either UsrExit or KeyedCounts.
UsrExit	Int	4	((n-1)*42)+ 130	This field is reserved for user extensions. This is the active union definition when you *have not* chosen to retain keyed counts through Options -> Store -> Accounting personalization.
KeyedCounts	Structure	4	((n-1)*42)+ 130	This is the active union definition when you *have* chosen to retain keyed counts through Options ->

Field Name	Type	Length	Decimal Offset	Description
				Store -> Accounting personalization. This field defines the KeyedDeptSales and KeyedUPCSales fields.
KeyedDeptSales	Int	2	((n-1)*42)+ 130	Keyed department sales count.
KeyedUPCSales	Int	2	((n-1)*42)+ 132	Keyed UPC sales count.

Department Totals/Records 2-84

You can maintain totals for departments 11 through 999 in records 2 through 84. Each record covers 12 departments. Totals from training, voided, or standalone transactions are not maintained in these records.

To determine which record a department is in, use this formula:

```
(department number + 2) / 12 = record number
```

If there is any remainder (no matter how slight), round the record number up to the next integer.

Field Name	Type	Length	Offset	Description
PDTTime	PD	4	0	Day and time of the transaction which last updated this record. Format: MMDDHHmm
Restart	Int	4	4	This field is used as a control field on a system restart to determine if the record has been updated by a particular transaction.

The rest of each of records 2 through 84 contains data for 12 departments. The *n* in the Offset formula can have a value from 11 to 22, which is the relative department number:

1. If the department number is 23 or more, subtract the last value that is evenly divisible by 12 from the department number by:
   a. Dividing the department number by 12 and rounding down to the integer value. (Truncate the remainder.)
   b. Multiply the integer by 12 and subtract it from the department number.

   Here is pseudo-code for this step:

   ```
   integer = integer value of department / 12
   relative_department = department - (integer*12)
   ```

2. If the result is less than 11, add 12 to it.

   Here is pseudo-code for this step:

   ```
   if relative_department < 11 then relative_department=relative_department + 12
   ```

Field Name	Type	Length	Offset for Department n	Description
Department	PD	2	((n-11)*42)+ 8	Department number (*n*)
NumSales	Int	4	((n-11)*42)+ 10	Number of item sales
AmtSales	Int	4	((n-11)*42)+ 14	Amount of item sales
NumCancl	Int	4	((n-11)*42)+ 18	Number of item sale cancels
AmtCancl	Int	4	((n-11)*42)+ 22	Amount of item sale cancels
NumRefnd	Int	4	((n-11)*42)+ 26	Number of refunds
AmtRefnd	Int	4	((n-11)*42)+ 30	Amount of refunds
NumStcpn	Int	4	((n-11)*42)+ 34	Number of store coupons
AmtStcpn	Int	4	((n-11)*42)+ 38	Amount of store coupons
NumTrans	Int	4	((n-11)*42)+ 42	Number of transactions that included an item from this department
UEUnion	Union	4	((n-11)*42)+ 46	Defines these four bytes as one of two possible fields, either UsrExit or KeyedCounts.
UsrExit	Int	4	((n-11)*42)+ 46	This field is reserved for user extensions. This is the active union definition when you *have not* chosen to retain keyed counts through Options -> Store -> Accounting personalization.
KeyedCounts	Structure	4	((n-11)*42)+ 46	This is the active union definition when you *have* chosen to retain keyed counts through Options -> Store -> Accounting personalization. This field defines the KeyedDeptSale and KeyedUPCSales fields.
KeyedDeptSales	Int	2	((n-11)*42)+ 46	Keyed department sales count.
KeyedUPCSales	Int	2	((n-11)*42)+ 48	Keyed UPC sales count.

Extended Department Totals

Logical name	<EAMDEPTC>, <EAMDEPTP>, <EAMDEPTO>, <EAMDEPCL>, <EAMDEPPL>, <EAMDEPOL>, <EAMDEPWK>
Data object reference	*ISDepartmentStatisticsData* <SADPSDAT.CPP>
Organization	Fixed direct

Distribution class	Mirrored
File copies	Short and long files for current, previous, and old periods
Record length	512 bytes

Record 1

Field Name	Type	Length	Offset	Description
PDTTime	PD	4	0	Date and time of the transaction that last updated this record. Format: `MMDDHHmm`
Restart	Int	4	4	Control field for a system restart. Checkout Support uses it to determine if this record has been updated by a particular transaction it is replaying.
The next part of the record maintains store totals on a store-wide basis. All counts and amounts include cancels but do not include totals from training or voided transactions.				
NumDepos	Int	4	8	Number of deposits.
AmtDepos	Int	4	12	Deposit amount.
NumDepcl	Int	4	16	Number of deposit cancels.
AmtDepcl	Int	4	20	Deposit cancel amount.
NumDeprt	Int	4	24	Number of deposit returns.
AmtDeprt	Int	4	28	Deposit return amount.
NumDisco	Int	4	32	Number of discounts.
AmtDisco	Int	4	36	Discount amount.
NumManuf	Int	4	40	Number of manufacturer coupons.
AmtManuf	Int	4	44	Manufacturer coupon amount.
GrossPos	Int	4	48	Standalone gross positive.
GrossNeg	Int	4	52	Standalone gross negative.
NumTrans	Int	4	56	Number of transactions. Does not include training or voided transactions.
DateTime	PD	5	60	Date and time when current period started or previous period ended, in the format `YYMMDDHHmm`.
AmtTaxsa	Int	4	65	Standalone tax amount.
Reserved	ASCII	3	69	Reserved.
UsrExit	Int	36	72	Nine 4-byte fields for use with extensions. The customer loyalty program can optionally use any or all of the first five user exit fields for

Field Name	Type	Length	Offset	Description
				accumulating automatic coupon counts and amounts, depending on settings for *Department File Exit Fields for Auto Coupon Totals* options in Loyalty -> Activity -> Totals personalization. See the online help or the *SurePOS ACE Planning and Installation Guide* for information on configuring SurePOS ACE for this use.
Totals are maintained for each of eight departments, 1 to 8, in this first record. (Totals from training or voided transactions are not maintained.) In the following table, *n* in the Offset formula can have a value from 1 to 8, which is the department number.				
Department	PD	2	((n-1)*50)+ 108	Department ID (n).
NumSales	Int	4	((n-1)*50)+ 110	Number of item sales.
AmtSales	Int	4	((n-1)*50)+ 114	Amount of item sales.
NumCancl	Int	4	((n-1)*50)+ 118	Number of item sale cancels.
AmtCancl	Int	4	((n-1)*50)+ 122	Amount of item sale cancels.
NumRefnd	Int	4	((n-1)*50)+ 126	Number of refunds.
AmtRefnd	Int	4	((n-1)*50)+ 130	Amount of refunds.
NumStcpn	Int	4	((n-1)*50)+ 134	Number of store coupons.
AmtStcpn	Int	4	((n-1)*50)+ 138	Amount of store coupons.
NumTrans	Int	4	((n-1)*50)+ 142	Number of transactions that included an item from this department.
UEUnion	Union	4	((n-1)*50)+ 146	Defines these four bytes as one of two possible fields, either UsrExit or KeyedCounts.
UsrExit	Int	4	((n-1)*50)+ 146	This field is reserved for user extensions. This is the active union definition when you *have not* chosen to retain keyed counts through Options -> Store -> Accounting personalization.
KeyedCounts	Structure	4	((n-1)*50)+ 146	This is the active union definition when you *have* chosen to retain keyed counts through Options -> Store -> Accounting personalization. This field defines the KeyedDeptSales and KeyedUPCSales fields.

Field Name	Type	Length	Offset	Description
KeyedDeptSales	Int	2	((n-1)*50)+ 146	Keyed department sales count.
KeyedUPCSales	Int	2	((n-1)*50)+ 148	Keyed UPC sales count.
AmtMlcpn	Int	4	((n-1)*50)+ 150	Amount of multiplied coupons when you have chosen the *Accumulate Bonus Coupons by Dept* option through Loyalty -> Activity -> Totals personalization.
XtndUsr1	Int	4	((n-1)*50)+ 154	Reserved for user extensions.

Department Totals/Records 2-101

You can maintain totals for departments 9 through 999 in records 2 through 101. Each record covers 10 departments. Totals from training, voided, or standalone transactions are not maintained in these records.

To determine which record a department is in, use this formula:

```
(department number + 2) / 10 = record number
```

If there is any remainder (no matter how slight), round the record number up to the next integer.

Field Name	Type	Length	Offset	Description
PDTTime	PD	4	0	Day and time of the transaction which last updated this record. Format: MMDDHHmm
Restart	Int	4	4	This field is used as a control field on a system restart to determine if the record has been updated by a particular transaction.

The rest of each of records 2 through 101 contains data for 10 departments. The *n* in the Offset formula can have a value from 9 to 18, which is the relative department number:

1. If the department number is 19 or more, subtract the last value that is evenly divisible by 10 from the department number.
2. If the result is less than 9, add 10 to it.

   Here is pseudo-code for this step:

   ```
   if (relative_department < 9)
      relative_department = relative_department + 10;
   ```

Field Name	Type	Length	Offset for Department n	Description
Department	PD	2	((n-9)*50)+ 8	Department number (*n*)
NumSales	Int	4	((n-9)*50)+ 10	Number of item sales

Field Name	Type	Length	Offset for Department n	Description
AmtSales	Int	4	((n-9)*50)+ 14	Amount of item sales
NumCancl	Int	4	((n-9)*50)+ 18	Number of item sale cancels
AmtCancl	Int	4	((n-9)*50)+ 22	Amount of item sale cancels
NumRefnd	Int	4	((n-9)*50)+ 26	Number of refunds
AmtRefnd	Int	4	((n-9)*50)+ 30	Amount of refunds
NumStcpn	Int	4	((n-9)*50)+ 34	Number of store coupons
AmtStcpn	Int	4	((n-9)*50)+ 38	Amount of store coupons
NumTrans	Int	4	((n-9)*50)+ 42	Number of transactions that included an item from this department
UEUnion	PD	4	((n-9)*50)+ 46	Defines these four bytes as one of two possible fields, either UsrExit or KeyedCounts.
UsrExit	Int	4	((n-9)*50)+ 46	This is a field reserved for user extensions. This is the active union definition when you *have not* chosen to retain keyed counts through Options -> Store -> Accounting personalization.
KeyedCounts	PD	4	((n-9)*50)+ 46	This is the active union definition when you *have* chosen to retain keyed counts through Options -> Store -> Accounting personalization. This field defines the KeyedDeptSales and KeyedUPCSales fields.
KeyedDeptSales	Int	2	((n-9)*50)+ 46	Keyed department sales count.
KeyedUPCSales	Int	2	((n-9)*50)+ 48	Keyed UPC sales count.
AmtMlcpn	Int	4	((n-9)*50)+ 50	Amount of multiplied coupons when you have chosen the *Accumulate Bonus Coupons by Dept* option through Loyalty -> Activity -> Totals personalization.
XtndUsr1	Int	4	((n-9)*50)+ 54	Reserved for user extensions.

EAMDMCTL (Delayed Data Maintenance Control File)

The Delayed Data Maintenance Control file defines a maintenance batch. The key for this file is the batch ID and host sequence number. Delayed data maintenance control records for batches created in-store are deleted when the batch is cleared. All externally created batch control records are only marked as cleared and are no longer displayed when cleared. These records must be deleted by the external process that created them.

An image copy of this file is kept on the alternate file server.

Note for SA Users

See Table 1 for a description of how the SurePOS ACE file differs from the 4680-4690 Supermarket Application file.

Logical name	<EAMDMCTL>
Data object reference	*ISSAAceTriggerData* <SAACTDAT.CPP>
Organization	Keyed
Distribution class	Mirrored per update
File copies	1
Record length	48 bytes
Key length	4 bytes: BatchId / Sequence

Host Record

Field Name	Type	Length	Decimal Offset	Description
BatchId	PD	3	0	The batch ID ranges from 010101 to 999996 for operator and immediate maintenance. For timer maintenance, the format is `MMDDHH` (month, day, hour).
Sequence	PD	1	3	Host sequence number
RequestType	Int	1	4	Request types: **Bit 0 X'80'** ADDMI-generated batch. **Bit 1 X'40'** Modify data file. **Bit 2 X'20'** Modify/restore data file. **Bit 3 X'10'** Report data from file. **Bit 4 X'08'** TOF batch. **Bit 5 X'04'** Protected. Source may not be edited.

Field Name	Type	Length	Decimal Offset	Description
				Bit 6 X'02' Source contains host messages only. Note: These batches can only be viewed, printed, or cleared. Item changes in these files cannot be executed. **Bit 7 X'01'** Do not write changes to Item Record Change file.
MaintenanceType	Int	1	5	Maintenance types: **Bit 0 X'80'** Permanent batch. Neither operator nor normal DDM cleanup can delete trigger. **Bit 1 X'40'** Operator-controlled maintenance. **Bit 2 X'20'** Scheduler-controlled maintenance. **Bit 3 X'10'** Immediate maintenance. **Bit 4 X'08'** Start executable named in TargetFile field. **Bit 5 X'04'** File Trigger Batch; used with bit 4. **Bit 6 X'02'** Reserved. **Bit 7 X'01'** Source file is in controller format.
ProcessingOptions	Int	1	6	Processing options: **Bit 0 X'80'** Reserved **Bit 1 X'40'** Do not print/display source file in-store. **Bit 2 X'20'** Replace can do an add.

Field Name	Type	Length	Decimal Offset	Description
				Bit 3 X'10' Add can do a replace. **Bit 4 X'08'** Source file contains price changes. **Bit 5 X'04'** Reserved. **Bit 6 X'02'** If add does a replace, log to error log. **Bit 7 X'01'** GTIN enabled
RestoreBatchId	PD	3	7	Restore batch ID, restore batch receives this ID. Also, after a batch is executed, this field contains time of execution in MMDDHH format.
StartBlock	Int	2	10	Starting 256 byte block in the source file for the batch's data if there are multiple batches in the source file. Use if `ProcessingOptions` bit 5 is set to 1.
SourceUnion	Union	8	12	Defines these eight bytes as one of two possible fields, either `ACESourceFile` or `EAMSourceFile`.
ACESourceFile	ASCII	8	12	Eight-character source file name. If the *Use Expanded Batch File Limit* option is on, this field might contain a logical name. For TOF requests, this field contains `EAMITEMR`.
EAMSourceFile	Structure	8	12	Defines the following `SourceFile` and `Reserved` fields for compatibility with 4680-4690 Supermarket Application.
SourceFile	ASCII	6	12	Source file name.
Reserved	ASCII	2	18	Reserved (must be two ASCII periods, X'2E2E').
TargetUnion	Union	8	20	Defines these eight bytes as one of two possible fields, either `ACETargetFile` or `EAMTargetFile`.
ACETargetFile	ASCII	8	20	Eight-character target file name. If the *Use Expanded Batch File Limit* option is on, this field might contain a logical name. For TOF requests, this field contains `ACETIRBL`.
EAMTargetFile	Structure	8	20	Defines the following `TargetFile` and `Reserved` fields for compatibility with 4680-4690 Supermarket Application.

Field Name	Type	Length	Decimal Offset	Description
TargetFile	ASCII	6	20	Target file name.
Reserved	ASCII	2	26	Reserved (must be two ASCII periods, X'2E2E').
ReportUnion	Union	8	28	Defines these eight bytes as one of two possible fields, either `ACEReportFile` or `EAMReportFile`.
ACEReportFile	ASCII	8	28	Eight-character report file name. If the *Use Expanded Batch File Limit* option is on, this field might contain a logical name.
EAMReportFile	Structure	8	28	Defines the following `ReportFile` and `Reserved` fields for compatibility with 4680-4690 Supermarket Application.
ReportFile	ASCII	6	28	Report file name.
Reserved	ASCII	2	34	Reserved (must be two ASCII periods, X'2E2E').
SourceRecordCnt	Int	2	36	Number of records in the source file.
ErrorsLogged	Int	2	38	Number of errors logged for this batch.
FirstError	Int	4	40	Result of *PTRRTN* at first error message.
ExecStatus	Int	1	44	Execution status: **Bit 0 X'80'** This batch is waiting for execution. **Bit 1 X'40'** Price required flag set **Bit 2 X'20'** Maintenance completed successfully. **Bit 3 X'10'** Operator posted batch for deletion by host. (The control record is deleted if `MaintenanceType` bit 7 is on.) **Bit 4 X'08'** Error in this control record prevented processing. **Bit 5 X'04'** Maintenance attempted but not completed because of errors. **Bit 6 X'02'** This batch is currently being processed.

Field Name	Type	Length	Decimal Offset	Description
				Bit 7 X'01' This batch is pending (available for execution).
ProcessingOptions2	Int	1	45	**Bit 4 X'08'** Reserved Processing options that are used when bit 5 in the *RequestType* field is on: **Bit 5 X'04'** off = TOF reload request; on = TOF rebuild and reload request. **Bit 6 X'02'** on = Started program runs in its own background slot, if bit 4 in *RequestType* field is on and bit 7 of this field is off. **Bit 7 X'01'** on = Asynchronous; off = Synchronous. Only applies when bit 4 in RequestType field is on.
Reserved	ASCII	2	46	Reserved

Store Record

Field Name	Type	Lengt h	Decimal Offset	Description
BatchId	PD	3	0	Batch identifier (999999)
SequenceNum	PD	1	3	Sequence number (00)
Counter	PD	2	4	Counter that is used to create unique file names for delayed maintenance data and report files created in the store. If the *Use Expanded Batch File Limit* option is off, the range is 0 - 999; if the option is on, the range is 0 - 9999.
Reserved	ASCII	42	6	Reserved

ADDMI Record

Field Name	Type	Length	Decimal Offset	Description
BatchId	PD	3	0	Batch identifier (999998)
SequenceNum	PD	1	3	Sequence number (00)
Reserved	ASCII	1	4	Counter ranging from 0 to 999 that is used to create unique file names for delayed maintenance data and report files created in the store.
MaintenanceType	Int	1	5	Maintenance types 0x8C, which are bits 0, 4, and 5. See the Host Record for a description of maintenance type bits.
Reserved	ASCII	6	6	Reserved.
SourceFile	ASCII	8	12	Source file name (EAMMAINT).
TargetFile	ASCII	8	20	Target file name (ACEADDML).
ReportFile	ASCII	8	28	Report file name (ASCII blanks).
BatchesProcessed	Int	2	36	Number of batches processed prior to error.
FileNumber	Int	2	38	Source file number being processed when error occurred. (EAM@@???, values 1-999).
Reset	Int	4	40	Restart indicator: **0** ADDMI process was successful, no restart condition. **>0** ADDMI processing failed. The value in this field represents the number of the batch being processed when the failure occurred.
Reserved	ASCII	4	44	Reserved.

EAMDMERR (Delayed Data Maintenance Error File)

Errors that occur during DDM processing are logged to the Delayed Maintenance Error file. The Alternate Delayed Data Maintenance Interface (ADDMI) logs errors in the Delayed Maintenance Error file when records are omitted from the delayed maintenance source file because input data is not valid or because a replacement record is not on file.

An image version of this file is on the alternate file server.

Note for SA Users

See Table 1 for a description of how the SurePOS ACE file differs from the 4680-4690 Supermarket Application file.

Logical name	<EAMDMERR>

Data object reference	*ISDelayedDataMaintBatchInstructionErrorDetail* <`DDMBIEDD.CPP`>
Organization	Variable sequential
Distribution class	Mirrored per Update
File copies	1
Record length	Variable

Field Name	Type	Length	Description
DateTime	PD	5	Date and time when error occurred, in the format `YYMMDDHHmm`.
BatchId	PD	3	Batch ID of batch with error.
SequenceId	PD	1	Sequence ID of batch with error.
MaintenanceType	ASCII	1	Maintenance type: 1 if operator-initiated 2 if timer maintenance 3 if immediate maintenance
TargetFile	ASCII	8	Target file name. If the *Use Expanded Batch File Limit* option is on, this field might contain a logical name.
SourceRecordNum	ASCII	v?	Source record number when error occurred. Zeros if indeterminate.
RecordKey	PD	13	Key of record when error occurred.
ErrorCode	ASCII	2	Error code.

These are the ErrorCode values:

07	Record to be added already exists. Will not replace the record.
09	Record to be added already exists. Will replace the record.
11	Unable to open source file.
12	Unable to open target file.
13	Unable to open report file.
24	Record to be replaced was not found. An add was done.
25	Record to be deleted was not found.
26	Record to be modified was not found. (Price change)
30	Error writing the record to the target file.
35	Record to be reported was not found.
42	Error writing record to report file.
43	Error writing the record to source file.
46	Unable to create the restore source file.
47	Error reading the target file.
49	Unable to write the record to the control file for the restore batch.
50	Unable to open, create, read, or write the source batch.

61	Command field in source data is not valid. Record was ignored.
62	No source data for add/replace. Record was ignored.
63	Control data for batch was not valid.
64	Modification of an aliased record (pricing method equal to five) is not allowed. Record was ignored.
65	ESL price change not allowed. Record was ignored.
70	Execute process is already active.
71	Process ended with errors.
72	Unknown process status.
73	DIF cannot be reached.
92	No space is available in EAMFBACT to add additional targeted coupons.
93	Failed to open EAMFBACT.DAT.
94	An ADDMI add caused a replace, but used the old price or department.
95	An ADDMI file that was not valid was discarded.
96	An ADDMI batch that was not valid was discarded.
97	An ADDMI record that was not valid was discarded.
98	A department number that was not valid was found. Record was discarded.
99	ADDMI discarded an update because the record was not on file.

EAMDR* (Item Movement Statistics Report Data File)

The eamdrxxx files are output files that are generated when a report item movement batch file is executed. The *xxx* in the report file name matches the value in the eamddxxx source file that contains the list of records to be reported.

Reporting is performed on the short item movement file for the current period at the time that the batch is executed. Records that are present in the source file but not in the item movement file will be flagged as errors (DDM error code 35), and will not be present in the item movement statistics report file.

- If the *Use Expanded Batch File Limit* option is turned off in Personalization, the counter value is always three numeric characters. There can be a maximum of 999 unique files at the same time.
- If the option is turned on, there can be a maximum of 9999 unique files at the same time.
  - If the counter value is 0 - 999, then the actual counter value is appended to the file name as follows:

Table 48. Source File Naming Convention for Counter Values 0-999

Source File Number	Batch Source	Report File
001-999	EAMDD:xxx	EAMDR:xxx
Example: 123	EAMDD:123	EAMDR:123

  - If the counter value is 1000 - 9999, then logical names are used to define the file name as follows:

Table 49. Source File Naming Convention for Counter Values 1000-9999

Source File Number	Batch Source	Report File
1000-9999	DMBx:yyy	DMRx:yyy
Example: 4712	DMB4:712	DMR4:712

Logical name	<EAMDR:>NNN
Data object reference	*ISDelayedDataMaintItemMovementStatisticsReportData* <DDMIMSRD.CPP>
Organization	Sequential
Distribution class	Mirrored at close
File copies	1
Record length	14 bytes

Field Name	Type	Length	Description
Key	PD	6	Item record.
Count	Long	4	Number of elements.
Restart	Long	4	Value used by Checkout Support during restart and recovery.

EAMEXCP* (Exception Log File)

The sequential Exception Log file contains entries describing exceptional user actions that impact totals or security. It does not attempt to define exceptional hardware or system software conditions, but rather in-store operational situations that are unusual enough to warrant tracking.

The Exception Log resides on the file server store controller with an image version on the alternate file server. Checkout Support renames the log to an old period file and allocates a new log when you request this during the store closing.

You can archive Exception Log files under any name. SurePOS ACE supports reports based on archived files that use the naming convention of eameyymd.xxz, where:

eame	Always first 4 characters of file name
yy	Last two digits of the year (for example, 04 if 2004)
m	Indicates month (1=January, ..., 9=September, A=October, B=November, C=December)
d	Indicates day of month (1=1, ..., 9=9, A=10, ..., V=31)
xx	Indicates whether the file is compressed. (DC=compressed, DB=uncompressed)
z	Number that starts at 0, increments for each file saved, and resets to 0 at the beginning of each day.

During a close, SurePOS ACE saves an uncompressed file in the archive directory unless it has already saved the specified number of files. During the next close, if the compressed limit is greater than 0, it compresses the oldest file. Then, SurePOS ACE saves the newest file uncompressed. During the next close after the compressed limit is reached, it erases the oldest compressed file.

SurePOS ACE saves archived Exception Log files in the directory you specify in the `Archive Directory` option in Options -> Archive personalization. The files have a compound at close distribution type.

If you select an archived file as the basis for the Exception Log Report, SurePOS ACE creates a temporary uncompressed copy of the file and stores the copy in the adx_idt4 directory. The naming convention for the temporary file is eamexcpy.dat, where the value of *y* is 0-5. The temporary work file is kept until the next time that the report is requested for an archived file. If you start the Reports function and request multiple Exception Log Reports using archived files, a maximum of six temporary files is kept. When you request the seventh report, the oldest temporary file is deleted.

You can set the maximum number of compressed and uncompressed Exception Log files to keep in personalization.

Checkout Support creates these Exception Log entries:

- X'01' - "Item Entry Exception (0x01)" on page 156
- X'02' - "Discount or Tax Exemption (0x02)" on page 162
- X'03' - "Tender Override or Exception (0x03)" on page 163
- X'04' - "Void Transaction (0x04)" on page 166
- X'05' - "Tender Exchange (0x05)" on page 167
- X'06' - "No-Sale Transaction (0x06)" on page 169
- X'07' - "Standalone Entry/Exit (0x07)" on page 170
- X'08' - "Invalid Item Rejected (0x08)" on page 171
- X'09' - "Terminal Price Change (0x09)" on page 172
- X'11' - "Operator Sign-on/Sign-off (0x11)" on page 173
- X'12' - "Non-Sales Transaction (0x12)" on page 174
- X'13' - "Cashier Loan (0x13)" on page 176
- X'14' - "Cashier Pickup (0x14)" on page 177
- X'15' - "Store Totals Closed (0x15)" on page 178
- X'16' - "Missing, Found, or Duplicate Transaction (0x16)" on page 179
- X'17' - "Critical Hardware Failure (0x17)" on page 180
- X'18' - "Tender Count Transaction (0x18)" on page 181
- X'19' - "User Data Entered (0x19)" on page 182

Processes other than Checkout Support create these Exception Log entries:

- X'20' - "Cashier Loan from Controller (0x20)" on page 182
- X'21' - "Cashier Pickup from Controller (0x21)" on page 183
- X'22' - "Tender Count from Controller (0x22)" on page 184
- X'23' - "Transfer Tender from Controller (0x23)" on page 185
- X'24' - "Carry Forward Tender from Controller (0x24)" on page 186
- X'25' - "Miscellaneous Transaction (0x25)" on page 187
- X'26' - "Request for Close of Store Totals (0x26)" on page 188
- X'30' - "Add, Alter, or Delete of an Operator Authorization Record (0x30)" on page 189
- X'31' - "Add, Alter, or Delete of an Item Record (0x31)" on page 190
- X'32' - "Add, Alter, or Delete of a Tender Verification Record (0x32)" on page 191
- X'33' - "Add, Alter, or Delete of an Item Movement List Record (0x33)" on page 192
- X'40' - "Report Executed (0x40)" on page 192
- X'41' - "Personalization Executed (0x41)" on page 193
- X'43' - "DDM Executed (0x43)" on page 195
- X'44' - "DDM Item Record Batch Editing (0x44)" on page 196
- X'45' - "DDM Tender Verification Batch Editing (0x45)" on page 197
- X'46' - "DDM Item Movement Batch Editing (0x46)" on page 198
- X'47' - "DDM Operator Authorization Batch Editing (0x47)" on page 198

- X'51' - “Unable to Read TLog Record (0x51)” on page 200
- X'52' - “Free Disk Space (0x52)” on page 200
- X'55' - “Transfer Processing (0x55)” on page 201
- X'80' - “WIC EBT (0x80)” on page 201
- X'81' - “WIC EBT Card (0x81)” on page 202
- X'82' - “Customer Entry (0x82)” on page 202
- X'83' - “Sales Transaction (0x83)” on page 203
- X'89' - “User Log (0x89)” on page 204
- X'99' - “User Log (0x99)” on page 205

Exception log record types X'89' and X'99' are reserved for user application use. Record type X'99' should be used for records created in the terminal or for records written in Transaction Summary Log strings of type 20 which are processed by checkout support. Record type X'89' should be used for entries written directly to the exception log file eamexcpt.

Logical name	<EAMEXCPT>, <EAMEXCPO>
Storage object reference	• *ISExceptionStorage* `<EXCPTSTR.CPP>` • *ISSAExceptionStorage* `<SAEXCSTR.CPP>` • *ISTSExceptionStorage* `<TSEXCSTR.CPP>` • *ISCSExceptionStorage* `<CSEXCSTR.CPP>`
Data layout object reference	• *ISSAExceptionData* `<SAEXCDAT.CPP>` • *ISTSExceptionData* `<TSEXCDAT.CPP>`
Organization	Variable sequential
Distribution class	Mirrored per update
File copies	Current, old
Record length	Variable

Item Entry Exception (0x01)

Data object reference: *ISTSExceptionData* <`TSEXCDAT.CPP`>

An item entry exception record is written for every item for which any of the following applies:

- A price is entered when it is not required
- An override is required to sell the item
- A log limit is exceeded for the item type
- A weight or volume is keyed
- The item record designates all sales of the item to be logged

Note:

1. If you use delayed manager overrides, SurePOS ACE does not record the override number in the OVERRIDE field, which is empty.
2. Attempts to sell a restricted item are recorded in record type X'08'.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'01').

Field Name	Type	Length	Description
Operator	PD	v5	Operator number.
ITEMCODE	PD	v7	Item code.
INDICAT0	PD	1	(Packed string representation of 1-byte integer) **Bit 0** Reserved. **Bit 1 - X'40'** Rain check item. **Bit 2 - X'20'** Fuel item. **Bit 3 - X'10'** All sales logged. **Bit 4 - X'08'** Price override. **Bit 5 - X'04'** Override entered. **Bit 6 - X'02'** Log limit exceed. **Bit 7 - X'01'** Weight item.
INDICAT1	ASCII	v2	0 Normal item sale. 1 Deposit. 2 Refund. 3 Deposit return. 4 Miscellaneous transaction receipt (sale). 5 Miscellaneous transaction payout (refund). 6 Manufacturer coupon. 7 Store coupon. 8 Item sale cancel. 9 Deposit cancel. 10 Markdown store coupon.
Amount	ASCII	v9	Amount ASCII v9 Amount - positive with the following exceptions: refunds, all coupons, deposit returns, deposit cancels, miscellaneous transaction refunds, and miscellaneous sales cancels.
SALEMEAS	ASCII	v6	Entered quantity, weight, or volume
FILEDEAL	ASCII	v2	Overridden item record deal quantity
FILEPRIC	ASCII	10	Overridden item record pricing data or imbedded price for items of number system 2
OVERRIDE	PD	v4	Index to manager override number

Field Name	Type	Length	Description
REASON	PD	v1	Override reason **X'00'** No override entered. **X'01'** Exceeded difference between keyed price and file price. **X'02'** Under department minimum price or over maximum limit. **X'03'** Exceeded entry limit by item type in transaction group limit. **X'04'** Price entered is less than minimum limit by item type. **X'05'** Exceeded maximum limit by item type. **X'06'** Cancel item not sold in transaction. **X'07'** Accept coupon for item not sold. **X'08'** Entered department lookup key with modifier. **X'09'** Exceeded weight limit. **X'10'** Bonus coupon value is more than item value. **X'11'** Non-WIC item not for sale in WIC transaction. **X'13'** Bypass date of birth check. **X'14'** Exceeded volume limit. **X'15'** Exceed discount rate limit **X'16'** Exceed discount amount limit

Field Name	Type	Length	Description
			X'17' Exceed tax exemption limit **X'20'** Bypass tender franking **X'21'** Exceed negative entry limit for transaction **X'22'** Exceed negative total limit for transaction **X'23'** Exceed tender type amount limit **X'25'** Exceed tender type change limit **X'27'** Exceed tender verification rejection **X'28'** Perform delayed override **X'29'** Bypass tender verification access failure **X'30'** Item Void in Negative Transaction. **X'35'** Exceed void transaction limit **X'36'** Coupon start date not met **X'37'** Coupon expiration date exceeded **X'38'** Coupon retailer ID not allowed **X'39'** Coupon weight not met **X'40'** Exceed tender exchange limit **X'41'** Bypass tender exchange ranking

Field Name	Type	Length	Description
			X'42' Exceed tender fee refund limit
REASON (cont.)	as above	as above	**X'44'** Special sign-off not authorized **X'45'** Exit special sign-off without password **X'46'** No-sale procedure inside transaction **X'47'** Exceed till exchange limit **X'50'** Sign-on when operator already active **X'51'** Bypass operator authorization access failure **X'52'** Re-initialize transferred terminal **X'55'** Exceed cashier loan limit **X'56'** Exceed cashier pickup limit **X'57'** Exceed maximum number of suspends allowed **X'58'** Force retrieve is required **X'59'** Force suspend is required **X'60'** No electronic signature **X'61'** Keyed account number **X'62'** SurePOS ACE EPS declined **X'63'** SurePOS ACE EPS offline limits exceeded.

Field Name	Type	Length	Description
			X'64' GIPC down **X'65'** Bypass paper error **X'66'** More points needed **X'67'** Keyed customer ID **X'68'** New Customer ID **X'69'** Daily card use limit exceeded **X'70'** Non-sale price verify/change **X'71'** Duplicate prescription **X'72'** Void prescription without a previous sale **X'73'** Return prescription without a previous sale **X'74'** Prescription information not found **X'75'** Pharmacy application inaccessible **X'77'** Operator age restriction overridden. **X'78'** MICR not present **X'79'** Low MICR signal **X'80'** Account number **X'81'** Transit code

Field Name	Type	Length	Description
			X'82' Check number **X'83'** Foreign transit **X'85'** Exceed rain check override limit **X'89'** Immediate quantity restriction
REASON (cont.)	as above	as above	The following reason codes are reserved for customers and Toshiba Global Commerce Solutions Business Partners: **X'90'-X'96'** Toshiba Global Commerce Solutions Business Partners **X'97'-X'99'** Customers
USREXIT	ASCII	v200	User field reserved for user extensions
EnteredReasonCode	PD	1	User-entered price-override reason code associated with the item entry
VoidReasonCode	PD	1	User-entered void reason code associated with the item entry
Index	PD	v1	The index in Personalization of the option that contains this manager override ID.
Initials	ASCII	v3	The initials defined in Personalization for this manager override ID.
ALTPRICE	PD	1	Specifies the type of pricing data being used. 00 = Primary pricing data; 01 = Alternate pricing data
BARCODE	ASCII	v80	Bar code value, if GS1 DataBar. If this is a GS1 DataBar Expanded bar code, any field delimiters (X'1D') are replaced with a hyphen (X'2D').

Discount or Tax Exemption (0x02)

Data object reference: *ISTSExceptionData* <`TSEXCDAT.CPP`>

This record is written whenever a transaction discount or tax exemption is taken.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number
TransNum	PD	2	Transaction number
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`
Code	PD	1	Type of Exception Log entry (value = X'02')

Field Name	Type	Length	Description
Operator	PD	v5	Operator number
AMTDISCO	ASCII	v8	Discount amount
AMTTAXEX	ASCII	v8	Total taxable amount for exemption
DISRATE	PD	v2	Discount rate
DISGROUP	PD	v1	Discount group number
OVERRIDE	PD	v4	Override number
REASON	PD	v1	Override reason: **X'00'** No override entered **X'15'** Exceed discount rate limit **X'16'** Exceed discount amount limit **X'17'** Exceed tax exemption limit **X'18'** Bypass tax exemption ID The following reason codes are reserved for customers and Toshiba Global Commerce Solutions Business Partners: **90-96** Toshiba Global Commerce Solutions Business Partners **97-99** Customers
PreferredCust	ASCII	v18	Preferred customer account number
USREXIT	ASCII	v200	User field for use with extensions
Index	PD	v1	The index in Personalization of the option that contains this manager override ID
Initials	ASCII	v3	The initials defined in Personalization for this manager override ID
TaxExemptID	ASCII	v20	Tax exempt ID

Tender Override or Exception (0x03)

Data object reference: *ISTSExceptionData* <`TSEXCDAT.CPP`>

The tender override or exception entry is written whenever a tender correction or tender override occurs or whenever a transaction ends with a negative balance due. A tender override can occur because verification status is overridden, a tender processing fee is forgiven, a tender limit is overridden, or a change limit is overridden.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`,
Code	PD	1	Type of Exception Log entry (value = X'03').
Operator	PD	v5	Operator number.
TENDTYPE	ASCII	1	Tender type 1 - Cash tender. 2 - Check tender. 3 - Food stamp tender. 4 - Miscellaneous tender 1. 5 - Miscellaneous tender 2. 6 - ;Miscellaneous tender 3.
TENDVAR	ASCII	1	Tender variety (numeric).
ACCOUNT	PD	v12	Card account number.
AMTTENDE	ASCII	v10	Tender amount in tender currency. Negative when the tender was canceled.
AMTBALAN	ASCII	v10	Balance due when tender was entered (final balance less earlier tenders).
OVERRIDE	PD	v4	Override number.
REASON	PD	v1	Override reason **X'00'** No override entered. **X'03'** Exceeded negative entry limit by item type or transaction group limit. **X'05'** Exceeded maximum limit by item type. **X'12'** EMV Security - No AID tendered with non-EMV card. **X'20'** Bypass tender franking. **X'21'** Exceed negative entry limit for transaction. **X'22'** Exceed negative total limit for transaction.

Field Name	Type	Length	Description
			X'23' EMV Security - No AID tendered with EMV card. **X'24'** Exceed tender type amount limit. **X'25'** Exceed tender type change limit. **X'26'** Bypass tender verification. **X'27'** Bypass tender verification rejection. **X'29'** Bypass tender verification access failure. **X'32'** Accept tender not allowed. **X'48'** EMV Security - EMV Chip Error tendered with non-EMV card. **X'49'** EMV Security - EMV Chip Error tendered with EMV card. **X'60'** No electronic signature. **X'61'** Keyed account number. **X'66'** More points needed. **X'78'** MICR not present. **X'79'** Low MICR signal. **X'80'** Account number. **X'81'** Transit code. **X'82'** Check number.

Field Name	Type	Length	Description
			X'83' Foreign transit. **X'84'** Foreign check not allowed. The following reason codes are reserved for customers and Toshiba Global Commerce Solutions Business Partners: **90-96** Toshiba Global Commerce Solutions Business Partners **97-99** Customers
PreferredCust	ASCII	v18	Preferred customer account number.
USREXIT	ASCII	v200	User field reserved for user extensions.
Index	PD	v1	The index in Personalization of the option that contains this manager override ID.
Initials	ASCII	v3	The initials defined in Personalization for this manager override ID.

Void Transaction (0x04)

Data object reference: *ISTSExceptionData* <`TSEXCDAT.CPP`>

This record is written for each voided transaction.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'04').
Operator	PD	v5	Operator number.
AMTVOID	ASCII	v10	Net sales for the voided transaction.
OVERRIDE	PD	v4	Override number.
REASON	PD	v1	Override reason X'00' - No override entered. X'35' - Exceed void transaction limit. X'64' - Approved value card cannot be deactivated (ACEGIPC is down). The following reason codes are reserved for customers and Toshiba Global Commerce Solutions Business Partners:

Field Name	Type	Length	Description
			90-96 - Toshiba Global Commerce Solutions Partners 97-99 - Customers
USREXIT	ASCII	v200	User field for use with exits.
Index	PD	v1	The index in Personalization of the option that contains this manager override ID.
Initials	ASCII	v3	The initials defined in Personalization for this manager override ID.
VOIDTRC	PD	v2	Void Transaction Reason Code: 0 - Manual/Normal Void 100 - Pump Test 101 - Suspended Transaction 102 - Terminal Transfer The following reason codes are reserved for base code, NRSC, and customers: 000-399 - Toshiba Base 400-699 - NRSC (Toshiba Global Commerce Solutions Business Partner 700-799 - Customers

Tender Exchange (0x05)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written every time a no-sale tender exchange is performed.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'05').
Operator	PD	v5	Operator number.
INDICAT0	ASCII	1	Indicators for the *from* tender type. **1** Cash tender **2** Check tender **3** Food stamp tender

Field Name	Type	Length	Description
			4 Miscellaneous tender 1 **5** Miscellaneous tender 2 **6** Miscellaneous tender 3 **7** Manufacturer coupon tender **8** Store coupon tender
INDICAT1	ASCII	1	Indicators for the *to* tender type, which are the same as for the *from* tender type.
AMTTENDE	ASCII	v8	Tender amount in tender currency.
OVERRIDE	PD	v4	Override number.
REASON	PD	v1	Override reason **X'00'** No override entered. **X'40'** Exceed tender exchange limit. **X'41'** Bypass tender exchange ranking. **X'42'** Exceed tender fee refund limit. The following reason codes are reserved for Toshiba Global Commerce Solutions Business Partners and customers: **90-96** Toshiba Global Commerce Solutions Business Partners **97-99** Customers
USREXIT	ASCII	v200	User field reserved for user extensions.
Index	PD	v1	The index in Personalization of the option that contains this manager override ID.
Initials	ASCII	v3	The initials defined in Personalization for this manager override ID.

No-Sale Transaction (0x06)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

Note: No record is written if a no sale price verify procedure is performed at the pumps.

This record is written every time a no-sale procedure is used to open the cash drawer or to perform a tender removal, till exchange, special sign-off, or special sign-on.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'06').
Operator	PD	v5	Operator number.
INDICAT0	ASCII	2	**1** Cash drawer open only **2** Tender removal **3** Till exchange **4** Special sign-off **5** Special sign-on **6** Till Contents Report **7** Price Verify **8** Reprint Tender Receipt **9** Operator Performance Report **10** Reprint Summary Receipts **11** Reprint Activation Receipts
TENDTYPE	ASCII	1	**1** Cash tender

Field Name	Type	Length	Description
			2 Check tender **3** Food stamp tender **4** Miscellaneous tender 1 **5** Miscellaneous tender 2 **6** Miscellaneous tender 3
OVERRIDE	PD	v4	Override number. SurePOS ACE does not record the item number in this field for a no-sale Price Verify transaction.
REASON	PD	v1	Override reason **X'00'** No override entered **X'45'** Exit special sign-off without password **X'46'** Open cash drawer inside transaction **X'47'** Exceed till exchange limit The following reason codes are reserved for Toshiba Global Commerce Solutions Business Partners and customers: **90-96** Toshiba Global Commerce Solutions Business Partners **97-99** Customers
USREXIT	ASCII	v200	Reserved for user extensions.
Index	PD	v1	The index in Personalization of the option that contains this manager override ID.
Initials	ASCII	v3	The initials defined in Personalization for this manager override ID.

Standalone Entry/Exit (0x07)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written every time that a terminal successfully enters or exits standalone mode.

Note: The Standalone Entry exception is only logged if the terminal can successfully despool the exception.

Field Name	Type	Length	Description
Terminal	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'07').
Operator	PD	v5	Operator number.
Type	ASCII	1	Reason for exception log entry. **1** Enter standalone **2** Exit standalone
NUMTRANS	ASCII	v4	Number of transactions in standalone mode.
GROSSPOS	ASCII	v8	Gross positive total for standalone mode.
GROSSNEG	ASCII	v8	Gross negative total for standalone mode.
NumPriceChanges	ASCII	v6	Number of offline price changes.

Invalid Item Rejected (0x08)

Data object reference: *ISTSExceptionData* <`TSEXCDAT.CPP`>

This record is written every time that an item sale is rejected because the item is not on file, is not for sale, or the item record contains invalid data. The *item not on file* condition is logged only on the second occurrence of a failed lookup attempt for an item code within a transaction with no intervening *item not on file* conditions for any other item codes.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'08').
Operator	PD	v5	Operator number.
ITEMCODE	PS	v7	Item code. Because this is a rejected item sale, it cannot be assumed that the item code contains only numeric characters.
INDICAT0	ASCII	1	**1** Not authorized for sale **2** Item not on file

Field Name	Type	Length	Description
			3 Invalid record data **4** Error in bar code **5** Sell by date past **6** Invalid alphanumeric item code
USREXIT	ASCII	v200	User field for use with exits.
BARCODE	ASCII	v80	Bar code value, if GS1 DataBar. If this is a GS1 DataBar Expanded bar code, any field delimiters (X'1D') are replaced with a hyphen (X'2D'). Carriage Return (X'0D') and New Line (X'0A') characters are replaced with the character (X'99').

Terminal Price Change (0x09)

Data object reference: *ISTSExceptionData* <`TSEXCDAT.CPP`>

This record is written every time that a price change is made from the terminal as an immediate or delayed change.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'09').
Operator	PD	v5	Operator number.
ITEMCODE	PD	v7	Item code.
NEWDEALQ	PD	v1	New deal quantity.
NEWPRICE	PD	v5	New price (as in item record).
OLDDEALQ	PD	v1	Old deal quantity.
OLDPRICE	PD	v5	Old price from item record.
BATCH	PD	v4	Batch and sequence, if delayed update.
USREXIT	ASCII	v200	User field for use with exits.

The item record data is a string containing all data from the item record except the descriptor. Item record fields are delimited by colons and all integer fields are expressed as ASCII numerics.

Operator Sign-on/Sign-off (0x11)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written every time an operator signs on or signs off an Toshiba POS terminal.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'11').
Operator	PD	v5	Operator number.
INDICAT0	ASCII	1	**1** Operator sign-on **2** Operator sign-on with new password **3** Operator sign-off **4** Operator sign-on rejected - operator not in authorization file **5** Operator sign-on rejected - incorrect password entered **6** Special sign-on rejected - incorrect password entered **7** Incorrect override number entered **8** Incorrect password entered for No-Sale Open Cash Drawer procedure **9** Unsuccessful password change **A** Unsuccessful password change - master controller not available
PASSWORD1	ASCII	v8	Password in authorization file or new password if changed. Note: When enhanced passwords are in use, this field will contain 99999999.
PASSWORD2	ASCII	v8	Invalid password or old password if changed.

Field Name	Type	Length	Description
			Note: When enhanced passwords are in use, this field will contain 99999999.
OVERRIDE	PD	v7	Override number Note: When a non-numeric override number is entered, this field will contain 0.
REASON	PD	v1	Override reason **X'00'** No override entered. **X'50'** Sign-on when already signed on. **X'51'** Bypass operator authorized access failure. **X'52'** Sign-on terminal that has not been re-initialized since it was transferred. The following reason codes are reserved for customers and Toshiba Global Commerce Solutions Business Partners: **90-96** Toshiba Global Commerce Solutions Business Partners **97-99** Customers
USREXIT	ASCII	v200	User field reserved for user extensions.
Index	PD	v1	The index in Personalization of the option that contains this manager override ID.
Initials	ASCII	v3	The initials defined in Personalization for this manager override ID.

Non-Sales Transaction (0x12)

Data object reference: *ISTSExceptionData* <`TSEXCDAT.CPP`>

This record is written whenever a training transaction, terminal transfer transaction, terminal monitor transaction, price verification/change transaction, tender listing transaction, or reprint tender receipt transaction occurs.

Note: No record is written if a no sale price verify procedure is performed at the pumps.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.

Field Name	Type	Length	Description
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'12').
Operator	PD	v5	Operator number.
INDICAT0	ASCII	1	**1** Training session. **2** Terminal transfer transaction. **3** Terminal monitor transaction. **4** Price change transaction. **5** Tender listing transaction. **6** Balance inquiry. **7** Reprint tender receipt. **8** Reprint previous sales transaction receipt.
TRANSDATA	ASCII	v6	Varies, depending on the value of INDICAT0: **1** Number of training transactions. **2** Terminal number of transferred terminal. **3** Terminal number of monitored terminal. **4** Batch identifier for price changes. **5** Not used. **6** Not used. **7** Not used.
USREXIT	ASCII	v200	User field reserved for user extensions.

Field Name	Type	Length	Description
OVERRIDE	PD	v4	Override number
REASON	PD	v1	Override reason **X'00'** No override entered **X'61'** Keyed account number **X'70'** Price change while offline **X'76'** Reprint tender receipt
Index	PD	v1	The index in Personalization of the option that contains this manager override ID.
Initials	ASCII	v3	The initials defined in Personalization for this manager override ID.

Cashier Loan (0x13)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written for every cashier loan transaction.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'13').
Operator	PD	v5	Operator number.
INDICAT0	ASCII	1	**1** Current period, operator **2** Current period, terminal
TENDAMT	ASCII	v8	Represents the 8 tender types.
CASH	ASCII	v8	Amount of cash loaned.
CHECKS	ASCII	v8	Amount of checks loaned.
FOODSTAMP	ASCII	v8	Amount of food stamps loaned.
MISC1	ASCII	v8	Amount of miscellaneous tender 1 loaned.
MISC2	ASCII	v8	Amount of miscellaneous tender 2 loaned.
MISC3	ASCII	v8	Amount of miscellaneous tender 3 loaned.

Field Name	Type	Length	Description
MFRCPN	ASCII	v8	Amount of manufacturer coupons loaned.
STRCPN	ASCII	v8	Amount of store coupons loaned.
OVERRIDE	PD	v4	Override number.
REASON	PD	v1	Override reason **X'00'** No override entered. **X'55'** Exceed cashier loan limit. The following reason codes are reserved for customers and Toshiba Global Commerce Solutions Business Partners: **90-96** Toshiba Global Commerce Solutions Business Partners **97-99** Customers
USREXIT	ASCII	v200	User field reserved for user extensions.
Index	PD	v1	The index in Personalization of the option that contains this manager override ID.
Initials	ASCII	v3	The initials defined in Personalization for this manager override ID.

Cashier Pickup (0x14)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written for every cashier pickup transaction.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'14').
Operator	PD	v5	Operator number.
INDICAT0	ASCII	1	**1** Current period, operator **2** Current period, terminal
TENDAMT	ASCII	v8	Represents the 8 tender types.

Field Name	Type	Length	Description
CASH	ASCII	v8	Amount of cash picked up.
CHECKS	ASCII	v8	Amount of checks picked up.
FOODSTAMP	ASCII	v8	Amount of food stamps picked up.
MISC1	ASCII	v8	Amount of miscellaneous tender 1.
MISC2	ASCII	v8	Amount of miscellaneous tender 2.
MISC3	ASCII	v8	Amount of miscellaneous tender 3.
MFRCPN	ASCII	v8	Amount of manufacturer coupons.
STRCPN	ASCII	v8	Amount of store coupons picked up.
OVERRIDE	PD	v4	Override number.
REASON	PD	v1	Override reason: **X'00'** No override entered. **X'56'** Exceed cashier pickup limit. The following reason codes are reserved for customers and Toshiba Global Commerce Solutions Business Partners: **90-96** Toshiba Global Commerce Solutions Business Partners **97-99** Customers
USREXIT	ASCII	v200	User field reserved for user extensions.
Index	PD	v1	The index in Personalization of the option that contains this manager override ID.
Initials	ASCII	v3	The initials defined in Personalization for this manager override ID.

Store Totals Closed (0x15)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written each time Checkout Support completes the close of a set of checkout totals.

Field Name	Type	Length	Description
TerminalNum	PD	2	Store controller ID.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'15').

Field Name	Type	Length	Description
Operator	PD	v5	Operator number = blank.
INDICAT0	ASCII	1	**1** No accounting or department total **2** Long reporting period closed **3** Short reporting period closed **4** Department totals closed short
INDICAT1	PD	1	(Packed string rep. of 1-byte integer) **Bit 0-2** Reserved **Bit 3** X'10' Tender Listing Log closed **Bit 4** X'08' Terminal Productivity Log closed **Bit 5** X'04' Exception Log closed **Bit 6** X'02' Item Movement Totals closed **Bit 7** X'01' Customer Account Totals closed
USREXIT	ASCII	v200	Reserved for user extension.

Missing, Found, or Duplicate Transaction (0x16)

Data object reference: *ISSAExceptionData* `SAEXCDAT.CPP`>

This record is written by Checkout Support whenever it detects a missing, found, or duplicate transaction in the TLOG.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'16').
Operator	PD	v5	Operator number.

Field Name	Type	Length	Description
INDICAT0	ASCII	1	**1** Missing transaction **2** Found transaction **3** Duplicate transaction **4** Duplicate transaction, processed
USREXIT	ASCII	v200	User field for use with exits.

Critical Hardware Failure (0x17)

Data object reference: *ISTSExceptionData* <`TSEXCDAT.CPP`>

A critical hardware failure exception record is written by the terminal sales application for the following conditions:

- Terminal initializes
- Terminal switches controllers
- 4610 printer paper errors are bypassed

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'17').
Operator	PD	v5	Operator number.
Indicat0	ASCII	1	**0** See Reason code. **2** Terminal initialized **4** Terminal switched controllers.
UserExit	ASCII	v200	User field for use with exits.
Override	PD	V4	Override number
Reason	PD	V1	Overide reason **X'00'** No override entered.

Field Name	Type	Length	Description
			X'65' Bypass printer paper errors The following reason codes are reserved for customers and Toshiba Global Commerce Solutions Business Partners: **90-98** Toshiba Global Commerce Solutions Business Partners **97-99** Customers
Index	PD	V1	The index in Personalization of the option that contains this manager override ID.
Initials	ASCII	V3	The initials defined in Personalization for this manager override ID.

Tender Count Transaction (0x18)

Data object reference: *ISSAExceptionData* <SAEXCDAT.CPP>

This record is written for every tender count transaction.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format YYMMDDHHmm.
Code	PD	1	Type of Exception Log entry (value = X'18').
Operator	PD	v5	Operator number.
INDICAT0	ASCII	1	**1** Current period, operator **2** Current period, terminal
TENDERAMT	ASCII	v8	Represents the 8 tender types.
CASH	ASCII	v8	Amount of cash counted
CHECKS	ASCII	v8	Amount of checks counted
FOODSTAMP	ASCII	v8	Amount of food stamps counted
MISC1	ASCII	v8	Amount of miscellaneous tender 1 counted
MISC2	ASCII	v8	Amount of miscellaneous tender 2 counted
MISC3	ASCII	v8	Amount of miscellaneous tender 3 counted
MFRCPN	ASCII	v8	Amount of manufacturer coupons counted
STRCPN	ASCII	v8	Amount of store coupons counted

Field Name	Type	Length	Description
USREXIT	ASCII	v200	User field for use with exits.

User Data Entered (0x19)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written whenever a cashier enters user data, including preferred customer numbers.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'19').
Operator	PD	v5	Operator number.
USRDATA	ASCII	v?	User-defined data or Loyalty Program Data. If you have enabled the Loyalty program and a preferred customer uses their card more than once in one day in the same store, SurePOS ACE logs the second and subsequent use of the customer number. SurePOS ACE also logs all uses of the rain check ID as defined in Loyalty personalization. This entry is intended to help detect fraudulent use of customer cards. The layout of Loyalty program entries follows:
Customer	ASCII	v18	Customer account number.
Points	ASCII	v9	Points for this transaction.
Coupons	ASCII	v5	Value of automatic coupons for this transaction.
Entry Method	ASCII	v1	If preferred customer, log entry method: • 0=Internal • 1-Keyboard • 2=OCR • 3=Scanner • 4=Wand • 5=MSR • 6=MICR
Reserved	ASCII	var	Reserved.

Cashier Loan from Controller (0x20)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written for every cashier loan from a store controller.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value X'20').
Operator	PD	v5	Store controller operator.
OPERATOR/TERMINAL	PD	v5	Operator/terminal receiving loan.
INDICAT0	ASCII	1	**1** Current period, operator **2** Current period, terminal **3** Previous period, operator **4** Previous period, terminal
TENDERAMT	ASCII	v8	Represents the 8 tender types.
CASH	ASCII	v8	Amount of cash loaned
CHECKS	ASCII	v8	Amount of checks loaned
FOODSTAMP	ASCII	v8	Amount of food stamps loaned
MISC1	ASCII	v8	Amount of miscellaneous tender 1 loaned
MISC2	ASCII	v8	Amount of miscellaneous tender 2 loaned
MISC3	ASCII	v8	Amount of miscellaneous tender 3 loaned
MFRCPN	ASCII	v8	Amount of manufacturer coupons loaned
STRCPN	ASCII	v8	Amount of store coupons loaned
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID
Reserved	ASCII	var	Reserved.

Cashier Pickup from Controller (0x21)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written for every cashier pickup from a store controller.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.

Field Name	Type	Length	Description
Code	PD	1	Type of Exception Log entry (value =X'21').
Operator	PD	v5	Controller operator.
OPERATOR/TERMINAL	PD	v5	Operator/terminal being picked up.
INDICAT0	ASCII	1	**1** Current period, operator **2** Current period, terminal **3** Previous period, operator **4** Previous period, terminal
TENDERAMT	ASCII	v8	Represents the 8 tender types.
CASH	ASCII	v8	Amount of cash picked up
CHECKS	ASCII	v8	Amount of checks picked up
FOODSTAMP	ASCII	v8	Amount of food stamps picked up.
MISC1	ASCII	v8	Amount of miscellaneous tender 1 picked up.
MISC2	ASCII	v8	Amount of miscellaneous tender 2 picked up.
MISC3	ASCII	v8	Amount of miscellaneous tender 3 picked up.
MFRCPN	ASCII	v8	Amount of manufacturer coupons picked up.
STRCPN	ASCII	v8	Amount of store coupons picked up.
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.
Reserved	ASCII	var	Reserved.

Tender Count from Controller (0x22)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written for every tender count from a store controller.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value =X'22').
Operator	PD	v5	Controller operator.
OPERATOR/TERMINALl	PD	v5	Operator/terminal for misc. entry.

Field Name	Type	Length	Description
INDICAT0	ASCII	1	**1** Current period, operator **2** Current period, terminal **3** Previous period, operator **4** Previous period, terminal
TENDERAMT	ASCII	v8	Represents the 8 tender types.
CASH	ASCII	v8	Amount of cash counted.
CHECKS	ASCII	v8	Amount of checks counted.
FOODSTAMP	ASCII	v8	Amount of food stamps counted.
MISC1	ASCII	v8	Amount of miscellaneous tender 1 counted.
MISC2	ASCII	v8	Amount of miscellaneous tender 2 counted.
MISC3	ASCII	v8	Amount of miscellaneous tender 3 counted.
MFRCPN	ASCII	v8	Amount of manufacturer coupons counted.
STRCPN	ASCII	v8	Amount of store coupons counted.
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.
Reserved	ASCII	var	Reserved.

Transfer Tender from Controller (0x23)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written every time tender is transferred to the office.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'23').
Operator	PD	v5	Controller operator.
OPERATOR/TERMINAL	PD	v5	Operator/terminal being transferred.
INDICAT0	ASCII	1	**1** Current period, operator

Field Name	Type	Length	Description
			2 Current period, terminal **3** Previous period, operator **4** Previous period, terminal
TENDERAMT	ASCII	v8	Represents the 8 tender types.
CASH	ASCII	v8	Amount of cash transferred.
CHECKS	ASCII	v8	Amount of checks transferred.
FOODSTAMP	ASCII	v8	Amount of food stamps transferred.
MISC1	ASCII	v8	Amount of miscellaneous tender 1 transferred.
MISC2	ASCII	v8	Amount of miscellaneous tender 2 transferred.
MISC3	ASCII	v8	Amount of miscellaneous tender 3 transferred.
MFRCPN	ASCII	v8	Amount of manufacturer coupons transferred.
STRCPN	ASCII	v8	Amount of store coupons transferred.
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.

Carry Forward Tender from Controller (0x24)

Data object reference: *ISSAExceptionData* <`.CPP`>

This record is written each time that the Carry Forward Office Tender is successfully performed at a store controller.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'24').
Operator	PD	v5	Controller operator.
TENDERAMT	ASCII	v8	Represents the 8 tender types.
CASH	ASCII	v8	Amount of cash carried forward.
CHECKS	ASCII	v8	Amount of checks carried forward.
FOODSTAMP	ASCII	v8	Amount of food stamps carried forward.
MISC1	ASCII	v8	Amount of miscellaneous tender 1 carried forward.
MISC2	ASCII	v8	Amount of miscellaneous tender 2 carried forward.

Field Name	Type	Length	Description
MISC3	ASCII	v8	Amount of miscellaneous tender 3 carried forward.
MFRCPN	ASCII	v8	Amount of manufacturer coupons carried forward.
STRCPN	ASCII	v8	Amount of store coupons carried forward.
Enhanced Operator ID	ASCII	v10	String for LDAP ID.

Miscellaneous Transaction (0x25)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written for every miscellaneous transaction performed at a store controller.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'25').
Operator	PD	v5	Controller operator.
OperatorTerminal	PD	v5	Operator/terminal for misc. entry.
INDICAT0	ASCII	1	**1** Current period, operator **2** Current period, terminal **3** Previous period, operator **4** Previous period, terminal **5** Current period office **6** Previous period office
TENDTYPE	ASCII	2	**1** Cash tender **2** Check tender **3** Food stamp tender

Field Name	Type	Length	Description
			4 Miscellaneous tender 1 **5** Miscellaneous tender 2 **6** Miscellaneous tender 3 **7** Manufacturer coupon tender **8** Store coupon tender
DIRECTION	ASCII	1	Direction of tender transfer. **1** Tender paid out of till. **2** Tender received into till.
AMOUNT	ASCII	v8	Amount of misc. entry.
ACCOUNT1	PD	v2	Account number decremented.
ACCOUNT2	PD	v2	Account number incremented.
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.
Reserved	ASCII	var	Reserved.

Request for Close of Store Totals (0x26)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written every time that a request is made to close store totals. Note that record 15 signifies that the close is complete.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value =X'26').
Operator	PD	v5	Controller operator.
INDICAT0	ASCII	1	Type of change. **00** No accounting or department totals closed

Field Name	Type	Length	Description
			01 Long reporting period closed **02** Short reporting period closed **03** Department totals closed short
INDICAT1	PD	1	(Packed string representation of 1-byte integer) **Bit 0-2** Reserved **Bit 3** X'10' Tender Listing Log closed **Bit 4** X'08' Terminal Productivity Log closed **Bit 5** X'04' Exception Log closed **Bit 6** X'02' Item Movement Totals closed **Bit 7** X'01' Customer Account Totals closed
CLOSENAME	ASCII	v20	Name of close reporting period procedure definition.
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.
Reserved	ASCII	var	Reserved.

Add, Alter, or Delete of an Operator Authorization Record (0x30)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written every time an operator authorization record is added, altered, or deleted by Data Maintenance (DM). The operator authorization data is a string containing all data from the operation authorization record except for the key. Data fields are delimited by colons and all integer fields are expressed as ASCII numerics. The NEWDATA string is null for a record deletion, and the OLDDATA string is null for a record addition.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'30').

Field Name	Type	Length	Description
Operator	PD	v5	Store controller operator.
TYPE	ASCII	1	Type of change **1** Record added **2** Record altered **3** Record deleted **4** Enhanced password changed
RECKEY	PD	5	Record key = operator number.
OLDDATA	ASCII	v150	Old operator authorization data.
NEWDATA	ASCII	v150	New operator authorization data.
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.
Reserved	ASCII	var	Reserved.

Add, Alter, or Delete of an Item Record (0x31)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written every time an item record is added, altered, or deleted by Data Maintenance (DM). The item record data is a string containing all data from the item record except for the key. Data fields are delimited by colons and all integer fields are expressed as ASCII numerics. The NEWDATA string is null for a record deletion and the OLDDATA string is null for a record addition.

Note: Reduced Quantity or Deal Limit is also in the record if the Limited quantity deal (LDQ) flag is selected.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'31').
Operator	PD	v5	Controller operator.
TYPE	ASCII	1	Type of change **1** Record added

Field Name	Type	Length	Description
			2 Record altered **3** Record deleted
RECKEY	PD	7	Record key = item code.
OLDDATA	ASCII	v500	Old item record data.
NEWDATA	ASCII	v500	New item record data.
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.
Reserved	ASCII	var	Reserved.

Add, Alter, or Delete of a Tender Verification Record (0x32)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written every time a tender verification record is added, altered, or deleted by DM. The tender verification data is a string containing all data from the tender verification record except for the key. Data fields are delimited by colons and all integer fields are expressed as ASCII numerics. The NEWDATA string is null for a record deletion and the OLDDATA string is null for a record addition.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'32').
Operator	PD	v5	Controller operator.
TYPE	ASCII	1	Type of change **1** Record added **2** Record altered **3** Record deleted
RECKEY	PD	13	Record key = tender type/account number.
OLDDATA	ASCII	v25	Old tender verification data.
NEWDATA	ASCII	v25	New tender verification data.
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.

Field Name	Type	Length	Description
Reserved	ASCII	var	Reserved.

Add, Alter, or Delete of an Item Movement List Record (0x33)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written every time the Personalization module adds, alters, or deletes an item movement list record.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'33').
Operator	PD	v5	Controller operator.
TYPE	ASCII	1	Type of change **1** Record added **2** Record altered **3** Record deleted
BATCH	PD	1	Batch number of changed list.
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.

Report Executed (0x40)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written each time a report is executed.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'40').
Operator	PD	v5	Controller operator.
REPORT	ASCII	v50	Report name.

Field Name	Type	Length	Description
OPERATOR/TERMINAL	PD	v5	Operator/terminal reported on.
TYPE	ASCII	1	Type of report **1** Displayed **2** Printed **3** Filed
ACCOUNTABILITY	ASCII	1	Type of accountability **1** Terminal **2** Operator **3** All terminals **4** All operators **5** Office **6** All operators/all terminals **7** Summary
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.

Personalization Executed (0x41)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written each time that personalization is executed.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'41').
Operator	PD	v5	Controller operator.

Field Name	Type	Length	Description
ACTIVITY	PD	1	Activity performed **1** Change store options **2** Print store options **3** Add terminal options **4** Change terminal options **5** Print terminal options **6** Delete terminal options **7** Copy terminal options **8** Change tax table **9** Add tax table **10** Print tax table **11** Delete tax table **12** Change sales description **13** Print sales description **14** Change report description **15** Print report description **16** Change feature description **17** Print feature description

Field Name	Type	Length	Description
			18 Add group options **19** Change group options **20** Print group options **21** Delete group options **22** Copy group options
FILEID	ASCII Array	3 x v3	Represents an array of 3 FILEID fields.
SourceTerminal	ASCII	v3	Terminal used for source of options copy.
StartTerminal	ASCII	v3	Start terminal for target of options copy.
EndTerminal	ASCII	v3	End terminal for target of options copy.
RECORD	ASCII	v3	Record number updated (0 = no updates).
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.
Reserved	ASCII	var	Reserved.

DDM Executed (0x43)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CP`>

This record is written each time Delayed Data Maintenance (DDM) processes a maintenance batch.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'43').
Operator	PD	v5	Controller operator.
BATCH	PD	4	Batch number acted on.
ACTION	ASCII	1	Action taken **1** No action

Field Name	Type	Length	Description
			2 Batch executed **3** Batch created **4** Batch erased **5** Batch modified **6** Price required flag set for batch **7** Source data created for batch **8** Source data erased for batch **9** Source data modified for batch
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.

DDM Item Record Batch Editing (0x44)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written each time DDM is used to edit the source data in an item record batch.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'44').
Operator	PD	v5	Controller operator.
BATCH	PD	4	Batch number edited.
TYPE	ASCII	1	Type of change **1** Record added **2** Record altered

Field Name	Type	Length	Description
			3 Record deleted
RECNUM	PD	v2	Relative record number within the batch.
OLDDATA	ASCII	v500	Old item record source data.
NEWDATA	ASCII	v500	New item record source data.
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.
Reserved	ASCII	var	Reserved.

DDM Tender Verification Batch Editing (0x45)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written each time delayed maintenance is used to edit the source data in a tender verification batch.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'45').
Operator	PD	v5	Controller operator.
BATCH	PD	4	Batch number edited.
TYPE	ASCII	1	Type of change **1** Record added **2** Record altered **3** Record deleted
RECNUM	PD	v2	Relative record number within the batch.
OLDDATA	ASCII	v25	Old tender verification source data.
NEWDATA	ASCII	v25	New tender verification source data.
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.
Reserved	ASCII	var	Reserved.

DDM Item Movement Batch Editing (0x46)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written each time delayed maintenance is used to edit the source data in an item movement report batch.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'46').
Operator	PD	v5	Controller operator.
BATCH	PD	4	Batch number edited.
TYPE	ASCII	1	Type of change **1** Record added **2** Record altered **3** Record deleted
RECNUM	PD	v2	Relative record number within the batch.
OLDDATA	ASCII	v300	Old item movement source data.
NEWDATA	ASCII	v300	New item movement source data.
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.
Reserved	ASCII	var	Reserved.

DDM Operator Authorization Batch Editing (0x47)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written each time delayed maintenance is used to edit the source data in an operator authorization batch.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'47').
Operator	PD	v5	Controller operator.

Field Name	Type	Length	Description
BATCH	PD	4	Batch number edited.
TYPE	ASCII	1	Type of change **1** Record added **2** Record altered **3** Record deleted
RECNUM	PD	v2	Relative record number within the batch.
OLDDATA	ASCII	v150	Old operator authorization source data.
NEWDATA	ASCII	v150	New operator authorization source data.
Enhanced Operator ID	ASCII	v10	String Data for LDAP ID.
Reserved	ASCII	var	Reserved.

Build/Reload Terminal Item Record File(0x48)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is written each time that the terminal Item Record File is built or the terminals are marked for reload.

Field Name	Type	Length	Description
TerminalNum	PD	2	Store controller ID.
TransNum	PD	2	Transaction number = 0.
DateTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'48').
Operator	PD	v5	Controller operator.
Event	ASCII	1	Type of event: **1** Build started **2** Build finished **3** Reload finished
BuildType	ASCII	1	Type of build: **R** Production terminal Item Record File

Field Name	Type	Length	Description
			P Pending terminal Item Record file This field is null when the Event field is set to 3.
NumRecords	ASCII	10	Number of records in the terminal Item Record file. This field contains a null for a reload request.
UserData	ASCII	v200	Reserved for user extensions.

Unable to Read TLog Record (0x51)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

Checkout support was unable to read a record in the TLog.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'51').
Operator	PD	v5	Operator number.
SUMMARYLOGNUMBER	ASCII	8	Name of summary log.
RECORDPOINTER	ASCII	v10	Byte offset in summary log to start of record which could not be read.
USREXIT	ASCII	v?	User-defined data.

Free Disk Space (0x52)

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>

This record is logged by the close reporting period process for the file server. It can also be used for other nodes in the system. This record will contain the free space available at the start of the close procedure.

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'52').
Operator	PD	v5	Operator number.
NODEID	ASCII	2	Controller ID containing drives when logged by close reporting period. It will be AC for the file server.

Field Name	Type	Length	Description
CDISKFREE	PD	v6	Number of bytes of free space on C drive.
DDISKFREE	PD	v6	Number of bytes of free space on D drive.

Transfer Processing (0x55)

This record is used when the Loyalty program processes a transferred activity file. When a store has activated the Loyalty Program with home and away stores, any data (such as points and coupons) for non-home store customers must be transferred to their home stores.

Each store must process incoming transfer data from other stores. This is done via transfer files EAM:ssss where *ssss* is the number of the store from which the data came.

The *ISTransferProgressDetails* object contains information about the transfer process which can be included in this exception data. An exception log record is written for every incoming transfer file processed by ACEFBXFR, the Loyalty Program activity transfer program.

Data object reference: *ISSAExceptionData* <`SAEXCDAT.CPP`>/<XFREXCPT.HPP>

Field Name	Type	Length	Description
TerminalNum	ASCII	2	Store Controller ID.
TransNum	PD	2	Transaction number = 0.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'55').
Operator	PD	v5	Operator number.
FileName	ASCII	v8	Name of the transfer file.
RecordsProcessed	ASCII	4	Number of records processed.
RecordsTransferred	ASCII	4	Number of records transferred.
CountOfStoresProcessed	ASCII	2	Number of stores processed.

WIC EBT (0x80)

This record provides an exception log entry for WIC EBT. The record is written whenever the inbound processing application processes a new inbound WIC EBT file.

Field Name	Type	Length	Description
TerminalNum	PD	2	Store controller ID
TransNum	PD	2	Transaction number = 0
DateAndTime	PD	5	Date and time, in the format `YYMMDDhhmm`
Code	PD	1	Type of Exception Log entry (value = X'80')
Operator	PD	v5	Operator number
FileName	ASCII	10	Name of inbound file that was processed

Field Name	Type	Length	Description
ProcRecCount	PD	3	Number of records that were processed
CompletionCode	PD	2	Return code, which indicates success or error

WIC EBT Card (0x81)

This record provides an exception log entry for WIC EBT. The record is written whenever the terminal receives an error response code from the Smart Card reader.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'81').
Operator	PD	v5	Operator number.
Identifier	PD	1	Substring identifier = 0x00 for entry that is not written to the claim file; 0x01 for entry that is written to the claim file
AgencyID	ASCII	2	Two-character identifier for the agency
PANLength	PD	1	Length of the participant account number (PAN), which is also the card account number
PAN	PD	10	Participant account number (PAN), which is also the card account number, left padded with zeros.
ErrorCode	ASCII	4	Error response code from the Smart Card reader
IssuingAgency	ASCII	15	Agency that last updated the Smart Card
ICCSystemData	PD	3	Card error data, if available. The format is `8101xx`, where *xx* is the error code.

Customer Entry (0x82)

A customer entry exception record is written for every customer with the following conditions:

- Customer number was keyed
- New customer ID is created
- The number of times a customer card is used beyond the maximum allowed per day.

Field Name	Type	Leng th	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'82').

Field Name	Type	Length	Description
Operator	PD	v5	Operator number.
Override	PD	V4	Override number.
Reason	PD	V1	Exception reason **19** Invalid customer ID. **67** Customer number was keyed. **68** New customer ID was created. **69** Customer card used > maximum allowed per day. The following reason codes are reserved for customers and Toshiba Global Commerce Solutions Business Partners: **90 through 96** Toshiba Global Commerce Solutions Business Partners **97 through 99** Customers
PreferredCust	ASCII	V18	Preferred customer account number
Index	PD	V1	The index in Personalization of the option that contains this manager override ID.
Initials	ASCII	V3	The initials defined in Personalization for this manager override ID.
UserData	ASCII	V200	Reserved for User Extensions

Sales Transaction (0x83)

A sales transaction exception record is written for every sales transaction with the following conditions:

- Number of suspends is greater than the maximum allowed per terminal
- Override needed for suspend
- Override needed for retrieve

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'83').
Operator	PD	v5	Operator number.

Field Name	Type	Length	Description
Override	PD	V4	Override number.
Reason	PD	V1	Override reason **57** Number of suspends greater than the maximum per terminal. **58** Retrieve required override. **59** Suspend required override. The following reason codes are reserved for customers and Toshiba Global Commerce Solutions Business Partners: **90 through 96** Toshiba Global Commerce Solutions Business Partners **97 through 99** Customers
Index	PD	V1	The index in Personalization of the option that contains this manager override ID.
Initials	ASCII	V3	The initials defined in Personalization for this manager override ID.
UserData	ASCII	V200	Reserved for User Extensions

User Log (0x89)

This record provides an exception log entry for user data.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'89').
Operator	PD	v5	Operator number.
User Data 0 (Originator)	PI	1	Represents the originating company for this user entry. Currently, these originator numbers are reserved: • X'00' or X'00' - Toshiba • X'01' - MGV • X'02' - MSDS • X'03' - Toshiba • X'05' - RHISCOM • X'10' - Customer

Field Name	Type	Length	Description
			• X'20' - NRSC • X'30' - Malloys • X'31' - Radius Solutions • X'40' - Excentus • X'50' - M-Dot • X'60' - EXIPOS • X'70' - CouponsCom • X'90' - ICR Other companies that wish to reserve originator numbers should contact Toshiba Global Commerce Solutions.
User Data 1 (SubType)	PI	1	Represents the type of user entry generated by the originator.
User Data 2	ASCII	Variable	Variable length user data
...			
User Data N	ASCII	Variable	Variable length user data

User Log (0x99)

This record is used when the Loyalty program processes a transferred activity file. Other than that, it is reserved for the user and is not written or processed by SurePOS ACE.

Field Name	Type	Length	Description
TerminalNum	PD	2	Terminal number.
TransNum	PD	2	Transaction number.
DateAndTime	PD	5	Date and time, in the format `YYMMDDHHmm`.
Code	PD	1	Type of Exception Log entry (value = X'99').
Operator	PD	v5	Operator number. This field is NULL for a record of a transferred activity file.
USREXIT	ASCII	v?	User-defined data or Loyalty Program data. If you have enabled the Loyalty Program, this field records the following data for every transferred activity file that the Loyalty program processes:
FileName	ASCII	8	Input file name.
Records	ASCII	v5	Number of records processed.
Updates	ASCII	v5	Number of records transferred.
Status	ASCII	v3	"ERR" if errors logged, otherwise "OK".

EAMFBACT (Preferred Customer Activity File)

The Preferred Customer Activity file contains a single record for each preferred home-store customer. Each record accumulates points and sales activity information that is needed at the terminal for customer transaction points processing and targeted coupon processing.

A copy of the file is maintained on all controllers. Checkout Support updates the file with transaction information. Operators can also update the file through Customer Data Maintenance.

You can save a copy of this file at store closing as eamfbacx.dat for additional backup (or to checkpoint activity data for a host dump) depending on your setting of the *Stage activity file at period close* option in Loyalty -> Activity -> Close Periods personalization.

Logical name	<EAMFBACT>, <EAMFBACX>
Data object reference	*ISSACustomerActivityData* <SACSADAT.CPP/HPP>
Organization	Keyed
Distribution class	Compound per update
File copies	1 per controller
Record length	31-508 bytes
Key length	9 bytes: CustomerID

Table 50. Preferred Customer Activity File

Field Name	Type	Length	Decimal Offset	Description
CustomerId	PD	9	0	Customer identification number (up to 18 digits).
TotalPoints	PD	4	9	Total points accumulated (0-999,999.99) Gross totals including adjustments.
TotalTrans	PD	2	13	Total number of transactions (0-9999).
RedeemedPoints	PD	4	15	Redeemed points.
AutoCouponAmount	PD	3	19	Automatic store coupon amount.
LastDate	PD	3	22	Date of last purchase (`YYMMDD`).
LastPoints	PD	3	25	Number of points from last transaction.
StatusLevel	PD	1	28	Customer status level (0-99).
MessageNum	PD	1	29	Message number (10-99) to be shown at next visit.
OptionFlags	PD	1	30	Flags defining optional processing X'80' Do not give preferred discounts X'40' Do not update non-home-store record X'20' Use alternate ID as alias ID X'10' Transfer data is available X'08' Do not give enrolled customer points X'04' Do not give automatic coupons X'02' Do not collect panel data X'01' Do not personalize trailer message
DiscountGroupId	PD	1	31	Discount group ID for customer.
Multiplier	PD	1	32	Points multiplier (0-99) in tenths 1 = 10% increase, 10 = 100% increase.

Field Name	Type	Length	Decimal Offset	Description
TargetedCouponIds	PD	10	33	Five 2-byte targeted coupon deal numbers (IDs) (0000-9999).
AltCustomerNum	PD	9	43	Alternate customer number (up to 18 digits).
PeriodStartDate	PD	2	52	Period starting date (MMDD).
PeriodPoints	PD	4	54	Points in this period (0-99999999).
PeriodTransCount	PD	2	58	Transactions in this period (0-9999).
PeriodRedeemPoints	PD	4	60	Points redeemed in this period.
LastRedeemDate	PD	3	64	Date of last redemption (YYMMDD).
CumSalesTotal	PD	4	67	Gross+ less Gross- for all transactions.
VarData	PD	v436	71	Variable data, including targeted coupon ID fields, secondary points data, and same card use count.
SecFlags115	Int	2	71+2n	Secondary Points Selection flags for clubs 1-15, where *n* = the number of extra targeted coupons • 0x8000 - Secondary points total 8 is active • 0x4000 - Secondary points total 7 is active • 0x2000 - Secondary points total 6 is active • 0x1000 - Secondary points total 5 is active • 0x0800 - Secondary points total 4 is active • 0x0400 - Secondary points total 3 is active • 0x0200 - Secondary points total 2 is active • 0x0100 - Secondary points total 1 is active • 0x0080 - Reserved - must be 0 • 0x0040 - Secondary points total 15 is active • 0x0020 - Secondary points total 14 is active • 0x0010 - Secondary points total 13 is active • 0x0008 - Secondary points total 12 is active • 0x0004 - Secondary points total 11 is active • 0x0002 - Secondary points total 10 is active • 0x0001 - Secondary points total 9 is active
SecPoints1	PD	4	73+2n	Secondary Points Accumulator 1.
SecPoints2	PD	4	77+2n	Secondary Points Accumulator 2.
SecPoints3	PD	4	81+2n	Secondary Points Accumulator 3.
SecPoints4	PD	4	85+2n	Secondary Points Accumulator 4.
SecPoints5	PD	4	89+2n	Secondary Points Accumulator 5.
SecPoints6	PD	4	93+2n	Secondary Points Accumulator 6.
SecPoints7	PD	4	97+2n	Secondary Points Accumulator 7.
SecPoints8	PD	4	101+2n	Secondary Points Accumulator 8.
SecPoints9	PD	4	105+2n	Secondary Points Accumulator 9.
SecPoints10	PD	4	109+2n	Secondary Points Accumulator 10.
SecPoints11	PD	4	113+2n	Secondary Points Accumulator 11.

Field Name	Type	Length	Decimal Offset	Description
SecPoints12	PD	4	117+2n	Secondary Points Accumulator 12.
SecPoints13	PD	4	121+2n	Secondary Points Accumulator 13.
SecPoints14	PD	4	125+2n	Secondary Points Accumulator 14.
SecPoints15	PD	4	129+2n	Secondary Points Accumulator 15.
If the number of secondary clubs is 16-30 or more, then SecFlags1630 and SecPoints16 through SecPoints30 are located here as shown.				
SecFlags1630	Int	2	133+2n	Secondary Points Selection flags for clubs 16-30, where *n* = the number of extra targeted coupons • 0x8000 - Secondary points total 23 is active • 0x4000 - Secondary points total 22 is active • 0x2000 - Secondary points total 21 is active • 0x1000 - Secondary points total 20 is active • 0x0800 - Secondary points total 19 is active • 0x0400 - Secondary points total 18 is active • 0x0200 - Secondary points total 17 is active • 0x0100 - Secondary points total 16 is active • 0x0080 - Reserved - must be 0 • 0x0040 - Secondary points total 30 is active • 0x0020 - Secondary points total 29 is active • 0x0010 - Secondary points total 28 is active • 0x0008 - Secondary points total 27 is active • 0x0004 - Secondary points total 26 is active • 0x0002 - Secondary points total 25 is active • 0x0001 - Secondary points total 24 is active
SecPoints16	PD	4	135+2n	Secondary Points Accumulator 16.
SecPoints17	PD	4	139+2n	Secondary Points Accumulator 17.
SecPoints18	PD	4	143+2n	Secondary Points Accumulator 18.
SecPoints19	PD	4	147+2n	Secondary Points Accumulator 19.
SecPoints20	PD	4	151+2n	Secondary Points Accumulator 20.
SecPoints21	PD	4	155+2n	Secondary Points Accumulator 21.
SecPoints22	PD	4	159+2n	Secondary Points Accumulator 22.
SecPoints23	PD	4	163+2n	Secondary Points Accumulator 23.
SecPoints24	PD	4	167+2n	Secondary Points Accumulator 24.
SecPoints25	PD	4	171+2n	Secondary Points Accumulator 25.
SecPoints26	PD	4	175+2n	Secondary Points Accumulator 26.
SecPoints27	PD	4	179+2n	Secondary Points Accumulator 27.
SecPoints28	PD	4	183+2n	Secondary Points Accumulator 28.
SecPoints29	PD	4	187+2n	Secondary Points Accumulator 29.
SecPoints30	PD	4	191+2n	Secondary Points Accumulator 30.

Field Name	Type	Length	Decimal Offset	Description
If the number of secondary clubs is 31-45 or more, then SecFlags3145 and SecPoints31 through SecPoints45 are located here.				
If the number of secondary clubs is 46-60 or more, then SecFlags4660 and SecPoints46 through SecPoints60 are located here.				
If the number of secondary clubs is 61-75 or more, then SecFlags6175 and SecPoints61 through SecPoints75 are located here.				
If the number of secondary clubs is 76-90, then SecFlags7690 and SecPoints76 through SecPoints90 are located here.				
OtherFlags	Byte	1	71 + VarData -2	Variable Data, flags related to customer operation. (Only stored if space available before card use count.) **X'02'** Suppress paper sales receipt **X'01'** Generate digital receipt
CardUseCount	ASCII	1	71+VarData-1	Variable data, same card use count. (Only stored if space available before last transaction number.)
LastTransNum	ASCII	1	71+VarData	Variable data, number of last transaction that updated this record.
Customer	ASCII	30	Optional	Customer name.

Loyalty -> Activity -> File Format personalization lets you control these interrelated file characteristics:

- Size of the record from 31-508 bytes
- What fields in the record are used at offsets greater than 31
- How many targeted coupons the record can identify
- Whether to store the daily card use count
- Whether to store the transaction number of the last update
- Whether to store the customer name in the record
- How many bytes to use to store the customer name
- The format of the customer name
- How many secondary points clubs will be used in the store

File characteristics are either directly specified by options or indirectly specified by the various possible combinations of settings of these options:

- *Size of activity file record* from 31-508 bytes
- *Number of secondary points clubs* from 1-90
- *Number of bytes before customer name* from 31-508 bytes, which determines what fields are available for use, whether to store the customer name, whether to store the transaction number of the last update, and how many bytes to use to store the customer name
- *Maximum number of targeted coupons* from 5-223, which determines how much of the variable data area is occupied by targeted coupon identifiers
- *Format of customer name,* which is either none, first name, surname, or full name

Each file characteristic is described below. Following the descriptions are examples that illustrate how certain settings interact.

Although the record is a fixed-length record, you determine the record length by your setting for the *Size of activity file record* option. Aside from general storage issues, there are only a few considerations for determining the size of the record:

1. Whether you need more than 5 targeted coupon identifiers
2. Whether you want to use the record to store this data:
   a. Customer discount group ID
   b. Points multiplier
   c. Targeted coupons
   d. Alternate customer number
   e. Starting date of the current period
   f. Current period points earned, redeemed, and transferred
   g. Last redemption date
   h. Total net sales amount from all purchases
   i. Daily card use count
   j. Points for secondary clubs
   k. Transaction number that last updated this file
3. Whether you want customer first or last names, full names, or no names
4. A record size that efficiently uses storage sectors

First determine whether you need more than 5 targeted coupon identifiers. If you do, you must store identifiers 6-91 in the variable data area at offset 71. If you are using this area, your records will have all the fields mentioned as the second criterion above (because they are all at offsets less than 71), and you do not have to consider them in your decision. Each identifier requires two bytes at offset 71 so you must determine how many identifiers you require.

If you do not need more than 5 targeted coupon identifiers, consider whether you need the fields listed in the second criterion. Consider them in reverse order. For example, if you need the transaction number of the last transaction that updated the file, you get it only by providing space in the record for all the fields before it. So, you need not consider whether you require a field at a lesser offset than a field you do require.

If you do not want a customer name in the record, you identify its offset as the end of the record. For example, if you want the last redemption date but not the total net sales amount and not the transaction number, you select 67 as both the size of the record and the number of bytes before the customer name. You must also select customer name format 0, which is no name.

If you need the customer name, you must decide whether you need the first or last name only. This helps determine the field size, which might range from 8-20 bytes for 8-20 characters. If you need the full name, you must allocate more space, perhaps as much as 30 bytes for 30 characters, which is the maximum allowed for the field.

Some efficient file sizes at reasonable field boundaries are: 31, 42, 72, 84, 101, 127, 169, and 254.

Table 51. Record Size, Sector Blocking, and 1MB Record Count

Length	Records/Sector	Records/1MB
31	16	32000 nominal / 25600 at 80% packing
42	12	24000 / 19200
72	7	14000 / 11200
84	6	12000 / 9600
101	5	10000 / 8000
127	4	8000 / 6400
169	3	6000 / 4800

Length	Records/Sector	Records/1MB
254	2	4000 / 3200

Sometimes you must iterate planning parameter values because of their interactions, before arriving at the exact personalization settings. For example, suppose you decide that the record must have room for 10 targeted coupon identifiers, have an extra byte in variable data for the transaction number, and have a customer last name. The targeted coupons field at offset 33 stores 5 identifiers. The other 5 must be stored in variable data at offset 71. Therefore, the extra byte for the transaction number is at offset 81. This makes the customer name start at offset 82. An efficient file size large enough to store enough bytes of the customer name is 101. The 101-byte record, however, allows 19 bytes for the customer name, which is more than you need (assuming you need 15 bytes). You can now iterate your planning by assuming a 101-byte record and moving the customer name to offset 86, leaving 15 bytes for the customer name and giving you 4 more bytes for storing 2 more targeted coupon identifiers. To define the record, you specify the *Size of activity file record* option as 101, the *Format of customer name* option as 2, the *Number of bytes before customer name* option as 86, and the *Maximum number of targeted coupons* option as 12.

Another example of a 101-byte record requires no extra targeted coupon identifiers and no transaction number but requires the maximum 30-byte customer name field at offset 71. To define the record, you specify the *Size of activity file record* option as 101, the *Format of customer name* option as 3, the *Number of bytes before customer name* option as 71, and the *Maximum number of targeted coupons* option as 5.

The minimum record size you can specify is 31 bytes. The 31-byte record includes no targeted coupon identifiers, no fields at offsets greater than 30, and no customer name. To define the record, you specify the *Size of activity file record* option as 31, the *Format of customer name* option as 0, the *Number of bytes before customer name* option as 31, and the *Maximum number of targeted coupons* option as 5.

Note: If you specify the *Size of activity file record* option as 31, there is no room for targeted coupons or the customer name and SurePOS ACE ignores values you specify for the *Maximum number of targeted coupons* option, the *Format of customer name* option, and the *Number of bytes before customer name* option.

You can store customer names anywhere in the record beginning at offset 31. For example, assume you specify the *Size of activity file record* option as 84, the *Format of customer name* option as 2, the *Number of bytes before customer name* option as 67, and the *Maximum number of targeted coupons* option as 5. All fields prior to SALES TOTAL (offset 67) are defined as shown in Table 50, a 17-character name field is defined, and the record is 84 bytes long.

Once you have selected the length of the customer name field by selecting the *Size of activity file record* option and the *Number of bytes before customer name* option, the field size applies to all records in the file. You can change the format of the customer name but you cannot change the length of the customer name field without recreating the file.

The *Maximum number of targeted coupons* option specifies how much of the VarData field is used for additional targeted coupon identifiers. If you specify the *Size of activity file record* option as 254 and the *Maximum number of targeted coupons* option as 96, there are 91 identifiers in the VarData field and there is no room for a customer name. You must also specify the *Number of bytes before customer name* option as 254. With a 254-byte record and a full 30-character customer name descriptor, 153 bytes of variable data are available, which can hold up to 76 additional targeted coupons identifiers. Added to the 5 stored at offset 33, this gives a total of 81 targeted coupons.

The LastTransNum field contains the low byte of the transaction number of the last transaction used to update this record. It distinguishes between two transactions for the same customer of the same amount in the same day. This ensures that totals are properly updated instead of being discarded as duplicates.

The LastTransNum field exists only if the activity record is large enough to contain variable data and if some of the variable data is not reserved for targeted coupons. When the field is present, it always occupies the last byte of the variable data area.

Secondary points data is stored in the VarData area after any targeted coupon data.

If there is only one byte available in the VarData area, it is used to store the last transaction number. If more than one byte is available, the space is used first for secondary points data. If there are still more bytes available, the card use count is stored in the byte before the last transaction number and the other flags field is stored in the byte before the card use count field.

Depending on your setting for the *Reusable Targeted Coupons Velocity Range* option in Options -> Coupon -> Preferred personalization, a targeted coupon is reusable. Its identifier stays in a customer's record until you remove it. Otherwise, SurePOS ACE removes an identifier from a customer's activity record after it awards the targeted coupon to the customer.

EAMFBAPP (Preferred Customer Previous Period Activity File)

The previous customer activity file is a keyed file that contains a single record for each preferred customer who is in the customer activity file when the points activity period is closed. This file and its contents are created by the period close procedure that resets points totals periodically. If you do not close the Preferred Customer Activity file, this file does not exist. When used, there is also a backup copy of this file. Close options are controlled by Loyalty -> Activity personalization (in Close Periods and Auto Process).

Each time the activity period is closed, prior period information for each customer is saved in this file. Neither the file nor any of its records are ever deleted, but rather, active records are overlaid at each successive close.

The file can contain up to 12 monthly records for each customer by keeping all records for the most recent year and overlaying only records with the same period starting month depending on your setting for the *Keep monthly previous points activity* option in Loyalty -> Activity -> Close Periods personalization.

Logical name	< EAMFBAPP>, < EAMFBACX>
Data object reference	*ISPreviousCustomerActivityData* <PRVCADAT.CPP/HPP>
Organization	Keyed
Distribution class	Mirrored per update
File copies	1 per controller
Record length	36 bytes
Key length	One of two possible values depending on personalization option: • 9 bytes: CustomerID for one record per customer • 10 bytes: CustomerID+MM from PeriodStartDate for monthly records

Table 52. Customer Previous Activity File Record Format

Field Name	Type	Length	Decimal Offset	Description
CustomerId	PD	9	0	Customer ID number (up to 18 digits).
PeriodStartDate	PD	2	9	Date of start of period (MMDD).
PeriodEndDate	PD	2	11	Date of end of period (MMDD).
LastPurchaseDate	PD	3	13	Date of last purchase (YYMMDD).

Field Name	Type	Length	Decimal Offset	Description
LastPoints	PD	3	16	Number of points from last transaction.
PeriodPoints	PD	4	19	Total points accumulated during period (gross total including adjustments).
PeriodTransCount	PD	2	23	Total number of transactions in period.
PeriodRedeemPoints	PD	4	25	Total points redeemed during period.
CarryoverPoints	PD	4	29	Points carried into the new period.
PriorDiscountGroupID	PD	1	33	Discount group number for prior period.
Flags	ASCII	1	34	Flags defining totals: X'01' Carry over amount is negative
Reserved	PD	1	35	Reserved for future use.

EAMFBAU* (Preferred Customer Audit Log)

The Preferred Customer Audit Log is used to log points redemption and points adjustment events for security auditing purposes. Two copies of the log are maintained. An old period copy of the audit file is maintained that consists of all the audit data appended since the last time that this old period file was dumped and purged. Current period audit data is appended to the old period file at the close of every reporting period. The current period file cannot be retrieved by the host because it must always be available for updates. The accumulation of old period audit log data is optional; the old period Audit Log need not exist.

The *Append audit data to panel file* Customer Loyalty personalization option causes the current period audit log data to be reformatted and appended to the panel file, eampanel.dat, at the close of a reporting period.

Logical name	<EAMFBAUD>, <EAMFBAUO>
Data object reference	*ISSACustomerAuditData* <SACADDAT.CPP> / <SACADLYT.HPP>
Organization	Sequential
Distribution class	Mirrored per update
File copies	Current, Old
Record length	40 bytes

Redemption Adjustment Record Format

Field Name	Type	Length	Decimal Offset	Description
Header	ASCII	1	0	Double quote.
Account	PD	9	1	Customer account number.
TranType	PD	1	10	"R" = redemption, "A" = adjustment.
DateTime	PD	5	11	Date and time of transaction, in the format `YYMMDDHHmm`.
Terminal	PD	2	16	Terminal where transaction took place.

Field Name	Type	Length	Decimal Offset	Description
Transaction	PD	2	18	Transaction number for this transaction.
Operator	PD	5	20	Operator/Employee who performed transaction.
PointsP	PD	4	25	Points adjusted/redeemed plus.
PointsM	PD	4	29	Points adjusted/redeemed minus.
SerialNum	PD	3	33	Form serial number or store number for rain check.
ClubNum	PD	1	36	Points Club Number. 0 = primary, >0 = secondary.
Trailer	ASCII	3	37	Double quote, CRLF.

Rain Check Record Format

Field Name	Type	Length	Decimal Offset	Description
Header	ASCII	1	0	Double quote.
Account	PD	9	1	Customer account number.
TranType	PD	1	10	"X" = rain check.
DateAndTime	PD	5	11	Date and time of original transaction.
Terminal	PD	2	16	Terminal where rain check transaction took place.
Transaction	PD	2	18	Transaction number for rain check transaction.
Operator	PD	5	20	Operator/employee who performed rain check transaction.
PointsP	PD	4	25	Points adjusted/redeemed plus.
Unused1	PD	4	29	Unused.
Unused2	PD	1	33	Unused.
StoreNum	PD	2	34	Store where rain check transaction took place.
ClubNum	PD	1	36	Points Club Number. 0 = primary, >0 = secondary.
Trailer	ASCII	3	37	Double quote, CRLF.

In the file data structure, this record is surrounded by quotes and followed by a carriage return/ line feed to yield a net usage of 40 bytes per record.

A rain check transaction performed from the controller enrollment procedure logs both an adjustment record and a rain check record. The adjustment record describes the situation at the time of the rain check adjustment. The rain check record describes the situation at the time the sales transaction is adjusted.

EAMFBCHG (Preferred Customer Enrollment Change Log)

The Preferred Customer Enrollment Change Log file is used to log changes to the enrollment file. This file is optional and is intended to aid the maintenance of a host copy of the enrollment data. Two copies of the log are maintained for backup.

These 256-byte records contain the 254 bytes of enrollment data from a Preferred Customer Enrollment File record followed by a carriage return/line feed (X'0D0A'). The 254 bytes of enrollment data is formatted exactly as defined in the Preferred Customer Enrollment file. (See "EAMFBCUS (Preferred Customer Enrollment File)" on page 217.) The carriage return/line feed delimits each record.

Records are written to this file when the in-store enrollment procedure is used to alter enrollment data. The file can contain multiple records for a customer if that customer's data was altered multiple times. In that case, the last record in the file represents the most recent data for the customer.

The records are written as the changes occur. The RSTAT field in each record shows the last enrollment function to act on this record. An RSTAT value of three indicates that the record has been deleted.

Logical name	<EAMFBCHG>
Data object reference	• Storage: *ISSACustomerEnrollmentLogStorage* `<SACELSTR.CPP>` • Data Layout: *ISSACustomerEnrollmentData* `<SACSEDAT.CPP>`<SACSELYT.HPP>
Organization	Sequential
Distribution class	Mirrored per Update
File copies	2
Record length	256 bytes

Field Name	Type	Length	Decimal Offset	Description
CustomerId	PD	9	0	Customer ID number left-padded with zeroes to length 18, then packed.
RecordStatus	ASCII	1	9	Record status "1"-"7" = last enrollment function to update this record. For example, record status "3" = record has been deleted.
LastName	ASCII	17	10	Customer last name.
FirstName	ASCII	13	27	Customer first name.
AddressLine1	ASCII	25	40	Apt., PO box, route number, etc.
AddressLine2	ASCII	25	65	Street address.
City	ASCII	20	90	City name.
State	ASCII	2	110	State abbreviation.
Zip	PD	5	112	ZIP code and ZIP code extension.

Field Name	Type	Length	Decimal Offset	Description
HomePhoneAreaCode	PD	2	117	Area code for home phone number.
HomePhone	PD	4	119	Home phone number.
BusPhoneAreaCode	PD	2	123	Area code for business phone.
BusPhone	PD	4	125	Business phone number.
DriverLicenseNum	ASCII	20	129	Driver's license number.
HomeStoreNum	PD	3	149	Home store number.
CustomerDemo	PD	3	152	Customer demographic field.
Reserved	PD	1	155	Reserved.
SSNumberExt	PD	2	156	Social Security number (extension).
SSNumber	PD	5	158	Social Security number.
Gender	ASCII	1	163	Gender (M or F).
Income	ASCII	1	164	Family annual income level indicator.
BirthDate	PD	3	165	Birth date (YYMMDD).
EnrollDate	PD	3	168	Enrollment date (YYMMDD).
FamilySize	PD	1	171	Family size.
ChildAge1	PD	1	72	Age of Child-1.
ChildAge2	PD	1	73	Age of Child-2.
ChildAge3	PD	1	74	Age of Child-3.
ChildAge4	PD	1	75	Age of Child-4.
ChildAge5	PD	1	76	Age of Child-5.
ChildAge6	PD	1	77	Age of Child-6.
ChildAge7	PD	1	78	Age of Child-7.
TotalAdjustCount	PD	2	179	Accumulated total number of adjustments.
TotalRedeemCount	PD	2	181	Accumulated total number of redemptions.
TotalPointsAdjPlus	PD	4	183	Total points adjusted plus.
TotalPointsAdjMinus	PD	4	187	Total points adjusted minus.
LastAdjustFormNum	PD	3	191	Last adjustment form number.
LastRedeemFormNum	PD	3	194	Last redemption form number.
LastAdjustDate	PD	3	197	Last adjustment date (YYMMDD).
LastRedeemDate	PD	3	200	Last redemption date (YYMMDD).
ChangeDate	PD	3	203	Last change date for record (YYMMDD).
LastRainCheckDate	ASCII	3	206	Last rain check date (YYMMDD).
CustomerAuthFlag	ASCII	1	209	Customer authorization (Y=Yes, N=No).
AliasFlag	ASCII	1	210	Referenced = Alias (Y=Yes, N=No).

Field Name	Type	Length	Decimal Offset	Description
ReferenceAccount	PD	9	211	Referenced account to get all points.
LastRainCheckAmount	PD	3	220	Last rain check amount.
StateExtension	ASCII	1	223	STATE (extension).
ZipAlphanumeric	ASCII	9	224	ZIP code and ZIP code extension (both alphanumeric).
DriverLicenseNumExt	ASCII	5	233	Driver's license number (extension).
VarData	ASCII	16	238	Reserved for variable data.

EAMFBCUS (Preferred Customer Enrollment File)

The Preferred Customer Enrollment file is not required in the store and is never accessed during the checkout transaction.

The Preferred Customer Enrollment file contains a single record for each preferred customer for their home store. This record contains the demographic data entered for a customer at enrollment. The enrollment program also tracks adjustments and redemptions in this file. The enrollment file gives an enterprise maximum flexibility to decide what customer information to obtain and where to enter and maintain this information.

The Preferred Customer Enrollment file is accessed and updated only by the enrollment program. It is never accessed during the checkout transaction. The file is optional, depending on whether the in-store enrollment program is used and whether this file is accessed by the program.

Two copies of the file are maintained for backup.

Logical name	<EAMFBCUS>
Data object reference	• Storage: *ISSACustomerStorage* <SACSTSTR.CPP> • Data Layout: *ISSACustomerEnrollmentData* `<SACSEDAT.CPP>` <SACSELYT.HPP>
Organization	Keyed
Distribution class	Mirrored per Update
File copies	2
Record length	254 bytes
Key length	9 bytes: Customer ID

Field Name	Type	Length	Decimal Offset	Description
CustomerId	PD	9	0	Customer ID number left-padded with zeroes to length 18, then packed.
RecordStatus	ASCII	1	9	Record status "1"-"7" = last enrollment function to update this record.
LastName	ASCII	17	10	Customer last name.
FirstName	ASCII	13	27	Customer first name.
AddressLine1	ASCII	25	40	Apt., PO Box, Rt number, etc.

Field Name	Type	Length	Decimal Offset	Description
AddressLine2	ASCII	25	65	Street address.
City	ASCII	20	90	City name
State	ASCII	2	110	State abbreviation
Zip	PD	5	112	ZIP code and ZIP code extension.
HomePhoneAreaCode	PD	2	117	Area code for home phone.
HomePhone	PD	4	119	Home phone number.
BusPhoneAreaCode	PD	2	123	Area code for business phone.
BusPhone	PD	4	125	Business phone number.
DriverLicenseNum	ASCII	20	129	Driver's license number.
HomeStoreNum	PD	3	149	Home store number
CustomerDemo	PD	3	152	Customer demographic field.
Reserved	PD	1	155	Reserved.
SSNumberExt	PD	2	156	Social Security number (extension).
SSNumber	PD	5	158	Social Security number.
Gender	ASCII	1	163	Gender (M or F).
Income	ASCII	1	164	Family annual income level indicator.
BirthDate	PD	3	165	Birth date (YYMMDD).
EnrollDate	PD	3	168	Enrollment date (YYMMDD).
FamilySize	PD	1	171	Family size.
ChildAge1	PD	1	72	Age of child 1.
ChildAge2	PD	1	73	Age of child 2.
ChildAge3	PD	1	74	Age of child 3.
ChildAge4	PD	1	75	Age of child 4.
ChildAge5	PD	1	76	Age of child 5.
ChildAge6	PD	1	77	Age of child 6.
ChildAge7	PD	1	78	Age of child 7.
TotalAdjustCount	PD	2	179	Accumulated total number of adjustments.
TotalRedeemCount	PD	2	181	Accumulated total number of redemptions.
TotalPointsAdjPlus	PD	4	183	Total points adjusted plus.
TotalPointsAdjMinus	PD	4	187	Total points adjusted minus.
LastAdjustFormNum	PD	3	191	Last adjustment form number.
LastRedeemFormNum	PD	3	194	Last redemption form number.
LastAdjustDate	PD	3	197	Last adjustment date (YYMMDD)
LastRedeemDate	PD	3	200	Last redemption date (YYMMDD).

Field Name	Type	Length	Decimal Offset	Description
ChangeDate	PD	3	203	Last change date for record (YYMMDD).
LastRainCheckDate	ASCII	3	206	Last rain check date (YYMMDD).
CustomerAuthFlag	ASCII	1	209	Customer authorization (Y=Yes, N=No).
AliasFlag	ASCII	1	210	Referenced = Alias (Y=Yes, N=No).
ReferenceAccount	PD	9	211	Referenced account to get all points.
LastRainCheckAmount	PD	3	220	Last rain check amount.
StateExtension	ASCII	1	223	STATE (extension).
ZipAlphanumeric	ASCII	9	224	ZIP code and ZIP code extension (both alphanumeric).
DriverLicenseNumExt	ASCII	5	233	DRIVER (extension).
VarData	ASCII	16	238	Reserved for variable data.

The records in the file intentionally have unused space in case you want to use the file to save additional demographic data. With two records per sector, only about 3200 records fit in a 1MB (MB = 1048576 bytes) file at an 80% packing factor.

EAMFBFLT (Preferred Customer Panel Filter File)

Logical name	<EAMFBFLT>, <EAMFBFL1>, <EAMFBFL2>, <EAMFBFL3>, <EAMFBFL4>, <EAMFBFL5>, <EAMFBFL6>, <EAMFBFL7>, <EAMFBFL8>, <EAMFBFL9>
Data object reference	*ISPanelFilter*, *ISPanelFilterElement*, *ISPanelFilterKey* <ACEPANEL.CPP>
Organization	Sequential
Distribution class	Compound per update
File copies	0 to 10 user-created files
Record length	Variable

Panel Filtering Overview

The Panel Diary file is a reformatted subset of TLog X'00'-X'11' strings that normally contain all data from sales transactions for preferred customers or for all customers. This data includes a nearly complete record of each transaction and is frequently more extensive than necessary. The panel filtering function can cull excess information from the file.

The panel filter can:

- Reduce the size of panel files and enhance their management by deleting nonessential data
- Delete all information for selected items, manufacturers, or departments
- Create different views of panel data that each contain a different subset of the data

The panel file contains about 3 million bytes for each 100,000.00 dollars of sales data. If you decide to process this data initially at a department level rather than at an item level, you can remove all item codes to reduce the size of this data by about 20%. You can achieve large savings by removing other information such as coupon family codes, taxes, and tenders if this information is not needed for preferred customer purchase history analysis. Reducing the contents and the size of the panel file reduces the cost of transmitting, storing, and analyzing its information.

Multiple Filter Files for Multiple Views

Panel data can be valuable to entities outside the enterprise, such as manufacturers, although you might not want to provide them all your information about sales and customers. By filtering data, you can show a manufacturer detailed information on who is buying his products without showing prices at which the products were purchased.

You might want to sell information on what types of customers are buying what types of products, excluding all data on sales from a specialty department or on house brands. By reducing the Panel Diary data to selected classes of items, the filter function reduces the effort and cost involved in selling subsets of purchase history data.

Filtered views of TLog data can highlight different aspects of customer purchase habits. For example, one subset of Panel Diary data might show promotion planning and analysis data. Another set might include information relevant to a marketing research company. You might trade a third subset to suppliers in exchange for additional participation in promotions. The panel filter function is faster and less costly than manual (or semi-automated) methods of compiling different versions of customer data.

There are ten possible panel diary filter files that activate and define filtering for the ten possible output panel files. With the exception of eamfbflt.dat, you must create, initialize, and place each filter file in adx_idt1 as a compound file.

Filtering TLog Strings

User-created panel filter files activate this filter function and provide the selection of information that is to be filtered out of the panel data. You must create Panel Filter files using the Toshiba Global Commerce Solutions-supplied file, eamfbflt.dat, as a template. The presence of a eamfbflx.dat file triggers creating and filtering data for the associated eampanlx.dat output file. Specify the ACEPANEL program in Misc -> User Programs personalization to run it after a store closing against the previous period TLog. ACEPANEL creates all output files based on the presence of filter files.

To understand filtering output data, you must understand the input TLog data. TLog strings represent item sales or tenders. Its substrings represent individual data elements such as item codes or prices. Data types in the TLog are delimited as shown in Table 68.

Delimiters in the panel file are the same as those in the TLog, but packed numeric data is converted to ASCII numeric data and flag data is converted to ASCII representations of hexadecimal values of the data. Filtering substrings nulls out their data and leaves their delimiters. Filtering strings or records removes their data and delimiters. This allows other programs to process panel data successfully whether or not it has been filtered. Filtering matches and removes data from the associated output Panel Diary file. It does not match and add data or merge or summarize data.

Filter data consists of ASCII strings that are identical to panel data strings with the addition of control characters that direct filtering:

```
"01:itemcode:price:dept:XXfamily1:family2:flag1:flag2:flag3:flag4"
```

For example, the filter input string above deletes (XX) the current coupon family code field from the item entry string (type X'01'). Each filter string is a separate record in the filter file and must be delimited from other records by a CRLF. Each record relates to a single input string type and processes only that string type. If the first two characters of a filter record are not 00 through 11, the record is ignored as a comment. If no filter record applies to a given input string, the input string is formatted without filtering.

A filter record without control characters, such as the one below, is a comment. It does not indicate filtering and does not affect data.

```
"05:type:tender:fee:account:status"
```

There can be up to 125 different active filter records. There can be multiple active filter records for a string type if a key field distinguishes a particular range of data to which each filter record applies. The order of filter records is important in this case because only the first filter record, or consecutive group of like filters, applies to a particular input string.

Filter Controls

Upper and lower case are equivalent characters in all filter records. Control characters are: ":", ":XX", ":ZZ", "KK:", and "KEY". All other filter record input is for documentation only.

Colon

The colon denotes a field delimiter in a filter string just as in an actual data string.

You need not specify a field name when delimiting substrings. The following two records are identical in function:

```
"01:itemcode:price:dept:family:XXfamily:XXflag:flag2:flag3:flag4"
```

The record above is the same as:

```
"01::::XX:XX""
```

Both indicate to null the fifth and sixth substrings in an item entry string.

XX

The XX control denotes a field to be filtered. The field is nulled but the delimiting colons remain. If all fields for a string type are discarded, the entire string is discarded.

For example:

```
01:itemcode:XXprice:department:family1:family2:flag1:flag2:flag3:flag4
01::XX::::::
01::XX
```

All three records null the price field in an item entry string.

The first record in the following example performs the same function as the last two. The first record filters the entire record. The other two together delete all fields in a type X'07' record, which filters the record because it also filters the record delimiter.

```
"07:XX"
```

```
"07:XXtaxa:XXtaxb:XXtaxc:XXtaxd:XXtaxe:XXtaxf:XXtaxg:XXtaxh"
"07:XXsalesa:XXsalesb:XXsalesc:XXsalesd:XXsalese:XXsalesf:XXsalesg:XXsalesh"
```

ZZ

The ZZ control denotes strings whose presence filters out the entire transaction.

For example:

```
"10:ZZoverride:reason"
```

The example filters out any transaction that involves a manager override.

KK

The KK control denotes the one key field that is evaluated against data values in the KEY parameter to determine whether to apply XX or ZZ filter controls in the filter record. This control is different than XX or ZZ because it applies to the string to its left. It appears as the last two characters in the delimited substring to which it applies.

If the filter record does not contain XX or ZZ, the string is processed without filtering.

For example:

```
01:itemcode:price:department:family1:family2:flag1:flag2:flag3:flag4
01:XX:XX:XX:XX:XX:XX:XX:XXKK:XXdummy KEY = 10 - 59
```

The first record is a comment that shows what each field in a TLog type X'01' string is. The KK in the second record applies to the Flag3 (Indicat3) field. The KEY indicates the criteria for Flag3 data values, which is a range from 10 to 59. This criteria identifies input methods 0 to 9 (which includes every input method) for entering record types 1 to 5, which are deposits, refunds, deposit returns, miscellaneous transactions receipts, and miscellaneous transaction payouts. Therefore, if the item entry is one of these types, all of its fields are nulled and SurePOS ACE deletes it from the panel diary. If the item entry is another kind (such as a normal item sale or a coupon item), none of its fields are nulled and it is added to the panel diary in its entirety.

Note: If a type one item string is discarded based on input key criteria, any associated item extension (type X'02' string) is also discarded.

As the key field designator must end with a colon, an additional dummy field might be needed in the filter input when a record is deleted based on the contents of its last field as in the last example. In this case, the filter record ends with :XXKK:XXdummy instead of :XXKK. The number of fields in a filter record need not match the number of fields in the actual data string. Up to 16 fields beyond the string type can be defined in any filter record.

Another example of the KK control is:

```
01:itemcode:price:department:family1:family2:flag1:flag2:flag3:flag4
01:XXitemcodeKK:XX:XX:XX:XX:XX:XX KEY=123450000000-123459999999
```

The first record is a comment. The second has KK applying to the item code field, looking for data in the range 1234500000-1234599999. It finds items made by manufacturer 12345, it nulls each field, which filters the record out of the panel diary.

This example drops items from department 123:

```
01:itemcode:price:department:family1:family2:flag1:flag2:flag3:flag4
01:XX:XX:XXdeptKK:XX:XX:XX:XX KEY = 123
```

This example drops redundant customer number data entry strings:

```
11:XXKK:XX KEY = ??P
```

KEY

The KEY control denotes data value criteria for a field designated with `KK:`. Key data can have these forms:

```
KEY = nn            KEY = nn-nnn
KEY NE nn           KEY NE nn-nnn
KEY = xxH           KEY = xxH-xxxH
KEY NE xxH          KEY NE xxH-xxxH
KEY = nn ON         KEY = xxH ON
KEY = nn OFF        KEY = xxH OFF
KEY = xxP           KEY NE xxP
```

Note:

1. The = sign is optional.
2. *nn* is numeric.
3. `xxH` is hexadecimal data.
4. `xxP` is one packed data byte.
5. `ON` means designated flags are on.
6. `OFF` means they are off.
7. - is an operator that indicates a range. A blank between two values also indicates a range.

   Blanks in key data are optional, but either a blank or an "-" should separate the two data fields in a range.
8. `NE` is an operator that means *not equal*.
9. Other operators are `{:;<=>?} = {ACDEF}`.

Comments

Comments can follow valid filter input in a filter record if an exclamation mark (!) separates comments from the data. For this reason, exclamation points are not allowed in filter data.

Filter Processing

ISPanelFilter, *ISPanelFilterElement*, and *ISPanelFilterKey* hold information for filtering TLog strings. They also determine whether to filter a field based on its value. Figure 2 shows the interaction between the objects.

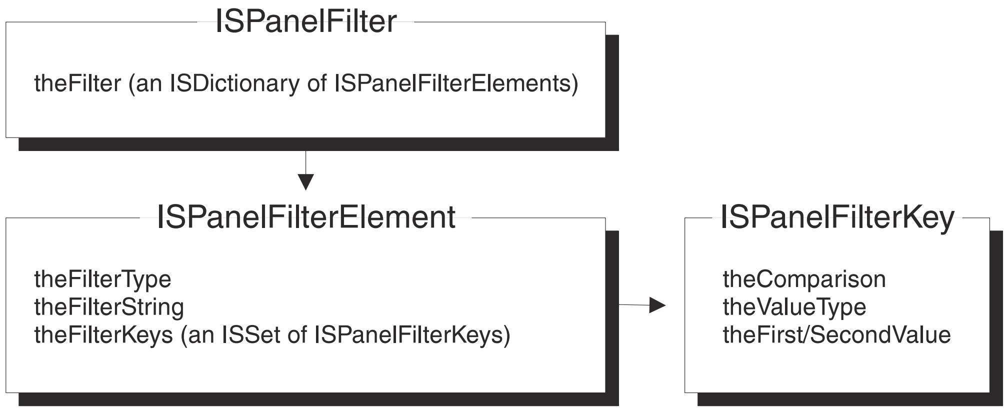


Figure 2. Filter Processing Overview

The key into *theFilter* dictionary is the TLog string type. When you specify filtering for certain fields, *theFilterString* has a value. When you specify KEY values, *theFilterKeys* has entries.

For example, assume a filter file (such as eamfbflt.dat) contains:

```
01:itemcode:price:dept:XX:XX:flag2:flag3:flag4
01:XXKK:XX:XX:XX:XX:XX:XX KEY=123
01:XXKK:XX:XX:XX:XX:XX:XX KEY 200-999
02:XX:XX:XX:XX:XX:qty/wgt:flags
07:XX
```

Filter processing is:

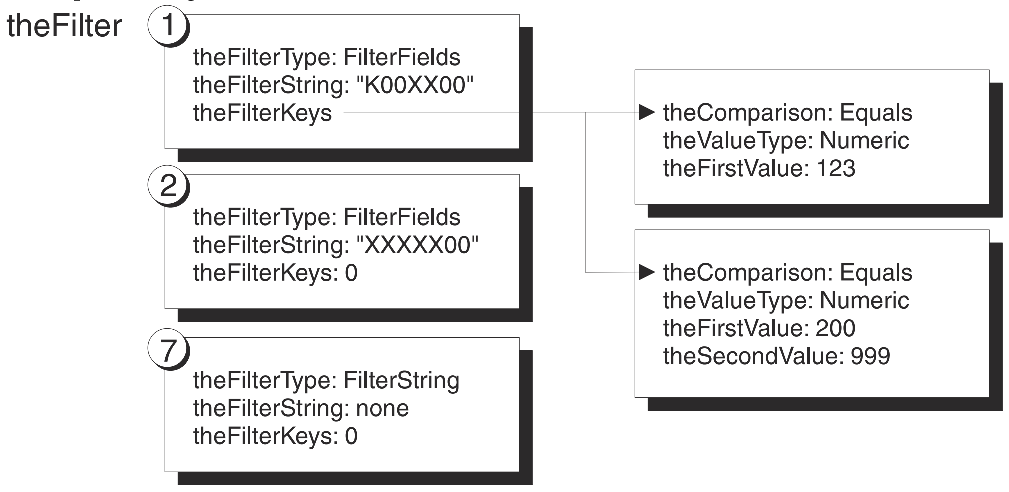


Figure 3. Filter Processing

All type 7 strings are deleted because *theFilterType* specification above is a *FilterString* type instead of a *FilterFields* type. The first five fields from type 2 strings are deleted. Type 1 strings with itemcodes of 123 or 200-999 are deleted. Fields 4 and 5 are deleted from type 1 strings that are not deleted.

Filtering Packed Decimal Data

Key packed decimal data in the TLog is matched against the unpacked value of the data. You can specify it in either ASCII format or hexadecimal format. For example, to match the value

X'0123' in the department number field in the TLog, specify the key as KEY = 123 or KEY = 7BP but do not specify it as KEY = 123P.

Default Filter File

A default filter file, shipped as eamfbflt.dat, reduces panel data by deleting fields that normally are not needed for purchase history analysis. This default file contains the following active records:

```
00:term:trans:time:type:strings:checker:0:total
+:total-:XX:XX:XX:XX:flgs:altprice:voidtrc
Drop performance timings from header

01:XX:XX:XX:XX:XX:XX:XXKK:XX KEY = 10-39
01:XX:XX:XX:XX:XX:XX:XXKK:XX KEY = 90-99
Drop deposits, deposit returns, refunds, and deposit cancels

01:itemcode:price:dept:XX:XX:flag2:flag3:flag4
Drop family codes and item record flags
02:XX:XX:XX:XX:XX:qty/wgt:flags
Drop item record pricing information

07:XX                   ! Drop tax
08:XX                   ! Drop tax refund
09:XX                   ! Drop change given
10:XX                   ! Drop manager&csq;s overrides
11:XX:XXKK:XX KEY = ==P  ! Drop coupon tracking data
11:XXKK:XX    KEY = ??P  ! Drop redundant customer number data
```

EAMFBTCM (Preferred Customer Targeted Coupon Messages File)

The targeted coupon messages file is an optional keyed file that contains a record for each targeted coupon for which the user wishes to provide message text. This message text is optionally printed on the receipt tape of the customers who are eligible for the targeted coupon along with information from the coupon item record. Up to three print lines, each with 38 characters of free formatted text, can be supplied for each targeted coupon to describe the promotion to the eligible customers. All blank print lines will not be printed on the customer receipt. A record with all blank text is the same as no record on file.

This file and its contents are maintained by the Data Maintenance -> Targeted Coupon Messages interface, which is described in the *SurePOS ACE Guide to Operations*. The size of the file is determined by the UPC range defined for targeted coupons, in the *Reusable Targeted Coupons Velocity Code Range* option in Options -> Coupons -> Preferred personalization.

Targeted coupon maintenance provides the ability to add, change, delete, or report on all coupon messages. The only other code that uses this file is the terminal code that prints the text on the receipt tape of customers to whom the coupons are targeted. A copy of this optional input file is kept on each controller so that each terminal can read its messages from a local controller.

Logical name	<EAMFBTCM>
Data object reference	*ISSATargetedCouponMessageData* <SATCMDAT.CPP>
Organization	Keyed
Distribution class	Compound per Update
File copies	1 per controller
Record length	127 bytes

Key length	2 bytes: CouponNumber

Field Name	Type	Length	Decimal Offset	Description
CouponNumber	PD	2	0	Coupon number (linkage to coupon).
CouponMessage	ASCII	114	2	Coupon message text (three 38-character lines).
Reserved	ASCII	11	116	Reserved.

EAMFBXFR (Preferred Customer Transfer File)

This file is optional. It is accessed only if you support multiple stores and need to transfer activity data from one store to another when customers shop in non-home stores. (This is controlled by the *Other home store numbers supported by this store* in Loyalty -> Home Store personalization.)

The Preferred Customer Transfer file is effectively an optional, low-use extension of the customer activity file. It contains a single record for each preferred customer who has transferred points from another store.

Checkout Support updates the file at store closings.

Logical name	<EAMFBXFR>
Organization	Keyed
Data object reference	*ISSACustomerTransferData* <SACTRDAT.CPP/HPP>
Distribution class	Compound per update
File copies	1 per controller
Record length	36 bytes
Key length	9 bytes: CustomerID

Field Name	Type	Length	Decimal Offset	Description
CustomerID	PD	9	0	Customer ID number.
PendingPoints	PD	4	9	Points in transfer (0-999,999.99).
PendingTransCount	PD	2	13	Transactions in transfer.
PendingCouponAmt	PD	3	15	Automatic coupons in transfer (.00-9999.99).
LastPurchaseDate	PD	3	18	Date of last transferred purchase (`YYMMDD`).
LastPoints	PD	3	21	Number of points from last transferred transaction.
TotalPoints	PD	4	24	Total points transferred.
TotalTransCount	PD	2	28	Total transactions transferred.
TotalCouponsCount	PD	3	30	Total automatic coupons transferred.
LastTransNum	PD	3	33	The number of the last transferred transaction.

Fourteen of these records fit in a sector and about 22000 records fit in a 1MB file. You can delete these records to minimize the file size. The maximum file size is the number of records in the Preferred Customer Activity file.

EAMHDPT* (Hourly Department Totals File)

The Hourly Department Totals File is a sequential file that contains totals data for all departments which have activity during a given hour. From 1 to 999 departments can be defined through personalization. This file maintains totals for each hour of each day until the department totals files have been closed. A separate file will exist for the previous period and the current period.

An image version of this file is kept on the alternate file server. This distribution takes place only when the file is closed to minimize the performance impact.

Logical name	<EAMHDPTC>, <EAMHDPTP>
Data object reference	*IntervalDepartmentStatisticsStorage* <INTDSSTR.CPP>
Organization	Sequential
Distribution class	Mirrored per Update
File copies	Current and Previous
Record length	Variable

Header Record

Field Name	Type	Length	Decimal Offset	Description
DeptNum	PD	2	0	Always 9999 for the Header record.
NumDepts	Int	2	2	The number of departments with activity in this hour.
DateTime	Int	4	4	Integer value of the date and time for which the following records apply. Format is YYMMDDHH.
NumTrans	Int	4	8	The number of transactions processed in this hour for all departments.

Data Record

Field Name	Type	Length	Decimal Offset	Description
DeptNum	PD	2	0	This value is always X'9997' for a period record. It is X'9999' for a header record, X'9998' for a trailer record, and is a department number for other (data) records.
Union	Union	6	2	Defines these six bytes as one of two possible fields, either OtherRecord or PeriodRecord.
OtherRecord	Structure	6	2	Defines the DepTrans and DepItems fields. This is the active union definition for a data record, which is one with a department number (in the DeptNum field) *not* equal to 9997, 9998, or 9999.

Field Name	Type	Length	Decimal Offset	Description
DepTrans	Int	2	2	The number of transactions for this hour for this department.
DepItems	Int	4	4	The number of items sold in this hour for this department.
PeriodRecord	Structur e	6	2	Defines the PeriodStart and Reserved fields. This is the active union definition for a period record, which is one with a department number (in the DeptNum field) equal to 9997.
PeriodStart	PD	5	2	Time that the period began, in the format `YYMMDDHHMMSS`.
Reserved	ASCII	1	7	Reserved.
AmtItems	Int	4	8	The sales amount of the items sold in this hour for this department.

Trailer Record

Field Name	Type	Length	Decimal Offset	Description
DeptNum	PD	2	0	Always `9998` for a Trailer record.
Reserved	Int	2	2	Reserved.
Reserved	Int	4	4	Reserved.
Restart	Int	4	8	The restart pointer used for recovery.

EAMIMG* (Check Image File)

The Check Image File is a sequential file that contains check images that have been captured during transactions. The previous file is used for host retrieval.

The Check Image File is not encrypted. Your store must handle security for accessing the controller that contains the file.

Check images can be captured and written to the Check Image File at any time. If a check image is captured near the time that a store close is being performed, the check image might be stored during a different period than the matching transaction in the Transaction Log.

During store close processing, if the Check Image File exists then a message is sent from GIPC to the check image executable to let it know that store close is in progress. This is done during the Close Reporting Period Encountered step of close processing for both daily and weekly closes.

Logical name	<EAMIMGC>, <EAMIMGH>
Organization	Sequential

Distribution class	Mirrored on update for EAMIMGC, mirrored on close for EAMIMGH
File copies	Current and previous
Record length	Tag data record is 508 bytes, check image record is variable

Tag Data Record

Field Name	Type	Length	Decimal Offset	Description
Date	ASCII	6	0	Date on which the check image was captured. The format is `yymmdd`, where `yy` is the two rightmost digits in the year, `mm` is the month, and `dd` is the day.
Time	ASCII	6	6	Time at which the check image was captured. The format is `hhmmss`, where `hh` is the hour using a 24-hour clock, `mm` is the minutes, and `ss` is the seconds.
Terminal number	ASCII	4	12	The terminal at which the check image was captured. Data is right justified and padded with zeros.
Store number	ASCII	4	16	The store number. Data is right justified and padded with zeros.
Record ID	ASCII	3	20	Identifies the level of layout of the tag record. Data is right justified and padded with zeros.
Image size	ASCII	5	23	The number of bytes in the check image. Data is right justified and padded with zeros.
Transaction number	ASCII	4	28	The number of the transaction for which the check image was captured. Data is right justified and padded with zeros.
Image format	ASCII	1	32	The format in which the check image is stored. (0 = TIFF-CCIT compression)
Tender amount	ASCII	8	33	The amount of the tendered check. Data is right justified and padded with zeros.
MICR length	ASCII	2	41	The number of bytes in the MICR data. Data is right justified and padded with zeros.
MICR data	ASCII	50	43	The MICR data, including the routing number, account number, and check number. Data is left justified and padded with spaces on the right.
Image number	ASCII	1	93	Indicates if both the front and the back of the check are being imaged: **0** Only the front is being imaged.

Field Name	Type	Length	Decimal Offset	Description
				1 This image is the front of the check, and both sides are being imaged. **2** This image is the back of the check, and both sides being imaged. **9** This is a results record only, and no image is stored with this record.
Void indicator	ASCII	1	94	Indicates if this tag record is for a check tender that was voided: **0** Normal tender. **1** Voided tender.
Reserved	ASCII	1	95	Reserved.
Results indicator	ASCII	1	96	When image number (above) is set to 9, this value indicates whether the check image records were successfully stored for this tender: **0** Success. **1** Problem encountered.
Reserved	ASCII	261	97	Reserved.
User tag data	ASCII	150	358	Customer-defined data, which is left justified and padded with spaces on the right.

Check Image Record

This record contains the check image in TIFF format.

EAMIMO* (Item Movement Totals File)

The Item Movement Totals file is a keyed file that accumulates movement for item sales. Movement is kept for all item codes flagged for movement in their item record. Item movement is not affected if the REFUND, DEPOSIT, STORE COUPON, or MFR COUPON key is used with an item code that represents a normal item sale. A separate file exits for the current period and for one old period.

Both Item Movement files and the records within them are created and maintained by the applications. An image version of each of the Item Movement files is kept on the alternate file server. This distribution takes place only when the file is closed, to minimize the performance impact.

If you select an archived file as the basis for the Item Movement Report, SurePOS ACE creates a temporary uncompressed copy of the file and stores the copy in the adx_idt4 directory. The naming convention for the temporary file is eamimovy.dat, where the value of *y* is 0-5. The temporary work file is kept until the next time that the report is requested for an archived file. If you start the Reports function and request multiple Item Movement Reports using archived files, a maximum of six temporary files is kept. When you request the seventh report, the oldest temporary file is deleted.

You can archive Item Movement files under any name. SurePOS ACE supports file names with the format eamkyymd.tzn, which is based on this naming convention:

eamk	Always the first four characters of the file name.
yy	The last two digits of the year. For example, 04 represents 2004.
m	The month of the year (1=1, ..., 9=9, A=10, B=11, C=12)
d	The day of the month: (1=1, ..., 9=9, A=10, ..., V=31)
t	Specifies what type of period: w - Long period d - Short period
z	Specifies whether the file is compressed: u - Uncompressed file. c - Compressed file.
n	Number that starts at 0, increments for each file close during a day, and resets to 0 at the beginning of each day.

File Layout for 12-digit Item Code

Logical name	<EAMIMOVE>, <EAMIMOVO>,<EAMIMOLC>, <EAMIMOLP>
Data object reference	*ISSAItemMovementStatisticsStorage* <SAIMSSTR.CPP/HPP>
Organization	Keyed
Distribution class	Mirrored at close
File copies	Current short, Previous short, Current long, Previous long
Record length	22 bytes
Key length	6 bytes: Item code

Store Records

Field Name	Type	Length	Decimal Offset	Description
ItemCode	PD	6	0	Item number = All 9s for the Store record.
DateTime	PD	5	6	Date and time when the period started or the old period ended, in the format `YYMMDDHHmm`.
Reserved	ASCII	11	11	Reserved.

Movement Record

Field Name	Type	Length	Decimal Offset	Description
ItemCode	PD	6	0	Item number.
SalesCount	Int	4	6	Count of the number of items sold.
SalesAmount	Int	4	10	Sum of the sales amount for the item.
Weight	Int	4	14	Sum of the weight or volume of the items sold.
Restart	Int	4	18	This field is used as a control field on a system restart. It is used to determine if this record has been updated by a particular transaction.

File Layout for 14-digit Item Code

Logical name	<EAMIMOVE>, <EAMIMOVO>,<EAMIMOLC>, <EAMIMOLP>
Data object reference	*ISSAItemMovementStatisticsStorage* <SAIMSSTR.CPP/HPP>
Organization	Keyed
Distribution class	Mirrored at close
File copies	Current short, Previous short, Current long, Previous long
Record length	23 bytes
Key length	7 bytes: Item code

Store Record

Field Name	Type	Length	Decimal Offset	Description
ItemCode	PD	7	0	Item number = All 9s for the Store record.

Field Name	Type	Length	Decimal Offset	Description
DateTime	PD	5	7	Date and time when the period started or the old period ended, in the format `YYMMDDHHmm`).
Reserved	ASCII	11	12	Reserved.

Movement Record

Field Name	Type	Length	Decimal Offset	Description
ItemCode	PD	7	0	Item number.
SalesCount	Int	4	7	Count of the number of items sold.
SalesAmount	Int	4	11	Sum of the sales amount for the item.
Weight	Int	4	15	Sum of the weight or volume of the items sold.
Restart	Int	4	19	This field is used as a control field on a system restart. It is used to determine if this record has been updated by a particular transaction.

EAMIRCHG (Item Record Change File)

The Item Record Change file contains the item codes for all item records that have been changed by in-store applications. The file is mirrored to the alternate file server at each update. The purpose of this file is to allow for in-store generation of shelf labels on all changed items. It is not intended to replace the Exception Log function of tracking file alterations for security purposes.

Logical name	<EAMIRCHG>
Data object reference	• Batch record, *ISSAItemRecordChangeBatch* <`SAITMRCB.CPP`> • Data record, *ISSAItemRecordChange* <`SAITMRCC.CPP`>
Organization	Variable sequential
Distribution class	Mirrored per update
File copies	1
Record length	Variable

Batch Record

Field Name	Type	Length	Description
Type	ASCII	1	Record types: **3** Defines a batch record created by price changes at the terminal.

Field Name	Type	Length	Description
			4 Defines a batch record created outside of SurePOS ACE. File maintenance batches created at a host site are in this category.
Batch	PD	3	Batch name.
SequenceNumber	PD	1	Sequence number.
DateTime	PD	5	Date and time when the batch was executed, in the format `YYMMDDHHmm`.

Data Record

Field Name	Type	Length	Description
Type	ASCII	1	Record types: **1** Defines a data record that is not associated with a batch of changes. **2** Defines a data record that is part of a batch of item record changes. The batch is defined by the closest batch record that precedes this record.
ItemCode	PD	v7	Item code of changed item.

EAMITEMR (Item Record File)

Overview

Customer Loyalty program processing makes special use of several fields in the item record.

Items Not Eligible for Points

Points are normally accumulated based on the value of all sales. But the awarding of points for certain items may be unwanted or even illegal. The points apply flag in the item record is used to define those items for which points may (or may not) be accumulated. This flag pertains to all items except for coupons. You can define whether coupons reduce points through the *Store coupons reduce points* option in Loyalty -> Points personalization.

Points-Only Items

Bit 5 in `INDICAT1A` (`IN_PointsItem` flag X'20') distinguishes points-only items. These items (or coupons) have no monetary value, but contribute to or reduce preferred customer points. The number of points associated with the item is in the price field. A normal item with this flag on provides bonus points to a customer. A coupon item with this flag on can either provide bonus points or require the redemption of points. The number of points that apply to a coupon are restricted to five digits because the price field for a coupon is restricted to five digits. The difference between a points-only item and a points-only coupon is that the coupon requires validation and requires the prior purchase of a matching item.

New accounting totals for the amount of redeemed and bonus points are accumulated by operator and/or terminal.

Item movement totals for points-only items can be maintained at the discretion of the user. Sales of points-only items can be logged in the exception file. The sale of a points-only item can affect item movement totals but does not influence any other transaction totals, including department totals and net sales accounting totals.

Point coupons are not tracked in the coupon tracking file.

Points-only items apply only to preferred customers. It is suggested, but not required, that these items use item codes within the unique preferred customer range (300 to 399), particularly when points-only items are linked to other items.

Preferred customer item codes may not be appropriate when used for items such as scannable paper coupons. If a points-only item is accepted for a customer who has not been identified as a preferred customer, the operator is prompted at the entry of the first tender to either enter the customer's ID number or void the points item.

Redemption Coupons

The discount flag (`INDICAT1`, bit 1, `IN_NoDiscount` flag X'02') for a points-only coupon defines whether the coupon provides bonus points or requires the redemption of points. This flag is interpreted uniquely only for a points-only coupon. In this case, a flag setting of zero denotes positive (bonus) points while a flag setting of one denotes negative (redeemed) points. After this flag is uniquely interpreted for a points only coupon, it is set to one to prevent the coupon from contributing to discountable totals.

Links to Deposit

Bit 4 in `INDICAT1A` (`IN_SeparateMinPerCpnORLinksToDeposit` flag X'10') is the *links to deposit* flag for an item record. It links the item both to a deposit and to electronic coupons. In this case, the coupon or coupons are linked to the item and the deposit for that item is linked to the last coupon in the chain. If this flag is turned on in the first item in the chain, the deposit on the item is always charged, regardless of whether the coupon chain is broken through a failed validation. This flag is generally used only for marking items that link to deposits, but it can be used to mark any item that links first to a coupon and then to a non-coupon to assure that the link between the item and the non-coupon is preserved whether or not the coupon is accepted.

Coupons that Require Separate Minimum Purchase Amounts

Bit 4 in `INDICAT1A` (`IN_SeparateMinPerCpnORLinksToDeposit` flag X'10') is the *separate minimum per coupon* flag for a coupon record. It is used to distinguish coupons that require separate sales to satisfy the minimum purchase requirement for each use of the coupon in a transaction. This flag has meaning only for coupons that have a minimum purchase requirement (USREXIT2). If this flag is set in a coupon with a $10.00 minimum purchase amount, then a customer must have $20.00 of purchases to redeem two of the same coupons in an order.

The same purchases can be used to satisfy the minimum purchase requirements for any two different coupons regardless of this flag. Coupons are considered different if they reference separate item records.

Coupon Validation and Minimum Purchase Combinations

Allow Coupon Validation by Department with Minimum Purchase by Transaction

To enable this feature, set the *Minimum by MFR* flag (bit 5, `IN_TaxableC`, in `INDICAT1`) of the coupon record to yes and set the MFR field (`SALEPRICE1` and `SALEPRICE2`) to 00000. This forces department validation and not manufacturer validation. Set the minimum purchase field (`USREXIT2`) to the desired amount. Finally, set the *Minimum by Dept* flag (bit 4, `IN_TaxableD`, in `INDICAT1`) to no.

Allow Minimum Purchase by Department with No Coupon Validation

To enable this feature, set the *Minimum by MFR* flag (bit 5, `IN_TaxableC`, in `INDICAT1`) to no and the MFR field (SALEPRICE1 and SALEPRICE2) to 00000. This bypasses department and manufacturer validation. Set the minimum purchase field (`USREXIT2`) to the desired amount. Finally, set the *Minimum by Dept* flag (bit 4, `IN_TaxableD`, in `INDICAT1`) to Yes to indicate that the minimum applies only to the coupon's department.

Allow Minimum Purchase by Department with Validation by MFR

To enable this feature, set the *Minimum by MFR* flag (bit 5, `IN_TaxableC`, in `INDICAT1`) to no and the *MFR* field (SALEPRICE1 and SALEPRICE2) to the valid manufacturer number. These settings force validation by manufacturer. Set the *minimum purchase* field (`USREXIT2`) to the desired amount. Finally, set the *Minimum by Dept* flag (bit 4, `IN_TaxableD`, in `INDICAT1`) to Yes to indicate that the minimum applies only to the coupon's department.

The following table defines the relationship between coupon record inputs, coupon validation, and minimum purchase requirements:

INPUTS				RESULTS	
Minimum Purchase > 0 **	Non-zero MFR in Record	Min by MFR or Validate Dept	Minimum by Dept	Coupon Validation	Minimum Purchase Required
No	No	N/A	N/A	None ***	None
No	Yes	N/A	N/A	By MFR	None
Yes	No	No	No	None ***	For Order
Yes	No	No	Yes	None ***	For Dept

INPUTS				RESULTS	
Minimum Purchase > 0 **	Non-zero MFR in Record	Min by MFR or Validate Dept	Minimum by Dept	Coupon Validation	Minimum Purchase Required
Yes	No	Yes	No	By Dept	For Order
Yes	No	Yes	Yes	By Dept	For Dept
Yes	Yes	No	No	By MFR	For Order
Yes	Yes	No	Yes	By MFR	For Dept
Yes	Yes	Yes	No	By MFR	For MFR
Yes	Yes	Yes	Yes	By MFR	For MFR
Note: ** A minimum purchase amount of .01 is adequate to meet this requirement. *** Validation will be by department if the option *All Coupons Require At Least Department Level Validation* is selected.					

Basic Item Record File Layouts

Toshiba Global Commerce Solutions ships these item record files with SurePOS ACE:

eamitemr.46

46-byte Item Record file with 12-digit item codes. This is the Item Record file that the 4680-4690 Supermarket Application uses.

eamitemr.127

127-byte Item Record file with 12-digit item codes

eamitemr.dat

169-byte Item Record file with 12-digit item codes

gtnitemr.127

127-byte Item Record file with 14-digit item codes

gtnitemr.dat

169-byte Item Record file with 14-digit item codes

The 169-byte item record file provides the ability to extend an item record by letting you enable and disable new item record fields in Database personalization. Data Maintenance supports all item record lengths that Toshiba Global Commerce Solutions has supported, as do the `optnparm.ini` file, BOB reports, the Data Maintenance Item report, and the Selective Item report.

The default item record layout is the 169-byte layout. To use the 46-byte item record as the default:

1. Rename eamitemr.dat, `sapath.ddf`, and `sapath.rvt` in case you later decide to use the 169-byte layout.
2. Copy eamitemr.046 as eamitemr.dat.
3. Copy `sap046.ddf` as `sapath.ddf`.
4. Copy `sap046.rvt` as `sapath.rvt`.

The key to each record in the Item Record layouts is the item code assigned to each item.

Note: Regardless of the layout you use, the prime version is on the master controller. An image version is kept on all other eligible controllers. Item records are distributed per update due to the performance implications of their usage. Any application can read item records, but only authorized applications can alter them.

The 169-Byte Item Record File - 12-digit Item Code

The number of item records is limited only by available disk space.

Logical name	<EAMITEMR>
Data object reference	*ISSAItemStorage* <SAITMSTR.CPP/HPP>
Organization	Keyed
Distribution class	Compound per update
File copies	1
Record length	169 to 508 bytes
Key length	6 bytes: Item code

Table 53. Layout of 169-byte Item Record - 12-digit Item Code

Field Name	Type	Length	Decimal Offset	Description
ITEMCODE	PD	6	0	Item code - right justified with leading zeroes
INDICAT0	Int	1	6	Bit 7 X'80' - IN_QuantityNotAllowed: Quantity not allowed. Bit 6 X'40' - IN_WeightOrPriceRequired: Weight or price required since item is sold by weight. Bit 5 X'20' - IN_QuantityRequired: Quantity required. Bit 4 X'10' - IN_PriceRequired: Price required. Bit 3 X'08' - IN_ExceptionLogged: Item sale is exception logged. Bit 2 X'04' - IN_NotForSale: Item is not authorized for sale. Bit 1 X'02' - IN_RestrictedSale: Item not authorized for sale during restricted hours. Bit 0 X'01' - IN_NoMovement: No item movement kept.
INDICAT1	Int	1	7	Bit 7 X'80' - IN_TaxableA: Tax plan A applies to item. Bit 6 X'40' - IN_TaxableB: Tax plan B applies to item. Bit 5 X'20' - IN_TaxableC: For coupon item records, this is the *minimum purchase by*

Field Name	Type	Length	Decimal Offset	Description
				manufacturer flag. For item records, this is the *Tax plan C applies to item* flag. Bit 4 X'10' - IN_TaxableD: For coupon item records, this is the *minimum purchase by department* flag. For item records, this is the *Tax plan D applies to item* flag. Bit 3 X'08' - IN_FoodStampable: Food stamp item. Bit 2 X'04' - IN_TradingStamps: For coupon item records, this is the *item excluded from purchase minimum* flag. For item records, this is the *points apply to item* flag. Bit 1 X'02' - IN_NoDiscount: For points items, this is the *redemption item* flag. For other item records, this is the *non-discountable item* flag. Bit 0 X'01' - IN_NoCouponMultiplication: Coupon value multiplication not allowed on this item.
INDICAT1A	Int	1	8	Bit 7 X'80' - IN_NoShelfLabelPrint: Log to Change File flag. Bit 6 X'40' - IN_FuelItem: Fuel item flag. Bit 5 X'20' - IN_PointsItem: Points item flag. Bit 4 X'10' - IN_SepMinPerCpnORLinksToDep: For coupon item records, this is the, *minimum per coupon* flag. For item records, this is the *links to deposit* flag. Bit 3 X'08' - IN_UsrFlag1: User flage 1. Bit 2 X'04' - IN_UsrFlag2: User flage 2. Bit 1 X'02' - IN_UsrFlag3: User flage 3. Bit 0 X'01' - IN_UsrFlag4: User flage 4.
UNION2	Union	1	9	Defines this byte as one of two possible fields, either INDICAT2 or STRUCTTYPEPM.
INDICAT2	PD	1	9	See the STRUCTTYPEPM field for a description of how each packed-decimal digit is mapped.
STRUCTTYPEPM	Structure	1	9	This is the mapping of the two packed-decimal digits in INDICAT2. This structure defines the ITEMTYPE and PRICINGMETHOD fields.
ITEMTYPE	PD	.5	9	0 -Normal item sale 1 - Deposit 2 - Refund

Field Name	Type	Length	Decimal Offset	Description
				3 - Deposit return 4 - Misc. trans. receipt (sale) 5 - Misc. trans. payout (refund) 6 - Manufacturer coupon 7 - Store coupon 8 - Reserved 9 - Reserved
PRICINGMETHOD	PD	.5	9.5	0 - Unit or split package pricing 1 - Base + 1 pricing 2 - Group threshold pricing 3 - Reduced deal pricing 4 - Increased deal pricing. For a coupon item record, this implies one of the following: • The value of the coupon is calculated to yield a net price that equals the value in the price field (Net Value coupon). • The value of the coupon is calculated based on the savings value multiplied by the number of matching items purchased, if the number of matching items required is met or exceeded (Multiple Items coupon). • A percent-off amount is calculated. 5 - Pricing data is in another record. The item code for this record is in the six bytes starting at SALEQUANTITY. >5 - Reserved.. • Fake coupons use pricing method 3. • Percent-off link and percent-off validation coupons use pricing method 4.
DEPARTMENT	PD	2	10	Department number (value 1 - 999)
FAMILIES	Union	3	12	Defines these three bytes as one of two possible fields, either STRUCTFAM or STRUCTMISC.
STRUCTFAM	Structure	3	12	This is the active definition when the item is associated with a coupon family. This structure defines the COUPONFAMILY1 and COUPONFAMILY2 fields.

Field Name	Type	Length	Decimal Offset	Description
COUPONFAMILY1	PD	1.5	12	Current coupon family number. A coupon with an equal family number can be applied to this item. (Value 0-999).
COUPONFAMILY2	PD	1.5	13.5	Previous coupon family number. A coupon with an equal family number can be applied to this item. (Value 0-999).
STRUCTMISC	Structure	3	12	This is the active definition when the item is associated with a miscellaneous account. This structure defines the MISCSALEACCOUNT1 and MISCSALEACCOUNT2 fields.
MISCSALEACCOUNT1	PD	1.5	12	Miscellaneous account number from which the item amount is added if this is a miscellaneous item. (Value 0 - 999, where 0 implies no update).
MISCSALEACCOUNT2	PD	1.5	13.5	Miscellaneous account number from which the item amount is subtracted if this is a miscellaneous item. (Value 0 - 999, where 0 implies no update).
MPGROUP	PD	1	15	Multipricing group number / Mix-and-match number when a combination of items comprise a deal. (Value 0 - 99, where 0 = mix only with self).
UNIONPINFO	Union	6	16	Defines these six bytes as one of two possible fields, either ALIASITEMCODE or STRUCTPINFO.
ALIASITEMCODE	PD	6	16	This is the active definition when the item does not contain any pricing information because it is chained to an alias item that contains the price. This field specifies the item code of an alias item record.
STRUCTPINFO	Structure	6	16	This is the active definition for a normal sales item that contains its own pricing information. This structure defines SALEQUANTITY and UNIONPRICE.
SALEQUANTITY	PD	1	16	Count or weight of items in the deal. (Whole units only, no fractional weights) (Value 0 - 99, where 0 is equivalent to 1). For coupon item records without a weight limit, this field is both the maximum number (*x*) of this coupon per transaction and the number (*y*) of matching items required to validate this coupon according to the equation `(x*10) + y`. For coupon item records with a weight limit, this field is both the maximum number (*x*) of pounds to which this coupon applies and the number (*y*) of matching items required to

Field Name	Type	Length	Decimal Offset	Description
				validate this coupon according to the equation `(x*10) + y`. For these coupons, both the weight flag and the Coupon Weight Limit flag must be on. Multiple-item coupons have a transaction limit value of 0, which means unlimited, and use this field for the matching quantity limit. For a field value of 1-49, when the Big Coupon Limit flag is on, the field contains the number of matching items required. The maximum number of coupons per transaction is implied to be 1. For a field value of 50-99 when the Big Coupon Limit flag is on, the field represents the maximum number of coupons per transaction. The actual value for the maximum number of coupons is calculated by subtracting X'50' from the field value. For example, if this field contains X'59', then the maximum number of coupons per transaction is X'9'. When the maximum number of coupons is equal to or greater than 9, then the number of matching items required is implied to be 1.
`UNIONPRICE`	Union	5	17	Defines these five bytes as one of three possible fields, either `SALEPRICE`, `STRUCTMULTIPRICE`, or `Tare`.
`SALEPRICE`	PD	5	17	This is the active definition for pricing methods 0 or 1. For pricing method 0 or 1, this is the price in the format `00pppppppp` where: **pppppppp** Unit price if `SALEQUANTITY` = 0 or 1. **pppppppp** Deal price if `SALEQUANTITY` > 1. For coupon item records, this field uses the format `mmmmmabcde` where: **mmmmm** The manufacturer's number for validation, if manufacturer validation is selected. Otherwise, contains all zeroes. **abcde** The meaning depends on the type of coupon item record:

Field Name	Type	Length	Decimal Offset	Description
				• Coupon - `abcde` is the value of the coupon. • Fake coupon - `abcde` is always `00001`. • Percent-off link coupon - `ab` is always `99`, `cde` is the percent-off rate with one decimal place (cd.e). • Percent-off validation coupon - `ab` is always `98`, `cde` is the percent-off rate, including one decimal place (`cd.e`). • Points-multiply coupon - `abc` is always `960`, `de` is the percentage to multiply points, including one decimal place (`d.e`). • Multiple-item coupon - `ab` is always `95`, `cde` is the savings amount, which must be in the range 0-999. For redemption coupons, if the pricing method is 0 (which means that manufacturer validation is not used) then all ten digits are used to represent points amount (the left most two digits must be 00, so really only eight digits can be used). If the pricing method is 3 (which means that manufacturer validation is used) then five digits are used for manufacturer validation and the remaining five digits are used for points amount.
`STRUCTMULTIPRICE`	Structure	5	17	This is the active definition for pricing methods 2, 3, and 4, and for coupon item records. This structure defines the `SALEPRICE1` and `SALEPRICE2` fields.
`SALEPRICE1`	PD	2.5	17	For pricing method 2, this field is the package deal price. For pricing method 3 or 4, this field is the unit price. For coupon item records, this field is the manufacturer's number for validation.
`SALEPRICE2`	PD	2.5	19.5	For pricing method 2, this field is the single unit price. For pricing method 3 or 4, this field is the reduced price. For coupon item records, this field is the value of the coupon.

Field Name	Type	Length	Decimal Offset	Description
`Tare`	Structure	5	17	This is the active definition when the item contains tare pricing information, which is stored in this field. The Tare field is enabled when using the split-package pricing method (pricing method 0), the *IN_WeightOrPriceRqd* flag is set, and the *Tare* option is enabled in Options -> Database personalization. This structure defines the `TareCodeValue` and `TareFiller` fields.
`TareCodeValue`	PD	2.5	17	The first five packed-decimal digits (length = 2.5) from offset 17 are the *TareCodeValue* field that stores a value. The value is either an ID (set in personalization), a fixed weight, or a percent (which is a percent of the total package weight that is subtracted from the item weight). Values of the Tare field can be: **1 - 99** Tare value = 1 through 99: A Tare ID defined in personalization that retrieves the weight or percent from the options file, eamoptns.dat. **100 - 9999** Erroneously entered values that result in operator message, *Check Item Data*. **1xxxx** The digits *xxxx* represent the tare weight in hundredths of a pound or in grams, depending on the locale. **2xxxx** The digits *xxxx* represent hundreds of a percent of the item weight.
`TareFiller`	PD	2.5	19.5	Reserved.
`LINKEDTO`	PD	2	22	Item code to be sold with the sale of this item. (Value 0-9999, 0 = no linked item)
`ITEMNAME`	CHR	18	24	Item descriptor to be printed on receipt (no commas allowed)
`USREXIT1`	Int	2	42	User field for use with extensions. This field can be the promotion code depending on the value of the *Use promotion code for coupon validation* option in Options ->Coupon personalization.
`USREXIT2`	Int	2	44	User field for use with extensions. For coupon item records, this is the *minimum purchase amount* for coupon to be accepted. For item records, this can be the date of last sale (DOLS) depending on the setting of the *Keep date of last sale in User Data 2* option in

Field Name	Type	Length	Decimal Offset	Description
				Database -> Totals personalization. The DOLS is updated at a store close that closes the Item Movement file.
The remainder of the 127-byte Item Record contains optional data. The options in Options -> Database personalization determine which fields are enabled and disabled. Fields and offsets shown in the remainder of this record are for the default record that 4680-4690 Supermarket Application ships with SurePOS ACE.				
`RestrictedSaleType`	Int	1	46	Codes for alcohol=1 tobacco=2
`ReportingCode`	PD	1	47	Codes to break out reports by department.
`Indicator5`	Int	1	48	**Bit 7 X'80'** WICFlag: WIC flag **Bit 6 X'40'** BigCrossPromo: Big cross promo flag. **Bit 5 X'20'** BigCouponLimit: Big coupon limit flag. This flag is set when the `SALEQUANTITY` field represents one of the following: • Matching items required, and the value is equal to or greater than 10. • Maximum coupons per transaction, and the value is equal to or greater than 9. See the description of the `SALEQUANTITY` field (offset 16) for more information. **Bit 4 X'10'** CouponWeightLimit: Coupon weight limit. **Bit 3 X'08'** InStoreAddedItem: In-store added item flag. **Bit 2 X'04'** ValueCardState: Value card state. 0=activation, 1=reload. **Bit 1 X'02'** Reserved2: Reserved. **Bit 0 X'01'** Reserved3: Reserved. The bit 3 flag is set when a new item is added from Data Maintenance. It is turned off when

Field Name	Type	Length	Decimal Offset	Description
				an item has Auxiliary Pricing data active, which occurs when an item is added or overwritten from a host.
`Commodity`	PD	2	49	Groups items by commodity (for example, soaps) for reporting purposes. This field can be updated from the host. At the store, it can be updated only on when a new item is added, or when the in-store flag (offset 48, bit 3, X'08') is on.
`Subcommodity`	PD	3	51	More granularity for the commodity code at offset 49. It has the same purpose and update characteristics.
`MSINumber`	PD	4	54	The MSI is an order entry number that ranges from 0-99999. It can be updated from the host. At the store, it can be updated only when a new item is added, or if the in-store flag (offset 48, bit 3, X'08') is on.
`Cost`	Long	4	58	The wholesale cost of the item.
`LQD_LimitQty`	PD	1	62	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity sold at the reduced price. Other items are sold at the unit price.
`LQD_DealQty`	PD	1	63	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity at the reduced price when there is a multiple price. For example, `3` in a 3/$1.00 deal.
Host pricing data fields are a contiguous block of 10 bytes of data. The *Host Price Data* option in Options -> Database -> Item(1) personalization enables or disables the entire group of fields.				
`HostQuantity`	PD	1	64	`HostQuantity` and `HostSalePrice` must be contiguous because they are loaded from an external source directly into the item record.
`UNIONHOSTP`	Union	5	65	Defines these five bytes as one of two possible fields, either `HostSalePrice` or `StructHostMultiPrice`.
`HostSalePrice`	PD	5	65	This is the active definition for pricing method 0 or 1. For pricing method 0 or 1, the price is in format `00pppppppp` where: **pppppppp** Unit price if `HostQuantity` = 0 or 1 **pppppppp** Deal price if `HostQuantity` > 1

Field Name	Type	Length	Decimal Offset	Description
StructHostMultiPrice	Structure	5	65	This is the active definition when for pricing methods 2, 3, or 4. This structure defines the HostSalePrice1 and HostSalePrice2 fields.
HostSalePrice1	PD	2.5	65	For pricing method 2, this is the package deal price. For pricing method 3 or 4, this is the unit price.
HostSalePrice2	PD	2.5	67.5	For pricing method 2, this is the single unit price. For pricing method 3 or 4, this is the reduced price.
HostLQDLimit	PD	1	70	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity set by the host to be sold at the reduced price. Other items are sold at the unit price.
HostLQDDeal	PD	1	71	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity set by the host to be sold at the reduced price when there is a multiple price. For example, 3 in a 3/$1.00 deal.
UNIONHostPricingMethod	Union	1	72	Defines this byte as one of two possible fields, either HostIndicator2 or StructHostPricingMethod.
HostIndicator2	PD	1	72	See the StructHostPricingMethod field for a description of how each packed decimal digit is mapped.
StructHostPricingMethod	Structure	1	72	This is the mapping of the two packed decimal digits in the HostIndicator2 field. This structure defines the HostItemType and HostPricingMethod fields.
HostItemType	PD	.5	72	0 - Normal item sale 1 - Deposit 2 - Refund 3 - Deposit return 4 - Misc. trans. receipt (sale) 5 - Misc. trans. payout (refund) 6 - Manufacturer coupon 7 - Store coupon 8 - Reserved

Field Name	Type	Length	Decimal Offset	Description
				9 - ;Reserved
`HostPricingMethod`	PD	.5	72.5	0 - Unit or split package pricing 1 - Base + 1 pricing 2 - Group threshold pricing 3 - Reduced deal pricing 4 - Increased deal pricing. For a coupon item record, this implies the value of the coupon is calculated to yield a net value in the price field. **5** Pricing data is in another record. The item code for this record is in the six bytes starting at `HostQuantity`. **6-8** Reserved.
`InStorePriceChangeFlag`	Int	1	73	This flag is set to FALSE when `SALEQUAN`, `SALEPRIC`, and `INDICAT2` fields are set from *Host Price* fields (from a host source outside the store). When a price change is recognized as coming from inside the store (either from Data Maintenance or from the terminal), it is set to TRUE.
`CompINDICAT2$`	PD	1	74	Comparison price pricing method
`CompSALEQUAN$`	PD	1	75	Comparison price deal quantity
`UnionComparisonPrice`	Union	5	76	Defines these five bytes as one of three possible fields, either `StructCompPrice2Byte`, `StructCompPrice4Byte`, or `CompPricePD`.
`StructCompPrice2Byte`	Structure	5	76	This is the active union definition when the *Comparison Price Format* option in Options -> Database -> Item(4) -> Comparison Price personalization is enabled. This structure defines the `CompPrice2Byte` and `Filler1` fields.
`CompPrice2Byte`	Int	2	76	Comparison price, 2 byte format.
`Filler1`	ASCII	3	78	Reserved.
`StructCompPrice4Byte`	Structure	5	76	This is the active union definition when the *Comparison Price Format* option in Options -> Database -> Item(4) -> Comparison Price personalization is enabled. This structure defines the `CompPrice4Byte` and `Filler` fields.

Field Name	Type	Length	Decimal Offset	Description
`CompPrice4Byte`	Int	4	76	Comparison price, 4 byte format.
`Filler`	ASCII	1	80	Reserved.
`CompPricePD`	PD	5	76	Note: Comparison price, packed decimal format.
`CompSaleType`	Int	1	81	Note: Comparison sale pricing type.
`CompetitorIndex`	Int	1	82	Note: Comparison pricing competition index.
`Price.Date`	PD	3	83	Note: Date of comparison price.
`AlternateDiscountByDept`	PD	2	86	Includes items in a Discount by Department promotion without changing the department number of the item.
`VATCode`	Int	1	88	Defaults to offset 0 (off)
`UNIONHostPriceChangeLog`	Union	11	89	Defines these eleven bytes as one of two possible fields, either `HostPriceChangeLog` or `HostPriceChangeLogStruct`.
Host price change log fields are contained in the 11 bytes that begin at offset 89. The *Host Price Change Log* option in Options -> Database -> Item(1) personalization enables or disables the entire group of fields.				
`HostPriceChangeLog`	ASCII	11	89	This field is used by older code; newer code uses the *HostPriceChangeLogStruct* field.
`HostPriceChangeLogStruct`	Structure	11	89	This is the active union definition when the *Host Price Change Log* option in Options -> Database -> Item(1) personalization is enabled. This structure defines the `HostPriceReasonCode`, `HostPriceOperatorID`, and `HostPriceChangeDateTime` fields.
`HostPriceReasonCode`	Int	1	89	Reason code for host price.
`HostPriceOperatorID`	Int	5	90	Operator ID.
`HostPriceChangeDateTime`	Timestamp	5	95	Date and time of host price change.
`SecondaryPointsClub`	PD	3	100	Up to three 1-byte secondary club numbers. If number of clubs is 0, the item only contributes to primary points. The bytes for this field must be contiguous. To use more than three secondary club numbers, you must redefine the SAPATH file. See "Using SAPATH.RVT" on page 608.

Field Name	Type	Length	Decimal Offset	Description
LargeLinkedTo	PD	6	103	Large linked-to item code.
ValueCardType	Int	1	109	Value card type. 0 - Not a value card. 61 - Gift card (X'3D') 62 - Phone card (X'3E') 63 - Blackhawk card (X'3F')
CouponLevel	PD	1	110	Coupon level.
CouponCoPay	Int	2	111	Coupon co-pay amount.
EASType	Int	1	113	EAS type.
Reserved4	PD	.5	114	Reserved.
PricingMethod_2	PD	.5	114.5	Alternate pricing method, which overlays the PRICINGMETHOD field. Only pricing methods 0 - 4 are valid.
SaleQuantity_2	PD	1	115	Sale quantity for alternate pricing, which overlays the SALEQUANTITY field. Valid values are 0 - 99. 0 - Do not use alternate pricing fields. Nonzero - Use alternate pricing fields.
UNIONPRICE_2	Union	5	116	Defines these five bytes as one of two possible fields, either SalePrice_2 or STRUCTMULTIPRICE_2.
SalePrice_2	PD	5	116	Sale price for alternate pricing, which overlays the SALEPRICE field.
STRUCTMULTIPRICE_2	Structure	5	116	This is the active alternate definition for item types 0 and 4 that use pricing methods 2, 3, and 4. This structure defines the SalePrice1_2 and SalePrice2_2 fields.
SalePrice1_2	PD	2.5	116	For pricing method 2, this field is the package deal price for alternate pricing. For pricing method 3 or 4, this field is the alternate unit price.
SalePrice2_2	PD	2.5	118.5	For pricing method 2, this field is the single unit price for alternate pricing. For pricing method 3 or 4, this field is the alternate reduced price.
Department_2	PD	2	121	Department number for alternate pricing, which overlays the DEPARTMENT field. If 0, then both primary and alternate pricing use the primary department number.

Field Name	Type	Length	Decimal Offset	Description
MPGroup_2	PD	1	123	Multipricing group number or Mix-and-match number when a combination of items comprise a deal. (Value 0 - 99, where 0 = mix only with self).
LQD_LimitQty_2	PD	1	124	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity sold at the alternate reduced price. Other items are sold at the unit price.
LQD_DealQty_2	PD	1	125	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity at the alternate reduced price when there is a multiple price. For example, 3 in a 3/$1.00 deal.
SCFINDICAT0	Int	1	126	Bit 7 X'80' - SalesAssociateRequired: Sale Associate Required Indicator Bit 6 X'40' - IN_NotForSelfCheckoutSales: Not For Self Checkout Sale Indicator Bit 5 X'20' - IN_SelfCheckoutFlag3: Reserved Bit 4 X'10' - IN_SelfCheckoutFlag4: Reserved Bit 3 X'08' - IN_SelfCheckoutFlag5: Reserved Bit 2 X'04' - IN_SelfCheckoutFlag6: Reserved Bit 1 X'02' - IN_SelfCheckoutFlag7: Reserved Bit 0 X'01' - IN_SelfCheckoutFlag8: Reserved
ItemSubType	Int	1	127	Item Sub Type 00 - No Sub Type 01 - Car Wash 02 - Foodstamp Forgiven Fee
Indicator4	Int	4	128	Bits 8-31: Reserved Bit 7 X'00000080': Tax Plan E Bit 6 X'00000040': Tax Plan F Bit 5 X'00000020': Tax Plan G Bit 4 X'00000010': Tax Plan H Bit 3 X'00000008': Scale Sentry Always Bit 2 X'00000004': Scale Sentry Never

Field Name	Type	Length	Decimal Offset	Description
				Bit 1 X'00000002': Qualified Healthcare Product (QHP) Bit 0 X'00000001': Prescription (Rx) Item
`IFPSCode`	PD	3	132	Associated IFPS code for the item
`TenderRestrictionGroup`	Int	1	135	Tender restriction group for this item
`Reserved5`	ASCII	33	136	Reserved

The 169-Byte Item Record File - 14-digit Item Code

The number of item records is limited only by the available disk space.

Logical name	<EAMITM14>
Data object reference	*ISSAItemStorage* <SAITMSTR.CPP/HPP>
Organization	Keyed
Distribution class	Compound per update
File copies	1
Record length	169 to 508 bytes
Key length	7 bytes: Item code

Table 54. Layout of 169-byte Item Record - 14-Digit (GTIN) Item Code

Field Name	Type	Length	Decimal Offset	Description
`ITEMCODE`	PD	7	0	Item code - right justified with leading zeroes
`INDICAT0`	Int	1	7	**Bit 7 X'80'** IN_QuantityNotAllowed: Quantity not allowed. **Bit 6 X'40'** IN_WeightOrPriceRequired: Weight or price required since item is sold by weight. **Bit 5 X'20'** IN_QuantityRequired: Quantity required. **Bit 4 X'10'** IN_PriceRequired: Price required.

Field Name	Type	Length	Decimal Offset	Description
				Bit 3 X'08' IN_ExceptionLogged: Item sale is exception logged. **Bit 2 X'04'** IN_NotForSale: Item is not authorized for sale. **Bit 1 X'02'** IN_RestrictedSale: Item not authorized for sale during restricted hours. **Bit 0 X'01'** IN_NoMovement: No item movement kept.
INDICAT1	Int	1	8	**Bit 7 X'80'** IN_TaxableA: Tax plan A applies to item **Bit 6 X'40'** IN_TaxableB: Tax plan B applies to item **Bit 5 X'20'** IN_TaxableC: For coupon item records, this is the *minimum purchase by manufacturer* flag. For item records, this is the *Tax plan C applies to item* flag. **Bit 4 X'10'** IN_TaxableD: For coupon item records, this is the *minimum purchase by department* flag. For item records, this is the *Tax plan D applies to item* flag. **Bit 3 X'08'** IN_FoodStampable: Food stamp item **Bit 2 X'04'** IN_TradingStamps: For coupon item

Field Name	Type	Length	Decimal Offset	Description
				records, this is the *item excluded from purchase minimum* flag. For item records, this is the *points apply to item* flag. **Bit 1 X'02'** IN_NoDiscount: For points items, this is the *redemption item* flag. For other item records, this is the *non-discountable item* flag. **Bit 0 X'01'** IN_NoCouponMultiplication: Coupon value multiplication not allowed on this item.
INDICAT1A	Int	1	9	**Bit 7 X'80'** IN_NoShelfLabelPrint: Log to Change File flag. **Bit 6 X'40'** IN_FuelItem: Fuel item flag. **Bit 5 X'20'** IN_PointsItem: Points item flag. **Bit 4 X'10'** IN_SepMinPerCpnORLinksToDep: For coupon item records, this is the *separate minimum per coupon* flag. For item records, this is the *links to deposit* flag. **Bit 3 X'08'** IN_UsrFlag1: User flag 1. **Bit 2 X'04'** IN_UsrFlag2: User flag 2. **Bit 1 X'02'** IN_UsrFlag3: User flag 3.

Field Name	Type	Length	Decimal Offset	Description
				Bit 0 X'01' IN_UsrFlag4: User flag 4.
`UNION2`	Union	1	10	Defines this byte as one of two possible fields, either `INDICAT2` or `STRUCTTYPEPM`.
`INDICAT2`	PD	1	10	See the `STRUCTTYPEPM` field for a description of how each packed decimal digit is mapped.
`STRUCTTYPEPM`	Structure	1	10	This is the mapping of the two packed-decimal digits in `INDICAT2`. This structure defines the `ITEMTYPE` and `PRICINGMETHOD` fields.
`ITEMTYPE`	PD	.5	10	0 - Normal item sale 1 - Deposit 2 - Refund 3 - Deposit return 4 - Misc. trans. receipt sale) 5 - Misc. trans. payout (refund) 6 - Manufacturer coupon 7 - Store coupon 8 - Reserved 9 - Reserved
`PRICINGMETHOD`	PD	.5	10.5	0 - Unit or split package pricing 1 - Base + 1 pricing 2 - Group threshold pricing 3 - Reduced deal pricing **4** Increased deal pricing. For a coupon item record, this implies one of the following: • The value of the coupon is calculated to yield a net price that equals the value in the price field (Net Value coupon).

Field Name	Type	Length	Decimal Offset	Description
				• The value of the coupon is calculated based on the savings value multiplied by the number of matching items purchased, if the number of matching items required is met or exceeded (Multiple Items coupon). • A percent-off amount is calculated. **5** Pricing data is in another record. The item code for this record is in the six bytes starting at `SALEQUANTITY`. **>5** Reserved. • Fake coupons use pricing method 3. • Percent-off link and percent-off validation coupons use pricing method 4.
`DEPARTMENT`	PD	2	11	Department number (value 1 - 999)
`FAMILIES`	Union	3	13	Defines these three bytes as one of two possible fields, either `STRUCTFAM` or `STRUCTMISC`.
`STRUCTFAM`	Structure	3	13	This is the active definition when the item is associated with a coupon family. This structure defines the COUPONFAMILY1 and COUPONFAMILY2 fields.
`COUPONFAMILY1`	PD	1.5	13	Current coupon family number. A coupon with an equal family number can be applied to this item. (Value 0-999).

Field Name	Type	Length	Decimal Offset	Description
COUPONFAMILY2	PD	1.5	14.5	Previous coupon family number. A coupon with an equal family number can be applied to this item. (Value 0-999).
STRUCTMISC	Structure	3	13	This is the active definition when the item is associated with a miscellaneous account. This structure defines the MISCSALEACCOUNT1 and MISCSALEACCOUNT2 fields.
MISCSALEACCOUNT1	PD	1.5	13	Miscellaneous account number from which the item amount is added if this is a miscellaneous item. (Value 0 - 999, where 0 implies no update).
MISCSALEACCOUNT2	PD	1.5	14.5	Miscellaneous account number from which the item amount is subtracted if this is a miscellaneous item. (Value 0 - 999, where 0 implies no update).
MPGROUP	PD	1	16	Multipricing group number / Mix-and-match number when a combination of items comprise a deal. (Value 0 - 99, where 0 = mix only with self).
UNIONPINFO	Union	7	17	Defines these seven bytes as one of two possible fields, either ALIASITEMCODE or STRUCTPINFO.
ALIASITEMCODE	PD	7	17	This is the active definition when the item does not contain any pricing information because it is chained to an alias item that contains the price. This field specifies the item code of an alias item record.
STRUCTPINFO	Structure	7	17	This is the active definition for a normal sales item that contains its own pricing information. This structure defines the SALEQUANTITY and UNIONPRICE fields.
Filler	ASCII	1	17	Reserved
SALEQUANTITY	PD	1	18	Count or weight of items in the deal. (Whole units only,

Field Name	Type	Length	Decimal Offset	Description
				no fractional weights) (Value 0 - 99, where 0 is equivalent to 1). For coupon item records without a weight limit, this field is both the maximum number (*x*) of this coupon per transaction and the number (*y*) of matching items required to validate this coupon according to the equation `(x*10) + y`. For coupon item records with a weight limit, this field is both the maximum number (*x*) of pounds to which this coupon applies and the number (*y*) of matching items required to validate this coupon according to the equation `(x*10) + y`. For these coupons, both the weight flag and the Coupon Weight Limit flag must be on. Multiple-item coupons have a transaction limit value of 0, which means unlimited, and use this field for the matching quantity limit. For a field value of 1-49, when the Big Coupon Limit flag is on, the field contains the number of matching items required. The maximum number of coupons per transaction is implied to be 1. For a field value of 50-99 when the Big Coupon Limit flag is on, the field represents the maximum number of coupons per transaction. The actual value for the maximum number of coupons is calculated by subtracting X'50' from the field value. For example, if this field contains X'59', then the maximum number of coupons per transaction is X'9'. When the maximum number of coupons is equal to or greater than 9, then the number of matching

Field Name	Type	Length	Decimal Offset	Description
				items required is implied to be 1.
UNIONPRICE	Union	5	19	Defines these five bytes as one of three possible fields, either SALEPRICE, STRUCTMULTIPRICE, or Tare.
SALEPRICE	PD	5	19	This is the active definition for pricing methods 0 or 1. For pricing method 0 or 1, this is the price in the format 00pppppppp where: **pppppppp** Unit price if SaleQuantity = 0 or 1. **pppppppp** Deal price if SaleQuantity > 1. For coupon item records, this field has the format mmmmmabcde where: **mmmmm** The manufacturer's number for validation, if manufacturer validation is selected. Otherwise, contains all zeroes. **abcde** The meaning depends on the type of coupon item record: • Coupon - abcde is the value of the coupon. • Fake coupon - abcde is always 00001. • Percent-off link coupon - ab is always 99, cde is the percent-off rate with one decimal place (cd.e).

Field Name	Type	Length	Decimal Offset	Description
				• Percent-off validation coupon - `ab` is always `98`, `cde` is the percent-off rate, including one decimal place (`cd.e`). • Points-multiply coupon - `abc` is always `960`, `de` is the percentage to multiply points, including one decimal place (`d.e`). • Multiple-item coupon - `ab` is always `95`, `cde` is the savings amount, which must be in the range 0-999.
`STRUCTMULTIPRICE`	Structure	5	19	This is the active definition for pricing methods 2, 3, and 4, and for coupon item records. This structure defines the `SALEPRICE1` and `SALEPRICE2` fields.
`SALEPRICE1`	PD	2.5	19	For pricing method 2, this field is the package deal price. For pricing method 3 or 4, this field is the unit price. For coupon item records, this field is the manufacturer's number for validation.
`SALEPRICE2`	PD	2.5	21.5	For pricing method 2, this field is the single unit price. For pricing method 3 or 4, this field is the reduced price. For coupon item records, this field is the value of the coupon.
`Tare`	Structure	5	19	This is the active definition when the item contains tare pricing information, which is stored in this field. The Tare field is enabled when using the split-package pricing method (pricing method 0), the *IN_WeightOrPriceRqd* flag

Field Name	Type	Length	Decimal Offset	Description
				is set, and the *Tare* option is enabled in Options -> Database personalization. This structure defines the `TareCodeValue` and `TareFiller` fields.
`TareCodeValue`	PD	2.5	19	The first five packed decimal digits (length = 2.5) from offset 19 are the *TareCodeValue* field that stores a value. The value is either an ID (set in personalization), a fixed weight, or a percent (which is a percent of the total package weight that is subtracted from the item weight). Values of the Tare field can be: **1 - 99** Tare value = 1 through 99: A tare ID defined in personalization that retrieves the weight or percent from the Options file, EAMOPTNS.DAT. **100 - 9999** Erroneously entered values that result in operator message, *Check Item Data*. **1xxxx** The digits *xxxx* represent the tare weight in hundredths of a pound or in grams, depending on the locale. **2xxxx** The digits *xxxx* represent hundreds of a percent of the item weight.
`TareFiller`	PD	2.5	21.5	Reserved.
`LINKEDTO`	PD	2	24	Item code to be sold with the sale of this item. (Value 0-9999, 0 = no linked item)

Field Name	Type	Length	Decimal Offset	Description
ITEMNAME	CHR	18	26	Item descriptor to be printed on receipt (no commas allowed)
USREXIT1	Int	2	44	User field for use with extensions. This field can be the promotion code depending on the value of the *Use promotion code for coupon validation* option in Options ->Coupon personalization.
USREXIT2	Int	2	46	User field for use with extensions. For coupon item records, this is the *minimum purchase amount* for coupon to be accepted. For item records, this can be the date of last sale (DOLS) depending on the setting of the *Keep date of last sale in User Data 2* option in Database -> Totals personalization. The DOLS is updated at a store close that closes the Item Movement file.
The remainder of the 127-byte Item Record contains optional data. The options in Options -> Database personalization determine which fields are enabled and disabled. Fields and offsets shown in the remainder of this record are for the default record that Toshiba ships with SurePOS ACE.				
RestrictedSaleType	Int	1	48	Codes for alcohol=1 tobacco=2
ReportingCode	PD	1	49	Codes to break out reports by department.
Indicator5	Int	1	50	**Bit 7 X'80'** WICFlag: WIC flag **Bit 6 X'40'** BigCrossPromo: Big cross promo flag. **Bit 5 X'20'** BigCouponLimit: Big coupon limit flag. This flag is set when the SALEQUANTITY field represents one of the following: • Matching items required, and the value is equal to or greater than 10.

Field Name	Type	Length	Decimal Offset	Description
				• Maximum coupons per transaction, and the value is equal to or greater than 9. See the description of the `SALEQUANTITY` field (offset 18) for more information. **Bit 4 X'10'** CouponWeightLimit: Coupon weight limit. **Bit 3 X'08'** InStoreAddedItem: In-store added item flag. **Bit 2 X'04'** ValueCardState: Value card state. 0=activation, 1=reload. **Bit 1 X'02'** Reserved2: Reserved. **Bit 0 X'01'** Reserved3: Reserved. The bit 3 flag is set when a new item is added from Data Maintenance. It is turned off when an item has Auxiliary Pricing data active, which occurs when an item is added or overwritten from a host.
`Commodity`	PD	2	51	Groups items by commodity (for example, soaps) for reporting purposes. This field can be updated from the host. At the store, it can be updated only on when a new item is added, or when the in-store flag (offset 50, bit 3, X'08') is on.
`Subcommodity`	PD	3	53	More granularity for the commodity code at offset 51. It has the same purpose and update characteristics.
`MSINumber`	PD	4	56	The MSI is an order entry number that ranges from 0-99999. It can be updated

Field Name	Type	Length	Decimal Offset	Description
				from the host. At the store, it can be updated only when a new item is added, or if the in-store flag (offset 50, bit 3, X'08') is on.
`Cost`	Long	4	60	The wholesale cost of the item.
`LQD_LimitQty`	PD	1	64	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity sold at the reduced price. Other items are sold at the unit price.
`LQD_DealQty`	PD	1	65	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity at the reduced price when there is a multiple price. For example, `3` in a 3/$1.00 deal.
Host pricing data fields are a contiguous block of 10 bytes of data. The *Host Price Data* option in Options -> Database -> Item(1) personalization enables or disables the entire group of fields.				
`HostQuantity`	PD	1	66	`HostQuantity` and `HostSalePrice` must be contiguous because they are loaded from an external source directly into the item record.
`UNIONHOSTP`	Union	5	67	Defines these five bytes as one of two possible fields, either `HostSalePrice` or `StructHostMultiPrice`.
`HostSalePrice`	PD	5	67	This is the active definition for pricing method 0 or 1. For pricing method 0 or 1, the price is in format `00pppppppp` where: **pppppppp** Unit price if `HostQuantity` = 0 or 1 **pppppppp** Deal price if `HostQuantity` > 1
`StructHostMultiPrice`	Structure	5	67	This is the active definition when for pricing methods 2, 3, or 4. This structure defines the

Field Name	Type	Length	Decimal Offset	Description
				HostSalePrice1 and HostSalePrice2 fields.
HostSalePrice1	PD	2.5	67	For pricing method 2, this is the package deal price. For pricing method 3 or 4, this is the unit price.
HostSalePrice2	PD	2.5	69.5	For pricing method 2, this is the single unit price. For pricing method 3 or 4, this is the reduced price.
HostLQDLimit	PD	1	72	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity set by the host to be sold at the reduced price. Other items are sold at the unit price.
HostLQDDeal	PD	1	73	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity set by the host to be sold at the reduced price when there is a multiple price. For example, 3 in a 3/$1.00 deal.
UNIONHostPricingMethod	Union	1	74	Defines this byte as one of two possible fields, either HostIndicator2 or StructHostPricingMethod.
HostIndicator2	PD	1	74	See the StructHostPricingMethod field for a description of how each packed decimal digit is mapped.
StructHostPricingMethod	Structure	1	74	This is the mapping of the two packed-decimal digits in the HostIndicator2 field. This structure defines the HostItemType and HostPricingMethod fields.
HostItemType	PD	.5	74	0 - Normal item sale 1 - Deposit 2 - Refund 3 - Deposit return 4 - Misc. trans. receipt (sale)

Field Name	Type	Length	Decimal Offset	Description
				5 - Misc. trans. payout (refund) 6 - Manufacturer coupon 7 - Store coupon 8 - Reserved 9 - Reserved
`HostPricingMethod`	PD	.5	74.5	0 - Unit or split package pricing 1 - Base + 1 pricing 2 - Group threshold pricing 3 - Reduced deal pricing **4** Increased deal pricing. For a coupon item record, this implies the value of the coupon is calculated to yield a net value in the price field. **5** Pricing data is in another record. The item code for this record is in the six bytes starting at `HOSTQUANTITY`. **6-8** Reserved.
`InStorePriceChangeFlag`	Int	1	75	This flag is set to FALSE when `SALEQUAN`, `SALEPRIC`, and `INDICAT2` fields are set from *Host Price* fields (from a host source outside the store). When a price change is recognized as coming from inside the store (either from Data Maintenance or from the terminal), it is set to TRUE.
`CompINDICAT2$`	PD	1	76	Comparison price pricing method
`CompSALEQUAN$`	PD	1	77	Comparison price deal quantity
`UnionComparisonPrice`	Union	5	78	Defines these five bytes as one of three possible fields: `StructCompPrice2Byte`,

Field Name	Type	Length	Decimal Offset	Description
				`StructCompPrice4Byte`, or `CompPricePD`.
`StructCompPrice2Byte`	Structure	5	78	This is the active union definition when the *Comparison Price Format* option in Options -> Database -> Item(4) -> Comparison Price personalization is enabled. This structure defines the `CompPrice2Byte` and `Filler1` fields.
`CompPrice2Byte`	Int	2	78	Comparison price, 2-byte format.
`Filler1`	ASCII	3	80	Reserved.
`StructCompPrice4Byte`	Structure	5	78	This is the active union definition when the *Comparison Price Format* option in Options -> Database -> Item(4) -> Comparison Price personalization is enabled. This structure defines the `CompPrice4Byte` and `Filler` fields.
`CompPrice4Byte`	Int	4	78	Comparison price, 4-byte format.
`Filler`	ASCII	1	82	Reserved.
`CompPricePD`	PD	5	78	Comparison price, packed-decimal format.
`CompSaleType`	Int	1	83	Comparison sale pricing type.
`CompetitorIndex`	Int	1	84	Comparison pricing competition index.
`Price.Date`	PD	3	85	Date of comparison price.
`AlternateDiscountByDept`	PD	2	88	Includes items in a Discount by Department promotion without changing the department number of the item.
`VATCode`	Int	1	90	Defaults to offset 0 (off)
`UNIONHostPriceChangeLog`	Union	11	91	Defines these eleven bytes as one of two possible fields, either `HostPriceChangeLog` or `HostPriceChangeLogStruct`.
Host price change log fields are contained in the 11 bytes that begin at offset 91. The *Host Price Change Log* option in Options -> Database -> Item(1) personalization enables or disables the entire group of fields.				

Field Name	Type	Length	Decimal Offset	Description
`HostPriceChangeLog`	ASCII	11	91	This field is used by older code; newer code uses the `HostPriceChangeLogStruct` field.
`HostPriceChangeLogStruct`	Structure	11	91	This is the active union definition when the *Host Price Change Log* option in Options -> Database -> Item(1) personalization is enabled. This structure defines the `HostPriceReasonCode`, `HostPriceOperatorID`, and `HostPriceChangeDateTime` fields.
`HostPriceReasonCode`	Int	1	91	Reason code for host price.
`HostPriceOperatorID`	Int	5	92	Operator ID.
`HostPriceChangeDateTime`	Timestamp	5	97	Date and time of host price change.
`SecondaryPointsClub`	PD	3	102	Up to three 1-byte secondary club numbers. If number of clubs is 0, the item only contributes to primary points. The bytes for this field must be contiguous. To use more than three secondary club numbers, you must redefine the SAPATH file. See "Using SAPATH.RVT" on page 608.
`LargeLinkedTo`	PD	7	105	Large linked-to item code.
`ValueCardType`	Int	1	112	Value card type. 0 - Not a value card. 61 - Gift card (X'3D') 62 - Phone card (X'3E') 63 - Blackhawk card (X'3F')
`CouponLevel`	PD	1	113	Coupon level.
`CouponCoPay`	Int	2	114	Coupon co-pay amount.
`EASType`	Int	1	116	EAS type.
`Reserved4`	PD	.5	117	Reserved.
`PricingMethod_2`	PD	.5	117.5	Alternate pricing method, which overlays the `PRICINGMETHOD` field. Only pricing methods 0 - 4 are valid.

Field Name	Type	Length	Decimal Offset	Description
`SaleQuantity_2`	PD	1	118	Sale quantity for alternate pricing, which overlays the `SALEQUANTITY` field. Valid values are 0 - 99. **0** Do not use alternate pricing fields. **Nonzero** Use alternate pricing fields.
`UNIONPRICE_2`	Union	5	119	Defines these five bytes as one of two possible fields, either `SalePrice_2` or `STRUCTMULTIPRICE_2`.
`SalePrice_2`	PD	5	119	Sale price for alternate pricing, which overlays the `SALEPRICE` field.
`STRUCTMULTIPRICE_2`	Structure	5	119	This is the active alternate definition for item types 0 and 4 that use pricing methods 2, 3, and 4. This structure defines the `SalePrice1_2` and `SalePrice2_2` fields.
`SalePrice1_2`	PD	2.5	119	For pricing method 2, this field is the package deal price for alternate pricing. For pricing method 3 or 4, this field is the alternate unit price.
`SalePrice2_2`	PD	2.5	121.5	For pricing method 2, this field is the single unit price for alternate pricing. For pricing method 3 or 4, this field is the alternate reduced price.
`Department_2`	PD	2	124	Department number for alternate pricing, which overlays the `DEPARTMENT` field. If 0, then both primary and alternate pricing use the primary department number.
`MPGroup_2`	PD	1	126	Multipricing group number or Mix-and-match number when a combination of items comprise a deal. (Value 0 - 99, where 0 = mix only with self).

Field Name	Type	Length	Decimal Offset	Description
`LQD_LimitQty_2`	PD	1	127	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity sold at the alternate reduced price. Other items are sold at the unit price.
`LQD_DealQty_2`	PD	1	128	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity at the alternate reduced price when there is a multiple price. For example, 3 in a 3/$1.00 deal.
`SCFINDICAT0`	Int	1	129	**Bit 7 X'80'** IN_SalesAssociateRequired: Sales Associated Required Indicator **Bit 6 X'40'** IN_NotForSelfCheckoutSales: Not For Self Checkout Sales Indicator **Bit 5 X'20'** IN_SelfCheckoutFlag3: Reserved **Bit 4X'10'** IN_SelfCheckoutFlag4: Reserved **Bit 3X'08'** IN_SelfCheckoutFlag5: Reserved **Bit 2X'04'** IN_SelfCheckoutFlag6: Reserved **Bit 1X'02'** IN_SelfCheckoutFlag7: Reserved **Bit 0X'01'** IN_SelfCheckoutFlag8: Reserved
`ItemSubType`	Int	1	130	Item Sub Type

Field Name	Type	Length	Decimal Offset	Description
				00 - No Sub Type 01 - Car Wash 02 - Foodstamp Forgiven Fee
`Indicator4`	Int	4	131	**Bits 8-31** Reserved **Bit 7 X'00000080'** Tax Plan E **Bit 6 X'00000040'** Tax Plan F **Bit 5 X'00000020'** Tax Plan G **Bit 4 X'00000010'** Tax Plan H **Bit 3 X'00000008'** Scale Sentry Always **Bit 2 X'00000004** Scale Sentry Never **Bit 1 X'00000002'** Qualified Healthcare Product (QHP) **Bit 0 X'00000001'** Prescription (Rx) Item
`IFPSCode`	PD	3	135	Associated IFPS code for the item
`TenderRestrictionGroup`	Int	1	138	Tender restriction group for this item
`Reserved5`	ASCII	30	139	Reserved

The 127-Byte Item Record File - 12-digit Item Code

The number of item records is limited only by available disk space.

Logical name	<EAMITEMR>
Data object reference	*ISSAItemStorage* <SAITMSTR.CPP/HPP>
Organization	Keyed
Distribution class	Compound per update
File copies	1

Record length	127 to 508 bytes
Key length	6 bytes: Item code

Table 55. Layout of 127-byte Item Record - 12-digit Item Code

Field Name	Type	Length	Decimal Offset	Description
ITEMCODE	PD	6	0	Item code - right justified with leading zeroes
INDICAT0	Int	1	6	**Bit 7 X'80'** IN_QuantityNotAllowed: Quantity not allowed. **Bit 6 X'40'** IN_WeightOrPriceRequired: Weight or price required since item is sold by weight. **Bit 5 X'20'** IN_QuantityRequired: Quantity required. **Bit 4 X'10'** IN_PriceRequired: Price required. **Bit 3 X'08'** IN_ExceptionLogged: Item sale is exception logged. **Bit 2 X'04'** IN_NotForSale: Item is not authorized for sale. **Bit 1 X'02'** IN_RestrictedSale: Item not authorized for sale during restricted hours. **Bit 0 X'01'** IN_NoMovement: No item movement kept.
INDICAT1	Int	1	7	**Bit 7 X'80'** IN_TaxableA: Tax plan A applies to item **Bit 6 X'40'** IN_TaxableB: Tax plan B applies to item **Bit 5 X'20'** IN_TaxableC: For coupon item records, this is the *minimum purchase by manufacturer* flag. For item records, this is the *Tax plan C applies to item* flag.

Field Name	Type	Length	Decimal Offset	Description
				Bit 4 X'10' IN_TaxableD: For coupon item records, this is the *minimum purchase by department* flag. For item records, this is the *Tax plan D applies to item* flag. **Bit 3 X'08'** IN_FoodStampable: Food stamp item **Bit 2 X'04'** IN_TradingStamps: For coupon item records, this is the *item excluded from purchase minimum* flag. For item records, this is the *points apply to item* flag. **Bit 1 X'02'** IN_NoDiscount: For points items, this is the *redemption item* flag. For other item records, this is the *non-discountable item* flag. **Bit 0 X'01'** IN_NoCouponMultiplication: Coupon value multiplication not allowed on this item.
`INDICAT1A`	Int	1	8	**Bit 7 X'80'** IN_NoShelfLabelPrint: Log to Change File flag. **Bit 6 X'40'** IN_FuelItem: Fuel item flag. **Bit 5 X'20'** IN_PointsItem: Points item flag. **Bit 4 X'10'** IN_SepMinPerCpnORLinksToDep: For coupon item records, this is the *separate minimum per coupon* flag. For item records, this is the *links to deposit* flag. **Bit 3 X'08'** IN_UsrFlag1: User flag 1. **Bit 2 X'04'** IN_UsrFlag2: User flag 2. **Bit 1 X'02'** IN_UsrFlag3: User flag 3.

Field Name	Type	Length	Decimal Offset	Description
				Bit 0 X'01' IN_UsrFlag4: User flag 4.
UNION2	Union	1	9	Defines this byte as one of two possible fields, either INDICAT2 or STRUCTTYPEPM.
INDICAT2	PD	1	9	See the STRUCTTYPEPM field for a description of how each packed-decimal digit is mapped.
STRUCTTYPEPM	Structure	1	9	This is the mapping of the two packed-decimal digits in INDICAT2. This structure defines the ITEMTYPE and PRICINGMETHOD fields.
ITEMTYPE	PD	.5	9	0 - Normal item sale 1 - Deposit 2 - Refund 3 - Deposit return 4 - Misc. trans. receipt (sale) 5 - Misc. trans. payout (refund) 6 - Manufacturer coupon 7 - Store coupon 8 - Reserved 9 - Reserved
PRICINGMETHOD	PD	.5	9.5	0 - Unit or split package pricing 1 - Base + 1 pricing 2 - Group threshold pricing 3 - Reduced deal pricing **4** Increased deal pricing. For a coupon item record, this implies one of the following: • The value of the coupon is calculated to yield a net price that equals the value in the price field (Net Value coupon). • The value of the coupon is calculated based on the savings value multiplied by the number of matching items purchased, if the number of matching items required is met or exceeded (Multiple Items coupon). • A percent-off amount is calculated. **5** Pricing data is in another record. The item code for this record is in the six bytes starting at SALEQUANTITY.

Field Name	Type	Length	Decimal Offset	Description
				>5 Reserved. • Fake coupons use pricing method 3. • Percent-off link and percent-off validation coupons use pricing method 4.
DEPARTMENT	PD	2	10	Department number (value 1 - 999)
FAMILIES	Union	3	12	Defines these three bytes as one of two possible fields, either STRUCTFAM or STRUCTMISC.
STRUCTFAM	Structure	3	12	This is the active definition when the item is associated with a coupon family. This structure defines the COUPONFAMILY1 and COUPONFAMILY2 fields.
COUPONFAMILY1	PD	1.5	12	Current coupon family number. A coupon with an equal family number can be applied to this item. (Value 0-999).
COUPONFAMILY2	PD	1.5	13.5	Previous coupon family number. A coupon with an equal family number can be applied to this item. (Value 0-999).
STRUCTMISC	Structure	3	12	This is the active definition when the item is associated with a miscellaneous account. This structure defines the MISCSALEACCOUNT1 and MISCSALEACCOUNT2 fields.
MISCSALEACCOUNT1	PD	1.5	12	Miscellaneous account number from which the item amount is added if this is a miscellaneous item. (Value 0 - 999, where 0 implies no update).
MISCSALEACCOUNT2	PD	1.5	13.5	Miscellaneous account number from which the item amount is subtracted if this is a miscellaneous item. (Value 0 - 999, where 0 implies no update).
MPGROUP	PD	1	15	Multipricing group number / Mix-and-match number when a combination of items comprise a deal. (Value 0 - 99, where 0 = mix only with self).
UNIONPINFO	Union	6	16	Defines these six bytes as one of two possible fields, either ALIASITEMCODE or STRUCTPINFO.
ALIASITEMCODE	PD	6	16	This is the active definition when the item does not contain any pricing information because it is chained to an alias item that contains the price. This field specifies the item code of an alias item record.
STRUCTPINFO	Structure	6	16	This is the active definition for a normal sales item that contains its own pricing information.

Field Name	Type	Length	Decimal Offset	Description
				This structure defines `SALEQUANTITY` and `UNIONPRICE`.
`SALEQUANTITY`	PD	1	16	Count or weight of items in the deal. (Whole units only, no fractional weights) (Value 0 - 99, where 0 is equivalent to 1). For coupon item records without a weight limit, this field is both the maximum number (*x*) of this coupon per transaction and the number (*y*) of matching items required to validate this coupon according to the equation `(x*10) + y`. For coupon item records with a weight limit, this field is both the maximum number (*x*) of pounds to which this coupon applies and the number (*y*) of matching items required to validate this coupon according to the equation `(x*10) + y`. For these coupons, both the weight flag and the Coupon Weight Limit flag must be on. Multiple-item coupons have a transaction limit value of 0, which means unlimited, and use this field for the matching quantity limit. For a field value of 1-49, when the Big Coupon Limit flag is on, the field contains the number of matching items required. The maximum number of coupons per transaction is implied to be 1. For a field value of 50-99 when the Big Coupon Limit flag is on, the field represents the maximum number of coupons per transaction. The actual value for the maximum number of coupons is calculated by subtracting X'50' from the field value. For example, if this field contains X'59', then the maximum number of coupons per transaction is X'9'. When the maximum number of coupons is equal to or greater than 9, then the number of matching items required is implied to be 1.
`UNIONPRICE`	Union	5	17	Defines these five bytes as one of three possible fields, either `SALEPRICE`, `STRUCTMULTIPRICE`, or `Tare`.
`SALEPRICE`	PD	5	17	This is the active definition for pricing methods 0 or 1. For pricing method 0 or 1, this is the price in the format `00pppppppp` where: **pppppppp** Unit price if `SaleQuantity` = 0 or 1.

Field Name	Type	Length	Decimal Offset	Description
				pppppppp Deal price if `SaleQuantity` > 1. For coupon item records, this field uses the format `mmmmmabcde` where: **mmmmm** The manufacturer's number for validation, if manufacturer validation is selected. Otherwise, contains all zeroes. **abcde** The meaning depends on the type of coupon item record: • Coupon - `abcde` is the value of the coupon. • Fake coupon - `abcde` is always `00001`. • Percent-off link coupon - `ab` is always `99`, `cde` is the percent-off rate with one decimal place (cd.e). • Percent-off validation coupon - `ab` is always `98`, `cde` is the percent-off rate, including one decimal place (`cd.e`). • Points-multiply coupon - `abc` is always `960`, `de` is the percentage to multiply points, including one decimal place (`d.e`). • Multiple-item coupon - `ab` is always `95`, `cde` is the savings amount, which must be in the range 0-999.
`STRUCTMULTIPRICE`	Structure	5	17	This is the active definition for pricing methods 2, 3, and 4, and for coupon item records. This structure defines the `SALEPRICE1` and `SALEPRICE2` fields.
`SALEPRICE1`	PD	2.5	17	For pricing method 2, this field is the package deal price. For pricing method 3 or 4, this field is the unit price. For coupon item records, this field is the manufacturer's number for validation.
`SALEPRICE2`	PD	2.5	19.5	For pricing method 2, this field is the single unit price. For pricing method 3 or 4, this field is the reduced price. For coupon item records, this field is the value of the coupon.

Field Name	Type	Length	Decimal Offset	Description
`Tare`	Structure	5	17	This is the active definition when the item contains tare pricing information, which is stored in this field. The Tare field is enabled when using the split-package pricing method (pricing method 0), the *IN_WeightOrPriceRqd* flag is set, and the *Tare* option is enabled in Options -> Database personalization. This structure defines the `TareCodeValue` and `TareFiller` fields.
`TareCodeValue`	PD	2.5	17	The first five packed-decimal digits (length = 2.5) from offset 17 are the *TareCodeValue* field that stores a value. The value is either an ID (set in personalization), a fixed weight, or a percent (which is a percent of the total package weight that is subtracted from the item weight). Values of the Tare field can be: **1 - 99** Tare value = 1 through 99: A Tare ID defined in personalization that retrieves the weight or percent from the options file, eamoptns.dat. **100 - 9999** Erroneously entered values that result in operator message, *Check Item Data*. **1xxxx** The digits *xxxx* represent the tare weight in hundredths of a pound or in grams, depending on the locale. **2xxxx** The digits *xxxx* represent hundreds of a percent of the item weight.
`TareFiller`	PD	2.5	19.5	Reserved.
`LINKEDTO`	PD	2	22	Item code to be sold with the sale of this item. (Value 0-9999, 0 = no linked item)
`ITEMNAME`	CHR	18	24	Item descriptor to be printed on receipt (no commas allowed)
`USREXIT1`	Int	2	42	User field for use with extensions. This field can be the promotion code depending on the value of the *Use promotion code for coupon validation* option in Options ->Coupon personalization.
`USREXIT2`	Int	2	44	User field for use with extensions. For coupon item records, this is the *minimum purchase amount* for coupon to be accepted. For item records, this can be the date of last sale (DOLS) depending on the setting of the *Keep date of last sale in User Data 2* option in Database -> Totals

Field Name	Type	Length	Decimal Offset	Description
				personalization. The DOLS is updated at a store close that closes the Item Movement file.
The remainder of the 127-byte Item Record contains optional data. The options in Options -> Database personalization determine which fields are enabled and disabled. Fields and offsets shown in the remainder of this record are for the default record that Toshiba Global Commerce Solutions ships with SurePOS ACE.				
`RestrictedSaleType`	Int	1	46	Codes for alcohol=1 tobacco=2
`ReportingCode`	PD	1	47	Codes to break out reports by department.
`Indicator5`	Int	1	48	**Bit 7 X'80'** WICFlag: WIC flag **Bit 6 X'40'** BigCrossPromo: Big cross promo flag. **Bit 5 X'20'** BigCouponLimit: Big coupon limit flag. This flag is set when the `SALEQUANTITY` field represents one of the following: • Matching items required, and the value is equal to or greater than 10. • Maximum coupons per transaction, and the value is equal to or greater than 9. See the description of the `SALEQUANTITY` field (offset 16) for more information. **Bit 4 X'10'** CouponWeightLimit: Coupon weight limit. **Bit 3 X'08'** InStoreAddedItem: In-store added item flag. **Bit 2 X'04'** ValueCardState: Value card state. 0=activation, 1=reload. **Bit 1 X'02'** Reserved2: Reserved. **Bit 0 X'01'** Reserved3: Reserved. The bit 3 flag is set when a new item is added from Data Maintenance. It is turned off when an item has Auxiliary Pricing data active, which occurs when an item is added or overwritten from a host.

Field Name	Type	Length	Decimal Offset	Description
`Commodity`	PD	2	49	Groups items by commodity (for example, soaps) for reporting purposes. This field can be updated from the host. At the store, it can be updated only on when a new item is added, or when the in-store flag (offset 48, bit 3, X'08') is on.
`Subcommodity`	PD	3	51	More granularity for the commodity code at offset 49. It has the same purpose and update characteristics.
`MSINumber`	PD	4	54	The MSI is an order entry number that ranges from 0-99999. It can be updated from the host. At the store, it can be updated only when a new item is added, or if the in-store flag (offset 48, bit 3, X'08') is on.
`Cost`	Long	4	58	The wholesale cost of the item.
`LQD_LimitQty`	PD	1	62	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity sold at the reduced price. Other items are sold at the unit price.
`LQD_DealQty`	PD	1	63	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity at the reduced price when there is a multiple price. For example, `3` in a 3/$1.00 deal.
Host pricing data fields are a contiguous block of 10 bytes of data. The *Host Price Data* option in Options -> Database -> Item(1) personalization enables or disables the entire group of fields.				
`HostQuantity`	PD	1	64	`HostQuantity` and `HostSalePrice` must be contiguous because they are loaded from an external source directly into the item record.
`UNIONHOSTP`	Union	5	65	Defines these five bytes as one of two possible fields, either `HostSalePrice` or `StructHostMultiPrice`.
`HostSalePrice`	PD	5	65	This is the active definition for pricing method 0 or 1. For pricing method 0 or 1, the price is in format `00pppppppp` where: **pppppppp** Unit price if `HostQuantity` = 0 or 1 **pppppppp** Deal price if `HostQuantity` > 1
`StructHostMultiPrice`	Structure	5	65	This is the active definition when for pricing methods 2, 3, or 4. This structure defines the `HostSalePrice1` and `HostSalePrice2` fields.
`HostSalePrice1`	PD	2.5	65	For pricing method 2, this is the package deal price.

Field Name	Type	Length	Decimal Offset	Description
				For pricing method 3 or 4, this is the unit price.
`HostSalePrice2`	PD	2.5	67.5	For pricing method 2, this is the single unit price. For pricing method 3 or 4, this is the reduced price.
`HostLQDLimit`	PD	1	70	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity set by the host to be sold at the reduced price. Other items are sold at the unit price.
`HostLQDDeal`	PD	1	71	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity set by the host to be sold at the reduced price when there is a multiple price. For example, `3` in a 3/$1.00 deal.
`UNIONHostPricingMethod`	Union	1	72	Defines this byte as one of two possible fields, either `HostIndicator2` or `StructHostPricingMethod`.
`HostIndicator2`	PD	1	72	See the `StructHostPricingMethod` field for a description of how each packed decimal digit is mapped.
`StructHostPricingMethod`	Structure	1	72	This is the mapping of the two packed decimal digits in the `HostIndicator2` field. This structure defines the `HostItemType` and `HostPricingMethod` fields.
`HostItemType`	PD	.5	72	0 - Normal item sale 1 - Deposit 2 - Refund 3 - Deposit return 4 - Misc. trans. receipt (sale) 5 - Misc. trans. payout (refund) 6 - Manufacturer coupon 7 - Store coupon 8 - Reserved 9 - Reserved
`HostPricingMethod`	PD	.5	72.5	0 - Unit or split package pricing 1 - Base + 1 pricing 2 - Group threshold pricing 3 - Reduced deal pricing **4** Increased deal pricing. For a coupon item record, this implies the value of the coupon is calculated to yield a net value in the price field.

Field Name	Type	Lengt h	Decima l Offset	Description
				5 Pricing data is in another record. The item code for this record is in the six bytes starting at `HostQuantity`. **6-8** Reserved.
`InStorePriceChangeFlag`	Int	1	73	This flag is set to FALSE when `SALEQUAN`, `SALEPRIC`, and `INDICAT2` fields are set from *Host Price* fields (from a host source outside the store). When a price change is recognized as coming from inside the store (either from Data Maintenance or from the terminal), it is set to TRUE.
`CompINDICAT2$`	PD	1	74	Comparison price pricing method
`CompSALEQUAN$`	PD	1	75	Comparison price deal quantity
`UnionComparisonPrice`	Union	5	76	Defines these five bytes as one of three possible fields, either `StructCompPrice2Byte`, `StructCompPrice4Byte`, or `CompPricePD`.
`StructCompPrice2Byte`	Structure	5	76	This is the active union definition when the *Comparison Price Format* option in Options -> Database -> Item(4) -> Comparison Price personalization is enabled. This structure defines the `CompPrice2Byte` and `Filler1` fields.
`CompPrice2Byte`	Int	2	76	Comparison price, 2 byte format.
`Filler1`	ASCII	3	78	Reserved.
`StructCompPrice4Byte`	Structure	5	76	This is the active union definition when the *Comparison Price Format* option in Options -> Database -> Item(4) -> Comparison Price personalization is enabled. This structure defines the `CompPrice4Byte` and `Filler` fields.
`CompPrice4Byte`	Int	4	76	Comparison price, 4 byte format.
`Filler`	ASCII	1	80	Reserved.
`CompPricePD`	PD	5	76	Note: Comparison price, packed decimal format.
`CompSaleType`	Int	1	81	Note: Comparison sale pricing type.
`CompetitorIndex`	Int	1	82	Note: Comparison pricing competition index.
`Price.Date`	PD	3	83	Note: Date of comparison price.

Field Name	Type	Length	Decimal Offset	Description
`AlternateDiscountByDept`	PD	2	86	Includes items in a Discount by Department promotion without changing the department number of the item.
`VATCode`	Int	1	88	Defaults to offset 0 (off)
`UNIONHostPriceChangeLog`	Union	11	89	Defines these eleven bytes as one of two possible fields, either `HostPriceChangeLog` or `HostPriceChangeLogStruct`.
Host price change log fields are contained in the 11 bytes that begin at offset 89. The *Host Price Change Log* option in Options -> Database -> Item(1) personalization enables or disables the entire group of fields.				
`HostPriceChangeLog`	ASCII	11	89	This field is used by older code; newer code uses the *HostPriceChangeLogStruct* field.
`HostPriceChangeLogStruct`	Structure	11	89	This is the active union definition when the *Host Price Change Log* option in Options -> Database -> Item(1) personalization is enabled. This structure defines the `HostPriceReasonCode`, `HostPriceOperatorID`, and `HostPriceChangeDateTime` fields.
`HostPriceReasonCode`	Int	1	89	Reason code for host price.
`HostPriceOperatorID`	Int	5	90	Operator ID.
`HostPriceChangeDateTime`	Timestamp	5	95	Date and time of host price change.
`SecondaryPointsClub`	PD	3	100	Up to three 1–byte secondary club numbers. If number of clubs is 0, the item only contributes to primary points. The bytes for this field must be contiguous. To use more than three secondary club numbers, you must redefine the SAPATH file. See "Using SAPATH.RVT" on page 608.
`LargeLinkedTo`	PD	6	103	Large linked-to item code.
`ValueCardType`	Int	1	109	Value card type. 0 - Not a value card. 61 - Gift card (X'3D') 62 - Phone card (X'3E') 64 - Blackhawk card (X'3F')
`CouponLevel`	PD	1	110	Coupon level.
`CouponCoPay`	Int	2	111	Coupon co-pay amount.
`EASType`	Int	1	113	EAS type.
`Reserved4`	ASCII	13	114	Reserved.

The 127-Byte Item Record File - 14-digit Item Code

The number of item records is limited only by the available disk space.

Note for SA Users

See Table 1 for a description of how the SurePOS ACE file differs from the 4680-4690 Supermarket Application file.

Logical name	<EAMITM14>
Data object reference	*ISSAItemStorage* <SAITMSTR.CPP/HPP>
Organization	Keyed
Distribution class	Compound per update
File copies	1
Record length	127 to 508 bytes
Key length	7 bytes: Item code

Table 56. Layout of 127-byte Item Record - 14-Digit (GTIN) Item Code

Field Name	Type	Length	Decimal Offset	Description
`ITEMCODE`	PD	7	0	Item code - right justified with leading zeroes
`INDICAT0`	Int	1	7	**Bit 7 X'80'** IN_QuantityNotAllowed: Quantity not allowed. **Bit 6 X'40'** IN_WeightOrPriceRequired: Weight or price required since item is sold by weight. **Bit 5 X'20'** IN_QuantityRequired: Quantity required. **Bit 4 X'10'** IN_PriceRequired: Price required. **Bit 3 X'08'** IN_ExceptionLogged: Item sale is exception logged. **Bit 2 X'04'** IN_NotForSale: Item is not authorized for sale. **Bit 1 X'02'** IN_RestrictedSale: Item not authorized for sale during restricted hours. **Bit 0 X'01'** IN_NoMovement: No item movement kept.

Field Name	Type	Length	Decimal Offset	Description
INDICAT1	Int	1	8	**Bit 7 X'80'** IN_TaxableA: Tax plan A applies to item **Bit 6 X'40'** IN_TaxableB: Tax plan B applies to item **Bit 5 X'20'** IN_TaxableC: For coupon item records, this is the *minimum purchase by manufacturer* flag. For item records, this is the *Tax plan C applies to item* flag. **Bit 4 X'10'** IN_TaxableD: For coupon item records, this is the *minimum purchase by department* flag. For item records, this is the *Tax plan D applies to item* flag. **Bit 3 X'08'** IN_FoodStampable: Food stamp item **Bit 2 X'04'** IN_TradingStamps: For coupon item records, this is the *item excluded from purchase minimum* flag. For item records, this is the *points apply to item* flag. **Bit 1 X'02'** IN_NoDiscount: For points items, this is the *redemption item* flag. For other item records, this is the *non-discountable item* flag. **Bit 0 X'01'** IN_NoCouponMultiplication: Coupon value multiplication not allowed on this item.
INDICAT1A	Int	1	9	**Bit 7 X'80'** IN_NoShelfLabelPrint: Log to Change File flag. **Bit 6 X'40'** IN_FuelItem: Fuel item flag. **Bit 5 X'20'** IN_PointsItem: Points item flag. **Bit 4 X'10'** IN_SepMinPerCpnORLinksToDep: For coupon item records, this is the *separate minimum per coupon* flag. For item records, this is the *links to deposit* flag.

Field Name	Type	Length	Decimal Offset	Description
				Bit 3 X'08' IN_UsrFlag1: User flag 1. **Bit 2 X'04'** IN_UsrFlag2: User flag 2. **Bit 1 X'02'** IN_UsrFlag3: User flag 3. **Bit 0 X'01'** IN_UsrFlag4: User flag 4.
`UNION2`	Union	1	10	Defines this byte as one of two possible fields, either `INDICAT2` or `STRUCTTYPEPM`.
`INDICAT2`	PD	1	10	See the `STRUCTTYPEPM` field for a description of how each packed decimal digit is mapped.
`STRUCTTYPEPM`	Structure	1	10	This is the mapping of the two packed-decimal digits in `INDICAT2`. This structure defines the `ITEMTYPE` and `PRICINGMETHOD` fields.
`ITEMTYPE`	PD	.5	10	0 - Normal item sale 1 - Deposit 2 - Refund 3 - Deposit return 4 - Misc. trans. receipt (sale) 5 - Misc. trans. payout (refund) 6 - Manufacturer coupon 7 - Store coupon 8 - Reserved 9 - Reserved
`PRICINGMETHOD`	PD	.5	10.5	0 - Unit or split package pricing 1 - Base + 1 pricing 2 - Group threshold pricing 3 - Reduced deal pricing **4** Increased deal pricing. For a coupon item record, this implies one of the following: • The value of the coupon is calculated to yield a net price that equals the value in the price field (Net Value coupon). • The value of the coupon is calculated based on the savings value multiplied by the number of matching items purchased, if the number of matching items required

Field Name	Type	Length	Decimal Offset	Description
				is met or exceeded (Multiple Items coupon). • A percent-off amount is calculated. **5** Pricing data is in another record. The item code for this record is in the six bytes starting at `SALEQUANTITY`. **>5** Reserved. • Fake coupons use pricing method 3. • Percent-off link and percent-off validation coupons use pricing method 4.
`DEPARTMENT`	PD	2	11	Department number (value 1 - 999)
`FAMILIES`	Union	3	13	Defines these three bytes as one of two possible fields, either `STRUCTFAM` or `STRUCTMISC`.
`STRUCTFAM`	Structure	3	13	This is the active definition when the item is associated with a coupon family. This structure defines the `COUPONFAMILY1` and `COUPONFAMILY2` fields.
`COUPONFAMILY1`	PD	1.5	13	Current coupon family number. A coupon with an equal family number can be applied to this item. (Value 0-999).
`COUPONFAMILY2`	PD	1.5	14.5	Previous coupon family number. A coupon with an equal family number can be applied to this item. (Value 0-999).
`STRUCTMISC`	Structure	3	13	This is the active definition when the item is associated with a miscellaneous account. This structure defines the `MISCSALEACCOUNT1` and `MISCSALEACCOUNT2` fields.
`MISCSALEACCOUNT1`	PD	1.5	13	Miscellaneous account number from which the item amount is added if this is a miscellaneous item. (Value 0 - 999, where 0 implies no update).
`MISCSALEACCOUNT2`	PD	1.5	14.5	Miscellaneous account number from which the item amount is subtracted if this is a miscellaneous item. (Value 0 - 999, where 0 implies no update).
`MPGROUP`	PD	1	16	Multipricing group number / Mix-and-match number when a combination of items comprise a deal. (Value 0 - 99, where 0 = mix only with self).
`UNIONPINFO`	Union	7	17	Defines these seven bytes as one of two possible fields, either `ALIASITEMCODE` or `STRUCTPINFO`.
`ALIASITEMCODE`	PD	7	17	This is the active definition when the item does not contain any pricing information because it is chained to an alias item that contains the price. This field specifies the item code of an alias item record.

Field Name	Type	Length	Decimal Offset	Description
STRUCTPINFO	Structure	7	17	This is the active definition for a normal sales item that contains its own pricing information. This structure defines the SALEQUANTITY and UNIONPRICE fields.
Filler	ASCII	1	17	Reserved
SALEQUANTITY	PD	1	18	Count or weight of items in the deal. (Whole units only, no fractional weights) (Value 0 - 99, where 0 is equivalent to 1). For coupon item records without a weight limit, this field is both the maximum number (*x*) of this coupon per transaction and the number (*y*) of matching items required to validate this coupon according to the equation (x*10) + y. For coupon item records with a weight limit, this field is both the maximum number (*x*) of pounds to which this coupon applies and the number (*y*) of matching items required to validate this coupon according to the equation (x*10) + y. For these coupons, both the weight flag and the Coupon Weight Limit flag must be on. Multiple-item coupons have a transaction limit value of 0, which means unlimited, and use this field for the matching quantity limit. For a field value of 1-49, when the Big Coupon Limit flag is on, the field contains the number of matching items required. The maximum number of coupons per transaction is implied to be 1. For a field value of 50-99 when the Big Coupon Limit flag is on, the field represents the maximum number of coupons per transaction. The actual value for the maximum number of coupons is calculated by subtracting X'50' from the field value. For example, if this field contains X'59', then the maximum number of coupons per transaction is X'9'. When the maximum number of coupons is equal to or greater than 9, then the number of matching items required is implied to be 1.
UNIONPRICE	Union	5	19	Defines these five bytes as one of three possible fields, either SALEPRICE, STRUCTMULTIPRICE, or Tare.
SALEPRICE	PD	5	19	This is the active definition for pricing methods 0 or 1. For pricing method 0 or 1, this is the price in the format 00pppppppp where: **pppppppp** Unit price if SaleQuantity = 0 or 1.

Field Name	Type	Length	Decimal Offset	Description
				pppppppp Deal price if `SaleQuantity` > 1. For coupon item records, this field has the format `mmmmmabcde` where: **mmmmm** The manufacturer's number for validation, if manufacturer validation is selected. Otherwise, contains all zeroes. **abcde** The meaning depends on the type of coupon item record: • Coupon - `abcde` is the value of the coupon. • Fake coupon - `abcde` is always `00001`. • Percent-off link coupon - `ab` is always `99`, `cde` is the percent-off rate with one decimal place (cd.e). • Percent-off validation coupon - `ab` is always `98`, `cde` is the percent-off rate, including one decimal place (`cd.e`). • Points-multiply coupon - `abc` is always `960`, `de` is the percentage to multiply points, including one decimal place (`d.e`). • Multiple-item coupon - `ab` is always `95`, `cde` is the savings amount, which must be in the range 0-999.
`STRUCTMULTIPRICE`	Structure	5	19	This is the active definition for pricing methods 2, 3, and 4, and for coupon item records. This structure defines the `SALEPRICE1` and `SALEPRICE2` fields.
`SALEPRICE1`	PD	2.5	19	For pricing method 2, this field is the package deal price. For pricing method 3 or 4, this field is the unit price. For coupon item records, this field is the manufacturer's number for validation.
`SALEPRICE2`	PD	2.5	21.5	For pricing method 2, this field is the single unit price. For pricing method 3 or 4, this field is the reduced price. For coupon item records, this field is the value of the coupon.
`Tare`	Structure	5	19	This is the active definition when the item contains tare pricing information, which is stored in this field. The Tare field is enabled when using the split-package pricing method (pricing method 0), the

Field Name	Type	Length	Decimal Offset	Description
				IN_WeightOrPriceRqd flag is set, and the *Tare* option is enabled in Options -> Database personalization. This structure defines the `TareCodeValue` and `TareFiller` fields.
`TareCodeValue`	PD	2.5	19	The first five packed decimal digits (length = 2.5) from offset 19 are the *TareCodeValue* field that stores a value. The value is either an ID (set in personalization), a fixed weight, or a percent (which is a percent of the total package weight that is subtracted from the item weight). Values of the Tare field can be: **1 - 99** Tare value = 1 through 99: A tare ID defined in personalization that retrieves the weight or percent from the Options file, EAMOPTNS.DAT. **100 - 9999** Erroneously entered values that result in operator message, *Check Item Data*. **1xxxx** The digits *xxxx* represent the tare weight in hundredths of a pound or in grams, depending on the locale. **2xxxx** The digits *xxxx* represent hundreds of a percent of the item weight.
`TareFiller`	PD	2.5	21.5	Reserved.
`LINKEDTO`	PD	2	24	Item code to be sold with the sale of this item. (Value 0-9999, 0 = no linked item)
`ITEMNAME`	CHR	18	26	Item descriptor to be printed on receipt (no commas allowed)
`USREXIT1`	Int	2	44	User field for use with extensions. This field can be the promotion code depending on the value of the *Use promotion code for coupon validation* option in Options ->Coupon personalization.
`USREXIT2`	Int	2	46	User field for use with extensions. For coupon item records, this is the *minimum purchase amount* for coupon to be accepted. For item records, this can be the date of last sale (DOLS) depending on the setting of the *Keep date of last sale in User Data 2* option in Database -> Totals personalization. The DOLS is updated at a store close that closes the Item Movement file.
The remainder of the 127-byte Item Record contains optional data. The options in Options -> Database personalization determine which fields are enabled and disabled. Fields and offsets shown in the remainder of this record are for the default record that Toshiba Global Commerce Solutions ships with SurePOS ACE.				

Field Name	Type	Length	Decimal Offset	Description
`RestrictedSaleType`	Int	1	48	Codes for alcohol=1 tobacco=2
`ReportingCode`	PD	1	49	Codes to break out reports by department.
`Indicator5`	Int	1	50	**Bit 7 X'80'** WICFlag: WIC flag **Bit 6 X'40'** BigCrossPromo: Big cross promo flag. **Bit 5 X'20'** BigCouponLimit: Big coupon limit flag. This flag is set when the `SALEQUANTITY` field represents one of the following: • Matching items required, and the value is equal to or greater than 10. • Maximum coupons per transaction, and the value is equal to or greater than 9. See the description of the `SALEQUANTITY` field (offset 18) for more information. **Bit 4 X'10'** CouponWeightLimit: Coupon weight limit. **Bit 3 X'08'** InStoreAddedItem: In-store added item flag. **Bit 2 X'04'** ValueCardState: Value card state. 0=activation, 1=reload. **Bit 1 X'02'** Reserved2: Reserved. **Bit 0 X'01'** Reserved3: Reserved. The bit 3 flag is set when a new item is added from Data Maintenance. It is turned off when an item has Auxiliary Pricing data active, which occurs when an item is added or overwritten from a host.
`Commodity`	PD	2	51	Groups items by commodity (for example, soaps) for reporting purposes. This field can be updated from the host. At the store, it can be updated only on when a new item is added, or when the in-store flag (offset 50, bit 3, X'08') is on.
`Subcommodity`	PD	3	53	More granularity for the commodity code at offset 51. It has the same purpose and update characteristics.

Field Name	Type	Length	Decimal Offset	Description
`MSINumber`	PD	4	56	The MSI is an order entry number that ranges from 0-99999. It can be updated from the host. At the store, it can be updated only when a new item is added, or if the in-store flag (offset 50, bit 3, X'08') is on.
`Cost`	Long	4	60	The wholesale cost of the item.
`LQD_LimitQty`	PD	1	64	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity sold at the reduced price. Other items are sold at the unit price.
`LQD_DealQty`	PD	1	65	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity at the reduced price when there is a multiple price. For example, `3` in a 3/$1.00 deal.
Host pricing data fields are a contiguous block of 10 bytes of data. The *Host Price Data* option in Options -> Database -> Item(1) personalization enables or disables the entire group of fields.				
`HostQuantity`	PD	1	66	`HostQuantity` and `HostSalePrice` must be contiguous because they are loaded from an external source directly into the item record.
`UNIONHOSTP`	Union	5	67	Defines these five bytes as one of two possible fields, either `HostSalePrice` or `StructHostMultiPrice`.
`HostSalePrice`	PD	5	67	This is the active definition for pricing method 0 or 1. For pricing method 0 or 1, the price is in format `00pppppppp` where: **pppppppp** Unit price if `HostQuantity` = 0 or 1 **pppppppp** Deal price if `HostQuantity` > 1
`StructHostMultiPrice`	Structure	5	67	This is the active definition when for pricing methods 2, 3, or 4. This structure defines the `HostSalePrice1` and `HostSalePrice2` fields.
`HostSalePrice1`	PD	2.5	67	For pricing method 2, this is the package deal price. For pricing method 3 or 4, this is the unit price.
`HostSalePrice2`	PD	2.5	69.5	For pricing method 2, this is the single unit price. For pricing method 3 or 4, this is the reduced price.
`HostLQDLimit`	PD	1	72	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity set by the host to be sold at the reduced price. Other items are sold at the unit price.
`HostLQDDeal`	PD	1	73	For pricing method 4 and limited quantity discount (LQD) pricing, this is the quantity set by the host to

Field Name	Type	Length	Decimal Offset	Description
				be sold at the reduced price when there is a multiple price. For example, `3` in a 3/$1.00 deal.
`UNIONHostPricingMethod`	Union	1	74	Defines this byte as one of two possible fields, either `HostIndicator2` or `StructHostPricingMethod`.
`HostIndicator2`	PD	1	74	See the `StructHostPricingMethod` field for a description of how each packed decimal digit is mapped.
`StructHostPricingMethod`	Structure	1	74	This is the mapping of the two packed-decimal digits in the `HostIndicator2` field. This structure defines the `HostItemType` and `HostPricingMethod` fields.
`HostItemType`	PD	.5	74	0 - Normal item sale 1 - Deposit 2 - Refund 3 - Deposit return 4 - Misc. trans. receipt (sale) 5 - Misc. trans. payout (refund) 6 - Manufacturer coupon 7 - Store coupon 8 - Reserved 9 - Reserved
`HostPricingMethod`	PD	.5	74.5	0 - Unit or split package pricing 1 - Base + 1 pricing 2 - Group threshold pricing 3 - Reduced deal pricing **4** Increased deal pricing. For a coupon item record, this implies the value of the coupon is calculated to yield a net value in the price field. **5** Pricing data is in another record. The item code for this record is in the six bytes starting at `HostQuantity`. **6-8** Reserved.
`InStorePriceChangeFlag`	Int	1	75	This flag is set to FALSE when `SALEQUAN`, `SALEPRIC`, and `INDICAT2` fields are set from *Host Price* fields (from a host source outside the store). When a price change is recognized as coming from inside the store (either from Data Maintenance or from the terminal), it is set to TRUE.

Field Name	Type	Length	Decimal Offset	Description
`CompINDICAT2$`	PD	1	76	Comparison price pricing method
`CompSALEQUAN$`	PD	1	77	Comparison price deal quantity
`UnionComparisonPrice`	Union	5	78	Defines these five bytes as one of three possible fields: `StructCompPrice2Byte`, `StructCompPrice4Byte`, or `CompPricePD`.
`StructCompPrice2Byte`	Structure	5	78	This is the active union definition when the *Comparison Price Format* option in Options -> Database -> Item(4) -> Comparison Price personalization is enabled. This structure defines the `CompPrice2Byte` and `Filler1` fields.
`CompPrice2Byte`	Int	2	78	Comparison price, 2-byte format.
`Filler1`	ASCII	3	80	Reserved.
`StructCompPrice4Byte`	Structure	5	78	This is the active union definition when the *Comparison Price Format* option in Options -> Database -> Item(4) -> Comparison Price personalization is enabled. This structure defines the `CompPrice4Byte` and `Filler` fields.
`CompPrice4Byte`	Int	4	78	Comparison price, 4-byte format.
`Filler`	ASCII	1	82	Reserved.
`CompPricePD`	PD	5	78	Comparison price, packed-decimal format.
`CompSaleType`	Int	1	83	Comparison sale pricing type.
`CompetitorIndex`	Int	1	84	Comparison pricing competition index.
`Price.Date`	PD	3	85	Date of comparison price.
`AlternateDiscountByDept`	PD	2	88	Includes items in a Discount by Department promotion without changing the department number of the item.
`VATCode`	Int	1	90	Defaults to offset 0 (off)
`UNIONHostPriceChangeLog`	Union	11	91	Defines these eleven bytes as one of two possible fields, either `HostPriceChangeLog` or `HostPriceChangeLogStruct`.
Host price change log fields are contained in the 11 bytes that begin at offset 91. The *Host Price Change Log* option in Options -> Database -> Item(1) personalization enables or disables the entire group of fields.				
`HostPriceChangeLog`	ASCII	11	91	This field is used by older code; newer code uses the `HostPriceChangeLogStruct` field.
`HostPriceChangeLogStruct`	Structure	11	91	This is the active union definition when the *Host Price Change Log* option in Options -> Database -> Item(1) personalization is enabled. This structure defines the `HostPriceReasonCode`, `HostPriceOperatorID`, and `HostPriceChangeDateTime` fields.
`HostPriceReasonCode`	Int	1	91	Reason code for host price.

Field Name	Type	Length	Decimal Offset	Description
HostPriceOperatorID	Int	5	92	Operator ID.
HostPriceChangeDateTime	Timestamp	5	97	Date and time of host price change.
SecondaryPointsClub	PD	3	102	Up to three 1-byte secondary club numbers. If number of clubs is 0, the item only contributes to primary points. The bytes for this field must be contiguous. To use more than three secondary club numbers, you must redefine the SAPATH file. See "Using SAPATH.RVT" on page 608.
LargeLinkedTo	PD	7	105	Large linked-to item code.
ValueCardType	Int	1	112	Value card type. 0 - Not a value card. 61 - Gift card (X'3D') 62 - Phone card (X'3E') 63 - Blackhawk card (X'3F')
CouponLevel	PD	1	113	Coupon level.
CouponCoPay	Int	2	114	Coupon co-pay amount.
EASType	Int	1	116	EAS type.
Reserved4	ASCII	10	117	Reserved.

The 46-Byte Item Record File

The number of item records is limited only by available disk space.

Logical name	<EAMITEMR>
Data object reference	*ISSAItemStorage* <SAITMSTR.CPP/HPP>
Organization	Keyed
Distribution class	Compound per update
File copies	1
Record length	46 to 508 bytes
Key length	6 bytes: Item code

Table 57. Layout of 46-byte Item Record

Field Name	Type	Length	Decimal Offset	Description
ITEMCODE	PD	6	0	Item code - right justified with leading zeroes

Field Name	Type	Length	Decimal Offset	Description
INDICAT0	Int	1	6	**Bit 7 X'80'** IN_QuantityNotAllowed: Quantity not allowed. **Bit 6 X'40'** IN_WeightOrPriceRequired: Weight or price required since item is sold by weight. **Bit 5 X'20'** IN_QuantityRequired: Quantity required. **Bit 4 X'10'** IN_PriceRequired: Price required. **Bit 3 X'08'** IN_ExceptionLogged: Item sale is exception logged. **Bit 2 X'04'** IN_NotForSale: Item is not authorized for sale. **Bit 1 X'02'** IN_RestrictedSale: Item not authorized for sale during restricted hours. **Bit 0 X'01'** IN_NoMovement: No item movement kept.
INDICAT1	Int	1	7	**Bit 7 X'80'** IN_TaxableA: Tax plan A applies to item **Bit 6 X'40'** IN_TaxableB: Tax plan B applies to item **Bit 5 X'20'** IN_TaxableC: For coupon item records, this is the *minimum purchase by manufacturer* flag. For item records, this is the *Tax plan C applies to item* flag. **Bit 4 X'10'** IN_TaxableD: For coupon item records, this is the *minimum purchase by department* flag. For item records, this is the *Tax plan D applies to item* flag. **Bit 3 X'08'** IN_FoodStampable: Food stamp item **Bit 2 X'04'** IN_TradingStamps: For coupon item records, this is the *item excluded from purchase minimum* flag. For item records, this is the *points apply to item* flag.

Field Name	Type	Length	Decimal Offset	Description
				Bit 1 X'02' IN_NoDiscount: For points items, this is the *redemption item* flag. For other item records, this is the *non-discountable item* flag. **Bit 0 X'01'** IN_NoCouponMultiplication: Coupon value multiplication not allowed on this item.
INDICAT1A	Int	1	8	**Bit 7 X'80'** IN_NoShelfLabelPrint: Log to Change File flag. **Bit 6 X'40'** IN_Reserved1: Reserved. **Bit 5 X'20'** IN_PointsItem: Points item flag. **Bit 4 X'10'** IN_SepMinPerCpnORLinksToDep: For coupon item records, this is the *separate minimum per coupon* flag. For item records, this is the *links to deposit* flag. **Bit 3 X'08'** IN_UsrFlag1: User flag 1. **Bit 2 X'04'** IN_UsrFlag2: User flag 2. **Bit 1 X'02'** IN_UsrFlag3: User flag 3. **Bit 0 X'01'** IN_UsrFlag4: User flag 4.
UNION2	Union	1	9	Defines this byte as one of two possible fields, either `INDICAT2` or `STRUCTTYPEPM`.
INDICAT2	PD	1	9	See the `STRUCTTYPEPM` field for a description of how each packed decimal digit is mapped.
STRUCTTYPEPM	Structure	1	9	This is the mapping of the two packed-decimal digits in `INDICAT2`. This structure defines the `ITEMTYPE` and `PRICINGMETHOD` fields.
ITEMTYPE	PD	.5	9	0 - Normal item sale 1 - Deposit 2 - Refund 3 - Deposit return 4 - Misc. trans. receipt (sale) 5 - Misc. trans. payout (refund)

Field Name	Type	Length	Decimal Offset	Description
				6 - Manufacturer coupon 7 - Store coupon 8 - Reserved 9 - Reserved
PRICINGMETHOD	PD	.5	9.5	0 - Unit or split package pricing 1 - Base + 1 pricing 2 - Group threshold pricing 3 - Reduced deal pricing **4** Increased deal pricing. For a coupon item record, this implies the value of the coupon is calculated to yield a net value in the price field. **5** Pricing data is in another record. The item code for this record is in the six bytes starting at `SALEQUANTITY`. **>5** Reserved.
DEPARTMENT	PD	2	10	Department number (value 1 - 999)
FAMILIES	Union	3	12	Defines these three bytes as one of two possible fields, either `STRUCTFAM` or `STRUCTMISC`.
STRUCTFAM	Structure	3	12	This is the active definition when the item is associated with a coupon family. This structure defines the `COUPONFAMILY1` and `COUPONFAMILY2` fields.
COUPONFAMILY1	PD	1.5	12	Current coupon family number. A coupon with an equal family number can be applied to this item. (Value 0-999).
COUPONFAMILY2	PD	1.5	13.5	Previous coupon family number. A coupon with an equal family number can be applied to this item. (Value 0-999).
STRUCTMISC	Structure	3	12	This is the active definition when the item is associated with a miscellaneous account. This structure defines the `MISCSALEACCOUNT1` and `MISCSALEACCOUNT2` fields.
`MISCSALEACCOUNT1`	PD	1.5	12	Miscellaneous account number from which the item amount is added if this is a miscellaneous item. (Value 0 - 999, where 0 implies no update).
`MISCSALEACCOUNT2`	PD	1.5	13.5	Miscellaneous account number from which the item amount is subtracted if this is a miscellaneous item. (Value 0 - 999, where 0 implies no update).
MPGROUP	PD	1	15	Multipricing group number / Mix-and-match number when a combination of items comprise a deal. (Value 0 - 99, where 0 = mix only with self).

Field Name	Type	Length	Decimal Offset	Description
UNIONPINFO	Union	6	16	Defines these six bytes as one of two possible fields, either `ALIASITEMCODE` or `STRUCTPINFO`.
ALIASITEMCODE	PD	6	16	This is the active definition when the item does not contain any pricing information because it is chained to an alias item that contains the price. This field specifies the item code of an alias item record.
STRUCTPINFO	Structure	6	16	This is the active definition for a normal sales item that contains its own pricing information. This structure defines `SALEQUANTITY` and `UNIONPRICE` fields.
`SALEQUANTITY`	PD	1	16	Count or weight of items in the deal. (Whole units only, no fractional weights) (Value 0 - 99, where 0 is equivalent to 1). For coupon item records, this field is both the maximum number (*x*) of this coupon per transaction and the number (*y*) of matching items required to validate this coupon according to the equation `(x*10) + y.`
UNIONPRICE	Union	5	17	Defines these five bytes as one of three possible fields: `SALEPRICE`, `STRUCTMULTIPRICE`, or `Tare`.
SALEPRICE	PD	5	17	This is the active definition for pricing methods 0 or 1. For pricing method 0 or 1, this is the price in format 00pppppppp where: **pppppppp** Unit price if `SaleQuantity` = 0 or 1 **pppppppp** Deal price if `SaleQuantity` > 1
`STRUCTMULTIPRICE`	Structure	5	17	This is the active definition for pricing methods 2, 3, and 4, and for coupon item records. This structure defines the `SALEPRICE1` and `SALEPRICE2` fields.
SALEPRICE1	PD	2.5	17	For pricing method 2, this field is the package deal price. For pricing method 3 or 4, this field is the unit price. For coupon item records, this field is the manufacturer's number for validation and the value of the coupon.
SALEPRICE2	PD	2.5	19.5	For pricing method 2, this field is the single unit price. For pricing method 3 or 4, this field is the reduced price. For coupon item records, this field is the manufacturer's number for validation and the value of the coupon.
Tare	Structure	5	17	This is the active definition when the item contains tare pricing information, which is stored in this field. The Tare field is enabled when using the split-package pricing method (pricing method 0), the *IN_WeightOrPriceRqd* flag is set, and the *Tare* option is

Field Name	Type	Length	Decimal Offset	Description
				selected in Options -> Database personalization. This structure defines the `TareCodeValue` and `TareFiller` fields.
TareCodeValue	PD	2.5	17	The first five packed decimal digits (length = 2.5) from offset 17 are the *TareCodeValue* field that stores a value. The value is either an ID (set in personalization), a fixed weight, or a percent (which is a percent of the total package weight that is subtracted from the item weight). Values of the Tare field can be: **1 - 99** Tare value = 1 through 99: A tare ID defined in personalization that retrieves the weight or percent from the Options file, EAMOPTNS.DAT. **100 - 9999** Erroneously entered values that result in operator message, *Check Item Data*. **1xxxx** The digits *xxxx* represent the tare weight in hundredths of a pound or in grams, depending on the locale. **2xxxx** The digits *xxxx* represent hundreds of a percent of the item weight.
TareFiller	PD	2.5	19.5	Reserved.
LINKEDTO	PD	2	22	Item code to be sold with the sale of this item. (Value 0-9999, 0 = no linked item)
ITEMNAME	CHR	18	24	Item descriptor to be printed on receipt (no commas allowed).
USREXIT1	Int	2	42	User field for use with extensions. This field can be the promotion code depending on the value of the *Use promotion code for coupon validation* option in Options ->Coupon personalization.
USREXIT2	Int	2	44	User field for use with extensions. For coupon item records, this is the minimum purchase amount for coupon to be accepted. For item records, this can be the date of last sale (DOLS) depending on the setting of the *Keep date of last sale in User Data 2* option in Database -> Totals personalization. The DOLS is updated at a store close that closes the Item Movement file.

EAMMAINT (ADDMI Maintenance File)

In order to support an item file with records greater than 69 bytes, SurePOS ACE supports the EAMMAINT record formats that are described in the sections that follow. These record formats are required to address item records that are greater than 69 bytes in length. Although these record formats are the default for SurePOS ACE, they are optional for those who are migrating from the 4680-4690 Supermarket Application. SurePOS ACE also supports the record formats supported by the 4680-4690 Supermarket Application.

The Alternate Delayed Data Maintenance Interface (ADDMI) Maintenance File contains the commands and associated data required to create a Delayed Data Maintenance batch. This file contains five types of records:

- "Batch Header Record" on page 302
- "Data Record" on page 304
- "Generic Record 1 - Item Code Range" on page 305
- "Generic Record 2 - Department Group" on page 307
- "Generic Record 3 - Multipricing Group" on page 308

Each batch file must begin with a batch header record. The data and generic records that follow are associated with this batch header record.

Records are delimited by `CR/LF X'0D0A'` but contain no internal field delimiters.

Note for SA Users

See Table 1 for a description of how the SurePOS ACE file differs from the 4680-4690 Supermarket Application file.

Logical name	<EAMMAINT>
Data object reference	• Apply immediate maintenance, *ISDDMAddmiData* <`DDMADDAT.CPP`> • ADDMI scheduled maintenance, *DDMAddmiStorage* <`DDMADSTR.CPP`>
Organization	Variable sequential
Distribution class	Mirrored on Close
File copies	1
Record length	Variable

ADDMI will log the following selected errors into the Delayed Maintenance Error file:

07	Record to be added already exists. Will not replace.
09	Record to be added already exists. Will replace record.
24	Record to be replaced not found. Add done.
25	Record to be deleted not found.
32	Error writing batch record to item record change file.
33	Error writing data record to item record change file.
53	Error opening the item record change file. No updates made.
94	Add caused replace but used old price and/or department.
95	Invalid ADDMI input file discarded.
96	Invalid ADDMI batch discarded.
97	Invalid ADDMI record discarded.
98	Invalid department number found. Update was discarded.
99	Record to be modified was not on file. Update was discarded.

When ADDMI updates item records, it also updates the Item Record Change File and the Exception Log, just as Delayed Data Maintenance would have done if it had processed the updates.

ADDMI source data records and control records will remain for the number of days specified by the *Retention Days* parameter in the `aceschdl.ini` file. After the number of retention days expires for any batch that has completed without error, the source file and control record are erased. Batches that end in error will remain until the user cleans them up.

Batch Header Record

Batch header records denote the beginning of a group of maintenance changes that are to be executed as a single batch.

Table 58. EAMMAINT Batch Header Record - 12-digit Item Code

Field Name	Type	Length	Decimal Offset	Description
HeaderId	PD	6	0	X'FFFFFFFFFF11' for header record
BatchId	PD	3	6	Batch number
Flags1	Int	1	9	Flag byte 1. See "Flags1 - Flag byte 1" on page 303 for the values.
Flags2	Int	1	10	Flag byte 2. See "Flags2 - Flag byte 2" on page 303 for the values.
MessageData	ASCII	v?	11	0 to 30,000 ASCII characters to be displayed in the store upon request of the operator. Note that the line length is 30 characters.

Table 59. EAMMAINT Batch Header Record - 14-digit (GTIN) Item Code

Field Name	Type	Length	Decimal Offset	Description
HeaderId	PD	6	0	X'FFFFFFFFFF14' for header record
BatchId	PD	3	6	Batch number
Flags1	Int	1	9	Flag byte 1. See "Flags1 - Flag byte 1" on page 303 for the values.
Flags2	Int	1	10	Flag byte 2. See "Flags2 - Flag byte 2" on page 303 for the values.
MessageData	ASCII	v?	11	0 to 30,000 ASCII characters to be displayed in the store upon request of the operator. Note that the line length is 30 characters.

If none or more than one of the maintenance type flags are on, the data will be batched for operator maintenance. If the last byte of the header ID is zeroed (HEADERID = X'FFFFFFFFFF00'), the batch is discarded with no error logged. This ability to cause a batch to be ignored is included so that batches can be deleted from the input (by a user preprocessor) without having to compress the input file.

Flags1 - Flag byte 1

Bit 0 X'80'
Reserved

Bit 1 X'40'
Operator-controlled maintenance

Bit 2 X'20'
Timer-controlled maintenance

Bit 3 X'10'
Immediate maintenance

Bit 4 X'08'
Report records in batch

Bit 5 X'04'
Do not validate department numbers

Bit 6 X'02'
Do not print/display source data in store

Bit 7 X'01'
Do not write changed item codes to IR change file.

Flags2 - Flag byte 2

Bit 0 X'80'
Not used

Bit 1 X'40'
Full 46-byte records are treated as replaces not adds

Bit 2 X'20'
Replace can do an add (with log)

Bit 3 X'10'
Add can do a replace

Bit 4 X'08'
Not used

Bit 5 X'04'
Not used

Bit 6 X'02'
If add does a replace log as error

Bit 7 X'01'
Unused

Data Record

Data records contain the information required to update, add, delete, or report a single item record.

Table 60. EAMMAINT Data Record - 12-digit Item Code

Field Name	Type	Length	Decimal Offset	Description
ItemCode	PD	6	0	Item code of the record.
Offset1	Int	1	6	Offset for Data1.
Length1	Int	1	7	Length of Data1.
Flags1	Int	1	8	Flags for Data1. See "Flagsn - Flags for DATAn" on page 305 for the values.
Data1	Mixed	v255	9	Item record data as it exists on 4680 or 4690. Zero bytes are valid for a delete or report request.
Offset*n*	Int	1	v?	Offset for Data*n*.
Length*n*	Int	1	v?	Length of Data*n*.
Flags*n*	Int	1	v?	Flags for Data*n*. See "Flagsn - Flags for DATAn" on page 305 for the values.
Data*n*	Mixed	v255	v?	Item record data.

Table 61. EAMMAINT Data Record - 14-digit (GTIN) Item Code

Field Name	Type	Length	Decimal Offset	Description
ItemCode	PD	7	0	Item code of the record.
Offset1	Int	1	7	Offset for Data1.
Length1	Int	1	8	Length of Data1.
Flags1	Int	1	9	Flags for Data1. See "Flagsn - Flags for DATAn" on page 305 for the values.
Data1	Mixed	v255	10	Item record data as it exists on 4680 or 4690. Zero bytes are valid for a delete or report request.
Offset*n*	Int	1	v?	Offset for Data*n*.
Length*n*	Int	1	v?	Length of Data*n*.
Flags*n*	Int	1	v?	Flags for Data*n*. See "Flagsn - Flags for DATAn" on page 305 for the values.

Field Name	Type	Length	Decimal Offset	Description
Data*n*	Mixed	v255	v?	Item record data.

Flags*n* - Flags for DATAn

Bit 0 X'80'
Delete this record.

Bit 1 X'40'
Report this record.

Bit 2 X'20'
Flags to turn OFF provided

Bit 3 X'10'
Flags to turn ON provided

Bit 4 X'08'
Reserved

Bit 5 X'04'
Reserved

Bit 6 X'02'
Unused

Bit 7 X'01'
Unused

Generic Record 1 - Item Code Range

Generic records contain the information required to update, add, delete, or report a group of item records. The data in these records is like the data in the data records except for the additional data required to define the group.

Table 62. EAMMAINT Generic Record 1 - 12-digit Item Code

Field Name	Type	Length	Decimal Offset	Description
HeaderId	PD	6	0	X'EEEEEEEEEEEE' for a generic record type 1.
ItemCodeLow	PD	6	6	Low end of item code range.
ItemCodeHigh	PD	6	12	High end of item code range.
Offset1	Int	1	18	Offset for Data1.
Length1	Int	1	19	Length of Data1.
Flags1	Int	1	20	Flags for Data1. See "Flagsn - Flags for DATAn" on page 306 for the values.

Field Name	Type	Length	Decimal Offset	Description
Data1	Mixed	v255	21	Item record data as it exists on 4680 or 4690.
Offset*n*	Int	1	v?	Offset for Data*n*.
Length*n*	Int	1	v?	Length of Data*n*.
Flags*n*	Int	1	v?	Flags for Data*n*. See "Flagsn - Flags for DATAn" on page 306 for the values.
Data*n*	Mixed	v255	v?	Item record data.

Table 63. EAMMAINT Generic Record 1 - 14-digit (GTIN) Item Code

Field Name	Type	Length	Decimal Offset	Description
HeaderId	PD	6	0	X'EEEEEEEEEEEE' for a generic record type 1.
ItemCodeLow	PD	7	6	Low end of item code range.
ItemCodeHigh	PD	7	13	High end of item code range.
Offset1	Int	1	20	Offset for Data1.
Length1	Int	1	21	Length of Data1.
Flags1	Int	1	22	Flags for Data1. See "Flagsn - Flags for DATAn" on page 306 for the values.
Data1	Mixed	v255	23	Item record data as it exists on 4680 or 4690.
Offset*n*	Int	1	v?	Offset for Data*n*.
Length*n*	Int	1	v?	Length of Data*n*.
Flags*n*	Int	1	v?	Flags for Data*n*. See "Flagsn - Flags for DATAn" on page 306 for the values.
Data*n*	Mixed	v255	v?	Item record data.

Flags*n* - Flags for DATAn

Bit 0 X'80'
Delete this record.

Bit 1 X'40'
Report this record.

Bit 2 X'20'
Flags to turn OFF provided.

Bit 3 X'10'
Flags to turn ON provided.

Bit 4 X'08'
Reserved.

Bit 5 X'04'

Reserved.

Bit 6 X'02'

Unused.

Bit 7 X'01'

Unused.

To search for generic records requires about 1 second for every 1000 items of item record capacity. At most, 5000 items will be processed in a single group. Additional items will be omitted.

Generic record type 1 requires that all records in the group have an item code greater than or equal to the low item code provided and less than or equal to the high item code provided. If both the low and high item codes are zero and this is a report request, then all item records listed in the item record change file will be reported.

On a request to report all changed item records, the delete flag tells the application to clear the item record change file after successful processing. If the delete flag is not set, then a harmless dummy record is written to the file. If this record is found in the same place at the next *report all changes* request, then only changes following this dummy record are reported.

If either flag is set in the length byte of a request to report all changed items, then an item record with item code zero precedes all changed items. This record is all zeros except for the final 22 characters (descriptor and user exits), which contain `"XXX,XXX ITEMS ON FILE"` where `XXX,XXX` is the total number of records in the item record file. A similar record follows all changed items to detail the number of changed records reported.

Generic Record 2 - Department Group

Generic records contain the information required to update, add, delete, or report a group of item records. The data in these records is like the data in the data records except for the additional data required to define the group.

Field Name	Type	Length	Decimal Offset	Description
HeaderId	PD	6	0	X'DDDDDDDDDDDD' for a generic record type 2.
DepartmentId	PD	2	6	Department number.
MPGroup	PD	1	8	Multipricing group.
Offset1	Int	1	9	Offset for Data1.
Length1	Int	1	10	Length of Data1.
Flags1	Int	1	11	Flags for Data1. See "Flagsn - Flags for Datan" on page 308 for the values.
Data1	Mixed	v255	12	Item record data as it exists on 4680 or 4690.
Offset*n*	Int	1	v?	Offset for Data*n*.
Length*n*	Int	1	v?	Length of Data*n*.
Flags*n*	Int	1	v?	Flags for Data*n*. See "Flagsn - Flags for Datan" on page 308 for the values.

Field Name	Type	Length	Decimal Offset	Description
Data*n*	Mixed	v255	v?	Item record data as it exists on 4680 or 4690.

Flags*n* - Flags for Data*n*

Bit 0 X'80'
Delete this record.

Bit 1 X'40'
Report this record.

Bit 2 X'20'
Flags to turn OFF provided.

Bit 3 X'10'
Flags to turn ON provided.

Bit 4 X'08'
Reserved.

Bit 5 X'04'
Reserved.

Bit 6 X'02'
Unused.

Bit 7 X'01'
Unused.

To search for generic records requires about 1 second for every 1000 items of item record capacity. At most, 5000 items will be processed in a single group. Additional items will be omitted.

Generic record type 2 requires that all records in the group have the same department and multipricing group number. If the provided multipricing group number is zero, then all items in the department are in the generic group. Note that the multipricing group field is not used for unit priced items.

Generic Record 3 - Multipricing Group

Generic records contain the information required to update, add, delete, or report a group of item records. The data in these records is like the data in the data records except for the additional data required to define the group.

Field Name	Type	Length	Decimal Offset	Description
HeaderId	PD	6	0	X'CCCCCCCCCCCC' for a generic record type 3.
Indicat2	PD	1	6	Type and pricing method.

Field Name	Type	Length	Decimal Offset	Description
DepartmentId	PD	2	6	Department number.
MPGroup	PD	1	9	Multipricing group.
SaleQuantity	PD	1	10	Deal quantity.
SalePrice	PD	5	11	Sale price.
Offset1	Int	1	16	Offset for Data1.
Length1	Int	1	17	Length of Data1.
Flags1	Int	1	18	Flags for Data1. See "Flagsn - Flags for Datan" on page 309 for the values.
Data1	Mixed	v255	19	Item record data as it exists on 4680 or 4690.
Offset*n*	Int	1	v?	Offset for Data*n*.
Length*n*	Int	1	v?	Length of Data*n*.
Flags*n*	Int	1	v?	Flags for Data*n*. See "Flagsn - Flags for Datan" on page 309 for the values.
Data*n*	Mixed	v255	v?	Item record data as it exists on 4680 or 4690.

Flags*n* - Flags for Data*n*

Bit 0 X'80'
Delete this record.

Bit 1 X'40'
Report this record.

Bit 2 X'20'
Flags to turn OFF provided.

Bit 3 X'10'
Flags to turn ON provided.

Bit 4 X'08'
Reserved.

Bit 5 X'04'
Reserved.

Bit 6 X'02'
Unused.

Bit 7 X'01'
Unused.

To search for generic records requires about 1 second for every 1000 items of item record capacity. At most, 5000 items will be processed in a single group. Additional items will be omitted.

Generic record type 3 requires that all records in the group have the same department, price, and type. Therefore, all fields provided to define the group must be identical for all items in the group.

EAMMTR* (Miscellaneous Transactions File)

The Miscellaneous Transaction file is a keyed file that contains totals data for miscellaneous accounts. A separate file exists for the current period, previous period, and one old period. A record exists for each account referenced by the controller miscellaneous transaction procedure or by the sale or refund of a miscellaneous transaction item.

Miscellaneous Transaction totals can be closed *short* or *long*, where a long close includes the totals from several short closes. This allows for daily totals as well as weekly totals. A separate file exists for the current period, previous period, and one old period. Therefore, SurePOS ACE maintains up to six Miscellaneous Transaction totals files.

All Miscellaneous Transaction files and records are created and maintained by the applications. An image version of each of the Miscellaneous Transaction files is kept on the alternate file server. This distribution takes place only when the file is closed, to minimize the miscellaneous transaction impact.

Logical name	<EAMMTRNC>, <EAMMTRNP>, <EAMMTRNO>, <EAMMTRCL>, <EAMMTRPL>, <EAMMTROL>, <EAMMTRWK>
Data object reference	*ISSAMiscTransactionAccountData* <SAMTRDAT.CPP>
Organization	Keyed
Distribution class	Mirrored at close
File copies	Current long, Previous long, Old long, Current short, Previous short, Old short, Weekly
Record length	36 bytes
Key length	2 bytes: Account

Store Record

Field Name	Type	Length	Decimal Offset	Description
Account	PD	2	0	Miscellaneous Account = X'9999'.
DateTime	PD	5	2	Date and Time when current period started or previous period ended. Format: YYMMDDHHmm.
Reserved	ASCII	21	7	Reserved.
UsrExit	ASCII	8	28	Reserved for user extensions.

Account Record

Field Name	Type	Length	Decimal Offset	Description
Account	PD	2	0	Miscellaneous Account (1-999).
Restart	Int	4	2	This field is used as a control field on a system restart. It is used to determine if this record has been updated by a particular transaction.
Amount	Int	4	6	Amount for the account.
Name	ASCII	18	10	Account descriptor.
UsrExit	ASCII	8	28	Reserved for user extensions.

EAMOPERA (Operator Authorization File)

The Operator Authorization file is a keyed file that controls the procedures an operator can perform. Any operator with a record in the file can sign on to the system; however, an operator can run only those procedures for which authorization is provided.

A single record exists in the file for each operator who is authorized to use the system. An image version of the file is maintained on each eligible controller so that authorization information is always accessible to any online terminal. The prime version of the file is on the master controller.

Operator numbers greater than 9999999990 are reserved for models and are not allowed to sign on to a terminal. The authorization data from record 9999999991 (model 1) is used as default authorization data.

Operator authorization is required to open the cash drawer using a no-sale procedure unless it is the first procedure after an operator sign-on and the operator is authorized for customer checkout.

Note for SA Users

See Table 1 for a description of how the SurePOS ACE file differs from the 4680-4690 Supermarket Application file.

Table 64. Logical name: EAMOPERA

Logical name: EAMOPERA

Logical name	<EAMOPERA>
Data object reference	*ISSAOperatorStorage* <`SAOPRSTR.CPP`>, *ISSAOperatorAuthorizationData* <`SAOPADAT.CPP`>
Organization	Keyed
Distribution class	Compound per update
File copies	1
Record length	84 bytes
Key length	5 bytes: Operator ID

In the following table, INDICAT0 through INDICAT15 contain boolean bit-flags that each indicate whether the operator is authorized for a particular transaction or procedure. If the bit value is 1, the operator is authorized. If the bit value is 0, the operator is not authorized.

Table 65. EAMOPERA (boolean bit-flags)

Field Name	Type	Length	Decimal Offset	Description
OperatorId	PD	5	0	Operator ID (only nine digits can be keyed).
Password	PD	4	5	Operator password (security code). If simple passwords are used at the controller or terminal, this field can contain either an encrypted or unencrypted packed decimal password. If enhanced passwords are used at both the controller and terminal, this field will contain all nulls.
OptionsLevel	PD	1	9	Operator options authorization level for performing personalization.
AuthUnion	Union	18	10	Defines these 18 bytes as one of three possible fields: `DetailAuth`, `RawAuthStruct`, or a series of indicator bits, `Indicat0` through `Indicat15`.
DetailAuth	Structure	18	10	Defines `AuthorizationFlags` and `ControllerAuthorization`, which follow:
AuthorizationFlags	Bit Array	32 x 1 bit	10	Various authorization flags as defined in `Indicat0` and `Indicat1`.
ControllerAuthorization	Int	14	14	Various authorization flags as defined in `Indicat2` through `Indicat15`.
RawAuthStruct	Structure	18	10	Defines the `RawAuth` array.
RawAuth	Int Array	9x2	10	Various authorization flags as defined in `Indicat0` through `Indicat15`, which follow.
The next 16 fields are bit flag controls:				
Indicat0	Bit Array	16 x 1 bit	10	X'8000' - Checkout transactions allowed X'4000' - Tender cashing allowed X'2000' - Tender exchange allowed X'1000' - Loan transaction allowed X'0800' - Pickup transaction allowed X'0400' - Tender Listing Allowed X'0200' - Price verification allowed X'0100' - Operator training allowed X'0080' - Terminal transfer allowed X'0040' - Terminal monitor allowed X'0020' - Tender count allowed X'0010' - Return item allowed X'0008' - WIC Transaction allowed X'0004' - Department Totals Report allowed X'0002' - User Non-sales 1 allowed X'0001' - User Non-sales 2 allowed
Indicat1	Bit Array	16 x 1 bit	12	X'8000' - No-sale open cash drawer allowed X'4000' - No-sale tender removal allowed

Field Name	Type	Length	Decimal Offset	Description
				X'2000' - No-sale till exchange allowed X'1000' - No-sale tender verify allowed X'0800' - No-sale till report allowed X'0400' - Refunds allowed X'0200' - Discounts allowed X'0100' - Misc. item payouts allowed X'0080' - Immediate I.R. changes allowed X'0040' - Delayed I.R. changes allowed X'0020' - More than price changes allowed X'0010' - No-sale price verify allowed X'0008' - Manager's Procedures allowed X'0004' - Reprint Tender Receipt allowed X'0002' - User function 1 allowed X'0001' - Front End Cashier allowed
Indicat2	Bit Array	8 x 1 bit	14	Bit 7 X'80' - Accounting Support Bit 6 X'40' - Reports Bit 5 X'20' - Delayed Maintenance Facility Bit 4 X'10' - Data Maintenance Bit 3 X'08' - Shelf Label Print Bit 2 X'04' - Application Personalization Bit 1 X'02' - Terminal Monitor Bit 0 X'01' - User Procedures
Indicat3	Bit Array	8 x 1 bit	15	These flags control accounting support: Bit 7 X'80' - Loans Bit 6 X'40' - Pickups Bit 5 X'20' - Tender Count Bit 4 X'10' - Transfer Tender Bit 3 X'08' - Carry Forward Office Tender Bit 2 X'04' - Record Misc. Transactions Bit 1 X'02' - Close Reporting Totals Bit 0 X'01' - Start/ Stop EPS
Indicat4	Bit Array	8 x 1 bit	16	These flags control reports: Bit 7 X'80' - Automatic Report Procedure Bit 6 X'40' - Oper/Term/Office Cash Report Bit 5 X'20' - Operator Sales Report Bit 4 X'10' - Store Totals Recap Report Bit 3 X'08' - Operator Performance Report Bit 2 X'04' - Over/Short Report Bit 1 X'02' - Cash Drawer Position Report Bit 0 X'01' - Tender Listing Report
Indicat5	Bit Array	8 x 1 bit	17	Bit 7 X'80' - Department Totals Report Bit 6 X'40' - Item Movement Report Bit 5 X'20' - Misc. Transaction Recap Report Bit 4 X'10' - Terminal Productivity Report Bit 3 X'08' - Operator Authorization Report Bit 2 X'04' - System Status Report Bit 1 X'02' - Print Filed Reports Bit 0 X'01' - User Reports

Field Name	Type	Length	Decimal Offset	Description
Indicat6	Bit Array	8 x 1 bit	18	Bit 7 X'80' - Manager's Audit Trail Report Bit 6 X'40' - Reset Item Movement Bit 5 X'20' - Exception Log Report Bit 4 X'10' - Transaction Log Report Bit 3 X'08' - Negative Sales Report Bit 2 X'04' - Item Sales Exception Report Bit 1 X'02' - Customer Audit Report Bit 0 X'01' - Customer Exception Report
Indicat7	Bit Array	8 x 1 bit	19	Bit 7 X'80' - User Report 1 Bit 6 X'40' - User Report 2 Bit 5 X'20' - User Report 3 Bit 4 X'10' - User Report 4 Bit 3 X'08' - User Report 5 Bit 2 X'04' - User Report 6 Bit 1 X'02' - User Report 7 Bit 0 X'01' - User Report 8
Indicat8	Bit Array	8 x 1 bit	20	These flags control Delayed Data Maintenance: X'80' - Apply Batches X'40' - Reserved X'20' - Report DDM Error Log X'10' - Reserved X'08' - Reserved X'04' - Create Batch Controls X'02' - Display Batch Controls X'01' - Reset DDM Error Log
Indicat9	Bit Array	8 x 1 bit	21	Bit 7 X'80' - Set Price Required Flag Bit 6 X'40' - Display / Erase Host Messages Bit 5 X'20' - Erase Batch Controls Bit 4 X'10' - Modify Batch Controls Bit 3 X'08' - Display / Print Batch Data Entries Bit 2 X'04' - Modify Batch Data Entries Bit 1 X'02' - Add / Erase Batch Data Entries Bit 0 X'01' - SUREPOS ACE EPS Reports
Indicat10	Bit Array	8 x 1 bit	22	These flags control Data Maintenance: Item Record Data Bit 7 X'80' - Add/change Bit 6 X'40' - Delete Bit 5 X'20' - Display/print Bit 4 X'10' - Change item price Tender Verification Data Bit 3 X'08' - Add/change Bit 2 X'04' - Delete Bit 1 X'02' - Display/print Bit 0 X'01' - Coupon Limit
Indicat11	Bit Array	8 x 1 bit	23	Operator Authorization Data Bit 7 X'80' - Add/change

Field Name	Type	Length	Decimal Offset	Description
				Bit 6 X'40' - Delete Bit 5 X'20' - Display/print Bit 4 X'10' - Customer Maintenance Item Movement Lists Bit 3 X'08' - Add/change Bit 2 X'04' - Delete Bit 1 X'02' - Display/print Bit 0 X'01' - Coupon Maintenance
Indicat12	Bit Array	8 x 1 bit	24	Options Authorization Lists Bit 7 X'80' - Add/change Bit 6 X'40' - Delete Bit 5 X'20' - Display/print Bit 4 X'10' - Misc. Account Maintenance Manager's Messages Bit 3 X'08' - Add/change Bit 2 X'04' - Delete Bit 1 X'02' - Display/print Bit 0 X'01' - Matrix Keyboard
Indicat13	Bit Array	8 x 1 bit	25	Bit 7 X'80' - Report list allowed Bit 6 X'40' - Reserved Bit 5 X'20' - Reserved Bit 4 X'10' - Reserved Bit 3 X'08' - Reserved Bit 2 X'04' - Reserved Bit 1 X'02' - Reserved Bit 0 X'01' - Reserved
Indicat14	Bit Array	8 x 1 bit	26	Bit 7 X'80' - Personalize Terminal/Group Options Bit 6 X'40' - Personalize Store Options Bit 5 X'20' - Personalize Tax Tables Bit 4 X'10' - Personalize Descriptors Bit 3 X'08' - Personalize User Procedures Bit 2 X'04' - Password Encrypted flag Bit 1 X'02' - Unattended Close Editor Bit 0 X'01' - Reserved Bit 2 (Password Encrypted flag) may be set when simple encrypted passwords are in use or may be set when enhanced passwords are in use at both the controller and the terminal.
Indicat15	Bit Array	8 x 1 bit	27	Bit 7 X'80' - User Procedure 1 Bit 6 X'40' - User Procedure 2 Bit 5 X'20' - User Procedure 3 Bit 4 X'10' - User Procedure 4 Bit 3 X'08' - User Procedure 5 Bit 2 X'04' - User Procedure 6 Bit 1 X'02' - User Procedure 7 Bit 0 X'01' - User Procedure 8

Field Name	Type	Length	Decimal Offset	Description
Indicat16	Bit Array	8 x 1 bit	28	Bit 7 X'80' - Manager's Journal Bit 6 X'40' - Electronic Journal Bit 5 X'20' - Journal Search Bit 4 X'10' - Journal Personalization Bit 3 X'08' - Personalization Bit 2 X'04' - Reserved Bit 1 X'02' - Item Record File Rebuild Bit 0 X'01' - Manage Files
PasswordChangeDat e	PD	3	29	The date of the last update to the password in format `YYMMDD`. This field is only maintained when changes are made to a simple password contained within this record. Changes to an operator's enhanced password are not recorded in this field.
Name	ASCII	20	32	Operator name.
Usr	ASCII	20	52	Reserved for user extensions.
OperatorBirthDate	PD	4	72	Operator date of birth in format `YYYYMMDD`.
Customer Reserved	Bit Array	1	76	Reserved for Customer refinement.
MigrationLevel	PD	1	77	Migration level of the record (set to 1)
Reserved	ASCII	6	78	Reserved for future use.

EAMOPnnn (Terminal Options File)

The random Terminal Options file contains personalization option values that are unique for the particular terminal whose number is appended to the name of the file. You can find information about terminal options in the description of personalization in the *SurePOS ACE: Planning and Installation Guide.*

This file is not updated directly but by the Personalization user interface, which is available from the SurePOS ACE Main Menu.

The prime version of this file is on the master controller. An image version is on all other eligible controllers.

Logical names	<EAMO:>nnn
Data object reference	*ISOptions* <OPTIONS.CPP/HPP>
Organization	Random
Distribution class	Compound on Close
File copies	1 per customized terminal, such as EAMOP001. for terminal 001
Record length	512 bytes

EAMOPGnn (Terminal Group Options File)

The random Terminal Group Options file contains personalization option values that are unique for the particular terminal group whose number is appended to the name of the file. You can find information about terminal group options in the description of personalization in the *SurePOS ACE: Planning and Installation Guide.*

This file is not updated directly but by the Personalization user interface, that is available from the SurePOS ACE Main Menu.

The prime version of this file is on the master controller. An image version is on all other eligible controllers.

Logical names	<EAMOG:>nn
Data object reference	*ISOptions* <OPTIONS.CPP/HPP>
Organization	Random
Distribution class	Compound
File copies	1 per customized terminal group, such as EAMOPG01 for terminal group 01
Record length	512 bytes

EAMOPTNS (Base Personalization Options File)

Operators use the SurePOS ACE Personalization interface (available from the SurePOS ACE Main Menu) to change option values in personalization .dat files.

Table 66. Companion Options INI and DAT Files

Options INI File	Options DAT File	Logical DAT File Name	Description
`optnparm.ini`	eamoptns.dat	<EAMOPTNS>	Base personalization options
`acecpntr.ini`	acecate5.dat	<ACECATE5>	Coupon translation options
`nlsparms.ini`	nlsoptns.dat	<NLSOPTNS>	National language support (NLS) personalization options
`apsparms.ini`	apsoptns.dat	<APSOPTNS>	SurePOS ACE EPS personalization options

Each personalization options .dat file is a random access file that contains values for all options declared in its companion `ini` file. SurePOS ACE creates each .dat file from its companion `ini` file if the .dat file does not exist.

For a description of options `ini` files, see “OPTNPARM (Options)” on page 534.

For a description of how to use DIF along with the ACEPERSL command, see “Using ACEPERSL to Maintain Personalization Options” on page 572.

For a description of how to enter user data in `ini` files, see “Tiered Overlay Files” on page 460.

Logical name	<EAMOPTNS>
Data object reference	*ISOptions* <OPTIONS.CPP/HPP>
Organization	Random access
Distribution class	Compound at close
File copies	1
Record length	512 bytes
User data	In the `INI` files

Table 67 contains a partial list of what occurs while the `LOADING OPTIONS` messages are displayed at a terminal. The options in nlsoptns.dat are loaded during static initialization.

See ISApplication2x20::construct() in 2xapplct.cpp for detailed information about the initial stage of options loading.

Table 67. Stages in Loading Options Data

Stage	Action
`LOADING OPTIONS DATA ACESDESC`	Load cached sales descriptors.
`LOADING OPTIONS DATA FORMAT FILES`	Load format files.
`LOADING OPTIONS DATA EAMOPTNS`	Read EAMOPTNS.DAT from disk into memory.
`LOADING OPTIONS DATA EAMOPnnn`	Read the terminal group or terminal specific options file, if one exists, and overlay the values that were read from EAMOPTNS.DAT. The *nnn* in EAMOPnnn is the terminal that is being loaded. The *nn* in EAMOPGnn is the terminal group number that is being loaded.
`LOADING OPTIONS DATA NLSOPTNS`	Read NLSOPTNS.DAT from disk into memory.
`LOADING OPTIONS DATA APSOPTNS`	Read APSOPTNS.DAT from disk into memory, if that file exists on the system.
`LOADING OPTIONS DATA ACECATE5`	Read ACECATE5.DAT from disk into memory.
`LOADING OPTIONS DATA EAMTAX:`	Load tax tables.
`LOADING OPTIONS DATA DATA FILES LOADED`	Reading options from disk has finished.
`LOADING OPTIONS DATA EXITING STANDBY`	This message is only seen when running a virtual terminal session that is in standby mode, and it indicates that the session is leaving standby mode and moving into failover mode.
`LOADING OPTIONS DATA INPUT DEVICES`	Get pointer to the IOP, MSR, and MICR.
`LOADING OPTIONS DATA OUTPUT DEVICES`	Get pointer to the printer, cash drawer, and display. Log `Terminal Application initialized` to the transaction log.
`LOADING OPTIONS DATA OPTIONS LOADED EVENT`	Raise the options loaded event.
`LOADING OPTIONS DATA TERMINAL INITIALIZED`	Initialization complete. Recover transaction, if one exists to recover.

Note:

1. Events will be raised for each of these stages. The events are documented in the optnsstr.hpp file (*ISOptionsStorage*), with enums defined for each stage.
2. An `Initialization` section has been added to the Descriptor Editor Sales Descriptors category in Personalization. This sections lists the messages that are displayed during ACE initialization.

EAMOSTAT (Operator Status File)

The Operator Status File is a keyed file that contains a record for each operator who has signed on to a terminal. The file and the records within it are created and managed by the applications.

The Operator Status File is a work file that the applications use to manage operator tills and limit operator sign-ons to a single terminal at a time. A record may exist for both the current period and previous period for each operator. The previous period records are used to print a previous period Cash Drawer Position Report. Previous period records are denoted by a 1 in the first character position of the operator number. This character is 0 for all current period operator numbers.

The Operator Status File resides on the file server controller. An image version exists on the alternate file server but it can be accessed only when the alternate is made the acting file server. The file is distributed to the alternate file server at each record update.

If the controller is down when an operator signs off from a terminal with operator accountability, an 0x13 string will be included in the Transaction Summary Log entry for the sign-off. The eamostat file will be missing transaction data until after the controller comes back online and Checkout Support catches up on its processing. If the sign-off data cannot be written to the Transaction Summary Log and a manager override is entered at the *Error writing transaction* prompt, then totals will be lost.

Because the eamostat file is unavailable while the master controller is down, you might encounter the following situations for an operator who signs off while the controller is down:

- An operator could incorrectly sign on to multiple terminals.
- An operator might need a manager override to sign on because the eamostat file is not yet aware of the signoff.

Logical name	<EAMOSTAT>
Data object reference	*ISSAOperatorStorage* <`SAOPRSTR.CPP`>
Organization	Keyed
Distribution class	Mirrored per update
File copies	1
Record length	56 bytes
Key length	5 bytes: OperatorID

Field Name	Type	Length	Decimal Offset	Description
Operator	PD	5	0	Operator number.
Terminal	PD	2	5	Terminal number most recently used by this operator.
NumLoans	Int	2	7	Number of loans to this till.
AmtLoans	Int	4	9	Amount of loans to this till.

Field Name	Type	Length	Decimal Offset	Description
NumPkups	Int	2	13	Number of pickups from this till.
AmtPkups	Int	4	15	Amount of pickups from this till.
TillContents	Array	8x4	19	This field represents the following eight 4-byte totals for till contents as of the end of the last transaction:
AmtCash	Int	4	19	Entered tender - cash.
AmtCheck	Int	4	23	Entered tender - checks.
AmtFoods	Int	4	27	Entered tender - food stamps.
AmtMisc1	Int	4	31	Entered tender - miscellaneous tender 1.
AmtMisc2	Int	4	35	Entered tender - miscellaneous tender 2.
AmtMisc3	Int	4	39	Entered tender - miscellaneous tender 3.
AmtManuf	Int	4	43	Entered tender - manufacturer coupons.
AmtStore	Int	4	47	Entered tender - store coupons.
Status	Int	1	51	True implies operator is currently signed-on. True = any nonzero value, False = 0.
Reserved	Int	4	52	Reserved

EAMPAN* (Preferred Customer Panel Diary File)

The Panel Diary function implements preferred customer transaction logging. It creates the eampanel.dat file by reformatting a subset of the Transaction Log. Transaction Log string types X'00' - X'11' are converted to ASCII, optionally reduced in size by filtering out fields (that you specify in Panel Filter files), and written to one or more Panel Diary files. The Panel Diary file contains all Transaction Log information for selected preferred customers. It can optionally contain sales information for all customers, regardless of whether they present preferred customer IDs. Select the *Collect panel data for all customers* option in Loyalty -> Activity -> Data Tracking personalization to collect data for all customers.

The *Add period summary string to panel file* option appends a summary string to the Panel Diary file.

SurePOS ACE ordinarily appends all Transaction Log data for points redemptions and adjustments to the Preferred Customer Audit Log. The *Append audit data to panel file* personalization option lets you also append the same data to the Panel Diary file. This combines all information about preferred customer points history in one file.

You can run the Panel Diary function automatically after a close accounting period by specifying the ACEPANEL program as a user program in Misc -> User Programs personalization. You can run ACEPANEL manually at any time. ACEPANEL extracts data from the previous period Transaction Log.

You are responsible for discarding data from the Panel Diary file periodically. This prevents the file from filling all available disk space. Data accumulates faster when collecting data for all customers.

Logical name	<EAMPANEL>, <EAMPANL1>, <EAMPANL2>, <EAMPANL3>, <EAMPANL4>, <EAMPANL5>, <EAMPANL6>, <EAMPANL7>, <EAMPANL8>, <EAMPANL9>

Data object reference	*ISPanelFormatDefinition* <`ACEPANEL.CPP`>, *ISSAEMTlogDataFactory* <`SAEMTLDF.CPP`>
Organization	Sequential
Distribution class	Mirrored per update
File copies	Default, Additional Filtered Panel Data Files 1-9
Record length	Variable

eampanel.dat retains appended panel data since the last time it was dumped and purged. The host can retrieve eampanel.dat in EBCDIC by requesting the file name $panel through Advanced Data Communications for Stores (ADCS).

Note: eampanel.dat can become very large, but its data is compressible. You can use system file compression facilities to compress the panel file before transmitting it across phone lines. This substantially reduces transmission time. Data can be decompressed at the host.

The panel file grows at a rate of about 3MB per $100,000.00 of contained sales data. For example, a store that produces $500,000.00 of business a week and keeps panel data for customers accounting for 80% of that volume should expect to accumulate 12MB of panel data a week. This panel data can be compressed to a compression ratio of four to one. For example, a 12MB panel file can be compressed into a 3MB file for transmission.

The panel filtering function ("EAMFBFLT (Preferred Customer Panel Filter File)" on page 219) can reduce the size of a Panel Diary file by deleting all information for selected items, manufacturers, or departments. There can be up to 10 Panel Filter files that each filter data for a corresponding Panel Diary file. SurePOS ACE always creates at least one panel file (eampanel.dat), even if the first filter file (eamfbflt.dat) does not exist. The presence of any additional Panel Filter files (eampanl1.dat - eampanl9.dat) triggers creating additional corresponding Panel Diary files (eampanl1.dat - eampanl9.dat).

The default eamfbflt.dat file that is shipped with SurePOS ACE automatically excludes Customer Identification (?) and Coupon Tracking (==) strings in the Transaction Log from being included in the eampanel files. You must edit eamfbflt.dat to include these strings in the eampanel file output.

Fields defined in eampanel.dat strings are delimited by ASCII characters.

Table 68. Transaction Summary Log File Delimiters

Items to be Delimited	ASCII Character
Fields/Substrings	: (Delimited by colons) X'3A'
Strings	"," (Bounded by quotes, delimited by commas) X'222C22'
Records	CRLF (Delimited by ASCII carriage return/line feed) X'0D0A'

You can view Panel Diary file contents by printing the file directly on your store printer with the `PRINT EAMPANEL` command. Or, you can dump the file to the host in EBCDIC and print it there.

Period Header String (81)

A period header is added to panel data to delimit the start of a new reporting period. This header string contains the date and time of the start of the period and the store number. The period header string appears in all panel files, but the date and time information might not be consistent. Each Panel Diary file contains the date and time from the first sales transaction in the period.

Field Name	Type	Length	Description
StringType	ASCII	2	Period Header string type of "81".
DateTime	ASCII	10	Date and time as YYMMDDHHmm.
StoreNumber	ASCII	4	Store number printed on customer receipt.

Marker for Preferred Customer Transactions

If a preferred customer ID is recognized in a sales transaction, SurePOS ACE records it in the Transaction Log Header string (X'00'), which is extracted for the Panel Diary file by default. The TRANTYPE field in the Transaction Log must be a normal sales transaction during customer checkout (00). The preferred customer transaction is indicated by flag bit 9 (X'00400000') in the INDICAT1 field.

Marker for Items to Which Preferred Customer Coupons Might Apply

Items to which a preferred customer coupon might apply are marked in the Transaction Log Item Entry string (X'01'), which is extracted for the Panel Diary file by default. The marker is flag bit 8 (X'0001') in the INDICAT2 field. Such items are also marked on customer receipts.

Marker for Coupon Tracking Transactions

You can specify options in Options -> Coupon -> Tracking personalization to track a coupon in the Coupon Tracking file (eamcoupc.dat). If you have done so, SurePOS ACE indicates that the coupon was tracked in the Transaction Log Header string (X'00'), which is extracted for the Panel Diary file by default. The TRANTYPE field must be for a normal sales transaction during customer checkout (00). The tracked coupon is indicated by flag bit 27 (X'00000010') in the INDICAT1 field.

Automatic Coupon Transaction Data String (11<<)

This data entry string shows the count and amount of automatic (linked-to) store or manufacturer coupons in a transaction. This string can be in a non-preferred customer transaction as well as a preferred customer transaction because automatic coupons can apply to all customers.

This information is critical to the function that automatically picks up paperless coupons.

Field Name	Type	Length	Description
StringType	ASCII	2	Data entry string type of "11".
Identifier	ASCII	2	Two characters of "< <" to define string.
MfgCouponAmount	ASCII	v5	Amount of automatic manufacturer coupons.

Field Name	Type	Length	Description
StoreCouponAmount	ASCII	v5	Amount of automatic store coupons and double coupon amount according to this formula: `StoreCouponAmount = StoreCouponAmount + DoubleCouponAmount`
MfgCouponCount	ASCII	v5	Count of automatic manufacturer coupons.
StoreCouponCount	ASCII	v5	Count of automatic store coupons and double coupon count according to this formula: `StoreCouponCount = StoreCouponCount + DoubleCouponCount`
Unused	ASCII	1	NULL string.

Customer Identification String (11?)

The customer identification string is actually two data entry strings in each Panel Diary file record for preferred customers. The first occurs when the cashier enters a customer number and can occur more than once in a transaction for multiple customer numbers, or if the same number is entered multiple times. The second data entry string occurs at the end of the transaction, immediately preceding the string for the final tender in a preferred customer transaction.

Field Name	Type	Length	Description
StringType	ASCII	2	Data entry string type of "11".
Identifier	ASCII	1	A character of "?" to define string.
CustomerAccountId	ASCII	v18	Entered customer account number.
NosaleData	ASCII	v4	Data keyed with NOSALE key (default = "0") Expiration date (YYMM) if MSR entry of customer number.
StoreNumber	ASCII	v4	Store number printed on customer receipt.
Unused	ASCII	1	NULL string.
Unused	ASCII	1	NULL string.

Preferred Customer Transaction Data String (11>>)

Field Name	Type	Length	Description
StringType	ASCII	2	Data entry string type of "11".
Identifier	ASCII	2	Two characters of "> >" to define string.
CustomerAccountID	ASCII	v18	Final customer account number.

Field Name	Type	Length	Description
Points	ASCII	v9	Primary points earned for the transaction (including bonus).
CouponAmount	ASCII	v5	Automatic preferred coupon total amount.
CouponCount	ASCII	v2	Automatic preferred coupon total count.
MessageCount	ASCII	v2	Number of messages given from activity file.
TransferredTransCount	ASCII	v2	Count of transferred transactions.
TransferredTransAmount	ASCII	v9	Amount of transferred transactions.
BonusPoints	ASCII	v9	Primary and secondary bonus points given in the transaction.
RedeemedPoints	ASCII	v9	Primary points redeemed during the transaction.

Preferred Customer Secondary Points Transaction Data String (11=>)

Field Name	Type	Length	Description
StringType	ASCII	2	Data entry string type of 0x11.
Identifier	ASCII	2	Two characters "=>"to define the substring.
ClubNumber	ASCII	v2	Secondary club number (1-90).
Points	ASCII	v9	Secondary points earned for transaction (including bonus).
RedeemedPoints	ASCII	v9	Secondary points redeemed during the transaction.
BonusPoints	ASCII	v9	Secondary bonus points given during the transaction.
Sales	ASCII	v9	Sales on which secondary points were earned.

New Tender Verification Account Number String (11;;)

This data entry string shows a new tender verification account number from a manual override of the account number from the Preferred Customer Activity file. This string applies only to preferred customer transactions. The new alternate account number replaces the old one in the

Preferred Customer Activity file. This string usually follows the tender string for the new account number used.

Field Name	Type	Length	Description
StringType	ASCII	2	Data entry string type of "11".
Identifier	ASCII	2	Two characters of ";;" to define string.
AccountNumber	ASCII	v18	New Tender Verification account number.
Unused	ASCII	1	NULL string.
Unused	ASCII	1	NULL string.
Unused	ASCII	1	NULL string.
Unused	ASCII	1	NULL string.

Alias Electronic Coupon Item Code String (11)

This data entry string shows the primary item code when an electronic coupon uses an alias item record to access a UPC that is too big to fit in the item record link field. This item code is needed so that item movement can be accumulated against both item codes that represent the coupon. The simple data entry string is recognized because it immediately precedes an item string for an electronic coupon.

Field Name	Type	Length	Description
StringType	ASCII	2	Data entry string type of "11".
ItemCode	ASCII	4	Linked-to item code for alias electronic coupon.
Unused	ASCII	1	NULL string.
Unused	ASCII	1	NULL string.
Unused	ASCII	1	NULL string.
Unused	ASCII	1	NULL string.
Unused	ASCII	1	NULL string.

Coupon Tracking String (11==)

If you select the *Log coupons in tracking file* option in Options -> Coupon -> Tracking personalization and if the coupon is in the range you are logging to the Coupon Tracking file, SurePOS ACE writes a data entry string to show the logging requirement and to provide extra data to log. This string type immediately follows the coupon string to which it applies.

Field Name	Type	Length	Description
StringType	ASCII	2	Data entry string type of "11".
Unused	ASCII	2	NULL string.
Identifier	ASCII	2	Two characters of "= =" to define string.

Field Name	Type	Length	Description
LogFlags	ASCII	v5	Flags to be logged in the EAMCOUPC record. This field corresponds to the ValidFlags field in the EAMCOUPC record.
CampaignNumber	ASCII	v6	Campaign number if keyed in.
MfgNumber	ASCII	v5	Manufacturer number from item record.
PromotionCode	ASCII	v5	Promotion code used to validate the coupon.

Used Targeted Coupons String (11=;)

This string contains the coupon codes of targeted coupons that were redeemed during the transaction.

Field Name	Type	Length	Description
StringType	ASCII	2	Data entry string type of "11".
Identifier	ASCII	2	Two characters of "= ;" to define string.
CustomerAccountID	ASCII	v18	Customer account number.
TargetedCoupon	ASCII	4	Coupon number of targeted coupon awarded during the transaction. This field is repeated once for each targeted coupon that was awarded. SurePOS ACE allows a maximum of 96 targeted coupons. Actual maximum for the store is specified in personalization. Coupon numbers are separated by colons (:).

Bonus/Redemption Points String (11<>)

This data entry string shows information associated with items or coupons that give bonus points or that require the redemption of preferred customer points. A bonus points string can appear in a regular customer transaction, but it has no meaning outside of a preferred customer transaction. This string replaces the item entry string that normally accompanies an item or coupon.

Note: The INDICAT1, INDICAT2 and INDICAT3 fields in this string all contain the DECIMAL representation of these flag fields. These same fields in the Transaction Log would be represented by hexadecimal values. For example, a value of X'2000' for INDICAT1 would be represented here as decimal 8192.

Field Name	Type	Length	Description
StringType	ASCII	2	Data entry string type of "11".
Identifier	ASCII	2	Two characters of "< >" to define string.
ItemCode	ASCII	v14	Item code for bonus item/coupon.
Value	ASCII	v8	Points associated with item/coupon.

Field Name	Type	Length	Description
MoreFlags	PD	1	• Bits 0-1 - Reserved • Bits 2-6 - Unused • Bit 7 - X'1' = Linked item; X'0' = Keyed item.
Families	ASCII	v6	Family codes for item/coupon. The first three bytes represent the current coupon family number and the last 3 bytes represent the previous coupon family number as defined in the item record.
Indicat1	ASCII	v8	Decimal value of INDICAT1 in Transaction Log string type 01: **Bit 0 X'80000000' through Bit 14 X'00020000'** Reserved. **Bit 15 X'00010000'** FuelItem: Fuel item. **Bit 16 X'00008000'** Reserved. **Bit 17 X'00004000'** WeightItem: Weight item. **Bit 18 X'00002000'** PriceEntered: Entered price used. **Bit 19 X'00001000'** PriceRequired: Price required. **Bit 20 X'00000800'** LogAll: Log all occurrences. **Bit 21 X'00000400'** LogEx: Log item as exception. **Bit 22 X'00000200'** Alias: Item code from alias record. **Bit 23 X'00000100'** NoMovement: No item movement kept. **Bit 24 X'00000080'** TaxableA: Tax plan A applies to this item. **Bit 25 X'00000040'** TaxableB: Tax plan B applies to this item.

Field Name	Type	Length	Description
			Bit 26 X'00000020' TaxableC: Tax plan C applies to this item. (Coupon items only-minimum purchase by manufacturer indicator.) **Bit 27 X'00000010'** TaxableD: Tax plan D applies to this item. (Coupon items only-minimum purchase by department indicator.) **Bit 28 X'00000008'** FoodStamp: Food stamp item. **Bit 29 X'00000004'** PointsItem: Points item. (Coupon items only-exclude item from minimum purchase indicator.) **Bit 30 X'00000002'** NonDiscountable: Non-discountable item. (Coupon items only-redemption item indicator.) **Bit 31 X'00000001'** CpnMultAllowed: Coupon multiplication allowed.
Indicat2	ASCII	v5	Decimal value of Indicat2 in Transaction Log string type 01: **Bit 7 X'8000'** Reserved. **Bit 6 X'4000'** OverrideRqd: Item requires override. **Bit 5 X'2000'** DataEntry: Data entry with item. **Bit 4 X'1000'** Multipriced: Multipriced item. **Bit 3 X'0800'** WeightOrQty: Weight or quantity entered, or points-only item indicator. **Bit 2 X'0400'** CouponMult: Store coupon created by coupon multiplier.

Field Name	Type	Length	Description
			Bit 1 X'0200' TaxKey: Tax key pressed. **Bit 0 X'0100'** FoodstampKey: Food stamp key pressed. **Bit 15 X'0080'** CancelKey: Cancel key pressed. **Bit 14 X'0040'** RefundKey: Refund key pressed. **Bit 13 X'0020'** StoreCouponKey: Store coupon key pressed. **Bit 12 X'0010'** MfrCouponKey: Manufacturer coupon key hit. **Bit 11 X'0008'** DepositKey: Deposit key pressed. **Bit 10 X'0004'** XPRICE: Negative price due to deal. **Bit 9 X'0002'** ExtensionExists: Extension follows this string. **Bit 8 X'0001'** Preferred customer coupon might apply to this item.
Indicat3	ASCII	v2	Hexadecimal value of Indicat3 in Transaction Log string type 01, where item type (T) and operator entry method (O) are in the format TO: **T=0** Normal item sale or bonus item sale **T=1** Deposit **T=2** Refund **T=3** Deposit return

Field Name	Type	Length	Description
			T=4 Misc. trans. receipt (sale) **T=5** Misc. trans. payout (refund) **T=6** Manufacturer coupon **T=7** Store coupon **T=8** Item sale cancel **T=9** Deposit cancel **T=10** Markdown store coupon **O=0** Scanned item code **O=1** Keyed item code **O=2** Item lookup key used **O=3** Item code linked to **O=4-7** Reserved **O=8** Redemption of points **O=9** Bonus points **O>9** Reserved
Indicat4	ASCII	v6	**Bits 8-31** Reserved **Bit 7 X'00000080'** Tax Plan E

Field Name	Type	Length	Description
			Bit 6 X'00000040' Tax Plan F **Bit 5 X'00000020'** Tax Plan G **Bit 4 X'00000010'** Tax Plan H **Bits 2-3** Reserved **Bit 1 X'00000002'** Qualified Healthcare Product (QHP) **Bit 0 X'00000001'** Prescription (Rx) Item
Unused	ASCII	1	NULL string.
Unused	ASCII	1	NULL string.

Redemption/Adjustment String (82)

Redemptions and adjustments data that occurs during enrollment can be appended to the end of EAMPANEL.DAT if you select the *Append audit data to panel file* option in Loyalty -> Activity -> Data Tracking personalization. It is the same data that is appended to the Preferred Customer Audit Log file. Doing this combines all points information in one file.

Information about redemptions and adjustments that occur at terminals already appears by default in the Panel Diary file in a Bonus/Redemption points string. It is not necessary to extract such strings twice. Therefore, the string 82 data that SurePOS ACE appends to eampanel.dat does not include redemptions and adjustments performed at a terminal. It contains only redemptions or adjustments performed through the enrollment procedure.

SurePOS ACE reformats the data to match all other Panel Diary file data. The content is the same as the Preferred Customer Audit Log except that redemptions from terminals are omitted. The transaction number is also omitted.

Field Name	Type	Length	Description
StringType	ASCII	2	Redemption/Adjustment string type of "82".
CustomerAccountId	ASCII	v18	Customer account number.
TransType	ASCII	1	"R"= redemption, "A"= adjustment, "X"= rain check.
DateTime	ASCII	10	Date and time of transaction (`YYMMDDHHmm`).
Reserved	ASCII	v3	NULL string.
Reserved	ASCII	v4	NULL string.

Field Name	Type	Length	Description
Operator	ASCII	v9	Operator/employee who performed transaction.
PointsP	ASCII	v8	Points adjusted/redeemed plus.
PointsM	ASCII	v8	Points adjusted/redeemed minus.
SerialNumber	ASCII	v6	Form serial number or store number for rain check.
Reserved	ASCII	1	Reserved.

Period Summary String (83)

A period summary string optionally can be appended to the end of the foreground panel file EAMPANEL at the end of a reporting period if you select the *Add period summary string to panel file* option in Loyalty -> Activity -> Data Tracking personalization.

This string contains Loyalty totals reported in the Summary Cash Report. These totals represent a summary for the accounting period. They are not necessarily a summary for Panel Diary file data because filtering can reduce the Panel Diary file to a subset of accounting data. This summary record is provided to allow validation of any process that summarizes information in the panel file.

Field Name	Type	Length	Description
StringType	ASCII	2	Period Summary string type of "83".
TotalTransCount	ASCII	v9	Number of sales transactions.
TotalTransAmount	ASCII	v9	Total sales amount.
PCTransCount	ASCII	v9	Number of preferred customer transactions.
PCTransAmount	ASCII	v9	Total amount sold to preferred customers.
PCCouponCount	ASCII	v9	Preferred customer coupon count.
PCCouponAmount	ASCII	v9	Preferred customer coupon amount.
PeriodItemPoints	ASCII	v9	Points given for item sales during period.
PeriodBonusPoints	ASCII	v9	Bonus points given during period.
PeriodRedeemedPoints	ASCII	v9	Points redeemed during period.
AutoMfgCouponAmount	ASCII	v9	Automatic (paperless) manufacturer coupon amount.
AutoStoreCouponAmount	ASCII	v9	Automatic (paperless) store coupon amount.

The net sales for preferred customers is the sum of the gross positive and gross negative amounts in all transaction header records that are marked for preferred customers. The transaction count should match the number of these records. The same totals for all sales transaction header strings should give the total sales amount and transaction count if data is kept for regular customers.

The preferred coupon count and amount reflect the sum of all coupon records with UPC codes within the preferred customer coupon range. Point totals reflect the sum of points from all preferred customer transaction data strings ("Preferred Customer Transaction Data String (11>>)" on page 323). Automatic coupon totals reflect the sum of paperless coupons from all automatic coupon transaction data strings ("Automatic Coupon Transaction Data String (11<<)" on page 322).

If there are no paper preferred coupons and automatic coupons are given only to preferred customers, automatic coupon totals should match the preferred customer coupon total.

EAMPER* (Operator Performance File)

The Operator Performance file is a keyed file that contains totals data for operators. A separate file exists for the current period, previous period, and one old period. A record exists for each operator who performed sales functions and for the store.

Operator Performance totals can be closed *short* or *long*. This allows for daily as well as weekly totals.

An image version of each Operator Performance file is on the alternate file server. This distribution takes place when the file is closed to minimize the performance impact.

Logical name	<EAMPERFC>, <EAMPERFP>, <EAMPERFO>, <EAMPERCL>, <EAMPERPL>, <EAMPEROL>, <EAMPERWK>
Data object reference	*ISSAPerformanceStorage* <SAPRFSTR.CPP>
Organization	Keyed
Distribution class	Mirrored at close
File copies	Current short, Previous short, Old short, Current long, Previous long, Old long, Weekly
Record length	508 bytes
Key length	6 bytes: Type/Operator

Store Record

Field Name	Type	Length	Decimal Offset	Description
Type	Int	1	0	Record type = 1 for the store record
Operator	PD	5	1	Operator = 0
DateTime	PD	5	6	Date and time when current period started or previous period ended. Format: YYMMDDHHmm.
Indicat0	Int	1	11	Bit 1 X'40' indicates that all operator performance totals have been printed since the last update for the period.
Terminal	PD	2	12	Terminal that last updated this record.

Field Name	Type	Length	Decimal Offset	Description
Restart	Int	4	14	This field is used as a control field on a system restart. It is used to determine if this record has been updated by a particular transaction.
The following are gross totals for all terminals in the store. (These totals are updated only when an operator is signed off. They are reset by each close of the reporting period.)				
GrossPos	Int	4	18	Gross positive. Accumulation of the positive sales entries. Does not include totals from voided or training transactions. (sales + deposits + taxes + tender fees)
GrossNeg	Int	4	22	Gross negative. Accumulation of the negative sales entries. Does not include totals from voided or training transactions. Note that this is a positive amount. (deposit returns + item refunds + tax refunds + discounts + cancels of sales, deposits, item refunds, deposit returns, and tender fees).
NumTrans	Int	4	26	Number of sales transactions
Reserved	ASCII	378	30	Reserved.
UserInts	Int	60	408	Fifteen 4-byte integers for use with user extensions.
UserExit	ASCII	40	468	Reserved for user extensions.

Operator Record

Field Name	Type	Length	Decimal Offset	Description
Type	ASCII	1	0	Record type = 2 for an operator record.
Operator	PD	5	1	Operator ID
DateTime	PD	5	6	Date and Time of transaction which last updated this record. Format: `YYMMDDHHmm`
Indicat0	Int	1	11	**Bit 7 X'80'** Reserved1: Reserved. **Bit 6 X'40'** IsPRPrinted: A performance report has been printed since the last update to this operator's record.

Field Name	Type	Length	Decimal Offset	Description
				Bit 5 X'20' IsMissing: The system is missing Transaction Summary Log records for this operator. **Bit 4 X'10'** IsCRPrinted: A short cash report has been printed since the last update to this operator's record. **Bits 3 - 0** Reserved2: Reserved.
TransNum	PD	2	12	Transaction number of the last update to this file
Restart	Int	4	14	This field is used as a control field on a system restart. It is used to determine if this record has been updated by a particular transaction.
The following fields are gross totals for the operator:				
GrossPos	Int	4	18	Gross positive. Accumulation of the positive sales entries. Does not include totals from voided or training transactions. (sales + deposits + taxes + tender fees)
GrossNeg	Int	4	22	Gross negative. Accumulation of the negative sales entries. Does not include totals from voided or training transactions. Note that this is a positive amount. (deposit returns + item refunds + tax refunds + discounts + cancels of sales, deposits, item refunds, deposit returns, and tender fees)
NumTrans	Int	4	26	Number of sales transactions
The following fields are tender accepted by the operator:				
AmtCash	Int	4	30	Entered tender - cash
AmtCheck	Int	4	34	Entered tender - checks
AmtFoods	Int	4	38	Entered tender - food stamps
AmtMisc1	Int	4	42	Entered tender - miscellaneous tender 1
AmtMisc2	Int	4	46	Entered tender - miscellaneous tender 2
AmtMisc3	Int	4	50	Entered tender - miscellaneous tender 3
AmtManuf	Int	4	54	Entered tender - manufacturer coupons
AmtStore	Int	4	58	Entered tender - store coupons

Field Name	Type	Length	Decimal Offset	Description
The following fields are sales amounts (voided and training transactions not included, amounts include cancels except for sales and deposits):				
AmtSale	Int	4	62	Item sale amount
AmtDepos	Int	4	66	Deposit amount
AmtRefun	Int	4	70	Refund amount
AmtDeprt	Int	4	74	Deposit return amount
AmtMirsa	Int	4	78	Amount of miscellaneous item record receipts
AmtMirpy	Int	4	82	Amount of miscellaneous item record payouts
AmtDisco	Int	4	86	Discount amount
AmtTaxex	Int	4	90	Taxable amount to which exemptions apply
AmtCance	Int	4	94	Item sale cancel amount
AmtDepcl	Int	4	98	Deposit cancel amount
AmtCredit	Int	4	102	Credit transaction amount
AmtTnfee	Int	4	106	Amount of tender fees (including checks) paid
AmtMisce	Int	4	110	Miscellaneous transactions amount credited to this operator by the controller procedure
AmtNstnd	Int	4	114	Amount of tender cashed with no-sale procedure
AmtTxchg	Int	4	118	Amount of tender exchanged with no-sale procedure
The following fields are tax information for the amount of tender exchanged with no-sale procedures:				
AmtSaleA	Int	4	122	Item sales to which tax plan A applies
AmtTaxA	Int	4	126	Tax paid for tax plan A
AmtSaleB	Int	4	130	Item sales to which tax plan B applies
AmtTaxB	Int	4	134	Tax paid for tax plan B
AmtSaleC	Int	4	138	Item sales to which tax plan C applies
AmtTaxC	Int	4	142	Tax plan for tax plan C
AmtSaleD	Int	4	146	Item sales to which tax plan D applies (where D = D + E + F + G + H)
AmtTaxD	Int	4	150	Tax paid for tax plan D (where D = D + E + F + G + H)
The following amounts are for training, voided, and standalone transactions:				
StandAloneGrossPos	Int	4	154	Standalone transactions gross positive
StandAloneGrossNeg	Int	4	158	Standalone transactions gross negative

Field Name	Type	Length	Decimal Offset	Description
AmtVoids	Int	4	162	Amount of voided transactions
AmtTTaf	Int	4	166	Amount of training transactions
The following fields are sales counts (voided and training transactions not included). Counts include cancels except for sales and deposits:				
NumItems	Int	4	170	Number of items sold
NumKeyed	Int	4	174	Number of items sold by keyed item codes
NumDepar	Int	4	178	Number of items sold by item lookup keys
Reserved	Int	4	182	Reserved
NumDepos	Int	2	186	Number of deposits
NumRefun	Int	2	188	Number of refunds
NumDeprt	Int	2	190	Number of deposit returns
NumMirsa	Int	2	192	Number of miscellaneous item record receipts
NumMirpy	Int	2	194	Number of miscellaneous item record payouts
NumDisco	Int	2	196	Number of discounts
NumTaxex	Int	2	198	Number of tax exemptions
NumCance	Int	2	200	Number of item sale cancels
NumDepcl	Int	2	202	Number of deposit cancels
NumCredit	Int	2	204	Number of credit transactions
NumSpsoff	Int	2	206	Number of special sign-offs
NumOpenc	Int	2	208	Count of the times the cash drawer was opened outside of a sales transaction
The following fields are tender counts:				
NumCash	Int	2	210	Number of cash transactions
NumCheck	Int	2	212	Number of checks cashed
NumFoods	Int	2	214	Number of food stamp transactions
NumMisc1	Int	2	216	Number of miscellaneous tender 1
NumMisc2	Int	2	218	Number of miscellaneous tender 2
NumMisc3	Int	2	220	Number of miscellaneous tender 3
NumManuf	Int	2	222	Number of manufacturer coupons
NumStore	Int	2	224	Number of store coupons
The following fields are counts for training, voided, and standalone transactions:				
NumStandaloneTrans	Int	2	226	Number of standalone transactions
NumVoids	Int	2	228	Number of voided transactions

Field Name	Type	Length	Decimal Offset	Description
NumTrainingTrans	Int	2	230	Number of training transactions
The following fields are operator statistical data - time breakout:				
RingTime	Int	4	232	Total ring time in seconds. Includes the time from the first line item until balance due. Also includes the time from the previous total or tender to a subsequent item sale.
Tenderl	Int	4	236	Total tender time in seconds. Includes the time between a total and a tender, the time between tenders, and the time between the final tender and the close of the cash drawer.
Special	Int	4	240	Total special sign-off time in seconds. Includes the time between a special sign-off and the subsequent special sign-on.
InActive	Int	4	244	Time between transactions in seconds
NonSales	Int	4	248	Time in seconds when non-sales procedures are being run. Voided, standalone, and training transactions are in this time.
The following counters are sums of various totals over a given range of departments and are used to determine the average item sale price over those departments. Department groups are defined by department group personalization. The *xx* in DeptCntxx and DeptAmtxx ranges from 1 to 20. There can be subtotals for up to the first 20 department groups.				
DeptCnt1	Int	4	252	Item count (sales - cancels - refunds) for the first range of departments
DeptAmt1	Int	4	256	Amount (sales - cancels - refunds) for first range of departments
DeptCnt2	Int	4	260	Item count (sales - cancels - refunds) for the second range of departments
DeptAmt2	Int	4	264	Amount (sales - cancels - refunds) for the second range of departments
DeptCnt3	Int	4	268	Item count (sales - cancels - refunds) for the third range of departments
DeptAmt3	Int	4	272	Amount (sales - cancels - refunds) for third range of departments
DeptCnt4	Int	4	276	Item count (sales - cancels - refunds) for the fourth range of departments
DeptAmt4	Int	4	280	Amount (sales - cancels - refunds) for fourth range of departments
DeptCnt5	Int	4	284	Item count (sales - cancels - refunds) for the fifth range of departments

Field Name	Type	Length	Decimal Offset	Description
DeptAmt5	Int	4	288	Amount (sales - cancels - refunds) for the fifth range of departments
DeptCnt6	Int	4	292	Item count (sales - cancels - refunds) for the sixth range of departments
DeptAmt6	Int	4	296	Amount (sales - cancels - refunds) for the sixth range of departments
DeptCnt7	Int	4	300	Item count (sales - cancels - refunds) for the seventh range of departments
DeptAmt7	Int	4	304	Amount (sales - cancels - refunds) for the seventh range of departments
DeptCnt8	Int	4	308	Item count (sales - cancels - refunds) for the eighth range of departments
DeptAmt8	Int	4	312	Amount (sales - cancels - refunds) for the eighth range of departments
DeptCnt9	Int	4	316	Item count (sales - cancels - refunds) for the ninth range of departments
DeptAmt9	Int	4	320	Amount (sales - cancels - refunds) for the ninth range of departments
DeptCnt10	Int	4	324	Item count (sales - cancels - refunds) for the tenth range of departments
DeptAmt10	Int	4	328	Amount (sales - cancels - refunds) for the tenth range of departments
DeptCnt11	Int	4	332	Item count (sales - cancels - refunds) for the eleventh range of departments
DeptAmt11	Int	4	336	Amount (sales - cancels - refunds) for the eleventh range of departments
DeptCnt12	Int	4	340	Item count (sales - cancels - refunds) for the twelfth range of departments
DeptAmt12	Int	4	344	Amount (sales - cancels - refunds) for the twelfth range of departments
DeptCnt13	Int	4	348	Item count (sales - cancels - refunds) for the thirteenth range of departments
DeptAmt13	Int	4	352	Amount (sales - cancels - refunds) for the thirteenth range of departments
DeptCnt14	Int	4	356	Item count (sales - cancels - refunds) for the fourteenth range of departments
DeptAmt14	Int	4	360	Amount (sales - cancels - refunds) for the fourteenth range of departments

Field Name	Type	Length	Decimal Offset	Description
DeptCnt15	Int	4	364	Item count (sales - cancels - refunds) for the fifteenth range of departments
DeptAmt15	Int	4	368	Amount (sales - cancels - refunds) for the fifteenth range of departments
DeptCnt16	Int	4	372	Item count (sales - cancels - refunds) for the sixteenth range of departments
DeptAmt16	Int	4	376	Amount (sales - cancels - refunds) for the sixteenth range of departments
DeptCnt17	Int	4	380	Item count (sales - cancels - refunds) for the seventeenth range of departments
DeptAmt17	Int	4	384	Amount (sales - cancels - refunds) for the seventeenth range of departments
DeptCnt18	Int	4	388	Item count (sales - cancels - refunds) for the eighteenth range of departments
DeptAmt18	Int	4	392	Amount (sales - cancels - refunds) for the eighteenth range of departments
DeptCnt19	Int	4	396	Item count (sales - cancels - refunds) for the nineteenth range of departments
DeptAmt19	Int	4	400	Amount (sales - cancels - refunds) for the nineteenth range of departments
DeptCnt20	Int	4	404	Item count (sales - cancels - refunds) for the twentieth range of departments
DeptAmt20	Int	4	408	Amount (sales - cancels - refunds) for the twentieth range of departments
AmtSATax	Int	4	412	Standalone tax amount
MfrCoupScanCount	Int	2	416	Count of manufacturer's coupons that were scanned
Reserved	Int	6	418	Reserved
SalesPoints	Int	4	424	Points given for participating customer sales
BonusPoints	Int	4	428	Bonus points given to enrolled customers
RedeemedPoints	Int	4	432	Points redeemed by enrolled customer
PCTransactionAmount	Int	4	436	Preferred customer transaction amount
PCAutoCouponCount	Int	2	442	Preferred customer auto coupon count
PCAutoCouponAmount	Int	4	444	Preferred customer auto coupon amount
UsrInts	Int	40	448	Ten 4-byte integers for use with extensions
UsrExit	ASCII	20	488	Reserved for user extensions

Operator Extension Record

Field Name	Type	Length	Decimal Offset	Description
Type	ASCII	1	0	Record type = 5 for extension
Operator	PD	5	1	Operator ID
DateTime	PD	5	6	Date and Time of transaction which last updated this record. Format: `YYMMDDHHmm`
Reserved	Int	1	11	Reserved
TransNum	PD	2	12	Transaction number of the last update to this file
Restart	Int	4	14	This field is used as a control field on a system restart. It is used to determine if this record has been updated by a particular transaction.
Reserved	Int	168	18	Forty-two 4-byte integers reserved by Toshiba
Reserved	Int	46	186	Twenty-three 2-byte integers reserved by Toshiba
Reserved	Int	20	232	Five 4-byte integers reserved by Toshiba
DeptCnt21	Int	4	252	Item count (sales - cancels - refunds) for the twenty-first range of departments
DeptAmt21	Int	4	256	Amount (sales - cancels - refunds) for twenty-first range of departments
DeptCnt22	Int	4	260	Item count (sales - cancels - refunds) for the twenty-second range of departments
DeptAmt22	Int	4	264	Amount (sales - cancels - refunds) for the twenty-second range of departments
DeptCnt23	Int	4	268	Item count (sales - cancels - refunds) for the twenty-third range of departments
DeptAmt23	Int	4	272	Amount (sales - cancels - refunds) for twenty-third range of departments
DeptCnt24	Int	4	276	Item count (sales - cancels - refunds) for the twenty-fourth range of departments
DeptAmt24	Int	4	280	Amount (sales - cancels - refunds) for twenty-fourth range of departments
DeptCnt25	Int	4	284	Item count (sales - cancels - refunds) for the twenty-fifth range of departments
DeptAmt25	Int	4	288	Amount (sales - cancels - refunds) for the twenty-fifth range of departments
DeptCnt26	Int	4	292	Item count (sales - cancels - refunds) for the twenty-sixth range of departments

Field Name	Type	Length	Decimal Offset	Description
DeptAmt26	Int	4	296	Amount (sales - cancels - refunds) for the twenty-sixth range of departments
DeptCnt27	Int	4	300	Item count (sales - cancels - refunds) for the twenty-seventh range of departments
DeptAmt27	Int	4	304	Amount (sales - cancels - refunds) for the twenty-seventh range of departments
DeptCnt28	Int	4	308	Item count (sales - cancels - refunds) for the twenty-eighth range of departments
DeptAmt28	Int	4	312	Amount (sales - cancels - refunds) for the twenty-eighth range of departments
DeptCnt29	Int	4	316	Item count (sales - cancels - refunds) for the twenty-ninth range of departments
DeptAmt29	Int	4	320	Amount (sales - cancels - refunds) for the twenty-ninth range of departments
DeptCnt30	Int	4	324	Item count (sales - cancels - refunds) for the thirtieth range of departments
DeptAmt30	Int	4	328	Amount (sales - cancels - refunds) for the thirtieth range of departments
DeptCnt31	Int	4	332	Item count (sales - cancels - refunds) for the thirty-first range of departments
DeptAmt31	Int	4	336	Amount (sales - cancels - refunds) for the thirty-first range of departments
DeptCnt32	Int	4	340	Item count (sales - cancels - refunds) for the thirty-second range of departments
DeptAmt32	Int	4	344	Amount (sales - cancels - refunds) for the thirty-second range of departments
DeptCnt33	Int	4	348	Item count (sales - cancels - refunds) for the thirty-third range of departments
DeptAmt33	Int	4	352	Amount (sales - cancels - refunds) for the thirty-third range of departments
DeptCnt34	Int	4	356	Item count (sales - cancels - refunds) for the thirty-fourth range of departments
DeptAmt34	Int	4	360	Amount (sales - cancels - refunds) for the thirty-fourth range of departments
DeptCnt35	Int	4	364	Item count (sales - cancels - refunds) for the thirty-fifth range of departments
DeptAmt35	Int	4	368	Amount (sales - cancels - refunds) for the thirty-fifth range of departments

Field Name	Type	Length	Decimal Offset	Description
DeptCnt36	Int	4	372	Item count (sales - cancels - refunds) for the thirty-sixth range of departments
DeptAmt36	Int	4	376	Amount (sales - cancels - refunds) for the thirty-sixth range of departments
DeptCnt37	Int	4	380	Item count (sales - cancels - refunds) for the thirty-seventh range of departments
DeptAmt37	Int	4	384	Amount (sales - cancels - refunds) for the thirty-seventh range of departments
DeptCnt38	Int	4	388	Item count (sales - cancels - refunds) for the thirty-eighth range of departments
DeptAmt38	Int	4	392	Amount (sales - cancels - refunds) for the thirty-eighth range of departments
DeptCnt39	Int	4	396	Item count (sales - cancels - refunds) for the thirty-nineth range of departments
DeptAmt39	Int	4	400	Amount (sales - cancels - refunds) for the thirty-nineth range of departments
DeptCnt40	Int	4	404	Item count (sales - cancels - refunds) for the fortieth range of departments
DeptAmt40	Int	4	408	Amount (sales - cancels - refunds) for the fortieth range of departments
Reserved	Int	4	412	One 4-byte integer reserved by Toshiba
Reserved	ASCII	32	416	Reserved
UsrInts	Int	40	448	Ten 4-byte integers for use with exits
UsrExit	ASCII	20	488	User field

EAMSRKY* (Suspend/Retrieve Keyed Index File)

The Suspend/Retrieve Keyed Index file is an index to the Suspended Transactions file (eamsrtrx.dat), which is a sequential file that holds data for suspended transactions in Transaction Log format.

See Table 1 for a description of how the SurePOS ACE file differs from the 4680-4690 Supermarket Application file.

Logical names	<EAMSRKYC>, <EAMSRKYP>, <EAMSRKLC>, <EAMSRKLP>, <LCLSRKYC:>, <LCLSRKYP:>, <LCLSRKLC:>, <LCLSRKLP:>
Data object reference	*ISSuspendRecallData* <SSPNRDAT.CPP>
Organization	Keyed

Distribution class	Prime Copy: Mirrored per update Local: Local
File copies	Current short, Previous short, Old short, Current long, Previous long
Record length	45 bytes
Key length	6 bytes: Key, RepKey

Field Name	Type	Length	Decimal Offset	Description
Key	ASCII	1	0	The Key consists of an unpacked ASCII number from 1 to 8.
RepKey	PD	5	1	The packed number that follows the Key that represents the store, operator, terminal, or transaction index key: • 1 is for store suspend totals, as in `310000000000`, for example • 2 is for store retrieve totals, as in `320000000000` • 3 is for terminal suspend totals, as in `330000000151` for terminal 151 • 4 is for terminal retrieve totals, as in `340000000151` for terminal 151 • 5 is for operator suspend totals, as in `350000000432` for operator 432 • 6 is for operator retrieve totals, as in `360000000432` for operator 432 • 7 is for the transaction index record, as in `371510000134` for terminal 151, transaction 134 • 8 is for the transaction index record of a transaction that was suspended in the previous period, as in 381510000134 for terminal 151, transaction 134.
TotalAmount	Int	4	6	Total amount of suspends or retrieves for the record type identified by the Key field.
TotalCount	Int	4	10	Number of suspends or retrieves for the record type identified by the Key field. Does not apply to record type 7 or 8.
Date	ASCII	6	14	Date when a transaction (Key field type 7 or 8) was suspended in this format: MMDDYY
Time	ASCII	6	20	Time when a transaction (Key field type 7 or 8) was suspended in this format: HHMMSS
Pointer	Int	4	26	Offset pointing to the beginning of this transaction record (Key field type 7 or 8) in the Suspended Transactions file.

Field Name	Type	Length	Decimal Offset	Description
TransSize	Int	4	30	This is the size of the transaction used to determine the mode of retrieval for a Key field type 7 or 8 suspended transaction. For Key field retrieve types 4 and 6, this is the total count of retrieved transactions of the particular type.
SuspendTraining	Int	1	34	Indicator that the record was created in training mode and is not to appear in accounting totals.
RetrievedFlag	Int	1	35	Indicates that this transaction (Key field type 7 or 8) has been retrieved if set to -1.
SuspendedOperator	PS	5	36	Operator ID of the cashier who suspended the transaction (Key field type 7 or 8). For Key field type 1, this field will contain the date/time of the store close processing.
RetrievingTerminal	Int	2	41	Terminal ID of the terminal that is currently retrieving the suspended transaction (Key field type 7 or 8). For key field type 2, this field will contain the number of remaining transactions that were suspended in the previous period and have not been retrieved in the current period. For key field type 4 and 6, this field will contain the number of transactions that were suspended in the previous period and retrieved in the current period for the record type identified by the Key field.
Reserved	Int	2	43	Reserved.

Note:

1. This file contains record types 1 and 2 for store totals, types 3 and 4 for each terminal, types 5 and 6 for each operator, and a type 7 or 8 record for each suspended transaction.
2. The keyed file size is determined by the *Suspend/Retrieve Keyed Index Records* option in Options -> Database -> Totals personalization. (Refer to the *SurePOS ACE: Planning and Installation Guide* for information about personalization.)
3. In addition to current, previous, and old versions of this file, there are long and short current and previous versions.

EAMSRTR* (Suspended Transactions File)

The Suspended Transactions file, adx_idt4\eamsrtrx.dat, is the file that holds suspended transactions. It is a single sequential file indexed by a keyed file, the Suspend Retrieve Keyed Index file, eamsrky*.dat.

SurePOS ACE uses a single, sequential file for all suspended transactions because multiple sequential files incur a great deal of processing overhead with the necessary closes, creates, and opens at every suspend, and with other closes, renames, and opens at every retrieval. These are additional reasons for a single file:

- A single file prevents cluster fragmentation from continuous creations and deletions of multiple files.

- Reporting suspend/retrieve activity requires opening and closing just one file per report rather than multiple files.
- A single sequential file is easier to manage than are several such files.
- The single sequential file is managed as if it were another Transaction Log. Updates to it are performed with simple write statements that are protected from controller failures in the same way as are Transaction Log entries, through spooling.
- Creating a single file at the controller lets it become distributable, which lets all terminals access it.

Logical name	<EAMSRTRX>, <EAMSRTRP>, <LCLSRTRX:>, <LCLSRTRP:>
Data object reference	*ISSASuspendedTransactionLibrary* <`SASSPNTL.CPP`>
Organization	Sequential
Distribution class	Prime Copy: Mirrored per update Local: Local
File copies	1
Record length	Variable

The Suspended Transactions file has the same format as the Transaction Log. See "EAMTRAN* (Transaction Summary Log)" on page 361 for a description of the Transaction Log file (*ISSATransactionData* <`SATRNDAT.CPP`>) layout.

The *Suspend needs manager's override* and *Retrieve needs manager's override* options in Options -> Store -> Suspend/Retrieve personalization control requiring a manager override. When a suspend or retrieve requires a manager override, SurePOS ACE logs it as a string type 0x10 with a reason code 0x57.

EAMTAXT* (Tax Table File)

The Tax Table file contains deltas for tax rates or tax brackets and amounts that are required to compute tax. Up to eight separate tax tables can be defined using eight file names. The file is sequential with three different record formats, each ending in X'0D0A'.

The prime versions of each file are on the master controller. Image versions are on all other eligible controllers so that terminals can always access them.

Logical names	<EAMTAX:>n
Data object reference	*ISUSTaxTable* <`USTAXTBL.CPP`>
Data object reference	*ISUSPercentageTax* <`USPRCNTX.CPP`>
Organization	Variable sequential
Distribution class	Compound at close, up to 4 per controller
File copies	1 per tax plan
Record length	Variable

Record 1 for Tax Table

Field Name	Type	Length	Description
Descriptor	ASCII	v24	Tax Table descriptor.

Field Name	Type	Length	Description
TaxRate	ASCII	6	Flat rate for tax rate plan, or start of repeat range for tax table. 0-999999, which assumes two decimal places.
FirstTax	ASCII	1	Boolean indicator of taxability for the first tax range. Value of zero (false) implies tax is not collected on the first range unless it is a tax rate plan. Any nonzero value (true) implies tax is collected.
DecimalPlaces	ASCII	1	Number of decimal places if this is a tax rate plan. If this field is not present, the default number of decimal places is 2.

Record 2 for Tax Table

Field Name	Type	Length	Description
NumberOfRanges	ASCII	2	Number of ranges in the tax table. The value is 0 for a tax rate plan.

Record 3 for Tax Table

Field Name	Type	Length	Description
RangeDelta	ASCII	2	Sales amount delta for high end of bracket for this range. Value is 0 for a tax rate plan.
AmountDelta	ASCII	2	Tax amount delta for this RangeDelta value. Value is 0 sfor a tax rate plan.

Record 1 for Percentage Tax

Field Name	Type	Length	Description
Descriptor	ASCII	v24	Percentage tax descriptor.
TaxRate	ASCII	6	Flat rate for tax rate plan. Range is 0-999999, which assumes two decimal places.
Reserved	ASCII	1	Reserved.
DecimalPlaces	ASCII	1	Number of decimal places if this is a tax rate plan. If this field is not present, the default number of decimal places is 2. If the value is 0, this is a tax table.

Record 2 for Percentage Tax

Field Name	Type	Length	Description
NumberOfRanges	ASCII	2	Number of ranges in the tax table. The value is 0 for a tax rate plan.

Record 3 for Percentage Tax

Field Name	Type	Length	Description
Reserved	ASCII	1	Reserved.
RoundingMethod	ASCII	1	Tax rounding method.
Reserved	ASCII	2	Reserved.

EAMTENDV (Tender Verification File)

The Tender Verification file is a keyed file that contains customer account data for verifying tenders. An image version of the file is maintained on all eligible controllers so that it will always be accessible to online terminals. The prime version is on the master controller. The key to the record is the tender type and the customer account number. The file can be used for positive or negative verification per tender type based on the options selected.

This file can contain either short or long format records. However, all records in the file must be the same length. The long format of the record contains additional limits to be used for verification.

Any tender type and variety which is entered with an account number can be verified against a Tender Verification file as selected by the user options. A single Tender Verification file is used for all tender types with records segregated by tender type, variety, and account number. The Tender Verification file is interpreted as a positive or negative file based on a user-specified option per tender variety. When a positive file is used, any account number not found on the file is rejected with a unique guidance code. When a negative file is used, any account number not found on the file is accepted. Whenever an account record is found on file, it is processed according to the record contents regardless of whether the file is positive or negative.

Logical name	<EAMTENDV>
Data object reference	*ISSATenderVerificationData* <`SATNVDAT.CPP`>
Organization	Keyed
Distribution class	Compound per update
File copies	1
Record length	14 bytes for the short file format; 26 bytes for the long. The first 14 bytes have the same layout in both files.
Key length	13 bytes: TenderId / AccountNumber

Field Name	Type	Length	Decimal Offset	Description
TenderId	PD	1	0	The TenderId is composed of the TenderType and TenderVariety: **X'1V'** Cash tender, variety V **X'2V'** Check tender, variety V **X'3V'** Food stamp tender, variety V **X'4V'** Misc. tender type 1, variety V **X'5V'** Misc. tender type 2, variety V **X'6V'** Misc. tender type 3, variety V V ranges from 1 to 9.
AccountNumber	PD	12	1	Customer account number.
AccountStatus	PD	1	13	Account status code: **X'50'** Accept tender **X'55'** Reject tender - risk 1 **X'56'** Reject tender - risk 2 **X'57'** Reject tender - risk 3 **X'58'** Reject tender - risk 4 **X'60'** Accept tender - no fee **X'70'** User reserved
The following data fields are included only if the long form of the file is specified:				
MaximumTransactionAmou nt	Int	4	14	Maximum value of an individual transaction against this account.
MaximumTotalAmount	Int	4	18	Limit of the sum of the transactions against this account. This field is used only if the customer

Field Name	Type	Length	Decimal Offset	Description
				account status file is selected because it accumulates the sum against which this limit is compared.
Usr	Int	4	22	Reserved for user extensions.

EAMTERMS (Terminal Status File)

The keyed Terminal Status file contains a record for each terminal and a record for the store. Checkout Support creates the file and the Store record. Terminal Sales and the Store Closing, Loan, and Pickup accounting functions add terminal records as required. A record can exist for both the current and previous periods for each terminal. Previous period records provide source data for a previous period Cash Drawer Position Report. Previous period records have 1 in the first character position of the terminal number. Current period terminal numbers have 0.

The Terminal Status file is a work file that controls data flow and passes data from one SurePOS ACE application to another. It contains such information as the current till contents and the operational status of each terminal. Reports use this file to provide cash drawer position data and system status data. Only Terminal Sales, Checkout Support, and the Store Closing, Loan, and Pickup segments of Accounting update this file.

The Terminal Status file resides on the file server controller as a mirrored file. The image version of the file is not accessed unless the alternate file server is made the file server. The file is distributed only when closed.

Having a single active version of the file provides a single point of control without undue performance impacts. Numerous control functions are lost when the file cannot be accessed because the local area network or the master controller is inoperative. Terminals can proceed in sales without this file and can restore and update the file when it becomes available again, using data saved in the terminals.

If the controller is down when an operator signs off, the eamterms file will be missing transaction data for terminals with operator accountability while the controller is down. After the controller comes back online and Checkout Support catches up on its processing, the eamterms file will be up-to-date the first time that an operator signs on.

Logical name	<EAMTERMS>
Data object references	• Store record - *ISSAStoreStatusData* <`SASTRDAT.CPP`> • Terminal record - *ISSATerminalStatusData* <`SATMSDAT.CPP`>
Organization	Keyed
Distribution class	Mirrored at close
File copies	1
Record length	72 bytes
Key length	2 bytes: Terminal

Store Record

Field Name	Type	Length	Decimal Offset	Description
Terminal	PD	2	0	The terminal ID is always 9999 for the store record.

Field Name	Type	Length	Decimal Offset	Description
SLogName	ASCII	8	2	Name of Transaction Log currently in use by the terminals
NumClose	Int	2	10	Bit flags used for control of file closes: **X'01'** Close Tender Listing File **X'02'** Close Item Movement File **X'04'** Close Terminal Productivity File **X'08'** Close Exception Log **X'10'** Close Customer Account Status File
CloseFlg	Int	1	12	True implies close of reporting period is pending but not yet completed by Sales Support. True = any nonzero value, False = 0.
DateTime	PD	5	13	Date and time of period start or of period close. Format: `YYMMDDHHmm`.
Monitor	PD	2	18	Number of last monitoring terminal.
ClseCntrl	Int	2	20	Close Control Flag.
Indicat0	Int	2	22	Close Type Flag - Bytes 0 and 1.
Indicat1	Int	2	24	Close Type Flag - Bytes 2 and 3.
Reserved	ASCII	46	26	Reserved.

Terminal Record

Field Name	Type	Length	Decimal Offset	Description
Terminal	PD	2	0	The terminal number. The first character of this field indicates whether the record is for the current period or a previous period: **0** Current period

Field Name	Type	Length	Decimal Offset	Description
				1 Previous period
Operator	PD	5	2	Operator number of current or most recent user of this terminal.
TransNum	PD	2	7	Transaction number of the current or most recent transaction.
NumLoans	Int	2	9	Number of loans to this till.
AmtLoans	Int	4	11	Amount of loans to this till.
NumPkups	Int	2	15	Number of pickups from this till.
AmtPkups	Int	4	17	Amount of pickups from this till.
The following four fields are net totals for this sign-on session:				
GrossPos	Int	4	21	Gross positive. Accumulation of the positive sales entries (sales + deposits + taxes + tender fees). Includes standalone totals but no totals from voided or training transactions.
GrossNeg	Int	4	25	Gross negative. Accumulation of the negative sales entries (deposits returns + item refunds + tax refunds + discounts + cancels of sales, deposits, item refunds, deposit returns, and tender fees). Includes standalone totals but no totals from voided or training transactions.
AmtMiscs	Int	4	29	Amount of miscellaneous item record entries.
NumTrans	Int	2	33	Number of sales transactions.
TillContents	Array	8x4	35	This represents the following eight 4-byte till content fields as of the end of the last transaction:
AmtCash	Int	4	35	Entered tender - cash
AmtCheck	Int	4	39	Entered tender - checks
AmtFoods	Int	4	43	Entered tender - food stamps
AmtMisc1	Int	4	47	Entered tender - miscellaneous tender 1
AmtMisc2	Int	4	51	Entered tender - miscellaneous tender 2
AmtMisc3	Int	4	55	Entered tender - miscellaneous tender 3
AmtManuf	Int	4	59	Entered tender - manufacturer coupons
AmtStore	Int	4	63	Entered tender - store coupons
The next three fields contain status indicators:				
TranType	PD	1	67	Transaction type in process:

Field Name	Type	Length	Decimal Offset	Description
				00 Checkout transaction **01** Tender cashing **02** Tender exchange **03** Cashier loan **04** Cashier pickup **05** Tender listing **06** Price verify/change **07** Training session **08** Terminal transfer **09** Terminal monitor **10** Tender count **12** Non-EBT WIC **13** Return item transaction. **16** Reprint tender receipt **20** EBT balance inquiry **21** Value card balance inquiry

Field Name	Type	Length	Decimal Offset	Description
				22 WIC EBT balance inquiry **80** Department totals report **99** No transaction in process
Status	Int	2	68	**Bit 7 X'8000'** UnattendedClose: Performing an unattended close. **Bit 6 X'4000'** ForceSignOff: Force a sign-off at the next opportunity. **Bit 5 X'2000'** ReOpenLog: Re-open the summary log at the next transaction. **Bit 4 X'1000'** ForceTillExchange: Force a till exchange at the next transaction start. **Bit 3 X'0800'** ReloadOptions: Reload options. **Bit 2 X'0400'** ProhibitSignOn: Prohibit sign-on. **Bit 1 X'0200'** ProhibitSignOff: Prohibit sign-off. **Bit 0 X'0100'** ProhibitTransStart: Prohibit transaction start. **Bit15 X'0080'** ProhibitTransEnd: Prohibit transaction end. **Bit14 X'0040'** TerminalMonitor: Terminal is being monitored. **Bit13 X'0020'** TerminalTransfer: Terminal has been transferred.

Field Name	Type	Length	Decimal Offset	Description
				Bit12 X'0010' TerminalSpecialSignOff: Terminal is in special sign-off. **Bit11 X'0008'** TerminalAccountability: Terminal accountability. **Bit10 X'0004'** OperatorSignedOn: Operator signed-on to terminal. **Bit 9 X'0002'** TenderVerification: Out-of-store verification of tenders is active. **Bit 8 X'0001'** ForceSignOffForClose: Forced signoff during a store close is required.
Status2	Int	2	70	**Bit 7 X'8000'** TOFReload; Reload of TOF file is required. **Bit 6 X'4000'** TOFCpnFraudReload;Reload of TOF Coupon Fraud Files is required. **Bit 5 X'2000'** Reserved. **Bit 4 X'1000'** Reserved. **Bit 3 X'0800'** Reserved. **Bit 2 X'0400'** Reserved. **Bit 1 X'0200'** Reserved. **Bit 0 X'0100'** Reserved. **Bit15 X'0080'** Reserved.

Field Name	Type	Length	Decimal Offset	Description
				Bit14 X'0040' Reserved. **Bit13 X'0020'** Reserved For User Extension - User Flag 6 Terminal Owned. **Bit12 X'0010'** Reserved For User Extension - User Flag 5 Terminal Owned. **Bit11 X'0008'** Reserved For User Extension - User Flag 4 Server Owned. **Bit10 X'0004'** Reserved For User Extension - User Flag 3 Server Owned. **Bit 9 X'0002'** Reserved For User Extension - User Flag 2 Server Owned. **Bit 8 X'0001'** Reserved For User Extension - User Flag 1 Server Owned.

EAMTLIS* (Tender Listing File)

The sequential Tender Listing file contains account numbers and amounts for tender types specified in Tender -> Tenders personalization. SurePOS ACE writes one record for each tender made for a selected tender type. Separate files exist for the current period, previous period, and one old period.

Image versions of each Tender Listing file are on the alternate file server. This distribution takes place at file close for minimal impact to performance.

Special delimiter records separate tenders for a given terminal or operator into groups based on the occurrence of a tender removal. An end-of-period record marks the end of an accounting period and is always the last record.

Logical names	<EAMTLIST>, <EAMTLISP>, <EAMTLISO>
Data object references	• Record types 1, 2, and 3 - *ISSATenderListingData* `<SATNLDAT.CPP>` • Record types 4 and 5 - *ISSACSTenderListingStorage* `<SACTLSTR.CPP>`
Organization	Variable sequential

Distribution class	Mirrored at close
File copies	Current, Previous, Old
Record length	Variable

Record Types 1 and 2

Field Name	Type	Length	Description
RecordType	ASCII	1	Record type **1** A tender record from an operator accountability terminal. **2** A tender record from a terminal accountability terminal.
TerminalId	PD	2	Terminal number
TenderType	PD	1	Tender ID, which is the one-digit tender type from 1 to 6 followed by the one-digit variety (V) from 1 to 9. Default tender types are: **1V** Cash type, variety V **2V** Check type, variety V **3V** Food stamp type, variety V **4V** Miscellaneous tender type 1, variety V **5V** Miscellaneous tender type 2, variety V **6V** Miscellaneous tender type 3, variety V
OperatorId	PD	v5	Operator number
AccountNumber	PD	v12	Customer account number.
Amount	ASCII	v10	Amount of the tender in the tender currency.
FeeAmount	ASCII	v10	Amount of the tender fee.
Usr	ASCII	v?	Reserved for user extensions.

Record Type 3 - Tender Removal (Group Delimiter)

Field Name	Type	Length	Description
RecordType	ASCII	1	Record type **3** Group delimiter.
TerminalId	PD	2	Terminal number.
TenderType	PD	1	Tender ID, which is the one-digit tender type from 1 to 6 followed by the one-digit variety (V) from 1 to 9. Default tender types are: **1V** Cash, variety V tender removal **2V** Check, variety V tender removal **3V** Food stamp, variety V tender removal **4V** Miscellaneous tender type 1, variety V tender removal **5V** Miscellaneous tender type 2, variety V tender removal **6V** Miscellaneous tender type 3, variety V tender removal
OperatorId	PD	v5	Operator ID.
TimeStamp	PD	5	Date and time of tender removal. Format: `YYMMDDHHmm`
Usr	ASCII	v?	Reserved for user extensions.

Record Types 4 and 5 - Tender Record (Period Delimiter)

Field Name	Type	Length	Description
RecordType	ASCII	1	Record Type **4** Start of period delimiter. **5** End of period delimiter.

Field Name	Type	Length	Description
DateTime	PD	5	Date and time of period open or close. Format: `YYMMDDHHmm`

EAMTMnnn (Terminal Monitor File)

The random Terminal Monitor file contains print and display lines from the terminal being monitored. There is one copy on the local controller for each terminal being monitored.

Logical names	<EAMW:><C_ROOT:>eamtm
Data object reference	*ISSAMonitorStorage* `<SAMNTSTR.CPP>`
Organization	Random
Distribution class	Local
File copies	1 per terminal being monitored
Record length	60 bytes Note: SurePOS ACE does not use the entire 60 bytes because the 2x20 display and the cash receipt each print a maximum of 40 characters.

Field Name	Type	Length	Description
Terminal	ASCII	2	Sequence indicator for record
LineType	ASCII	1	Line type: **P** Print line on receipt. **D** Display line.
LineData	ASCII	v48	Line data to print or display.

EAMTPRO* (Terminal Productivity File)

SurePOS ACE updates the sequential Terminal Productivity file hourly with records for each active terminal during that hour. Records contain data required to analyze terminal performance that can help you predict labor scheduling requirements.

Image versions of each Terminal Productivity file are on the alternate file server. Distribution takes place at file close, to minimize the performance impact.

Terminal Productivity log records might not be in chronological order. There might also be more than one record for a given hour, or a single record might contain more than one hour's worth of data. These situations occur because of LAN failures or distribution of transactions during an hour.

Logical names	<EAMTPROD>, <EAMTPROO>
Data object reference	*ISSATPStatisticsStorage* <`SATPSSTR.CPP`>
Organization	Sequential

Distribution class	Mirrored at close
File copies	Current, Old
Record length	Variable

Productivity Record

Field Name	Type	Length	Description
TerminalId	PD	2	Terminal ID.
TimeStamp	PD	5	Date and Time. Format: `YYMMDDHHmm`.
NetSalesAmount	ASCII	v8	Gross positive - gross negative for other than training, voided, or standalone transactions.
ItemCount	ASCII	v8	Net number of item entries. Includes sales, deposits, refunds, coupons, deposit returns, cancels and miscellaneous items all summed together. It does not include items from training, standalone, or voided transactions.
CouponCount	ASCII	v8	Net number of store and manufacturer coupons. This does not include coupons from training, voided, or standalone transactions.
TransactionCount	ASCII	v8	Number of transactions. This does not include training, voided, or standalone transactions.
RingTime	ASCII	v8	Total ring time in seconds. Includes the time from the first line item until balance due. Also includes the time from the previous total or tender to a subsequent item sale.
TenderTime	ASCII	v8	Total tender time in seconds. Includes the time between a total and a tender, the time between tenders, and the time between the final tender and the close of the cash drawer.
SpecialSignOffTime	ASCII	v8	Total special sign-off time in seconds. Includes the time between a special sign-off and the subsequent special sign-on.
InactiveTime	ASCII	v8	Time between transactions in seconds.
NonSalesTime	ASCII	v8	Time in seconds when non-sales procedures are being run. Voided, standalone, and training transactions are in this time.
Usr	ASCII	v?	Reserved for user extensions.

Period Delimiter Record

Field Name	Type	Length	Description
TerminalId	PD	2	The terminal number is always 9999 for a delimiter record.
PeriodTimeStamp	PD	5	Date and time of period start or of period close. Format: `YYMMDDHHmm`.

EAMTRAK* (Transaction Tracking File)

The Transaction Tracking file is a keyed file that contains information about any transactions that might have been logged in the Exception Log as missing, duplicate, or found. The file contains 100 entries per terminal for a maximum of 50 terminals.

Logical name	<EAMTRAK>, <EAMTRAKO>
Data object reference	*ISSATransactionTrackingStorage* `<SATRTSTR.CPP>`, *ISTransactionTrackingData* `<TRNSTDAT.CPP>`
Organization	Keyed
Distribution class	Mirrored per update
File copies	Current, Old
Record length	9 bytes
Key length	4 bytes: TerminalID / TransactionNumber

Field Name	Type	Length	Decimal Offset	Description
TerminalId	PD	2	0	Terminal number
TransactionNumber	PD	2	2	Transaction number
Restart	Int	4	4	The offset in the Transaction Log file of the last transaction which updated this record.
Indicat	ASCII	1	8	Transaction status: **1** Missing **2** Found **3** Duplicate **4** Duplicate, processed

EAMTRAN* (Transaction Summary Log)

The sequential Transaction Summary Log (Transaction Log) file contains a record for each transaction run at a terminal and a record for some events that occur on the controller (such as initiating a store close in accounting). Record sizes vary depending upon the size of the sales or non-sales transaction. Terminal Sales writes data to the Transaction Log only after a transaction successfully completes.

The store closing procedure, which is performed by the Checkout Support program, creates a new Transaction Log at the close of reporting periods and at periodic intervals. The same procedures purge an old copy if it is not required for user processing. Two Transaction Logs are always on the disk although Terminal Sales uses only one at any given time.

SurePOS ACE writes all Transaction Log records to the file server controller, which the 4690 OS distributes to the alternate file server. Such data duplication provides totals integrity in the event of hardware failures. SurePOS ACE purges the Transaction Log to reclaim disk space only after the 4690 OS distributes individual report files to the alternate file server controller.

Transactions in Transaction Log records are delimited by a carriage return and line feed. Strings that make up these records (such as those that represent items and tenders, for example) are delimited by commas. Substrings that make up strings (such as those that represent amounts and flags, for example) are delimited by colons. The 4690 OS delimits records and strings while SurePOS ACE delimits and concatenates substrings. Substring data is predominately packed numeric format but occasionally is ASCII numeric format. Packed numeric substrings do not contain leading zeros and are null if the value of the substring field is zero.

You can archive Transaction Log files and index files under any name. SurePOS ACE supports file names with the format eamjyymd.xxn, which is based on this naming convention (which is the 4680-4690 Supermarket Application naming convention):

eamj	Always the first four characters of the file name.
yy	The last two digits of the year. For example, 03 represents 2003.
m	The month of the year (1=1, ..., 9=9, A=10, B=11, C=12).
d	The day of the month: (1=1, ..., 9=9, A=10, ..., V=31).
xx	The type of file and type of compression: db - Uncompressed Transaction Log dc - Compressed Transaction Log dd - Uncompressed Electronic Journal Index df - Compressed Electronic Journal Index de - Uncompressed Manager's Journal Index dg - Compressed Manager's Journal Index
n	Number that starts at 0, increments for each Transaction Log file saved, and resets to 0 at the beginning of each day.

Archived Transaction Log files have a compound at close distribution type.

For each Transaction Log file that is saved, an Electronic Journal Index file and a Manager's Journal Index file is also saved. During a close, uncompressed files are saved in the archive directory until the specified number are saved. During the next close, if the compressed limit is greater than 0, the oldest file is compressed. Then, the newest file is saved uncompressed. During the next close after the compressed limit is reached, the oldest compressed file is erased.

Archived Transaction Logs and index files are saved in the directory you specify in the `Archive Directory` option in Options -> Archive personalization. You can also set the maximum number of compressed and uncompressed Transaction Log files to keep.

If you select an archived file as the basis for the Transaction Log Report, SurePOS ACE creates a temporary uncompressed copy of the file and stores the copy in the adx_idt4 directory. The naming convention for the temporary file is eamtrany.dat, where the value of *y* is 0-5. The temporary work file is kept until the next time that the report is requested for an archived file. If you start the Reports function and request multiple Transaction Log Reports using archived files, a maximum of six temporary files is kept. When you request the seventh report, the oldest temporary file is deleted.

The Transaction Log contains these strings as defined by *ISTransactionTypes*:

Table 69. Transaction Log String Types and Data Object References

String Type	Description	Class Object	Source Code File
00	Transaction header	*ISSATransactionSummaryData*	<SATRNDAT.CPP>
01	Item entry	*ISSATransactionEntryData*	<SATRNDAT.CPP>
02	Item entry extension	*ISSAItemEntryData*	<SATRNDAT.CPP>
03	Discount	*ISSADiscountEntryData*	<SATRNDAT.CPP>
04	Voided discount	*ISSADiscountEntryData*	<SATRNDAT.CPP>
05	Tender	*ISSATenderEntryData*	<SATRNDAT.CPP>
06	Tender correction	*ISSATenderEntryData*	<SATRNDAT.CPP>
07	Tax	*ISSATransactionTaxData*	<SATRNDAT.CPP>
08	Tax refund	*ISSATransactionTaxData*	<SATRNDAT.CPP>
09	Change	*ISSAChangeEntryData*	<SATRNDAT.CPP>
10	Manager override	*ISSAOverrideData*	<SATRNDAT.CPP>
11	Data entry	*ISSADataEntryData*	<SATRNDAT.CPP>
13	Till change	*ISSATillChangeData*	<SATRNDAT.CPP>
16	SurePOS ACE EPS tender	*ISEPSTenderLoggingPolicy*	<EPSTNDLP.CPP>
20	Exception log		
21	Store closing	*ISSAClosePeriodData*	<SACLPDAT.CPP>
80	WIC EBT data	*ISSAWICEBTItemData*	<SATRNDAT.CPP>
97	Extra data	*ExtraDataStringType*	<SATRNDAT.CPP>
99	User data	*ISUserEntryFactory*	<USRENFCT.CPP>

These are the Transaction Log file characteristics:

Logical name	<EAMTRANA>, <EAMTRANB>, <EAMTRANC>, <EAMTR:nn>
Data object reference	*ISTransactionTypes* <SATRNDAT.CPP>
Organization	Sequential
Distribution class	Mirrored per update
File copies	Current, Previous, Old (as compressed and uncompressed archives)
Record length	Variable

While a terminal is in Terminal Offline (TOF) mode, transaction data is stored on the terminal in the eamtlog.dat file. If the Transaction Summary Log is not valid when the terminal IPLs, the eamtlog.dat file is appended to eamtl*xxx*.dat (*xxx* is the terminal number) on the locally-attached controller.

Transaction Header String (0x00)

Every transaction record begins with a transaction header string, except for the store closing and exception log records that contain only a single string. A user exit string can also comprise a record by itself.

Note:

1. Transaction types 5, 7, 8, 9, and 17 normally contain only a header string. A sign-off record (type 17) can contain a till change string if an operator signed off an operator accountability terminal while the LAN was down. A transaction type 18 string usually has till change strings, tax strings, or both with the header string.

   For transaction types 5, 7, 8, 9, 17, and 18, the NUMSTRING field is used for data other than the number of strings in the record:

05	Tender listing contains the listed tender type (X'99' defines all that were listed)
07	Training session contains the number of transactions
08	Terminal transfer contains transferred terminal
09	Terminal monitor contains the number of the monitored terminal
17	Operator sign-off contains the number of transactions
18	Standalone session contains the number of offline transactions

2. The gross positive and gross negative fields represent the totals for the specified transaction(s). The training, standalone, and sign-off records contain the totals for the given session.
3. Tender cashing and tender exchange transaction records may contain any normally-logged string that is associated with a tender in addition to the header string. These are some of the string types that might be included:
   - 0x09 - change given
   - 0x10 - override
   - 0x16 - SurePOS ACE data
   - 0x97 - supplemental currency information
   - 0x99 - signature data
4. Loan, pickup, and tender count transaction records may contain only till change and override strings in addition to the header string.
5. Standalone transaction records may contain only tax and till change strings in addition to the header string.
6. A normal customer checkout transaction record may contain all string types except for price change, till change, exception log, and store closing strings.

Field Name	Type	Length	Description
TYPE	PD	1	String type = X'00' for header string
TERMINAL	PS	2	Terminal which created this string
TRANSNUM	PI	2	Transaction number
DATETIME	5x1 PI	5	Date and Time as printed on the receipt (Format: `YYMMDDHHmm`)
TRANTYPE	PI	1	Transaction type as defined by *ISTransactionEntryTypes*: **00** Checkout transaction **01** Tender cashing

Field Name	Type	Length	Description
			02 Tender exchange **03** Cashier loan **04** Cashier pickup **05** Tender listing **06** Price verify / change **07** Training session **08** Terminal transfer **09** Terminal monitor **10** Tender count **11** Reserved **12** Return item transaction **13** WIC transaction **14** Reserved **15** Reprint tender receipt **16** Voided checkout transaction **17** Operator sign-off **18** Standalone session

Field Name	Type	Length	Description
			20 EBT balance inquiry **21** Value card balance inquiry **22** WIC EBT balance inquiry **80** Department totals report
NUMSTRING	PD	v3	Number of strings in the logical record (zero for header with no strings)
OPERATOR	PS	v5	Operator number
PASSWORD	PS	v4	Operator password Note: When enhanced passwords are in use, this field will contain 99999999.
GROSSPOS	PL	v5	Gross positive (transaction)
GROSSNEG	PL	v5	Gross negative (transaction)
RINGTIME	PL	v4	Transaction ring time
TENDERTI	PL	v4	Tender time in seconds
SPECIAL	PL	v4	Special sign-off time
INACTIVE	PL	v4	Inactive time in seconds
INDICAT1	PL	v5	Packed string representation of four-byte integer header flags defined in `SATRNDAT.CPP`: **Bit 0 X'80000000'** Reserved. **Bit 1 X'40000000'** TOF Recovered. **Bit 2 X'20000000'** Transaction copied after previous TLog close string. **Bit 3 X'10000000'** CVB Paper: WIC transaction. **Bit 4 X'08000000'** Reserved.

Field Name	Type	Length	Description
			Bit 5 X'04000000' SelfCheckoutTerminal Transaction run on self checkout lane. **Bit 6 X'02000000'** Digital Receipt Created. A digital version of sales transaction receipt was created. **Bit 7 X'01000000'** Paper Receipt Suppressed. The sales transaction paper receipt was not printed. **Bit 8 X'00800000'** EatInTransaction: Eat-in transaction. **Bit 9 X'00400000'** PreferredCustomer: Preferred customer transaction. **Bit 10 X'00200000'** GROSSPOS: Gross positive field is negative. **Bit 11 X'00100000'** GROSSNEG: Gross negative field is negative. **Bit 12 X'00080000'** AdditionalRecords: More records exist for transaction. **Bit 13 X'00040000'** NotFirstRecord: Not first record for transaction. **Bit 14 X'00020000'** TillChange: Signoff record with till change string due to LAN failure. **Bit 15 X'00010000'** TermInitialized: Initialized before or during transaction.
INDICAT1	PL	v5	**Bit 16 X'00008000'** RollbackPriceItem **Bit 17 X'00004000'** Reserved.

Field Name	Type	Length	Description
			Bit 18 X'00002000' TransTransferred: Transaction transferred. **Bit 19 X'00001000'** TransactionMonitored: Transaction monitored. **Bit 20 X'00000800'** TenderRemoval: No-sale tender removal before or during transaction. **Bit 21 X'00000400'** TillContents: No-sale till contents report before this transaction. **Bit 22 X'00000200'** TillExchanged: No-sale till exchange before this transaction. **Bit 23 X'00000100'** TenderVerified: No-sale tender verification before transaction. **Bit 24 X'00000080'** NewPassword: New password with operator sign-on. Note: This bit is set whenever the signon includes a new password, regardless of the password type used (simple or enhanced). **Bit 25 X'00000040'** OperatorSignon: Operator sign-on prior to transaction. **Bit 26 X'00000020'** TenderRejected: Tender rejected in transaction. **Bit 27 X'00000010'** SignoffIsFalse: *TRANTYPE 17*: Not true sign-off; NRTs logged. *TRANTYPE 00*: Coupons tracked in eamcoupc.dat. **Bit 28 X'00000008'** DataEntry: In this transaction in next string.

Field Name	Type	Length	Description
			Bit 29 X'00000004' OpenDrawer: Opened before or during transaction. **Bit 30 X'00000002'** SpecialSignoff: Before or in transaction. **Bit 31 X'00000001'** TerminalAcct: Terminal accountability.
ALTPRICE	PD	1	Specifies the type of pricing data being used. 00 = Primary pricing data; 01 = Alternate pricing data
VOIDTRC	PD	v2	Void Transaction Reason Code: • 0: Manual/Normal Void • 100: Pump Test • 101: Suspended Transaction • 102: Terminal Transfer • 103: Suspended Transaction (Local)

Item Entry String (0x01)

This string is contained in customer checkout transaction types only. It is defined by the *ISSATransactionEntryData* class object.

Field Name	Type	Length	Description
TYPE	PD	1	String type = X'01' for an item entry
ITEMCODE	PD	v7	Item code Note: For a GS1 DataBar coupon, the item code is `811099999999` packed.
XPRICE	PD	v5	Extended price
DEPARTME	PD	v2	Department number (value 1 - 999)
FAMILYNU (CURRENT)	PD	1.5	Coupon family number. A coupon with an equal family number can be applied to this item (value 0-999) or a miscellaneous account number to which the item amount is added. If this is a miscellaneous item (value 0-999), 0 implies no update.
FAMILYNU (PREVIOUS)	PD	1.5	Coupon family number. A coupon with an equal family number can be applied to this item (value 0-999) or a miscellaneous account number from which the item amount is subtracted. If this is a miscellaneous item (value 0 - 999), 0 implies no update.
INDICAT1	PD	v4	Packed string representation of four-byte integer:

Field Name	Type	Length	Description
			Bit 0 X'80000000' through Bit 11 X'00100000' Reserved. **Bit 0 X'80000000' through Bit 12 X'00080000'** Free item coupon. **Bit 13 X'00040000'** GS1DataBar: GS1 DataBar. (Indicates an X'99' string follows that contains the value of the GS1 DataBar.) Currently the Non-GS1 barcodes are logged with this flag, when the barcode exceeds the item lookup key length (12 or 14 digits) depending on whether a 6-byte or 7-byte keyed item file is used. **Bit 14 X'00020000'** RainCheckItem: Rain check item. **Bit 15 X'00010000'** FuelItem: Fuel item. **Bit 16 X'00008000'** RollbackPriceItem: Roll Back Price Item **Bit 17 X'00004000'** WeightItem: Weight item. **Bit 18 X'00002000'** PriceEntered: Entered price used. **Bit 19 X'00001000'** PriceRequired: Price required. **Bit 20 X'00000800'** LogAll: Log all occurrences. **Bit 21 X'00000400'** LogEx: Log item as exception. **Bit 22 X'00000200'** Alias: Item code from alias record. **Bit 23 X'00000100'** NoMovement: No item movement kept. **Bit 24 X'00000080'** TaxableA: Tax plan A applies to this item. **Bit 25 X'00000040'** TaxableB: Tax plan B applies to this item.

Field Name	Type	Length	Description
			Bit 26 X'00000020' TaxableC: Tax plan C applies to this item. (Coupon items only-minimum purchase by manufacturer indicator.) **Bit 27 X'00000010'** TaxableD: Tax plan D applies to this item. (Coupon items only-minimum purchase by department indicator.)
INDICAT1 (continued)	PD	v4	**Bit 28 X'00000008'** FoodStamp: Food stamp item. **Bit 29 X'00000004'** PointsItem: Points item. (Coupon items only-exclude item from minimum purchase indicator.) **Bit 30 X'00000002'** NonDiscountable: Non-discountable item. (Coupon items only-redemption item indicator.) **Bit 31 X'00000001'** CpnMultAllowed: Coupon multiplication not allowed.
INDICAT2	PD	v3	Packed string representation of two-byte integer: **Bit 7 X'8000'** Reserved. **Bit 6 X'4000'** OverrideRqd: Item requires override for exceeding price limit when Price key hit. **Bit 5 X'2000'** DataEntry: Data entry with item. **Bit 4 X'1000'** Multipriced: Multipriced item. **Bit 3 X'0800'** WeightOrQtyOrVolume: Weight or quantity or volume entered, or points-only item indicator. **Bit 2 X'0400'** CouponMult: Store coupon created by coupon multiplier. **Bit 1 X'0200'** TaxKey: Tax key pressed. **Bit 0 X'0100'** FoodstampKey: Food stamp key pressed.

Field Name	Type	Length	Description
			Bit 15 X'0080' CancelKey: Cancel key pressed. **Bit 14 X'0040'** RefundKey: Refund key pressed. **Bit 13 X'0020'** StoreCouponKey: Store coupon key pressed. **Bit 12 X'0010'** MfrCouponKey: Manufacturer coupon key hit. **Bit 11 X'0008'** DepositKey: Deposit key pressed. **Bit 10 X'0004'** XPRICE: Negative price due to deal. **Bit 9 X'0002'** ExtensionExists: Extension follows this string. **Bit 8 X'0001'** Reserved
INDICAT3	PD	1	Item type (T) and operator entry method (O), in format TO: **T=0** Normal item sale or bonus item sale **T=1** Deposit **T=2** Refund **T=3** Deposit return **T=4** Misc. trans. receipt (sale) **T=5** Misc. trans. payout (refund) **T=6** Manufacturer coupon **T=7** Store coupon **T=8** Item sale cancel

Field Name	Type	Length	Description
			T=9 Deposit cancel **O=0** Scanned item code **O=1** Keyed item code **O=2** Item lookup key used **O=3** Item code linked to **O=4** Reserved **O=5** Item created by service **O=6-7** Reserved **O=8** Redemption of points **O=9** Bonus points
INDICAT4	PD	V4	Packed string representation of a four-byte integer. **Bit 31** Reserved **Bits 27-30** Reserved for NRSC Use **Bit 26** Reserved **Bit 25 X'02000000'** WIC Forgiven Fee Void **Bit 24 X'01000000'** Foodstamp Forgiven Fee **Bits 8-23** Reserved **Bit 7 X'00000080'** Tax Plan E

Field Name	Type	Length	Description
			Bit 6 X'00000040' Tax Plan F **Bit 5 X'00000020'** Tax Plan G **Bit 4 X'00000010'** Tax Plan H **Bit 3 X'00000008'** Automatic Void **Bit 2 X'00000004'** Markdown Coupon **Bit 1 X'00000002'** Qualified Healthcare Product (QHP) **Bit 0 X'00000001'** Prescription (Rx) Item
OrdinalNumber	PD	V3	The base ordinal number of the entry. If this is a multi-quantity entry, it takes up an ordinal number for each individual item starting at the base ordinal number.
VoidedOrdinalNumber	PD	V3	The base ordinal number of the original item being voided. If this is a multi-quantity entry, it takes up an ordinal number for each individual item starting at the base ordinal number. If a linked item has a voided ordinal number, each item (not coupon) in that link chain following that item will also have a voided ordinal number. This new field will be empty (only an extra ":" (colon) at the end of the record) if the entry is not a programmatic void or touch void by ordinal number entry.

Item Entry Extension String (0x02)

This string is written for each item entry that is multipriced, had a price override, or had a weight, quantity, or volume entered. It always follows the item entry string to which it applies. It is defined by the *ISSAItemEntryData* <`SATRNDAT.CPP`> class object.

Field Name	Type	Length	Description
TYPE	PD	1	String type = X'02' for an item entry extension
MPGROUP	PD	v1	Multipricing group number / Mix and match number (used when a combination of items comprise a deal; value 0-99, 0 = mix only with self)
DEALQUAN	PD	v4	Number of items in the deal up to this point

Field Name	Type	Length	Description
METHOD	PD	v1	Pricing method as defined by the *ISPricingMethodPolicy* class <`PRCNGPLC.CPP`>: **0** Unit or split package pricing **1** Base + 1 pricing **2** Group threshold pricing **3** Group adjusted (reduced deal) pricing **4** Unit adjusted (increased deal) pricing Note: If an item uses the alias pricing method, the pricing method recorded here is taken from the item code it is aliasing.
SALEQUAN	PD	v1	Pricing quantity from the item record count or weight of items in the deal (value 0-99, 0 is equivalent to 1).
SALEPRIC	PD	v5	Pricing data from the item record: Pricing method 0 or 1: format 00pppppppp where: **pppppppp** = unit price if SALEQUAN = 0 or 1 = deal price if SALEQUAN > 1 Pricing method 2: format pppppsssss where: • ppppp = package deal price • sssss = single unit price Pricing method 3 or 4: format uuuuurrrrr where: • uuuuu = unit price • rrrrr = reduced price
QTYORWGTORVOL	PD	v5	Entered quantity, weight, or volume
INDICAT1	PD	v1	Packed string representation of one-byte integer **Bit 7 X'80'** DealQtyKeyed: Deal Quantity was keyed. **Bit 6 X'40'** WeightOrVolumeKey: Weight or volume was keyed.

Field Name	Type	Length	Description
			Bit 5 X'20' QtyKey: Quantity was keyed. **Bit 4 X'10'** TareKey: Tare code was keyed. **Bit 3 X'08'** ScaleWeight: Weight came from scale. **Bit 2 X'04'** QtyRqd: Quantity required. **Bit 1 X'02'** RepeatQty: Price includes discount. **Bit 0 X'01'** CouponQty: Coupon enlarged quantity.
TareWeight	PD	v5	Tare weight if a weight item.

Discount or Voided Discount String (0x03, 0x04)

These strings are contained in customer checkout transaction records only. It is defined by the *ISSADiscountEntryData* <`SATRNDAT.CPP`> class object.

Field Name	Type	Length	Description
TYPE	PD	1	String type: **X'03'** Discount **X'04'** Voided discount
DISGROUP	PI	1	Discount group number (1–99).
DISRATE	PI	v2	Discount rate (0–100).
AMOUNT	PL	v4	Discount amount.
TAXEXEMP	PL	v4	Tax amount exempted.
TAXABLEEXEMPTAMOUNT	PL	v4	Taxable amount exempted
TAXEXEMPTID	ASCII	v20	Tax exempt ID
TAXEXEMPTAMOUNTA	PL	v4	Amount of tax A exempted
TAXEXEMPTAMOUNTB	PL	v4	Amount of tax B exempted
TAXEXEMPTAMOUNTC	PL	v4	Amount of tax C exempted
TAXEXEMPTAMOUNTD	PL	v4	Amount of tax D exempted
TAXABLEEXEMPTAMOUNTA	PL	v4	Taxable A amount exempted

Field Name	Type	Length	Description
TAXABLEEXEMPTAMOUNTB	PL	v4	Taxable B amount exempted
TAXABLEEXEMPTAMOUNTC	PL	v4	Taxable C amount exempted
TAXABLEEXEMPTAMOUNTD	PL	v4	Taxable D amount exempted
TAXEXEMPTAMOUNTE	PL	v4	Amount of tax E exempted
TAXEXEMPTAMOUNTF	PL	v4	Amount of tax F exempted
TAXEXEMPTAMOUNTG	PL	v4	Amount of tax G exempted
TAXEXEMPTAMOUNTH	PL	v4	Amount of tax H exempted
TAXABLEEXEMPTAMOUNTE	PL	v4	Taxable E amount exempted
TAXABLEEXEMPTAMOUNTF	PL	v4	Taxable F amount exempted
TAXABLEEXEMPTAMOUNTG	PL	v4	Taxable G amount exempted
TAXABLEEXEMPTAMOUNTH	PL	v4	Taxable H amount exempted

Tender or Tender Correction String (0x05, 0x06)

Both strings have the same format. These strings are contained in customer checkout, tender cashing, and tender exchange transaction records. It is defined by the *ISSATenderEntryData* <`SATRNDAT.CPP`> class object.

Field Name	Type	Length	Description
TYPE	PD	1	String type: **X'05'** Tender **X'06'** Tender correction
TENDTYPE	PI	1	Tender ID, which is the one-digit tender type from 1 to 8 followed by the one-digit variety (V) from 1 to 9. Default tender types are: **1V** Cash type, variety V **2V** Check type, variety V **3V** Food stamp type, variety V **4V** Miscellaneous tender type 1, variety V **5V** Miscellaneous tender type 2, variety V

Field Name	Type	Length	Description
			6V Miscellaneous tender type 3, variety V **7V** Manufacturer coupon tender type, variety V **8V** Store coupon tender type, variety V
AMTTENDE	PL	v5	Tender amount in native currency
AMTTNFEE	PL	v2	Tender cashing fee amount
CUSTOMER	PS	v12	Customer account number
STATUS	PI	1	Account status code **X'00'** None **X'50'** Accept tender **X'51'** Reject tender - risk 1 **X'52'** Reject tender - risk 2 **X'53'** Reject tender - risk 3 **X'54'** Reject tender - risk 4 **X'60'** Accept tender - no fee

Tax or Tax Refund String (0x07, 0x08)

Both strings have the same format. These strings are contained in customer checkout transaction records only. It is defined by *ISSATransactionTaxData* <`SATRNDAT.CPP`>.

Field Name	Type	Length	Description
TYPE	PD	1	String type: **X'07'** Normal tax **X'08'** Tax refund

Field Name	Type	Length	Description
AMTTAXA	PL	v4	Amount of tax A paid.
AMTTAXB	PL	v4	Amount of tax B paid.
AMTTAXC	PL	v4	Amount of tax C paid.
AMTTAXD	PL	v4	Amount of tax D paid.
AMTSALEA	PL	v4	Taxable A amount.
AMTSALEB	PL	v4	Taxable B amount.
AMTSALEC	PL	v4	Taxable C amount.
AMTSALED	PL	v4	Taxable D amount.
AMTTAXE	PL	v4	Amount of tax E paid.
AMTTAXF	PL	v4	Amount of tax F paid.
AMTTAXG	PL	v4	Amount of tax G paid.
AMTTAXH	PL	v4	Amount of tax H paid.
AMTSALEE	PL	v4	Taxable E amount.
AMTSALEF	PL	v4	Taxable F amount.
AMTSALEG	PL	v4	Taxable G amount.
AMTSALEH	PL	v4	Taxable H amount.

Change String (0x09)

This string is contained in customer checkout transaction records when change is given to the customer. It is defined by the *ISSAChangeEntryData* class object.

Field Name	Type	Length	Description
TYPE	PD	1	String type = X'09' for change
TENDTYPE	PI	1	Tender ID, which is the one-digit tender type from 1 to 8 followed by the one-digit variety (V) from 1 to 9. Default tender types are: **1V** Cash type, variety V **2V** Check type, variety V **3V** Food stamp type, variety V **4V** Miscellaneous tender type 1, variety V **5V** Miscellaneous tender type 2, variety V

Field Name	Type	Length	Description
			6V Miscellaneous tender type 3, variety V **7V** Manufacturer coupon tender type, variety V **8V** Store coupon tender type, variety V
AMTCHANGE	PL	v4	Change amount

Manager/Operator Override String (0x10)

This string appears in any transaction record which required a manager or operator override. It normally precedes the string that required the override. It is defined by *ISSAOverrideData* <`SATRNDAT.CPP`>.

Field Name	Type	Length	Description
TYPE	PD	1	String type = X'10' for a manager or operator override
OVERRIDE	PS	v4	Manager or operator override number
REASON	PI	v128	Override reason as defined by the *ISSAContraintReasonCode* class <SACNSTRR.HPP>. There can be more than one of these reasons: 01 Exceed price override limit 02 Fall short of minimum price limit 03 Exceed negative entry limit 04 Fall short of minimum misc. item limit 05 Exceed item/misc. limit 06 Cancel item not sold in transaction 07 Accept coupon for item not sold 08 Enter coupon/refund with Department Key 09 Exceed weight override limit 10 Multiply coupon value > item value 11 Item not for sale 12 EMV Security - No AID tendered with non-EMV card 13 Date of birth bypass 14 Exceed volume override limit 15 Exceed discount rate limit 16 Exceed discount amount limit 17 Exceed tax exemption limit 18 Bypass tax exemption ID 19 Invalid Alpha customer ID 20 Bypass tender franking 21 Exceed negative entry limit for trans. 22 Exceed negative total limit for trans. 23 EMV Security - No AID tendered with EMV card. 24 Exceed tender type amount limit 25 Exceed tender type change limit 26 Bypass tender verification

Field Name	Type	Length	Description
			27 Exceed tender verification rejection 28 Perform delayed override 29 Bypass tender verification access failure 30 Item void in negative transaction 31 Item not on scale correctly 32 Accept tender that is not allowed 33 Expired License 34 Keyed Date of Birth 35 Exceed void transaction limit 36 Coupon start date not met 37 Coupon expiration date exceeded 38 Coupon retailer ID not allowed 39 Coupon weight not met 40 Exceed tender exchange limit 41 Bypass tender exchange ranking 42 Exceed tender fee refund limit 44 Special sign-off not authorized 45 Exit special sign-off without password 46 No-sale procedure inside transaction 47 Exceed till exchange limit 48 EMV Security - EMV Chip Error tendered with non-EMV card 49 EMV Security - EMV Chip Error tendered with EMV card 50 Sign-on when operator already active 51 Bypass operator auth. access failure 52 Re-initialize transferred terminal 55 Exceed cashier loan limit 56 Exceed cashier pickup limit 57 Exceed maximum number of suspends allowed 58 Force retrieve is required 59 Force suspend is required 60 No electronic signature 61 Keyed account number 62 SurePOS ACE EPS declined 63 SurePOS ACE EPS limits exceeded

Field Name	Type	Length	Description
REASON	PI	v128	64 GIPC Down 65 Bypass paper error 66 More points needed 67 Keyed customer ID 68 New Customer ID 69 Daily card use limit exceeded 70 Non-sale price verify/change 71 Duplicate prescription 72 Void prescription without a previous sale 73 Return prescription without a previous sale 74 Prescription information not found 75 Pharmacy application inaccessible 76 Reprint tender receipt 77 Operator age restriction overridden. 78 MICR not present. 79 Low MICR signal. 80 Account number

Field Name	Type	Length	Description
			81 Transit code 82 Check number 83 Foreign transit 85 Exceed rain check override limit 86 Preferred customer not present in transaction 87 Tender Restriction 88 Item Sell By Date Past 89 Immediate quantity restriction The following reason codes are reserved for customers and Toshiba Global Commerce Solutions Business Partners: **90-96** Toshiba Global Commerce Solutions Business Partners **97-99** Customers
Index	PD	1	The index in Personalization of the option that contains this manager override ID.
Initials	ASCII	v3	The initials defined in Personalization for this manager override ID.

Data Entry String (0x11)

This string can be present in any transaction record, following a header, item entry, or tender in a customer checkout transaction. It contains data entered for the transaction, item, or tender.

It is defined by *ISSADataEntryData* <`SATRNDAT.CPP`>.

Note: Only entered data fields are present, up to the highest numbered field that was entered. Thus, for an item sale or tender, only data field 1 is present. For a response to a prompt for user data that was specified in Personalization, data field 1 is empty.

Field Name	Type	Length	Description
TYPE	PD	1	String type = X'11' for a data entry
Data1	PS	v10	User defined numeric data field 1
Data2	PS	v10	User defined numeric data field 2
Data3	PS	v10	User defined numeric data field 3
Data4	PS	v10	User defined numeric data field 4
Data5	PS	v10	User defined numeric data field 5
Data6	PS	v10	User defined numeric data field 6

Preferred Customer Data Entry Strings (0x11)

The Loyalty Program (when enabled) writes strings to the Data Entry string. The first two bytes of Data1 indicate the type of string as shown in the following list. “EAMPAN* (Preferred

Customer Panel Diary File)" on page 320 describes the layout of the data after the Loyalty Program extracts it from the Transaction Log and reformats it for the Panel Diary file.

Table 70. Preferred Customer 0x11 Data String Identifiers

Identifier		
Hexadecimal Value	ASCII Value	Description
0xBB	;;	"New Tender Verification Account Number String (0xBB)" on page 384
0xBC	;<	Reserved for NRSC use
0xBD	;=	Reserved for NRSC use
0xBE	;>	Reserved for NRSC use
0xBF	;?	Reserved for NRSC use
0xCC	<<	"Automatic Coupon Transaction Data String (0xCC)" on page 384
0xC1	<1	Reserved for customers or Toshiba Global Commerce Solutions Business Partners
0xC2	<2	Reserved for customers or Toshiba Global Commerce Solutions Business Partners
0xC3	<3	Reserved for customers or Toshiba Global Commerce Solutions Business Partners
0xC4	<4	Reserved for customers or Toshiba Global Commerce Solutions Business Partners
0xC5	<5	Reserved for customers or Toshiba Global Commerce Solutions Business Partners
0xCE	<>	"Bonus/Redemption Points String (0xCE)" on page 385
0xDB	=;	"Used Targeted Coupons String (0xDB)" on page 389
0xDD	==	"Coupon Tracking String (0xDD)" on page 390
0xDE	=>	"Preferred Customer Secondary Points Transaction Data String (0xDE)" on page 390
0xEB	>;	Reserved for customers or Toshiba Global Commerce Solutions Business Partners
0xEC	><	Reserved for customers or Toshiba Global Commerce Solutions Business Partners
0xED	>=	Reserved for customers or Toshiba Global Commerce Solutions Business Partners
0xEE	>>	"Preferred Customer Transaction Data String (0xEE)" on page 391. When used for offline transactions, this string contains the raincheck ID.
0xEF	>?	Reserved for NRSC use
0xFF	?	"Customer Identification String (0xFF)" on page 391. When used for offline transactions, this string contains the customer data.

Each string, except those that are reserved for use by the NRSC or the customer, is described in the sections that follow.

New Tender Verification Account Number String (0xBB)

This data entry string shows a new tender verification account number from a manual override of the account number from the Preferred Customer Activity file. This string applies only to preferred customer transactions. The new alternate account number replaces the old one in the Preferred Customer Activity file. This string usually follows the tender string for the new account number used.

Field Name	Type	Lengt h	Description
StringType	PD	1	Data entry string type of 0x11.
Identifier	PD	1	Two characters, 0xBB, to define the substring.
AccountNumber	PD	v9	New Tender Verification account number.
Unused	PD	2	NULL string.

Automatic Coupon Transaction Data String (0xCC)

This data entry string shows the count and amount of automatic (linked-to) store or manufacturer coupons in a transaction. This string can be in a non-preferred customer transaction as well as a preferred customer transaction because automatic coupons can apply to all customers.

This information is critical to the function that automatically picks up paperless coupons.

Field Name	Type	Lengt h	Description
StringType	PD	1	Data entry string type of 0x11.
Identifier	PD	1	Two characters, 0xCC, to define the substring.
MfgCouponAmount	PD	v3	Amount of automatic manufacturer coupons.
StoreCouponAmount	PD	v3	Amount of automatic store coupons and double coupon amount according to this formula: `StoreCouponAmount = StoreCouponAmount + DoubleCouponAmount`
MfgCouponCount	PD	v3	Count of automatic manufacturer coupons.
StoreCouponCount	PD	v3	Count of automatic store coupons and double coupon count according to this formula: `StoreCouponCount = StoreCouponCount + DoubleCouponCount`
Unused	PD	1	NULL string.

Bonus/Redemption Points String (0xCE)

This data entry string shows information associated with items or coupons that give bonus points or that require the redemption of preferred customer points. A bonus points string can appear in a regular customer transaction, but it has no meaning outside of a preferred customer transaction. This string replaces the item entry string that normally accompanies an item or coupon.

Note: The INDICAT1 and INDICAT2 fields in this string contain the hexadecimal representation of these flag fields. These same fields in the eampanel would be represented by decimal values. For example, a value of X'2000' for INDICAT1 here would be represented in the eampanel file as decimal 8192.

Field Name	Type	Lengt h	Description
StringType	ASCII	2	Data entry string type of 0x11.
Identifier	ASCII	2	Two characters, 0xCE, to define the substring.
ItemCode	PD	7	Item code for bonus item/coupon.
Value	PD	4	Points associated with item/coupon.
MoreFlags	PD	1	• Bits 0-1 - Reserved • Bits 2-6 - Unused • Bit 7 - X'1' = Linked item; X'0' = Keyed item.
Families	PD	v3	Family codes for item/coupon. The first three bytes represent the current coupon family number and the last 3 bytes represent the previous coupon family number as defined in the item record.
Indicat1	PD	v4	Decimal value of INDICAT1 in Transaction Log string type 01: **Bit 0 X'80000000' through Bit 14 X'00020000'** Reserved. **Bit 15 X'00010000'** FuelItem: Fuel item. **Bit 16 X'00008000'** Reserved. **Bit 17 X'00004000'** WeightItem: Weight item. **Bit 18 X'00002000'** PriceEntered: Entered price used. **Bit 19 X'00001000'** PriceRequired: Price required. **Bit 20 X'00000800'** LogAll: Log all occurrences. **Bit 21 X'00000400'** LogEx: Log item as exception.

Field Name	Type	Lengt h	Description
			Bit 22 X'00000200' Alias: Item code from alias record. **Bit 23 X'00000100'** NoMovement: No item movement kept. **Bit 24 X'00000080'** TaxableA: Tax plan A applies to this item. **Bit 25 X'00000040'** TaxableB: Tax plan B applies to this item. **Bit 26 X'00000020'** TaxableC: Tax plan C applies to this item. (Coupon items only-minimum purchase by manufacturer indicator.) **Bit 27 X'00000010'** TaxableD: Tax plan D applies to this item. (Coupon items only-minimum purchase by department indicator.) **Bit 28 X'00000008'** FoodStamp: Food stamp item. **Bit 29 X'00000004'** PointsItem: Points item. (Coupon items only-exclude item from minimum purchase indicator.) **Bit 30 X'00000002'** NonDiscountable: Non-discountable item. (Coupon items only-redemption item indicator.) **Bit 31 X'00000001'** CpnMultAllowed: Coupon multiplication allowed.
Indicat2	PD	v3	Decimal value of INDICAT2 in Transaction Log string type 01: **Bit 7 X'8000'** Reserved. **Bit 6 X'4000'** OverrideRqd: Item requires override. **Bit 5 X'2000'** DataEntry: Data entry with item.

Field Name	Type	Lengt h	Description
			Bit 4 X'1000' Multipriced: Multipriced item. **Bit 3 X'0800'** WeightOrQty: Weight or quantity entered, or points-only item indicator. **Bit 2 X'0400'** CouponMult: Store coupon created by coupon multiplier. **Bit 1 X'0200'** TaxKey: Tax key pressed. **Bit 0 X'0100'** FoodstampKey: Food stamp key pressed. **Bit 15 X'0080'** CancelKey: Cancel key pressed. **Bit 14 X'0040'** RefundKey: Refund key pressed. **Bit 13 X'0020'** StoreCouponKey: Store coupon key pressed. **Bit 12 X'0010'** MfrCouponKey: Manufacturer coupon key hit. **Bit 11 X'0008'** DepositKey: Deposit key pressed. **Bit 10 X'0004'** XPRICE: Negative price due to deal. **Bit 9 X'0002'** ExtensionExists: Extension follows this string. **Bit 8 X'0001'** Preferred customer coupon might apply to this item.
Indicat3	PD	v1	Hexadecimal value of Indicat3 in Transaction Log string type 01, where item type (T) and operator entry method (O) are in the format TO: **T=0** Normal item sale or bonus item sale **T=1** Deposit

Field Name	Type	Lengt h	Description
			T=2 Refund **T=3** Deposit return **T=4** Misc. trans. receipt (sale) **T=5** Misc. trans. payout (refund) **T=6** Manufacturer coupon **T=7** Store coupon **T=8** Item sale cancel **T=9** Deposit cancel **T=10** Markdown store coupon **O=0** Scanned item code **O=1** Keyed item code **O=2** Item lookup key used **O=3** Item code linked to **O=4-7** Reserved **O=8** Redemption of points **O=9** Bonus points **O>9** Reserved

Field Name	Type	Lengt h	Description
Indicat4	PD	v6	Decimal value of Indicat4 in Transaction Log string type 01: **Bits 8-31** Reserved **Bit 7 X'00000080'** Tax Plan E **Bit 6 X'00000040'** Tax Plan F **Bit 5 X'00000020'** Tax Plan G **Bit 4 X'00000010'** Tax Plan H **Bits 2-3** Reserved **Bit 1 X'00000002'** Qualified Healthcare Product (QHP) **Bit 0 X'00000001'** Prescription (Rx) Item
Unused	PD	1	NULL string.

Used Targeted Coupons String (0xDB)

This string contains the coupon codes of targeted coupons that were redeemed during the transaction.

Field Name	Type	Lengt h	Description
StringType	PD	1	Data entry string type of 0x11.
Identifier	PD	1	Two characters, 0xDB, to define the substring.
CustomerAccountID	PD	v9	Customer account number
TargetedCoupon	PD	2	Coupon number of targeted coupon awarded during the transaction. This field is repeated once for each targeted coupon that was awarded. SurePOS ACE allows a maximum of 96 targeted coupons. Actual maximum for the store is specified in personalization. Coupon numbers are separated by colons (:).

Coupon Tracking String (0xDD)

If you select the *Log coupons in tracking file* option in Options -> Coupon -> Tracking personalization and if the coupon is in the range you are logging to the Coupon Tracking file, SurePOS ACE writes a data entry string to show the logging requirement and to provide extra data to log. This string type immediately follows the coupon string to which it applies.

Field Name	Type	Length	Description
StringType	PD	1	Data entry string type of 0x11.
Unused	PD	1	NULL string.
Identifier	PD	1	Two characters, 0xDD, to define the substring.
LogFlags	PD	v3	Flags to be logged in the EAMCOUPC record. This field corresponds to the ValidFlags field in the EAMCOUPC record.
CampaignNumber	PD	v6	**Non-GS1 DataBar coupons** Campaign number, if keyed in **GS1 DataBar coupons** Offer code from the bar code
MfgNumber	PD	v6	Manufacturer number from item record. Note: For GS1 DataBar coupons, if the coupon had the purchase requirements specified with OR conditions, then the purchase requirement used to satisfy the coupon is used for this field. For purchase requirements specified with AND conditions, the primary purchase requirement is used for this field. For purchase requirements specified with AND/OR conditions (APRC 3), the first purchase requirement that has a quantity greater than zero is used for these fields.
PromotionCode	PD	v3	Promotion code used to validate the coupon.

Preferred Customer Secondary Points Transaction Data String (0xDE)

Field Name	Type	Length	Description
StringType	PD	1	Data entry string type of 0x11.
Identifier	PD	1	Two characters, 0xDE, to define the substring.
ClubNumber	PD	1	Secondary club number (1-90).
Points	PD	v5	Secondary points earned for transaction (including bonus).

Field Name	Type	Lengt h	Description
RedeemedPoints	PD	5	Secondary points redeemed during the transaction.
BonusPoints	PD	v5	Secondary bonus points given during the transaction.
Sales	PD	v5	Sales on which secondary points were earned.

Preferred Customer Transaction Data String (0xEE)

Field Name	Type	Lengt h	Description
StringType	PD	1	Data entry string type of 0x11.
Identifier	PD	1	Two characters, 0xEE, to define the substring.
CustomerAccountID	PD	v9	Final customer account number.
Points	PD	v5	Primary points earned for transaction (including bonus).
CouponAmount	PD	v3	Automatic preferred coupon total amount.
CouponCount	PD	v1	Automatic preferred coupon total count.
MessageCount	PD	v1	Number of messages given from activity file.
TransferredTransCount	PD	v1	Count of transferred transactions.
TransferredTransAmoun t	PD	v5	Amount of transferred transactions.
BonusPoints	PD	v5	Primary and secondary bonus points given in the transaction.
RedeemedPoints	PD	v5	Primary points redeemed during the transaction.
Entry Method	PD	v2	0=Internal 1=Keyboard 2=OCR 3=Scanner 4=Wand 5=MSR 6=MICR 1000-1999=Reserved for Toshiba BPs 2000-2999=Reserved for customers

Customer Identification String (0xFF)

The customer identification string is actually two data entry strings in each Panel Diary file record for preferred customers. The first occurs when the cashier enters a customer number and can occur more than once in a transaction for multiple customer numbers, or if the same number is entered multiple times. The second data entry string occurs at the end of the transaction, immediately preceding the string for the final tender in a preferred customer transaction.

Field Name	Type	Lengt h	Description
StringType	PD	1	Data entry string type of 0x11.
Identifier	PD	1	Two characters, 0xFF, to define the substring.
CustomerAccountId	PD	v9	Entered customer account number.
NosaleData	PD	v2	Data keyed with NOSALE key (default = "0") Expiration date (YYMM) if MSR entry of customer number.
StoreNumber	PD	v2	Store number printed on customer receipt.
Entry Method	PD	v2	0=Internal 1=Keyboard 2=OCR 3=Scanner 4=Wand 5=MSR 6=MICR 1000-1999=Reserved for Toshiba BPs 2000-2999=Reserved for customers
Unused	PD	1	NULL string.

Till Change String (0x13)

This string may be contained in a Loan, Pickup, Tender Count, Standalone, or Sign-off transaction record. Amounts are positive if they represent money into the till and are negative if they represent money out of the till. Thus, pickup amounts are normally negative and loan amounts are normally positive. It is defined by *ISSATillChangeData* <`SATRNDAT.CPP`>.

This string is logged in the Sign-off transaction record only if both of the following conditions exist:

- The operator is signing off a terminal that has operator accountability.
- The master controller is down.

When this string is logged in the Sign-off transaction record, tender ID totals are combined by like tender type. They are not split out by variety, as happens with loans and pickups.

Field Name	Type	Length	Description
TYPE	PD	1	String type = X'13' for a till change
NUMLOANS	PI	v2	Number of loans to this till
AMTLOANS	PL	v4	Amount of loans to this till
NUMPKUPS	PI	v2	Number of pickups from this till
AMTPKUPS	PL	v4	Amount of pickups from this till

Field Name	Type	Length	Description
TENDTYP1	PI	1	ID of first positive tender. The tender ID is the one-digit tender type from 1 to 8 followed by the one-digit variety (V) from 1 to 9. Default tender types are: **1V** Cash type, variety V **2V** Check type, variety V **3V** Food stamp type, variety V **4V** Miscellaneous tender type 1, variety V **5V** Miscellaneous tender type 2, variety V **6V** Miscellaneous tender type 3, variety V **7V** Manufacturer coupon tender type, variety V **8V** Store coupon tender type, variety V
AMTTEND1	PL	v4	The amount of the first positive tender in the native currency amount.
TENDTYP2	PI	1	ID of second positive tender.
AMTTEND2	PL	v4	The amount of the positive tender in the native currency amount.
TENDTYP3	PI	1	ID of third positive tender.
AMTTEND3	PL	v4	The amount of the positive tender in the native currency amount.
TENDTYP4	PI	1	ID of fourth positive tender.
AMTTEND4	PL	v4	The amount of the positive tender in the native currency amount.
TENDTYP5	PI	1	ID of fifth positive tender.
AMTTEND5	PL	v4	The amount of the positive tender in the native currency amount.
TENDTYP6	PI	1	ID of sixth positive tender.
AMTTEND6	PL	v4	The amount of the positive tender in the native currency amount.
TENDTYP7	PI	1	ID of seventh positive tender.
AMTTEND7	PL	v4	The amount of the positive tender in the native currency amount.
TENDTYP8	PI	1	ID of eighth positive tender.
AMTTEND8	PL	v4	The amount of the positive tender in the native currency amount.

Field Name	Type	Length	Description
Negative	PD	1	**X'99'** Delimiter for negative tenders
TENDTYP1	PI	1	ID of first negative tender. The tender ID is the one-digit tender type from 1 to 8 followed by the one-digit variety (V) from 1 to 9. Default tender types are the same as shown for the first positive tender.
AMTTEND1	PL	v4	The amount of the first negative tender in the native currency amount.
TENDTYP2	PI	1	ID of second negative tender.
AMTTEND2	PL	v4	The amount of the negative tender in the native currency amount.
TENDTYP3	PI	1	ID of third negative tender.
AMTTEND3	PL	v4	The amount of the negative tender in the native currency amount.
TENDTYP4	PI	1	ID of fourth negative tender.
AMTTEND4	PL	v4	The amount of the negative tender in the native currency amount.
TENDTYP5	PI	1	ID of fifth negative tender.
AMTTEND5	PL	v4	The amount of the negative tender in the native currency amount.
TENDTYP6	PI	1	ID of sixth negative tender.
AMTTEND6	PL	v4	The amount of the negative tender in the native currency amount.
TENDTYP7	PI	1	ID of seventh negative tender.
AMTTEND7	PL	v4	The amount of the negative tender in the native currency amount.
TENDTYP8	PI	1	ID of eighth negative tender.
AMTTEND8	PL	v4	The amount of the negative tender in the native currency amount.

SurePOS ACE EPS Tender String (0x16)

This string contains only SurePOS ACE EPS tender data. The *ISEPSTenderLoggingPolicy* <`EPSTNDLP.CPP`> object class is responsible for building and managing this record.

Field Name	Type	Length	Description
TYPE	PD	1	String type = X'16' for an SurePOS ACE EPS Tender String
HostID	ASCII	2	Identifies the SurePOS ACE EPS Host Server (the servicer) that processed this request
TotalsType	PI	1	Indicates how this transaction was approved: **X'1'** Sale approved by the host (online) **X'5'** Void approved by the host (online)

Field Name	Type	Length	Description
			X'7' Sale approved by SurePOS ACE EPS **X'11'** Void approved by SurePOS ACE EPS **X'19'** Refund approved by the host (online) **X'23'** Refund approved by SurePOS ACE EPS Approval by SurePOS ACE EPS can mean either SurePOS ACE EPS only or SurePOS ACE EPS plus operator input, such as an override.
TotalAmount	PL	v6	The absolute value tendered for this transaction
DateTime	ASCII	12	The date and time of this transaction in the format `YYMMDDhhmmss`, where `YYYY` is the two rightmost digits of the year, `MM` is the month, `DD` is the day, `hh` is the hour (0-23), `mm` is the minutes, and `ss` is the seconds.
ActionCode	ASCII	3	Identifies an action to take as a result of the response code and reason code values. It is built as a zero-indexed array of 3 characters as follows: Character 0 is set as a result of the response code as follows: **A** Approved (0), or LocalApproved (120) **C** CallForAuth (1) **D** DuplicateTransaction (94) **I** InvalidTender (104), TranslationFailed (105), TrxNotSupported (40) **R** All other response codes Characters 1 and 2 are a packed-string representation of the last two digits of the reason code.
RRN	ASCII	v20	Response Reference Number. The default value is 0.
Sequence	PL	5	A sequential identifier for this transaction returned by the server.
MessageType	PI	1	Identifies the type of message this record represents. Possible types are defined in <APSDEFS.H> as follows:

Field Name	Type	Length	Description
			1=FirstEPSType 1=CreditRequest 2=CreditResponse 3=DebitRequest 4=DebitResponse 5=ChequeRequest 6=ChequeResponse 7=EBTRequest 8=EBTResponse 9=TotalsRequest 10=TotalsResponse 11=TotalsPrevRequest 12=TotalsPrevResponse 13=DetailsRequest 14=DetailsResponse 15=SettlementRequest 16=SettlementResponse 17=PINPadLoadRequest 18=PINPadLoadResponse 19=ParameterRequest 20=ParameterResponse 21=HandshakeRequest 22=HandshakeResponse 23=SAFCreditRequest 24=SAFCreditResponse 25=SAFDebitRequest 26=SAFDebitResponse 27=SAFChequeRequest 28=SAFChequeResponse 29=SAFEBTRequest 30=SAFEBTResponse 31=InquiryRequest 32=InquiryResponse 33=PassThroughRequest 34=PassThroughResponse 34=LastEPSType 51=FirstInternalType 51=TotalsUpdateRequest 52=StoreCAPComplete 53=TerminalCAPComplete 54=RenameLogsRequest 55=PassThroughFailToSend 56=ReconciliationRequest 57=IsOkToReconcile 58=OkToReconcile 59=ElectronicVoucherProcessingRequest 60=ElectronicVoucherProcessingResponse 60=LastInternalType 99=Unknown
ResponseCode	ASCII	v9	See EPS Users Guide, Appendix A, "Response Codes" section.
ReasonCode	ASCII	v9	SurePOS ACE EPS failure reason (*ISEPSFailureReason*):

Field Name	Type	Length	Description
			0 Timeout **1** PipeErrorReason **2** LinkDown **3** InternalMessage **4** ExternalMessage **5** TryLocalApproval **6** TurnedAround SurePOS ACE EPS log reason (*ISEPSLogReason*): **300** Informational **301** LinkStatus **302** TraceData **303** UnexpectedEvent **304** InvalidEvent **305** PipeErrorEvent **306** LogError **307** ProblemDetermination
ExpDate	PI	2	The expiration date for the card used in the transaction
TENDTYPE	PI	1	Tender ID, which is the one-digit tender type from 1 to 8 followed by the one-digit variety (V) from 1 to 9. Default tender types are:

Field Name	Type	Length	Description
			1V Cash type, variety V **2V** Check type, variety V **3V** Food stamp type, variety V **4V** Miscellaneous tender type 1, variety V **5V** Miscellaneous tender type 2, variety V **6V** Miscellaneous tender type 3, variety V **7V** Manufacturer coupon tender type, variety V **8V** Store coupon tender type, variety V
CashBack	PL	v4	The amount of money returned to the card holder as a result of this transaction
Account	ASCII	v30	The account number of the card being used. This data is not packed.
EntryMethod	ASCII	1	Identifies the method used to enter the card data as follows: C - OCR H - Swiped track 1 D - Swiped track 2 E - Swiped track 3 R - ;Raw swiped track data F - Contactless track 2 G - Contactless track 1 K - Keyed check T - Keyed track Y - Keyed license M - MICR data S - Scanner L - EMV Fallback swiped I - EMV Fallback keyed

Field Name	Type	Length	Description
			V - EMV Contact Chip W - EMV Contactless
ID1	ASCII	v80	If the tender presented for this transaction is a check, then supplemental personal identification data, provided by the customer, is stored based on its type. In each type of ID, fields after the 2-byte `idType` are variable length and are delimited by hyphens (-). Swiped data has the 2-byte `idType` field followed by track 2 data. See the description of the EPSTLIST.DAT file in the *SurePOS ACE EPS: User's Guide* for more information. If the data contains an embedded colon, the application changes the character to an asterisk to avoid problems with parsing the transaction log entry. This change is most likely to occur in Discretionary Data for MSR input. **Drivers License** idType(2)-State(v2)-DOB(v6 MMDDYY)-D/LNum(v29) **Check (MICR)** idType(2)-TransitNumber (v9)-Acct(v20)-ChkNum(v8) **Mgr's Override** idType(2)-OverrideID(v8) **Credit Card** idType(2)-Acct(v20)-ExpDate(v6) **Other** idType(2)-Acct(v20)
ID2	ASCII	v80	If the tender presented for this transaction is a check, then a second optional piece of supplemental personal identification can be requested of the customer and stored based on its type. In each type of ID, fields after the 2-byte `idType` are variable length and are delimited by hyphens (-). Swiped data has the 2-byte `idType` field followed by track 2 data. See the description of the epstlist.dat file in the *SurePOS ACE EPS: User's Guide* for more information. If the data contains an embedded colon, the application changes the character to an asterisk to avoid problems with parsing the transaction log entry. This change is most likely to occur in Discretionary Data for MSR input. **Drivers License** idType(2)-State(v2)-DOB(v6 MMDDYY)-D/LNum(v29)

Field Name	Type	Length	Description

			Check (MICR)
			idType(2)-TransitNumber (v9)-Acct(v20)-ChkNum(v8)
			Mgr's Override
			idType(2)-OverrideID(v8)
			Credit Card
			idType(2)-Acct(v20)-ExpDate(v6)
			Other
			idType(2)-Acct(v20)

ApprovalCode	ASCII	v??	Approval code
RCDescription	ASCII	v??	Response display description
ApprovalCodeSource	ASCII	1	SurePOS ACE EPS message origin (*ISEPSMessageOrigin*): **A** Called for authorization **E** From SurePOS ACE EPS application **H** From host **W** From workstation
AccountType	ASCII	v1	SurePOS ACE EPS card type (*ISEPSCardType*). If the length = 0, then the account type is NotSpecified. If the length = 1, then the field has one of the following meanings: **1** Saving Account (Debit) or EBT Foodstamps **2** Checking Account (Debit) or EBT Cash **5** Flexible Spending Account (FSA)
VoucherNumber	ASCII	v??	Voucher number
AcquireInst.#	ASCII	v??	Acquiring institution number
OriginalHostResponseCode	ASCII	v?	The original response code received from the payment host
HostResponseIndicator	ASCII	v1	Host Response Indicator. 1 = host returned a response other than "host unavailable" (host is considered online).

Field Name	Type	Length	Description
CardValidationNumber	ASCII	v4	If a CVV2, CVC2, or CID number was keyed, this field contains a 1 (padded with blanks); otherwise, the field is empty.
EntryMethodSecondID	ASCII	v1	Identifies the method used to enter the card data as follows: H - Swiped track 1 D - Swiped track 2 E - Swiped track 3 F - Contactless track 2 G - Contactless track 1 K - Keyed check T - Keyed track Y - Keyed license M - MICR data V - EMV Contact Chip W - EMV Contactless
Flags1	PD	v3	Packed string representation of two-byte integer: **X'0001'** FSA Tender **X'0002'** Online WIC Tender
QHP Amount	PL	v4	Qualified Health Plan Amount
Rx Amount	PL	v4	Prescription Amount
Original Tender Amount	PL	v4	Original Tender Amount
Alternate Tender ID	PD	v1	Alternate Tender ID
Token	ASCII	v30	Tokenized account number

Exception Log String (0x20)

Note: When a No sale/Non-Sales price verify transaction is performed at the pump, no Exception Log String with Exception Type X'06' or X'12' is written to the Exception or Transaction Log.

This string comprises a record by itself and defines data to be written to the Exception Log.

Because this string is in the Exception Log, it is not translated to the host during the normal translation process of the Transaction Log.

Data to be exception logged is a string that can be written directly to the exception log once imbedded colons have been translated to commas. This requires the terminal to delimit non-numeric fields in the log data with three colons.

The terminal logs these entries to the Exception Log:

X'01'
: "Item Entry Exception (0x01)" on page 156

X'02'
: "Discount or Tax Exemption (0x02)" on page 162

X'03'
: "Tender Override or Exception (0x03)" on page 163

X'04'
: "Void Transaction (0x04)" on page 166

X'06'
: "No-Sale Transaction (0x06)" on page 169

X'08'
: "Invalid Item Rejected (0x08)" on page 171

X'09'
: "Terminal Price Change (0x09)" on page 172

X'11'
: "Operator Sign-on/Sign-off (0x11)" on page 173

X'12'
: "Non-Sales Transaction (0x12)" on page 174

X'15'
: "Store Totals Closed (0x15)" on page 178

X'17'
: "Critical Hardware Failure (0x17)" on page 180

X'19'
: "User Data Entered (0x19)" on page 182

X'82'
: "Customer Entry (0x82)" on page 202

X'83'
: "Sales Transaction (0x83)" on page 203

X'89'
: "User Log (0x89)" on page 204 for refund reason codes

The no-sale entry is also used to log tender removals to the Tender Listing Log.

Field Name	Type	Length	Description
TYPE	PD	1	String type = X'20' for Exception Log
LOGDATA	ASCII	v30	Data to be exception logged

Store Closing String (0x21)

ISSAClosePeriodData <`SACLPDAT.CPP`> is the class that defines and manages the manipulation of this Transaction Log string.

This string comprises a record by itself and marks the closing point for a set of files.

Note:

1. A close of the reporting period automatically closes these ledger files:
   - Accounting Database Ledger files
   - Coupon Tracking Database Ledger files
   - Customer Activity Database Ledger files
   - Customer Audit Database Ledger files
   - Department Totals Database Ledger files
   - Miscellaneous Transaction Database Ledger files
   - Negative Sales Database Ledger files
   - Operator Performance Database Ledger files
   - Signature Data file
   - SurePOS ACE EPS Message Database Ledger files
   - Transaction Log Processor Control Database Ledger files
   - Transaction Tracking Database Ledger files
2. A close of the reporting period optionally closes the User Database Ledger and Item Movement Database Ledger files, depending on your personalization settings.
3. A close of the reporting period zeros the till status data in the Terminal Status and Operator Status files.
4. During a close long reporting period, the following ledgers close both short and long:
   - Accounting Database Ledger
   - Department Totals Database Ledger
   - Miscellaneous Transaction Database Ledger
   - Operator Performance Database Ledger

Field Name	Type	Length	Description
TYPE	PD	1	String type = X'21' for store closing
DateTime	PS	5	Date and time of close: (format: `YYMMDDHHmm`)
INDICAT1	PI	1	Close type indication **99** Invalid close type **0** Close files **1** Close reporting period long **2** Close reporting period short **3** Close department totals

Field Name	Type	Length	Description
INDICAT2	PI	1	String representation of a one byte unsigned integer.Additional files that are to be closed as a result of this close: **Bit 0 X'01'** Customer Account Status Database Ledger files **Bit 1 X'02'** Exception Log Database Ledger files **Bit 2 X'04'** Terminal Productivity Database Ledger files **Bit 3 X'08'** Item Movement short files **Bit 4 X'10'** Tender Listing Database Ledger files **Bit 5 X'20'** Item Movement long files **Bit 6 X'40'** Reserved **Bit 7 X'80'** Reserved
AutoPickup	ASCII	v3	String representation of a one byte unsigned integer. During an automated close period, this list defines the tender types to carry forward. Each type represents a bit in the 1-byte field: **Bit 0 X'01'** Tender ID 1 (cash) is carried forward **Bit 1 X'02'** Tender ID 2 (check) is carried forward **Bit 2 X'04'** Tender ID 3 (food stamp) is carried forward **Bit 3 X'08'** Tender ID 4 (miscellaneous type 1) is carried forward **Bit 4 X'10'** Tender ID 5 (miscellaneous type 2) is carried forward **Bit 5 X'20'** Tender ID 6 (miscellaneous type 3) is carried forward **Bit 6 X'40'** Tender ID 7 (manufacturer coupon) is carried forward

Field Name	Type	Length	Description
			Bit 7 X'80' Tender ID 8 (store coupon) is carried forward
CloseProcedure	PI	1	The ID of the close period procedure that requested this close

WIC EBT Data String (0x80)

This string is defined by the *ISSAWICEBTItemData* <`SATRNDAT.CPP`> class object.

This string exists in any transaction entry that has a WIC EBT tender. The WIC EBT tender string is preceded by one WIC EBT data string (identifier is 0x01) for each corresponding 0x01 string that had any of its items purchased with the WIC EBT tender. The WIC EBT tender string is followed by one WIC EBT data string (identifier is 0x02) summarizing the WIC EBT tender data. Finally, there is a WIC EBT data continuation string (identifier is 0x03) that is used to store information relating to *childrenArray* and *zeroChildrenArray* that cannot fit in the 80:01 string.

Table 71. Layout of 0x80 String (Identifier = 0x01) in EAMTRAN.DAT*

Field Name	Type	Length	Description
String Type	PD	1	String type = X'80' for WIC EBT data
Identifier	PD	1	Substring identifier = X'01' for item substring
AgencyID	ASCII	2	Two-character agency ID
Category	PD	1	Food product category
Subcategory	PD	2	Food product subcategory
BenefitQuantity	PD	v3	Amount of benefit debited for a single quantity of this item. The total benefit debited for this entry, for the nonzero subcategory and the zero subcategory, is this amount times the correct PurchaseQuantity for the subcategory.
PurchaseQuantity	PD	v2	Quantity of single items, from this item entry, that were purchased with WIC EBT tender against the nonzero subcategory.
ZeroPurchaseQuantity	PD	v2	Quantity of single items, from this item entry, that were purchased with WIC EBT tender against the zero subcategory.
Cost	PD	v6	Total cost of the total quantity that was purchased with WIC EBT tender against the nonzero subcategory purchase.
ZeroCost	PD	v6	Total cost of the total quantity that was purchased with WIC EBT tender against the zero subcategory purchase.
TransactionEntryIndex	PD	2	Index into transaction entry array for the corresponding item entry.
ItemCode	PD	v8	UPC/PLU including the trailing check digit. This is the UPC/PLU that matched an entry in the APL file, which may be a mapped IFPS or alias item code. If it is a mapped or alias code, the OriginalUPC field contains the code that was keyed/ scanned. This field, in combination with the UPC/PLU Indicator field below, is used to construct the UPC/PLU Data field in the WIC EBT claim file.

Field Name	Type	Length	Description
ChildrenArray	PD	v125	Indicates which single items in a multi-quantity item entry were purchased with WIC EBT tender against a nonzero subcategory. After unpacking, every four characters are used to represent the position of a single item (in a multi-quantity item entry) that was purchased with WIC EBT tender against a nonzero subcategory. Used to link this data back to the correct item entry.
ZeroChildrenArray	PD	v125	Indicates which single items in a multi-quantity item entry were purchased with WIC EBT tender against a zero subcategory. After unpacking, every four characters are used to represent the position of a single item (in a multi-quantity item entry) that was purchased with WIC EBT tender against a zero subcategory. Used to link this data back to the correct item entry.
Continuation	PD	1	**0** no more continuation for this item entry **1** WIC continuation string follows this string
OriginalUPC	PD	v7	Original UPC/PLU code used for item entry, if a mapped IFPS or alias code is used for APL file lookup
UPC/PLU Indicator	ASCII	1	**0** ItemCode field contains a UPC item code **1** ItemCode field contains a PLU item code
WIC Item Flag	PD	1	**X'00'** None **X'01'** WIC Tender associated with this item has been voided. **X'02'** CVB Item **X'04'** Item sold using partial benefits (split tender). **X'08'** Item sold using combined benefits (straddle).

Table 72. Layout of 0x80 String (Identifier = 0x02) in EAMTRAN.DAT*

Field Name	Type	Length	Description
String Type	PD	1	String type = X'80' for WIC EBT data
Identifier	PD	1	Substring identifier = X'02' for summary substring
AgencyID	ASCII	2	Two-character agency ID

Field Name	Type	Length	Description
PAN Length	PD	1	Length of the participant account number (PAN), which is also the card account number.
PAN	PD	10	PAN, left padded with zeros.
ClaimAmount	PD	v6	Total WIC EBT tender amount for this transaction. This amount is the total after any WIC EBT discount amount has been deducted.
POSDataCode	PD	v6	This field is not used.
IssuingAgency	ASCII	v15	ID of the agency that most recently issued benefits to the Smart Card. The ID is read from the Smart Card.
FirstDateToSpend	PD	4	Start date for this benefit period. The date is read from the Smart Card.
LastDateToSpend	PD	4	End date for this benefit period. The date is read from the Smart Card.
DiscountAmount	PD	v6	Total WIC EBT discount amount for this transaction. This amount includes discount and coupon amounts for the WIC EBT items.
TransactionSignature	ASCII	v99	Cryptographic security information about the current WIC EBT transaction. This information is returned by the Smart Card device.
TransactionDateTime	PD	7	Date and time of the WIC EBT transaction. Format of the date and time is `YYYYMMDDhhmmss`, where `YYYY` is the year, `MM` is the month, `DD` is the day of the month, `hh` is the hour, `mm` is the minutes, and `ss` is the seconds.
WIC Summary Flag	PD	1	**X'00'** None **X'01'** WIC tender was voided

Table 73. Layout of 0x80 String (Identifier = 0x03) in EAMTRAN.DAT*

Field Name	Type	Length	Description
String Type	PD	1	String type = X'80' for WIC EBT data
Identifier	PD	1	Substring identifier = X'03' for item continuation substring
ChildrenArray	PD	v125	Indicates which single items in a multi-quantity item entry were purchased with WIC EBT tender against a nonzero subcategory. After unpacking, every four characters are used to represent the position of a single item (in a multi-quantity item entry) that was purchased with WIC EBT tender against a nonzero subcategory.
ZeroChildrenArray	PD	v125	Indicates which single items in a multi-quantity item entry were purchased with WIC EBT tender against a zero subcategory. After unpacking, every four characters are used to represent the position of a single item (in a multi-quantity item entry) that was purchased with WIC EBT tender against a zero subcategory.

Field Name	Type	Length	Description
Continuation	PD	1	**0** no more continuation for this item entry **1** WIC continuation string follows this string

Extra Data String (0x97)

This string records extra data for certain transactions. It is defined by the *ExtraDataStringType* class object.

Field Name	Type	Length	Description
TYPE	PD	1	String type = X'97' for extra data
SUBTYPE	PD	1	Substring type: **X'05'** Extra tender data for foreign currency support **X'06'** Extra tender correction data
TendAmt	PD	v4	Amount of currency

Field Name	Type	Length	Description
TYPE	PD	1	String type = X'97' for extra data
SUBTYPE	PD	1	Substring type = X'13' for extra till change data
There can be up to eight sets of TENDTYPx and AMTTENDx data for positive tenders.			
TENDTYP1	PI	1	ID of first positive tender. The tender ID is the one-digit tender type from 1 to 8 followed by the one-digit variety (V) from 1 to 9. Default tender types are: **1V** Cash type, variety V **2V** Check type, variety V **3V** Food stamp type, variety V **4V** Miscellaneous tender type 1, variety V **5V** Miscellaneous tender type 2, variety V

Field Name	Type	Length	Description
			6V Miscellaneous tender type 3, variety V **7V** Manufacturer coupon tender type, variety V **8V** Store coupon tender type, variety V
AMTTEND1	PL	v4	The amount of the first positive tender in the foreign currency amount.
TENDTYP2	PI	1	ID of second positive tender.
AMTTEND2	PL	v4	The amount of the positive tender in the foreign currency amount.
TENDTYP3	PI	1	ID of third positive tender.
AMTTEND3	PL	v4	The amount of the positive tender in the foreign currency amount.
TENDTYP4	PI	1	ID of fourth positive tender.
AMTTEND4	PL	v4	The amount of the positive tender in the foreign currency amount.
TENDTYP5	PI	1	ID of fifth positive tender.
AMTTEND5	PL	v4	The amount of the positive tender in the foreign currency amount.
TENDTYP6	PI	1	ID of sixth positive tender.
AMTTEND6	PL	v4	The amount of the positive tender in the foreign currency amount.
TENDTYP7	PI	1	ID of seventh positive tender.
AMTTEND7	PL	v4	The amount of the positive tender in the foreign currency amount.
TENDTYP8	PI	1	ID of eighth positive tender.
AMTTEND8	PL	v4	The amount of the positive tender in the foreign currency amount.
Negative	PD	1	Delimiter for negative tenders = X'99'
There can be up to eight sets of TENDTYPx and AMTTENDx data for negative tenders.			
TENDTYP1	PI	1	ID of first negative tender. The tender ID is the one-digit tender type from 1 to 8 followed by the one-digit variety (V) from 1 to 9. Default tender types are the same as shown for the first positive tender above.
AMTTEND1	PL	v4	The amount of the first negative tender in the foreign currency amount.
TENDTYP2	PI	1	ID of second negative tender.
AMTTEND2	PL	v4	The amount of the negative tender in the foreign currency amount.
TENDTYP3	PI	1	ID of third negative tender.
AMTTEND3	PL	v4	The amount of the negative tender in the foreign currency amount.
TENDTYP4	PI	1	ID of fourth negative tender.
AMTTEND4	PL	v4	The amount of the negative tender in the foreign currency amount.
TENDTYP5	PI	1	ID of fifth negative tender.
AMTTEND5	PL	v4	The amount of the negative tender in the foreign currency amount.

Field Name	Type	Length	Description
TENDTYP6	PI	1	ID of sixth negative tender.
AMTTEND6	PL	v4	The amount of the negative tender in the foreign currency amount.
TENDTYP7	PI	1	ID of seventh negative tender.
AMTTEND7	PL	v4	The amount of the negative tender in the foreign currency amount.
TENDTYP8	PI	1	ID of eighth negative tender.
AMTTEND8	PL	v4	The amount of the negative tender in the foreign currency amount.

User Data String (0x99)

This string is defined by the *ISUserEntryFactory* <`USRENFCT.CPP`> class object.

The `Originator` and `SubType` fields together form a key that should uniquely identify any type of user entry. Therefore, there are no restrictions on the particular numbers used in the `SubType` field, provided a given subtype number is unique within types defined by that particular originator.

The existing set of defined originators is listed in the description for the `Originator` field in Table 74. Also listed are existing subtypes currently in use by Toshiba Global Commerce Solutions wanting to use the User Data Record for their own use (or for use by other companies) are encouraged to contact Toshiba Global Commerce Solutions to reserve a number. A unique number guarantees a unique identifier for a company's user entries that is different from those in use by base SurePOS ACE and different from other companies' extensions that might be installed.

Table 74. General Layout of 0x99 String in EAMTRAN.DAT*

Field Name	Type	Length	Description
TYPE	PD	1	String type = X'99' for user data
Originator	PI	1	Represents the originating company for this user entry. Currently, these companies have reserved originator numbers: **X'F0' or X'00'** Toshiba **X'F1'** MGV **X'F2'** MSDS **X'F3'** Toshiba **X'F5'** RHISCOM **X'10'** Customer

Field Name	Type	Length	Description
			X'11' Customer 1 **X'12'** Customer 2 **X'13'** Customer 3 **X'14'** Customer 4 **X'15'** Customer 5 **X'20'** NRSC **X'30'** Malloys **X'31'** Radius Solutions **X'40'** Excentus **X'50'** M-Dot **X'60'** EXIPOS **X'70'** CouponsCom **X'80'** Plenti **X'90'** ICR Other companies that want to reserve originator numbers should contact Toshiba Global Commerce Solutions.
SubType	PI	1	Represents the type of user entry generated by the originator. For Originator=X'00', these are the subtypes: **X'F0'** Age Entry (see Table 75) **X'F1'** Tiered Coupon Multiplication (see Table 76)

Field Name	Type	Length	Description
			X'F2' Memory Debug (see Table 77) **X'F3'** Signature Data (see Table 78) **X'F4'** Supplemental Signature Data (see Table 79) **X'F5'** Coupon Item Association - This entry will not show up in the TLOG (see Table 80). **X'06'** Pharmacy (see Table 81) **X'F7'** Suspended Transaction Summary (see Table 82) **X'F8'** Fuel Item (see Table 83) **X'F9'** Item Subtype Indicator, General (see Table 84) and Car Wash (see Table 85) **X'10'** Rain Check (see Table 86) **X'11'** Bar Code Data (see Table 87) **X'21'** Promotion Service Coupon Details (see Table 88) **X'22'** Additional Matching Items (see Table 89) **X'23'** Promotion Service Coupon Recovery - This entry will not show up in the TLOG. **X'24'** Promotion Service Gift Card Recovery. This entry will not show up in the TLOG. **X'25'** PSI Tender Entry **X'26'** PSI Value Card Entry

Field Name	Type	Length	Description
			X'31' Encrypted Account Information Details. This entry will not show up in the TLOG. **X'32'** EMV Tag Data - This entry will not show up in the TLOG. **X'33'** Scale Sentry (see "Scale Sentry Entry" on page 420) **X'34'** Chec Scale Sentry (see "Chec Scale Sentry Entry" on page 421) **X'50'** Refund Reason Code (see Table 92) **X'51'** Price Override Reason Code (see Table 93) **X'52'** Void Reason Code (see Table 94) **X'60'** AEF User Data (see Table 95) **X'70'** Local Retrieve Totals (see Table 96) **X'78'** Line Item Discount (see Table 97) **X'79'** Department Discount (see Table 98) **X'80'** WIC EBT (see Table 99) **X'81'** WIC EBT (see Table 100) **X'96'** Value Card (see Table 101) **X'99'** EMV PSI Receipt Data - This entry will not show up in the TLOG.
UsrData	var	v97	User-defined data

Table 75. 0x99 TLog Entry for Age Entry (SubType X'F0')

Field Name	Type	Length	Description
ID Type	PD	2	String type = X'99' for user data.
Originator ID	PI	1	X'F0'
SubType ID	PI	1	X'F0'
Age Date	ASCII	v22	Contains the date in this format: up to 6-character abbreviation for the day of the week, 1 space, up to 6-character abbreviation for the month, 1 space, 2 digits for the day of the month, 1 comma, 1 space, 4 digits for the year. If the cashier bypassed the age entry prompt or the option “Allow storing scanned information” in Personalization is disabled, this field will be blank.
Entry Method	ASCII	1	0=Internal 1=Keyboard 2=OCR 3=Scanner 4=Wand 5=MSR 6=MICR
Age Date (`YYYYMMDD`)	ASCII	8	Contains the date in `YYYYMMDD` format. If the cashier bypassed the age entry prompt, this field will contain the 99999999 string. If the age was scanned from a driver's license and the option Allow storing scanned information in Personalization is disabled, this field will contain the 88888888 string.

Table 76. 0x99 TLog Entry for Tiered Coupon Multiplication (SubType X'F1')

Field Name	Type	Length	Description
ID Type	PD	2	String type = X'99' for user data.
Originator ID	PI	1	X'F0'
SubType ID	PI	1	X'F1'
Tier Data	PD	56	Sign indicator and amount data for all seven tiers. X'00' = positive sign indicator; X'01' = negative sign indicator.

Table 77. 0x99 TLog Entry for Memory Debug (SubType X'F2')

Field Name	Type	Length	Description
ID Type	PD	2	String type = X'99' for user data.
Originator ID	PI	1	X'F0'
SubType ID	PI	1	X'F2'

Field Name	Type	Length	Description
Free Memory	ASCII	v94	Free memory

Signature Data

There are two types of 0x99 entries for signature data. There could be more than one of the subtype=3 records, which contain the actual signature data. There should be only one subtype=4 record, which contains information about the signature data.

Table 78. 0x99 TLog Entry for Signature Data (SubType X'F3')

Field Name	Type	Length	Description
ID Type	PD	2	String type = X'99' for user data.
Originator ID	PI	1	X'F0'
SubType ID	PI	1	X'F3'
Signature Length	PD	2	Total length of the signature data (might span multiple signature entries).
Signature Data	Byte	v450	Signature data. The high-order bit in each byte of the signature data is set.
Reserved	ASCII	20	Reserved.

Table 79. 0x99 TLog Entry for Supplemental Signature Data (SubType X'F4')

Field Name	Type	Length	Description
ID Type	PD	2	String type = X'99' for user data
Originator ID	PI	1	X'F0'
SubType ID	PI	1	X'F4'
Signature Type	ASCII	1	E = electronic; P = paper, S = skipped
Signature Source	ASCII	1	F = financial; P = pharmacy
Signature Format	ASCII	1	A = 3-byte ASCII; H = Hypercom UUEncoded
Signature Name	ASCII	v30	Name associated with the signature. For tender-related signatures, the name is in the same format as the Track 1 data on the card, if it is available.
Signature Key	ASCII	v16	Signature key provided by the host in the authorization response, if it is available.

Coupon Item Association

There will be one subtype=5 record for each item in the group of items associated with the coupon.

Table 80. 0x99 Entry for Coupon Item Association (SubType X'F5')

Field Name	Type	Length	Description
ID Type	PD	1	String type = X'99' for user data.
Originator ID	PI	1	X'F0'
SubType ID	PI	1	X'F5'
TransactionIndex	PI	v2	Index of Item Associated with Coupon in the Transaction.
Index in Group	PI	1	Index of Item Associated with Coupon in the Associated Items Group.

Pharmacy Transactions

Table 81. 0x99 TLog Entry for Pharmacy Entry (SubType X'06')

Field Name	Type	Length	Description
ID Type	PD	2	String type = X'99' for user data.
Originator ID	PI	1	X'00'
SubType ID	PI	1	X'06'
Rx Transaction Number	PI	7	Prescription identifier for the pharmacy application.

Fuel Kiosk and Fuel Pump Suspended Transactions

SurePOS ACE logs a 0x99 string (X'F7') that contains the balance due and total amount at the time a transaction was suspended. This string is contained only in the Suspended Transaction File, and *not* in the Transaction Log. Only one string is logged per transaction. See Table 82 for a description of the 0x99 string.

Table 82. 0x99 TLog Entry for Suspended Transaction Summary (SubType X'F7')

Field Name	Type	Length	Description
ID Type	PD	2	String type = X'99' for user data.
Originator ID	PI	1	X'F0'
SubType ID	PI	1	X'F7'
Balance Due Sign Indicator	PI	1	Sign indicator for balance due. X'00'=positive, X'01'=negative.
Balance Due	PD	v4	Balance due at time the transaction was suspended
Total Amount Sign Indicator	PI	1	Sign indicator for total amount. X'00'=positive, X'01'=negative.
Total Amount	PD	v4	Total amount for the transaction.

Table 83. 0x99 TLog Entry for Fuel Item (SubType X'F8')

Field Name	Type	Length	Description
ID Type	PD	2	String type = X'99' for user data.
Originator ID	PI	1	X'F0'
SubType ID	PI	1	X'F8'
PumpId	PS	v2	1-999 Fuel pump number.
NetUnitPrice	PD	v4	Net unit price.
NetXPrice	PD	v4	Net extended price.

Table 84. 0x99 TLog Entry for Item Subtype Indicator (SubType X'F9') General

Field Name	Type	Length	Description
ID Type	PD	1	String type = X'99' for user data.
Originator ID	PI	1	X'F0'
ItemSubTypeIndicator	PI	1	X'F9'
ItemSubtypeValue	PD	v2	Item sub type value from item record file
ItemSubType Data	ASCII	v255	Varies depending on subtype

Table 85. 0x99 TLog Entry for Item Subtype Indicator (SubType X'F9') Car Wash Subtype 99 String

Field Name	Type	Length	Description
ID Type	PD	1	String type = X'99' for user data.
Originator ID	PI	1	X'F0'
ItemSubTypeIndicator	PI	1	X'F9'
ItemSubtypeValue	PD	v2	X'01' Car wash subtype
Auth code	ASCII	v10	Authcode

Table 86. 0x99 TLog Entry for Item Subtype Indicator (SubType X'10') Rain Check Entry Subtype 99 String

Field Name	Type	Length	Description
ID Type	PD	1	String type = X'99' for user data
Originator ID	PI	1	X'F0'
ItemSubTypeIndicator	PI	1	X'10'
Attribute Indicator	PI	1	**Bit 0 X'01'** Quantity **Bit 1X'02'** Weight **Bit 2 to Bit 5** Reserved

Field Name	Type	Length	Description
			Bit 6 X'40' Refund Indicator **Bit 7 X'80'** Void Indicator
Qty/Weight	PD	3	Quantity or weight of this item entry
Itemcode	PD	7	Item code of rain check
Extended Rain Check Price	PD	4	Extended rain check price for this item entry
Extended item's original price	PD	4	Extended original price for this item entry. Original price can be based on the item file price or can be the price entered by the cashier.

Bar Code Data

The 0x99 string stores bar codes in the X'11' subtype. For a GS1 DataBar expanded bar code, the 0x99 string is logged only if there are Application Identifiers (AIs) other than `01` in the bar code

Table 87. 0x99 TLog Entry for Bar Code Data (SubType X'11')

Field Name	Type	Length	Description
ID Type	PD	1	String type = X'99' for user data.
Originator ID	PI	1	X'F0'
SubType ID	PI	1	X'11'
Bar Code Type	PI	1	**X'F0' =** GS1 DataBar Coupon **X'F1' =** Value Card or Code 128 **X'F2' =** UPC **X'F3' =** UPC-E **X'F4' =** EAN **X'F5' =** GS1 DataBar or GS1 DataBar Expanded **X'F6' =** Unknown Bar code type

Field Name	Type	Length	Description
Bar Code	ASCII	v80	Bar code value. Bar code value. If this is a GS1 DataBar Expanded bar code, any field delimiters (X'1D') are replaced with a hyphen (X'2D'). If the bar code type is EAN or GS1 DataBar Expanded, this field includes a check digit.
GS1 Primary Qty	PL	v4	Number of items that were used to match the primary purchase requirement for a GS1 DataBar coupon (see Note)
GS1 Second Qty	PL	v4	Number of items that were used to match the second purchase requirement for a GS1 DataBar coupon (see Note)
GS1 Third Qty	PL	v4	Number of items that were used to match the third purchase requirement for a GS1 DataBar coupon (see Note)

Note: For a weight-required GS1 Databar Coupon, this field will have a value of 1 even if multiple weighted items were used to satisfy the weight purchase requirement. For barcodes with a type other than GS1 Databar Coupon, this field will be blank.

Promotion Service Entries

Table 88. 0x99 TLog Entry for Promotion Service Coupon Details (SubType X'21')

Field Name	Type	Length	Description
Type	PD	1	X'99' for user data.
Originator	PI	1	X'F0' for Toshiba.
SubType	PI	1	X'21' for Promotion Service Coupon Details.
Promotion Service ID	ASCII	V2	Promotion service identifier.
Promotion Service Coupon ID	ASCII	V15	Promotion service coupon identifier.
Discount Type	PI	1	The type of discount this coupon gave: 0 = PERCENT_OFF 1 = FIXED_VALUE 2 = NET_VALUE
Discount Type Value	PI	V5	The original value of the coupon. Meaning differs depending on discount type: 0 = The percentage the coupon discounted its matching items by. One digit of precision assumed. Maximum of 999. 1 = The total value of the coupon in cents. Maximum of 99999999. 2 = the value the coupon discounted its matching items to in cents. Maximum of 99999999.
PS User Data	PI	V8	User data sent by the promotion service. This is not used by ACE in any way.
Number of Matching Items	PI	V3	The number of items the coupon discounts.

Field Name	Type	Length	Description
Matching Item Ordinal Number	PI	V3	Ordinal number of the coupon's first matching item.
Matching Item Discount Amount	PI	V5	Amount the first matching item was discounted in cents.
Matching Item Price	PI	V5	The price of the matching item.

Table 89. 0x99 TLog Entry for Additional Matching Items (SubType X'22')

Field Name	Type	Length	Description
Type	PD	1	X'99' for user data.
Originator	PI	1	X'F0'
SubType	PI	1	X'22' for Additional Matching Items.
Matching Item 2 Ordinal Number	PI	V3	Ordinal number of the coupon's second matching item.
Matching Item 2 Discount Amount	PI	V5	Amount the second matching item was discounted in cents.
Matching Item 2 Price	PI	V5	Price of the second matching item.
Matching Item 3 Ordinal Number	PI	V3	Ordinal number of the coupon's third matching item.
Matching Item 3 Discount Amount	PI	V5	Amount the third matching item was discounted in cents.
Matching Item 3 Price	PI	V5	Price of the third matching item.
Matching Item N Ordinal Number	PI	V3	Ordinal number of the coupon's Nth matching item.
Matching Item N Discount Amount	PI	V5	Amount the Nth matching item was discounted in cents.
Matching Item N Price	PI	V5	Price of the Nth matching item.

Scale Sentry Entry

Table 90. 0x99 TLog Entry for Scale Sentry Entry (SubType X'33')

Field Name	Type	Length	Description
IDType	PD	2	String type = X'99' for user data.
OriginatorID	PI	1	X'F0'.
SubTypeID	PI	1	X'33'.
Action	PI	1	X'01' Beam Block Override. X'02' Beam Block No Override.

Field Name	Type	Length	Description
			X'04' Beam Block Fixed.
Item Code	PD	V7	Item code.
Weight	PD	V5	Item weight.
Original Weight	PD	V5	The original weight used for X'04' Beam Block Fixed action.
Price Savings Amount	PD	V5	Price savings amount between the corrected and original weighted price amount. Used when Action is Beam Block Fixed.

Chec Scale Sentry Entry

Table 91. 0x99 TLog Entry for Scale Sentry Entry (SubType X'33')

Field Name	Type	Length	Description
IDType	PD	2	String type = X'99' for user data.
OriginatorID	PI	1	X'F0'.
SubTypeID	PI	1	X'34'.
Action	PI	1	X'01' Beam Block Override. X'02' Beam Block No Override. X'04' Beam Block Fixed.
Item Code	ASCII	V7	Item code.
Weight	ASCII	V5	Item weight.
Original Weight	ASCII	V5	The original weight used for X'04' Beam Block Fixed action.
Price Savings Amount	ASCII	V5	Price savings amount between the corrected and original weighted price amount. Used when Action is Beam Block Fixed.

Refund, Void, Discount, and Other Transactions

Table 92. 0x99 TLog Entry for Refund Reason Code (SubType X'50')

Field Name	Type	Length	Description
IDType	PD	2	String type = X'99' for user data.
OriginatorID	PI	1	X'F0'
SubTypeID	PI	1	X'50'
RefundReason	PD	1	Refund reason code.

Table 93. 0x99 TLog Entry for Price Override Reason Code (SubType X'51')

Field Name	Type	Length	Description
IDType	PD	2	String type = X'99' for user data.
OriginatorID	PI	1	X'F0'
SubTypeID	PI	1	X'51'
PriceOverrideReason	PD	1	Reason code for price override.

Table 94. 0x99 TLog Entry for Void Reason Code (SubType X'52')

Field Name	Type	Length	Description
IDType	PD	2	String type = X'99' for user data.
OriginatorID	PI	1	X'F0'
SubTypeID	PI	1	X'52'
VoidReason	PD	1	Reason code for void.

Table 95. 0x99 TLog Entry for AEF User Data (SubType X'60')

Field Name	Type	Length	Description
IDType	PD	1	String type = X'99' for user data.
User Data 0 (Originator ID)	PD	1	User data (For AEF generated 99 strings, this will be the AEF Originator ID).
User Data 1 (SubType ID)	PD	1	User data (For AEF generated 99 strings, this will be the AEF SubType ID)
User Data 2	ASCII	Variabl e	Variable length user data
…			
User Data N	ASCII	Variabl e	Variable length user data

Table 96. 0x99 TLog Entry for Local Retrieve Totals (SubType X'70')

Field Name	Type	Length	Description
IDType	PD	1	String type = X'99' for user data.
OriginatorID	PI	1	X'F0'
SubTypeID	PI	1	X'70'
ActionFlag	PD	1	Always X'01'; indicates that the transaction was retrieved.
DataSourceFlag	PD	1	Always X'01'; indicates that the transaction was retrieved from the local controller.
Controller	ASCII	2	ID of the controller from which the transaction was retrieved.
OriginalTerminalID	PS	2	Terminal ID of the suspending terminal.
OriginalTransactionNumber	PI	2	Transaction number of the suspended transaction.
DateTime	PD	5	Date and time of the original transaction in the format `YYMMDDHHmm`.

Field Name	Type	Length	Description
GrossPositive	PL	v4	Gross plus of the retrieved transaction (does not include changes to the transaction made by the operator after retrieval).
GrossNegative	PL	v4	Gross minus for the retrieved transaction (does not include changes to the transaction made by the operator after retrieval).
UserData	ASCII	v255	User data.

Table 97. 0x99 TLog Entry for Line Item Discount (SubType X'78')

Field Name	Type	Length	Description
IDType	PD	2	String type = X'99' for user data.
OriginatorID	PI	1	X'F0'
SubTypeID	PI	1	X'78'
ItemCode	PD	v7	The item code to which the discount was applied.
DiscountRate	PD	v4	The discount rate.
OriginalPrice	PD	v4	The original price of the item.
SoldatPrice	PD	v4	The price at which the item was sold.
DiscountGroupNumber	PD	1	The discount group number that was applied to the item.

Table 98. 0x99 TLog Entry for Department Discount (SubType X'79')

Field Name	Type	Length	Description
IDType	PD	2	String type = X'99' for user data.
OriginatorID	PI	1	X'00'
SubTypeID	PI	1	X'79'
ItemCode	PD	v7	The item code to which the discount was applied.
DiscountRate	PD	v4	The discount rate.
OriginalPrice	PD	v4	The original price of the item.
SoldAtPrice	PD	v4	The price at which the item was sold.
DepartmentNumber	PD	1	The department number that is assigned to the item.

Table 99. 0x99 TLog Entry for WIC EBT (SubType X'80')

Field Name	Type	Length	Description
IDType	PD	2	String type = X'99' for user data.
OriginatorID	PI	1	X'F0'
SubTypeID	PI	1	X'80'
Prescription	ASCII	v?	Initial prescription for smart card and online WIC.

Table 100. 0x99 TLog Entry for WIC EBT (SubType X'81')

Field Name	Type	Length	Description
IDType	PD	2	String type = X'99' for user data.
OriginatorID	PI	1	X'F0'
SubTypeID	PI	1	X'81'
ErrorResponse	ASCII	36	WIC smart card error response. The data below is not colon delimited.
Type	PD	1	Does Claim file exception detail record exist? **X'00'** No **X'01'** Yes
AgencyID	ASCII	2	ID of benefit-issuing agency
PANLength	PD	1	Length of primary account number
PAN	PD	10	Primary account number extracted from WIC card
ErrorCode	ASCII	4	Error code from device
IssuingAgency	ASCII	15	Agency that issued benefits to the card
ICCSignature	PD	3	Card error code in the format `8101xx`, where `xx` is a 2-byte error code

Table 101. 0x99 TLog Entry for Value Card (SubType X'96')

Field Name	Type	Length	Description
TYPE	PD	2	String type = X'99' for user data.
Originator	PI	1	X'F0'
SubType	PI	1	X'96'
Plan	String	2	Card plan. **GI** Gift Card **PC** Phone Card **GB** Blackhawk Card **GT** Lottery Item
AccountType	PI	1	Indicates action taken. **1** Activation

Field Name	Type	Length	Description
			2 Reload
RequestType	PI	1	Type of transaction. **X'01'** Sale **X'02'** Void **X'03'** Refund **X'04'** Void of a refund
Amount	PL	v4	Value card amount.
Sequence	PL	5	Sequence identifier for this transaction that was returned by the SurePOS ACE EPS server.
DateTime	String	12	Date and time of this transaction in the format `YYMMDDhhmmss`.
AccountNumber	String	v30	Account number of the value card.
EntryMethod	PI	1	Method used to enter the card data: **X'00'** Unknown **X'01'** Keyed track **X'02'** Track 1, track 2 or track 3 **X'03'** Scanner
UserField	String	8	Terminal ID for the host.
Track1	String	v80	Track 1 data.
Track2	String	v80	Track 2 data.
Track3	String	v80	Track 3 data.
ApprovalCode	String	variable	Approval code.
ApprovalCodeSource	ASCII	1	Message origin: **A** Called for authorization **E** From SurePOS ACE EPS application

Field Name	Type	Length	Description
			H From the host **W** From the workstation
ResponseCode	PI	v2	Response from the SurePOS ACE EPS host.
ValueCardFlags	String	2	Value card flags: **X'00'** None **X'01'** Post Tender **X'02'** Pre-Authorization Request **X'04'** Pre-Authorization Complete **X'08'** Operator Override **X'10'** Manager Override
Promotion Service ID	String	V2	Promotion service identifier
Promotion Service Gift Card ID	String	V15	Promotion service gift card identifier
RRN	String	V15	The RRN for a lottery item as provided by the host. In the case of a post tender authorized value card, this field contains data only in the string logged at the time of authorization.
Lottery player number	String	V19	The player number for a lottery item.

EAMTRAN*.IDX and EAMTRAN*.MDX (Transaction Log Index Files)

The index files are direct sequential files used to quickly locate transactions in the associated transaction log.

Logical names	<EAMIDXA>, <EAMIDXB>, <EAMIDXC>, <EAMMDXA>, <EAMMDXB>, <EAMMDXC>
Data object reference	*ISSATransactionStorage* `<SATRNSTR.CPP>`
Organization	Fixed direct
Distribution class	Mirrored per update
File copies	Current, Previous, oldas compressed and uncompressed archives for both index files (IDX) and manager index (MDX) files

Record length 64

Field Name	Type	Length	Decimal Offset	Description
TRANSOFFSET	Long	4	0	Offset of the current transaction in the TLog.
TERMINAL	Int	2	4	Terminal number.
TRANSNUM	Int	2	6	Transaction number.
DATETIME	ASCII	5	8	Date and time as printed on the receiptFormat: yymmddhhmm where yy = year, mm = month, dd = day, hh = hour, and mm = minute.
TRANSTYPE	Byte	1	13	Transaction type as defined by *ISTransactionEntryTypes*: **00** Checkout transaction **01** Tender cashing **02** Tender exchange **03** Cashier loan **04** Cashier pickup **05** Tender listing **06** Price verify/change **07** Training session **08** Terminal transfer **09** Terminal monitor **10** Tender count **11** Reserved.

Field Name	Type	Length	Decimal Offset	Description
				12 Return item transaction **13** WIC transaction **14** Reserved. **15** Reprint tender receipt **16** Voided checkout transaction **17** Operator sign-off **18** Standalone session **20** EBT balance inquiry **21** Value card balance inquiry **22** WIC EBT balance inquiry **80** Department totals report
NUMSTRING	Int	2	14	Number of strings in the logical record (zero for header with no strings).
OPERATOR	Long	4	16	Operator number.
GROSSPOS	Long	4	20	Gross positive (transaction).
GROSSNEG	Long	4	24	Gross negative (transaction).
HEADERFLG	Long	4	28	Packed string representation of four-byte integer header flags defined in `SATRNDAT.CPP` **Bit 0 X'80000000'** Reserved. **Bit 1 X'40000000'** TOF Recovered.

Field Name	Type	Length	Decimal Offset	Description
				Bit 2 X'20000000' Transaction copied after previous TLog close string. **Bit 3 X'10000000' - Bit 7 X'01000000'** Reserved. **Bit 8 X'00800000'** EatInTransaction: Eat-in transaction. **Bit 9 X'00400000'** PreferredCustomer: Preferred customer transaction. **Bit 10 X'00200000'** GROSSPOS: Gross positive field is negative. **Bit 11 X'00100000'** GROSSNEG: Gross negative field is negative. **Bit 12 X'00080000'** AdditionalRecords: More records exist for transaction. **Bit 13 X'00040000'** NotFirstRecord: Not first record for transaction. **Bit 14 X'00020000'** TillChange: Signoff record with till change string due to LAN failure. **Bit 15 X'00010000'** TermInitialized: Initialized before or during transaction.
HEADERFLAG	Long	4	28	**Bit 16 X'00008000'** RollbackPriceItem **Bit 17 X'00004000'** Reserved. **Bit 18 X'00002000'** TransTransferred: Transaction transferred.

Field Name	Type	Length	Decimal Offset	Description
				Bit 19 X'00001000' TransactionMonitored: Transaction monitored. **Bit 20 X'00000800'** TenderRemoval: No-sale tender removal before or during transaction. **Bit 21 X'00000400'** TillContents: No-sale till contents report before this transaction. **Bit 22 X'00000200'** TillExchanged: No-sale till exchange before this transaction. **Bit 23 X'00000100'** TenderVerified: No-sale tender verification before transaction. **Bit 24 X'00000080'** NewPassword: New password with operator sign-on. **Bit 25 X'00000040'** OperatorSignon: Operator sign-on prior to transaction. **Bit 26 X'00000020'** TenderRejected: Tender rejected in transaction. **Bit 27 X'00000010'** SignoffIsFalse: *TRANTYPE 17*: Not true sign-off; NRTs logged. *TRANTYPE 00*: Coupons tracked in eamcoupc.dat. **Bit 28 X'00000008'** DataEntry: In this transaction in next string. **Bit 29 X'00000004'** OpenDrawer: Opened before or during transaction. **Bit 30 X'00000002'** SpecialSignoff: Before or in transaction.

Field Name	Type	Length	Decimal Offset	Description
				Bit 31 X'00000001' TerminalAcct: Terminal accountability.
Item Flag	Int	2	32	Packed string representation of one-byte integer: **Bit 0 X'8000'** Reserved. **Bit 1 X'4000'** WeightItem: Weight item. **Bit 2 X'2000'** PriceEntered: Entered price used. **Bit 3 X'1000'** PriceRequired: Price required. **Bit 4 X'0800'** LogAll: Log all occurrences. **Bit 5 X'0400'** LogEx: Log item as exception. **Bit 6 X'0200'** Alias: Item code from alias record. **Bit 7 X'0100'** NoMovement: No item movement kept. **Bit 8 X'0080'** TaxableA: Tax plan A applies to this item. **Bit 9 X'0040'** TaxableB: Tax plan B applies to this item. **Bit 10 X'0020'** TaxableC: Tax plan C applies to this item. (Coupon items only - minimum purchase by manufacturer indicator.) **Bit 11 X'0010'** TaxableD: Tax plan D applies to this item. (Coupon items only - minimum purchase by department indicator.)

Field Name	Type	Length	Decimal Offset	Description
				Bit 12 X'0008' FoodStamp: Food stamp item. **Bit 13 X'0004'** PointsItem: Points item. (Coupon items only - exclude item from minimum purchase indicator.) **Bit 14 X'0002'** NonDiscountable: Non-discountable item. (Coupon items only - redemption item indicator.) **Bit 15 X'0001'** CpnMultAllowed: Coupon multiplication not allowed.
Item Flag 2	Int	2	34	Packed string representation of one-byte integer: **Bit 15 X'0080'** CancelKey: Cancel key pressed. **Bit 14 X'0040'** RefundKey: Refund key pressed. **Bit 13 X'0020'** StoreCouponKey: Store coupon key pressed. **Bit 12 X'0010'** MfrCouponKey: Manufacturer coupon key hit. **Bit 11 X'0008'** DepositKey: Deposit key pressed. **Bit 10 X'0004'** XPRICE: Negative price due to deal. **Bit 9 X'0002'** ExtensionExists: Extension follows this string. **Bit 8 X'0001'** Preferred customer coupon might apply to this item. **Bit 7 X'8000'** Reserved.

Field Name	Type	Length	Decimal Offset	Description
				Bit 6 X'4000' OverrideRqd: Item requires override. **Bit 5 X'2000'** DataEntry: Data entry with item. **Bit 4 X'1000'** Multipriced: Multipriced item. **Bit 3 X'0800'** WeightOrQtyOrVolume: Weight or quantity or volume entered, or points-only item indicator. **Bit 2 X'0400'** CouponMult: Store coupon created by coupon multiplier. **Bit 1 X'0200'** TaxKey: Tax key pressed. **Bit 0 X'0100'** FoodstampKey: Food stamp key pressed.
Void Reason Code	Int	2	36	Void Transaction Reason Code
Item Flag Long*	Long	4	38	Packed string representation of two-byte integer: **BIT 0 X'80000000' through BIT 14X'00020000'** Reserved. **BIT 15 X'00010000'** FuelItem: Fuel item. **BIT 16 X'00008000'** Reserved. **BIT 17 X'00004000'** WeightItem: Weight item. **BIT 18 X'00002000'** PriceEntered: Entered price used. **BIT 19 X'00001000'** PriceRequired: Price required. **BIT 20 X'00000800'** LogAll: Log all occurrences.

Field Name	Type	Length	Decimal Offset	Description
				BIT 21 X'00000400' LogEx: Log item as exception. **BIT 22 X'00000200'** Alias: Item code from alias record. **BIT 23 X'00000100'** NoMovement: No item movement kept. **BIT 24 X'00000080'** TaxableA: Tax plan A applies to this item. **BIT 25 X'00000040'** TaxableB: Tax plan B applies to this item. **BIT 26 X'00000020'** TaxableC: Tax plan C applies to this item. (Coupon items only - minimum purchase by manufacturer indicator.) **BIT 27 X'00000010'** TaxableD: Tax plan D applies to this item. (Coupon items only - minimum purchase by department indicator.) **BIT 28 X'00000008'** FoodStamp: Food stamp item. **BIT 29 X'00000004'** PointsItem: Points item. (Coupon items only - exclude item from minimum purchase indicator.) **BIT 30 X'00000002'** NonDiscountable: Non-discountable item. (Coupon items only - redemption item indicator.) **BIT 31 X'00000001'** CpnMultAllowed: Coupon multiplication not allowed.
Item Flag 4	Long	4	42	Packed string representation of four-byte integer: **Bits 8-31** Reserved

Field Name	Type	Length	Decimal Offset	Description
				Bit 7 X'00000080' Tax Plan E **Bit 6 X'00000040'** Tax Plan F **Bit 5 X'00000020'** Tax Plan G **Bit 4 X'00000010'** Tax Plan H **Bits 2-3** Reserved **Bit 1 X'00000002'** Qualified Healthcare Product (QHP) **Bit 0 X'00000001'** Prescription (Rx) Item
Reserved	ASCII	14	46	Reserved.
Restart pointer	Long	4	60	Number of transaction processed (used during restart recovery).

Note: (*) The Item Flag long field is a duplicated of the Item Flag field, expanded to 4 bytes to allow for further details while preserving support for the legacy Item Flag field. Thus, if the Item Flag Long field is empty, you can still use the information in the Item Flag field.

EAMTSnnn (Totals Save File)

The sequential Totals Save file is a work file that a single terminal uses. Every terminal has its own, which is located on the local controller. Terminal Sales creates and manages Totals Save files and is the only user of the files.

The file contains partial Transaction Log records that consist of all strings written by Terminal Sales to the Transaction Log during sales transactions. Terminal Sales writes data to the Totals Save file when it can no longer write data in the terminal hard totals area or when the terminal might need to be transferred to maintain totals integrity. This preserves data in a file for potential recovery purposes.

Logical name	<EAMX:>*nnn*
Data object reference	*ISRecoverableMatrixStorageArray* <RCVMTRSA.CPP>
Organization	Variable sequential
Distribution class	Local
File copies	1 per terminal
Record length	Variable

Records in the Totals Save file have the same format as records in the Transaction Log except for fewer strings in each record. See "EAMTRAN* (Transaction Summary Log)" on page 361 for the layout of the records. The Totals Save file does not contain the till change and store closing strings. The sensitive customer data (credit and debit account numbers, track data and CVV number) may be encrypted when written to this file, if the related encryption personalization option for the tender or value card is enabled. If encryption is enabled, the data written to totals save file is encrypted. At recovery time, we can get the original account information by decrypting the data, and a void of a previously accepted EPS tender or value card will be successful. However, if the encryption personalization option for the tender or value card is disabled, and the EPS -> EPS Security -> Masking options in Personalization are enabled, the sensitive customer data will be masked when written to this file. If you use the file for recovery, then a void of a previously accepted EPS tender or value card might be unsuccessful.

Strings are in the same order but are delimited differently to reflect the size of the hard totals area rather than the size of memory.

EAMX* (Preferred Customer Activity Transfer File)

The Preferred Customer Activity Transfer file is used to move preferred customer information from one store to another. Each record contains the information needed to transfer a preferred customer transaction. There is one outbound file per store but there can be many inbound files. The files are named eamxssss, where *ssss* is the originating store number for any outbound file. The application maintains two copies of the outbound file. The distribution attributes of the inbound files are determined by the mechanism used to move them into the store. It is suggested that these files also be mirrored for backup.

Inbound files can have numbers in the range of 0000 to 0009 or in the range defined in the *Valid non-home store numbers* Loyalty personalization option. A record in an inbound file is processed in a given store only if the record did not originate in that store and the destination store number is supported there.

Organization	Sequential
Data object reference	*ISSACustomerActivityTransferData* <`SACAXFDT.CPP`>
Distribution class	Mirrored per Update
File copies	1
Record length	32 bytes

Field Name	Type	Length	Decimal Offset	Description
HomeStoreNumber	PD	2	0	Destination store number.
UsedStore	PD	2	2	Originating store number.
CustomerAccountId	PD	9	4	Customer account number.
TransPoints	PD	3	13	Points for this transaction.
TransCoupons	PD	2	16	Coupon amount for this transaction.
DateTime	PD	5	18	Date and time of transaction (`YYMMDDHHmm`).
Terminal	PD	2	23	Terminal where transaction took place.
TransNumber	PD	2	25	Transaction number for this transaction.
TransRedeemedPoints	PD	4	27	Points redeemed in transaction.
Options	PD	1	31	Options flags:

Field Name	Type	Length	Decimal Offset	Description
				X'04' Redemption points are negative X'02' Transaction points are negative X'01' Immediate activity update

All data in the record is packed decimal numerics and all fields are fixed-length with no separating delimiters. Each record contains 32 bytes of data and there are no record delimiters. Each record requires exactly 32 bytes of file space.

GIPCSPOL* (GIPC Spool File)

The guaranteed interprocess communications (GIPC) spool file contains data which is used to communicate between applications.

Logical Name	<GIPCSPOL:>, <TempGIPC:>
Data Object Reference	ISGIPCMessageRecord <GIPCMSGR.CPP>
Organization	Direct
Distribution class	Local
File copies	Spool file, work file

Field Name	Type	Length	Description
Record Length	Int	2	The length of the GIPC record. This field is in big endian format.
Status	ASCII	1	Message status: **O** OK to send **S** Sent **E** Expired **R** Rejected
Message Length	Int	2	Length of the message data.
Message Data	Varies	Variable	Message data. The format is defined by the sender or receiver of the message.
ReturnTo	ASCII	Variable	The "return to" pipe name. This ASCII string is null terminated.
Destination	ASCII	Variable	The "destination" pipe name. This ASCII string is null terminated.
MessageID	Int	4	The GIPC message identifier for the message.
ExpiryDate	Int	4	The Julian day on which the message expires.

Field Name	Type	Length	Description
Filler	BIN	Variable	Padded with X'00' up to a multiple of 512 less the length of the record delimiter.
Delimiter	BIN	4	Always X'EDEDEDED'.

GPCMSGID.* (GIPC Message ID File)

The guaranteed interprocess communications (GIPC) Message ID File contains the message ID of the last received message.

Logical Name	<GPCMsgID:>
Data Object Reference	None
Organization	Direct
Distribution class	Local
File copies	One

Field Name	Type	Length	Decimal Offset	Description
Message ID	Int	4	0	The message ID (starting offset) of the last message stored in the GIPC spool file.

NLSOPTNS (NLS Personalization Options File)

Operators use the SurePOS ACE Personalization interface (available from the SurePOS ACE Main Menu) to change option values in personalization DAT files.

Operators use the SurePOS ACE Personalization interface (available from the SurePOS ACE Main Menu) to change option values in personalization DAT files.

Table 102. Companion Options INI and DAT Files

Options INI File	Options DAT File	Logical DAT File Name	Description
`optnparm.ini`	eamoptns.dat	<EAMOPTNS>	Base personalization options
`acecpntr.ini`	acecate5.dat	<ACECATE5>	Coupon translation options
`nlsparms.ini`	nlsoptns.dat	<NLSOPTNS>	National language support (NLS) personalization options
`apsparms.ini`	apsoptns.dat	<APSOPTNS>	SurePOS ACE EPS personalization options

Each personalization options .dat file is a random access file that contains values for all options declared in its companion `.ini` file. SurePOS ACE creates each dat file from its companion `.ini` file if the .dat file does not exist.

For a description of options `.ini` files, see "OPTNPARM (Options)" on page 534.

For a description of how to use DIF along with the ACEPERSL command, see "Using ACEPERSL to Maintain Personalization Options" on page 572.

For a description of how to enter user data in `.ini` files, see "Tiered Overlay Files" on page 460.

Logical name	<NLSOPTNS>
Data object reference	*ISOptions* <OPTIONS.CPP/HPP>
Organization	Random access
Distribution class	Compound on Close
File copies	1
Record length	512 bytes
User data	In the `INI` files

Reprint Sales Receipt Index File

This file is used to store an index file to sales receipts in the Reprint Sales Receipt File. There is one record for each sales receipt that is stored in the Reprint Sales Receipt File. There is one index file per day, stored up to the number of days defined in personalization Print->Sales Receipt->Reprint Receipt.

File name	RmddIDX (m is month 1-9, A=10, B=11, C=12, dd is day of month)
Data Object Reference	ISReprintSalesReceiptIndexData (rpntidat.cpp)
Organization	Direct
Distribution class	Mirrored per update
Record length copies	64

Field Name	Type	Length	Decimal Offset	Description
DateTime	5x1 PS	5	0	Sales transaction date and time as printed on the receipt. Format: `yymmddhhmm where yy = year, mm = month, dd = day, hh = hour, and mm = minute.`
TerminalNum	PS	2	5	Terminal Number. This field, along with Date, will be used to determine the file name where the receipt is stored.
TransNum	PS	3	7	Transaction Number
Amount	PS	5	10	Transaction Total Amount (does not include cash back amount).
CustomerNumber	ASCII	18	15	Customer loyalty number
ReceiptOffset	PS	4	33	Offset of current sales receipt in Reprint Sales Receipt File.
Flags	Int	1	37	x01 – training mode transaction receipt x02 – amount value is negative (refund) x04 – x80 - Reserved
Reserved for Toshiba		16	38	Toshiba reserved for future use.
Reserved for customer		10	54	Reserved for future use.

Reprint Sales Receipt File

This file is used to store sales transaction receipts. There will be one file per terminal, per day, stored up to the number of days defined in personalization Print->Sales Receipt->Reprint Receipt.

File name	Rmddttt (m is month 1-9, A=10, B=11, C=12, dd is day of month, ttt is terminal number)
Data Object Reference	ISReprintSalesReceiptData (rpntrdat.cpp)
Organization	Sequential
Distribution class	Mirrored per update
Record length	Variable

Field Name	Type	Length	Decimal Offset	Description
Line Feed	Int	1	0	Number of line feeds after the line is printed.
Font Id	Int	1	1	Font to use when line is printed.
Color Id	Int	1	2	Color to use when line is printed.
Flags	Int	1	3	x01 – Center line flag x02 – logo in print line x04 – barcode in print line x08 – signature line x10 – inline logo (used by NRSC) x20 – reserved x40 - reserved x80 - reserved
LOGO Id	PI	1	4	LOGO Id from Personalization. The printer being used for the reprint must have the same logos loaded in the printer; that were loaded in the printer from the original transaction.
LineLength	PI	1	5	Length of the receipt line.
ReceiptLine	ASCII	V38	6	Sales receipt line. If the barcode flag is set, this line contains the barcode string value. The barcode string layout is (each field is stored as a string): 1- byte symbology 2- bytes leading digits 1- byte data position 2 - bytes ID 3 - bytes height scale factor Barcode data (variable) If the signature line flag is set, this line contains the following (each field is stored as a string):

Field Name	Type	Length	Decimal Offset	Description
				2- byte signature record number 2 - byte total number of signature records 1 - byte signature type 1 - byte signature source 1 - byte signature format 4 - byte receipt line number where signature will print
SignatureLength	PS	2		Length of portion of signature (up to 450). This field only exists if Signature line flag is set.
SignatureData		V450		Portion of the signature data (high order bits are turned on when the signature is stored). This field only exists if Signature line flag is set.

SGNATURE (Signature Data File)

The Signature Data Set contains electronic signatures that have been captured during SurePOS ACE EPS transactions. There is one record in the file for each signature that has been captured. The file also contains a header record and a trailer record.

If the *Write Signature Data to Signature File* option in Options -> Database -> E-Sign personalization has been selected:

- If the file does not exist, the file will be created and a header record will be written to the file when the first SurePOS ACE EPS transaction that requires an electronic signature is processed.
- When signature data is detected during processing of the Transaction Log file, a record will be created and written to the Signature file.
- At store close, a trailer record will be written to the current Signature file. Whether the current file is appended to or replaces the previous Signature file (`sgnaturp.dat`) depends on options in Options -> Database -> E-Sign personalization. (Refer to the *SurePOS ACE: Planning and Installation Guide* for detailed information.) At the end of the store close, a new current Signature file is created and a header record is written to the file.

Logical name	`<SGNATURE>`, `<SGNATURP>`, `<SGNATURT>`
Data object reference	*ISSignatureFileData*
Organization	Sequential
Distribution class	Mirrored on Close
File copies	Current, Previous

Table 103. Header Record for Signature File

Field Description	Type	Length	Description
ID Type	ASCII	2	00
Date	ASCII	14	Date that the file was created. Format `YYYYMMDDhhmmss`.
Store Number	ASCII	4	Numeric digits, zero-filled at the left

Table 104. Signature Record Entry for Signature File

Field Description	Type	Length	Description
ID Type	ASCII	2	01
Date	ASCII	14	Date of the transaction in the format `YYYYMMDDhhmmss` (matches the date in the 0x16 string in the transaction log).
Term	ASCII	3	Terminal number.
Trans#	ASCII	4	Transaction number.
Sequence #	ASCII	6	Sequence number.
Stand-in Indicator	ASCII	1	O = online; S = stand-in
Signature Type	ASCII	1	E = electronic; P = paper; S = skipped A skipped signature entry means that a signature was not required because the tender amount was less than the limit specified in the Signature Required Limit option in EPS -> EPS Tender -> E-Signature Personalization.
Signature Source	ASCII	1	F = financial; P = pharmacy
Card Number	ASCII	20	Card account number (left justified with trailing blanks).
Expiration Date	ASCII	4	Card expiration date. Format is *YYMM*
Operator Number	ASCII	6	Numeric operator number (right justified with leading zeros).
Customer Name	ASCII	30	Customer name from Track 1 data (if necessary, truncated or right-padded with blanks).
Host Response Code	ASCII	4	Original host response code (right justified with leading zeros, if present; otherwise, blank).
Manager Index	ASCII	3	Manager override index (right justified with leading zeros, if override is present; otherwise, blank). This field is set to the index of the manager override ID, regardless of the reason for the override, if a manager override was needed for the tender.
Approval Code	ASCII	6	Approval code from the host (blank if not available).
Tender ID	ASCII	2	2-digit tender ID.
Tender Type	ASCII	1	P = Purchase; V = Void; R = Refund (no signature file entries are currently generated with a "V" in this field).
Amount	ASCII	7	Amount of tender (right justified with leading zeros).
Signature Format	ASCII	1	A = 3-byte ASCII; H = Hypercom format
Reserved	ASCII	9	Reserved.

Field Description	Type	Length	Description
Signature Data	Byte	var	Signature data as received from the PIN pad.

Table 105. Trailer Record for Signature File

Field Description	Type	Length	Description
ID Type	ASCII	2	99
Date	ASCII	14	Date of the store close that occurred at the end of the file. Format `YYYYMMDDhhmmss`.

Chapter 3. .ini Files

SurePOS ACE provides `.ini` files as part of its standard programming interface. These `.ini` files retain processing parameter values, some of which you are likely to change as you customize your SurePOS ACE system. All .ini files in SurePOS ACE have the same general structure, which is described in "Basic .ini File Structure" on page 452.

Toshiba Global Commerce Solutions and its Business Partners are likely to customize .ini files as they offer new function or maintenance releases. However, your `.ini` files represent a significant portion of your customized data. You must be able to apply new function releases and maintenance from Toshiba Global Commerce Solutions and Toshiba Global Commerce Solutions Business Partners but retain your previously customized `.ini` files. To allow you, Toshiba Global Commerce Solutions, and its Business Partners to coordinate customizing `.ini` files, SurePOS ACE provides a *tiered overlay* interface for handling tiered overlay `.ini` files. This is described in "Tiered Overlay Files" on page 460.

Note: You should not modify parameters in any base .ini file but always create a tiered overlay .ini file for your changes. There are no exceptions to this rule.

Although the tiered overlay method provides some protection for changing `.ini` files, there are some `.ini` files that you should not modify even with the tiered overlay method. There are also some fields within `.ini` files that you should not modify.

The second column in Table 106 identifies the programming interface classification of each `.ini` file. The `.ini` files that you can modify using the tiered overlay method are described in "Tiered Overlay Files" on page 460. The programming interface classification of such `.ini` files is *modifiable.*

Although an `.ini` file might be generally modifiable, sometimes there are fields within it that are reserved. You should review comments at the beginning of the .ini source file to determine current programming interface classifications. *Modifiable* `.ini` files have one or more modifiable fields.

Still other `.ini` files contain internal system variables and should not be changed at all. An `.ini` file that you should not modify is classified as *reserved*. Such `.ini` files are reserved for internal system variables. Changing a reserved file, even with a tiered overlay `.ini` file, can impact the ability of Toshiba Global Commerce Solutions to provide service for SurePOS ACE.

Note: This book generally does not describe controls and sections within reserved `.ini` files. However, if an `.ini`file is identified with an asterisk (*Reserved**) in Table 106, it is possible that user extensions that you code might require you to change the `.ini` file to reflect your code changes. Because you might have to change such `.ini` files, this book describes their sections and controls.

If you do modify an `.ini` file and there is a Build Process section in its description, follow the procedure to rebuild any related files. After modifying `.ini` files, rebuild SurePOS ACE, stop all background processes, reload SurePOS ACE, and restart all background processes. You might prefer to IPL your controller after rebuilding and reloading SurePOS ACE.

Note: If you do modify an `.ini` file, you are responsible for migrating your changes to any subsequent releases of SurePOS ACE. Fields that you modify might not exist in a new release. If a field does exist in a new release, its use might change and your modification might affect performance and function in ways that you do not intend it to. Toshiba Global Commerce Solutions will document changes it makes to *modifiable* `.ini` files in any future releases to aid your migration efforts.

In addition to identifying the programming interface for `.ini` files, file descriptions also identify the programming interface of significant fields in modifiable files. Fields are identified as in the following examples.

Example of Identifying a Field as Modifiable

ReportName

Modifiable. Specifies the name that appears at the top of the report.

Example of Identifying a Field as Reserved

Base0

Reserved. Specifies the source file used to declare BOB class tables.

Table 106 lists all `.ini` files in SurePOS ACE. Each file listed in Table 106 is described in detail in the section referred to in the last column.

Table 106. Alphabetical Listing of SurePOS ACE .ini Files

.ini File Name	Programming Interface Classification	Purpose	Description
`aceacrsr.ini`	Reserved	Resource file for the Accounting interface	"ACEACRSR (Accounting Resources)" on page 467
`aceadrsr.ini`	Reserved	Resource file for the ADDMI Batch Separator	"ACEADRSR (ADDMI Batch Separator Resource)" on page 467
`acebinlg.ini`	Modifiable	Controls how ACEBIRBL rolls the acebinlg.dat file	"ACEBINLG (BIN Report Log)" on page 467
`acebnrsr.ini`	Reserved	Resource file for ACEBIRBL, the BIN File Rebuild utility	"ACEBNRSR (ACEBIRBL Resources)" on page 468
`aceborsr.ini`	Modifiable	Resource file for the SurePOS ACE Main Menu	"ACEBORSR (Main Menu Resources)" on page 468
`acecdrsr.ini`	Modifiable	Resource file for acecnvdl utility	"ACECDRSR (ACECNVDL Resources)" on page 473
`acecfrsr.ini`	Reserved	Resource file for ACEFIRBL, the fraudulent coupon rebuild utility	"ACECFRSR (ACEFIRBL Resources)" on page 473
`aceclslg.ini`	Modifiable	Rollover settings for the Close Progress Control (aceclslg.dat) file	"ACECLSLG (Close Log Rollover Settings)" on page 473
`acecmisc.ini`	Reserved	Resource file for suspend/retrieve processing	"ACECMISC (Miscellaneous Resources)" on page 474

.ini File Name	Programming Interface Classification	Purpose	Description
`acecpntr.ini`	Reserved	Coupon translation personalization options	"ACECPNTR (Coupon Translation Options)" on page 474
`acecpyrt.ini`	Reserved	Resource file for copyright and other legal information	"ACECPYRT (Copyright Information Resources)" on page 474
`acecsrsr.ini`	Reserved	Resource file for database IDs and miscellaneous messages	"ACECSRSR (Global Resources)" on page 474
`aceddelt.ini`	Modifiable	Maintenance changes to department descriptors	When present, the file uses the same organization as "ACEDDESC (Department Descriptors)" on page 474
`aceddesc.ini`	Reserved	Default department descriptors	"ACEDDESC (Department Descriptors)" on page 474
`aceddmi.ini`	Reserved*	Delayed data maintenance parameters	"ACEDDMI (Delayed Data Maintenance)" on page 475
`aceddrsr.ini`	Reserved	Resource file for Delayed Data Maintenance	"ACEDDRSR (Delayed Data Maintenance Resources)" on page 478
`acedersr.ini`	Reserved	Resource file for Descriptor Editor	"ACEDERSR (Descriptor Editor Resources)" on page 478
`acedmp.ini`	Modiafiable	Application dump control settings	"ACEDMP (Application Dump Control)" on page 478
`acedmrsr.ini`	Reserved	Resource file for Data Maintenance	"ACEDMRSR (Data Maintenance Resources)" on page 480
`acedscre.ini`	Reserved	Descriptor Editor initialization parameters	"ACEDSCRE (Descriptor Editor)" on page 481
`acedtmni.ini`	Reserved*	Data maintenance parameters	"ACEDTMNI (Data Maintenance)" on page 483
`aceejrsr.ini`	Reserved	Resource file for Electronic Journal	"ACEEJRSR (Electronic Journal Resources)" on page 490
`aceesrsr.ini`	Modifiable	Resource file for aceesclm, the shelf label utility	"ACEESRSR (ACEESCLM Resources)" on page 490

.ini File Name	Programming Interface Classification	Purpose	Description
`aceevrsr.ini`	Reserved	Resource file for electronic voucher authorization	"ACEEVRSR (Electronic Voucher Authorization Resources)" on page 490
`acefirlg.ini`	Modifiable	Controls how ACEFIRBL rolls the acefirlg.dat file	"ACEFIRLG (ACEFIRBL Log File)" on page 79
`acefmtds.ini`	Modifiable	Define output formatting for the display	"ACEFMTDS (Output Formatting for 2x20 Display)" on page 490
`acefmtfs.ini`	Modifiable	Define output formatting for the fullscreen display	"ACEFMTFS (Output Formatting for Fullscreen Display)" on page 496
`acefmtgr.ini`	Modifiable	Define output formatting for gift receipts	"ACEFMTGR (Output Formatting for Gift Receipts)" on page 496
`acefmtpr.ini`	Modifiable	Define output formatting for the printer	"ACEFMTPR (Output Formatting for Printer)" on page 496
`acegipc.ini`	Reserved	Guaranteed interprocess communication parameters	"ACEGIPC (GIPC)" on page 497
`aceglrsr.ini`	Reserved	Resource file for global, generic interface text	"ACEGLRSR (Global Generic Interface Text)" on page 498
`acegprsr.ini`	Reserved	Resource file	"ACEGPRSR (GIPC Resources)" on page 498
`acehcrsr.ini`	Reserved	Resource file	"ACEHCRSR (HCH Messages)" on page 498
`acehprsr.ini`	Reserved	Resource file for SurePOS ACE EPS HPH messages	"ACEHPRSR (HPH Messages)" on page 498
`aceicrsr.ini`	Reserved	Resource file for keyed file rebuild utility	"ACEICRSR (Item Code Migration Utility Resources)" on page 498
`acejrrsr.ini`	Reserved	Resource file for ACE Transaction Search Tool	"ACEJRRSR (ACE Transaction Search Tool Resources)" on page 498
`acekfrsr.ini`	Reserved	Resource file for keyed file rebuild dialogs	"ACEKFRSR (Keyed File Rebuild Dialogs)" on page 499

.ini File Name	Programming Interface Classification	Purpose	Description
`acemtops.ini`	Reserved	Miscellaneous transactions options initialization parameters	"ACEMTOPS (Miscellaneous Transactions Groups)" on page 500
`acemtrsr.ini`	Reserved	Resource file for Matrix Keyboard Editor	"ACEMTRSR (Matrix Keyboard Editor Resources)" on page 500
`aceomrsr.ini`	Reserved	Resource file for Operator Authorization Records Data Maintenance	"ACEOMRSR (Operator Data Maintenance Resources)" on page 500
`acepnfmt.ini`	Reserved	Transaction log strings for panel diary processing	"ACEPNFMT (Panel Diary Format)" on page 501
`acerdelt.ini`	Modifiable	Maintenance changes to report descriptors	When present, the file uses the same organization as "ACERDESC (Report Descriptors)" on page 501
`acerdesc.ini`	Reserved	Default reports descriptors	"ACERDESC (Report Descriptors)" on page 501
`acerdisk.ini`	Modifiable	RAM disk initialization parameters	"ACERDISK (RAM Disk Parameters)" on page 501
`acereini.ini`	Modifiable	Report engine initialization parameters	"ACEREINI (Report Engine)" on page 502
`acerersr.ini`	Reserved	Resource file for Report Engine	"ACERERSR (Report Engine Resources)" on page 510
`acerfrsr.ini`	Reserved	Message strings for processing points transfers	"ACERFRSR (Loyalty Program Resources)" on page 510
`acerlrsr.ini`	Reserved	Resource file for Reporting List Editor	"ACERLRSR (Reporting List Editor Resources)" on page 510
`aceschdl.ini`	Modifiable	Delayed data maintenance batch scheduler	"ACESCHDL (Delayed Data Maintenance Scheduler)" on page 510
`acesdelt.ini`	Modifiable	Maintenance changes to sales descriptors	When present, the file uses the same organization as "ACESDESC (Sales Descriptors)" on page 514

.ini File Name	Programming Interface Classification	Purpose	Description
`acesdesc.ini`	Reserved	Default sales descriptors	"ACESDESC (Sales Descriptors)" on page 514
`aceslcrd.ini`	Reserved	Customer Reporting and Maintenance Workbench initialization parameters	"ACESLCRD (Customer Reporting and Maintenance Workbench)" on page 514
`aceslird.ini`	Modifiable	Item Reporting and Maintenance Workbench initialization parameters	"ACESLIRD (Item Reporting and Maintenance Workbench)" on page 514
`aceslrsr.ini`	Reserved	Resource file for Item Reporting and Maintenance Workbench and Customer Reporting and Maintenance Workbench	"ACESLRSR (Selective Item Report Resources)" on page 526
`acesql.ini`	Reserved	SQL engine initialization parameters	"ACESQL (SQL Engine)" on page 526
`acetirlg.ini`	Modifiable	Controls how ACETIRBL rolls the acetirlg.dat file	"ACETIRLG (ACETIRBL Log Configuration)" on page 526
`acetirni.ini`	Reserved	Keyword-value pairs used by the ACETIRBL utility	"ACETIRNI (ACETIRBL Constants)" on page 527
`acetirsr.ini`	Reserved	Resource file for ACETIRBL, the Terminal Item Record File Rebuild application	"ACETIRSR (ACETIRBL Resources)" on page 527
`acetmrsr.ini`	Reserved	Resource file for the Terminal Monitor interface	"ACETMRSR (Terminal Monitor Resources)" on page 527
`acetvrsr.ini`	Reserved	Resource file for TurboVision library functions	"ACETVRSR (Library Resources for TurboVision)" on page 528
`acetxrsr.ini`	Reserved	Resource file for tax plans interface	"ACETXRSR (Tax Plans Resources)" on page 528

.ini File Name	Programming Interface Classification	Purpose	Description
aceubrsr.ini	Reserved	Resource file for unattended close processing	"ACEUBRSR (Resources for Unattended Close Program)" on page 528
aceuprsr.ini	Modifiable	Resource file for the aceubatp utility program	"ACEUPRSR (ACEUBATP Resources)" on page 528
aceversr.ini	Reserved	Generic resource file for TurboVision view engine	"ACEVERSR (Generic Resources for TurboVision View Engine)" on page 528
acewersr.ini	Modifiable	Resource file for the acewerbl utility program	"ACEWERSR (ACEWERBL Resources)" on page 528
acexfrsr.ini	Reserved	Resource file for transferring points activity	"ACEXFRSR (Loyalty Program Points Transfer Resources)" on page 528
adxcsozf.ini	Reserved*	Messages	"ADXCSOZF (Background Error Messages)" on page 528
apsgrant.ini	Modifiable	Grant file for SurePOS ACE EPS operator options authorizations	"APSGRANT (SurePOS ACE EPS Operator Options Authorization Grant)" on page 529
apsparms.ini	Reserved	SurePOS ACE EPS personalization options	"APSPARMS (SurePOS ACE EPS Personalization Options)" on page 529
cpngrant.ini	Modifiable	Grant file for coupon translation operator options authorizations	"CPNGRANT (Coupon Translation Operator Options Authorization Grant)" on page 529
fdsln.ini	Reserved	Logical file names	"FDSLN (Logical Names)" on page 530
gstfmtds.ini	Modifiable	Define output formatting for the GST display	"GSTFMTDS (Output Formatting for GST Display)" on page 530
gstfmtpr.ini	Modifiable	Define output formatting for the GST printer	"GSTFMTPR (Output Formatting for GST Printer)" on page 530
iop.ini	Modifiable	Windows Simulator file	"IOP (Toolkit I/O Processor Simulator)" on page 530
msrwndw.ini	Modifiable	Windows Simulator file	"MSRWNDW (Toolkit Magnetic Stripe Reader Window Simulator)" on page 532

.ini File Name	Programming Interface Classification	Purpose	Description
`nlsgrant.ini`	Modifiable	Grant file for NLS personalization options	"NLSGRANT (NLS Operator Options Authorization Grant)" on page 533
`nlsparms.ini`	Reserved	NLS personalization options	"NLSPARMS (NLS Options)" on page 533
`optgrant.ini`	Modifiable	Grant file for SurePOS ACE personalization options	"OPTGRANT (Operator Options Authorization Grant)" on page 533
`optnparm.ini`	Modifiable	SurePOS ACE personalization options	"OPTNPARM (Options)" on page 534
`scnnrwnd.ini`	Modifiable	Windows Simulator file	"SCNNRWND (Toolkit Scanner Window Simulator)" on page 546

Basic .ini File Structure

SurePOS ACE `.ini` files may share certain characteristics. For example, many begin with an `INIControl` section that SurePOS ACE uses to control the `.ini` file and apply its values throughout the system.

Note: Any line in an `.ini` file which exceeds 512 bytes is assumed invalid and is not processed.

This section describes the following things that can apply to SurePOS ACE `.ini` files:

- "INIControl Section Parameters" on page 452
- "Common TurboVision Controls" on page 453
- "Input Validation Data Types" on page 460
- "Tiered Overlay Files" on page 460

INIControl Section Parameters

The INIControl section is required in all SurePOS ACE `.ini` files that initialize programs. (The section is not required in resource `.ini` files.) SurePOS ACE uses this section to order, identify, and control all other sections in the `.ini` file and to imbed any external or linked files.

An INIControl section can include these parameters:

External

Specifies a user-defined section name that refers to an external file specified in the `File` parameter of this section. The external file becomes an addendum to the file section of the `.ini` file.

SurePOS ACE requires all controls on a panel to be defined in one file, either the `.ini` file or the external file. It does not allow control on any one panel to be mixed (some from the `.ini` file and some from an external file).

File

Parameters are positional. `OPT` is required for an options `.ini` file. It identifies files that are related to options.

The second parameter identifies another section of the .ini file, FileNames. The FileNames section identifies related files such as the HPP file. "FileNames Section" on page 535 describes the FileNames section.

The third parameter also identifies another section of the .ini file, File0. The File0 section identifies other sections of the `.ini` file that each contain one record of personalization option values. "File0 Section" on page 539 describes the File0 section.

Link

Specifies the name of another section that specifies other `.ini` files that this `.ini` file links.

Menu

Specifies the name of the section in the `.ini` file that defines all menus for this user interface.

RestoreMenus

Specifies the Boolean value for allowing menus to be restored to their original default values.

Validate

Specifies the name of another section in the `.ini` file (often the InputValidation section) that specifies valid data types for field validations.

Common TurboVision Controls

Some `.ini` files define menus, submenus, dialogs and processes for a TurboVision user interface. This section describes those controls, which are the same for each `.ini` file that accesses them.

There are two types of controls, described by these sections:

- "Menu Controls" on page 453
- "Dialog Controls" on page 455

Menu Controls

Menu sections are composed of menu options sections, which each appear on the menu. A menu option can be one of these controls:

- "Dialog" on page 454
- "NewLine" on page 454
- "Process" on page 454
- "Submenu" on page 455

Dialog

A *dialog* control specifies that a dialog box is to be called when an operator selects the menu option it defines. It also identifies another section that defines controls for the dialog. For example, in the NLS section of the `nlsparms.ini` file there is a Numbers dialog option. Therefore, there is another section, NLS.Numbers, that defines granular controls for the dialog.

Other parameters in a dialog are positional:

- *Upper left corner x coordinate* from 0 to 80
- *Upper left corner y coordinate* from 0 to 24
- *Lower right corner x coordinate* from 0 to 80
- *Lower right corner y coordinate* from 0 to 24
- *PreCommand*, which is a code from 0 to 255 that is processed before displaying the dialog. (0 means no command is used.)
- *PostCommand*, which is a code from 0 to 255 that is processed after closing the dialog. (0 means no command is used.)
- *Help context*, which is a topic number in the online help file
- *Record list*, which is either *record name* or *record list, record name*. This specification lists all records that should be accessed before opening the dialog box. They do not have to be specified in the same order as their related options appear in the dialog box.

An example of a dialog control is:

```
Numbers   = Dialog, "~N~umbers  " ,1, 0,80,22, 0,0, 7400, NLSOptions
```

which defines the Numbers dialog, assigns help topic 7400 to the dialog, and specifies that the NLSOptions section must be called before opening the dialog.

NewLine

A *NewLine* control creates a new line using the characters that preceded it. For example, in the File section of the `optnparm.ini` file, there are two uses of the NewLine control. All 15 characters of the menu option are line characters for each new line. So, each draws a line across the File pull-down menu.

Process

A *Process* control includes a coded command process call:

-0	Database operation: load, save, or erase.
-1	Exit menu.
-10	Confirmation button to close dialog.
-11	Cancel button to close dialog.
101	Open terminal options file.
nnn	Other values are specific to various `.ini` files.

It also includes the protection level for the process, which is either standard or protected. Standard is the default.

These values are TurboVision command codes. Some command codes are reserved by TurboVision. The *Borland TurboVision for DOS Programmer's Guide* lists command codes that TurboVision reserves.

Submenu

A *Submenu* control creates a type of menu. It should only be used within a `MainMenu` section. It appears as a pull-down menu if it is a submenu of a main menu section.

Dialog Controls

A dialog section specifies dialog controls that each appear on the panel. Dialog controls can be any of these:

- "AmountInputLine" on page 455
- "Button" on page 456
- "CheckBox" on page 456
- "Combo" on page 456
- "InputKey" on page 458
- "InputLine" on page 458
- "NoteBook" on page 458
- "RadioButtons" on page 459
- "StaticLine" on page 459
- "StaticText" on page 459

AmountInputLine

An *AmountInputLine* control automatically adjusts the input field to include a decimal symbol. Its functionality is equivalent to that of the `InputLine` control.

Note: Always define an `AmountInputLine` with a length 1 greater than the maximum allowed input length to account for the decimal symbol. Otherwise, the first digit in the field is truncated.

Comma-delimited parameters for an `AmountInputLine` control are:

- *x coordinate* from 0 to 80
- *y coordinate* from 0 to 24
- *width of control* from 0 to 80
- *PreCommand*, which is a code from 0 to 255 that is a command to process when the field gains focus. (0 means no command is used.)
- *PostCommand*, which is a code from 0 to 255 that is a command to process when the field loses focus. (0 means no command is used.)
- *TurboVision option value*
- *Validation type*. This is any validation policy. Allowed values are described in "Input Validation Data Types" on page 460.
- *Record list*, which is in one of two formats:
  - *recordName,fieldName*, such as Security,DrawerErrorLimit
  - *recordName::blockField, fieldName*, such as Security, PriceOverrideLimit, 1

Button

A *Button* control has comma-delimited positional parameters:

- *x coordinate* from 0 to 80
- *y coordinate* from 0 to 24
- *width of control* from 0 to 80
- *text*, the label with &tilde; delimiters around the action character
- *Command*, which is a code from 0 to 255. (0 means no command is used.)

CheckBox

A *CheckBox* control has comma-delimited positional parameters:

- *x coordinate* from 0 to 80
- *y coordinate* from 0 to 24
- *width of control* from 0 to 80
- *PreCommand*, which is a code from 0 to 255 that is a command to process when the field gains focus. (0 means no command is used.)
- *PostCommand*, which is a code from 0 to 255 that is a command to process when the field loses focus. (0 means no command is used.)
- *TurboVision option value*
- *Validation type*. This is any validation policy listed in the specified validation section.
- *Record list*, which is in one of two formats:
    - *recordName,fieldName*, such as Security,DrawerErrorLimit
    - *recordName::blockField, fieldName*, such as Security, PriceOverrideLimit, 1

Combo

A *Combo* (box) control has comma-delimited positional parameters:

- *x coordinate* from 0 to 80
- *y coordinate* from 0 to 24
- *width of control* from 0 to 80
- *TurboVision option value*. If the TurboVision option value is 128, the user can enter any value allowed by *inputline* validation. If the option is not used, the user can select from values in the list box or can enter values within the range of values in the list box.
- *InputLine control*. The name of a control that corresponds to the associated InputLine in this section. (A.Ctrl06 in `optnparm.ini` is an example of such a control.)
- *TextList*, which is a list of comma-delimited strings that populate the combo box.

WeightInputLine

A WeightInputLine control automatically adjusts the input field to include a decimal symbol. Its functionality is equivalent to that of the InputLine control.

Note: Always define a WeightInputLine with a length 1 greater than the maximum allowed input length to account for the decimal symbol. Otherwise, the first digit in the field is truncated.

.

Comma-delimited parameters for a WeightInputLine control are:

- *x coordinate* from 0 to 80
- *y coordinate* from 0 to 24
- *width of control* from 0 to 80
- *PreCommand*, which is a code from 0 to 255 that is a command to process when the field gains focus. (0 means no command is used.)
- *PostCommand*, which is a code from 0 to 255 that is a command to process when the field loses focus. (0 means no command is used.)
- *TurboVision* option value.
- *Validation type*. This is any validation policy. Allowed values are described in "Input Validation Data Types" on page 460.
- *Record list*, which is in one of two formats:
  - recordName,fieldName, such as Security,DrawerErrorLimit
  - recordName::blockField, fieldName, such as Security, PriceOverrideLimit, 1

CharacterInputLine

An CharacterInputLine control has these special character parameters for defining an area for data entry. Special characters for CharacterInputLine control are defined as coma, dot and space.

Comma-delimited parameters for a PercentInputLine control are:

- *x coordinate* from 0 to 80
- *y coordinate* from 0 to 24
- *width of control* from 0 to 80
- *PreCommand*, which is a code from 0 to 255 that is a command to process when the field gains focus. (0 means no command is used.)
- *PostCommand*, which is a code from 0 to 255 that is a command to process when the field loses focus. (0 means no command is used.)
- *TurboVision* option value.
- *Validation type*. This is any validation policy. Allowed values are described in "Input Validation Data Types" on page 460.
- *Record list*, which is in one of two formats:
  - recordName,fieldName, such as Security,DrawerErrorLimit
  - recordName::blockField, fieldName, such as Security, PriceOverrideLimit, 1

PercentInputLine

A PercentInputLine control automatically adjusts the input field to include a decimal symbol. Its functionality is equivalent to that of the InputLine control.

Note: Always define a PercentInputLine with a length 1 greater than the maximum allowed input length to account for the decimal symbol. Otherwise, the first digit in the field is truncated.

Comma-delimited parameters for a PercentInputLine control are:

- *x coordinate* from 0 to 80
- *y coordinate* from 0 to 24
- *width of control* from 0 to 80
- *PreCommand*, which is a code from 0 to 255 that is a command to process when the field gains focus. (0 means no command is used.)

- *PostCommand*, which is a code from 0 to 255 that is a command to process when the field loses focus. (0 means no command is used.)
- *TurboVision* option value.
- *Validation type*. This is any validation policy. Allowed values are described in "Input Validation Data Types" on page 460.
- *Record list*, which is in one of two formats:
  - recordName,fieldName, such as Security,DrawerErrorLimit
  - recordName::blockField, fieldName, such as Security, PriceOverrideLimit, 1

InputKey

An *InputKey* control has comma-delimited positional parameters that are the same as those for an `AmountInputLine` control. See "AmountInputLine" on page 455.

InputLine

An *InputLine* control has these comma-delimited positional parameters for defining an area for data entry:

- *x coordinate* from 0 to 80
- *y coordinate* from 0 to 24
- *width of control* from 0 to 80
- *PreCommand*, which is a code from 0 to 255 that is a command to process when the field gains focus. (0 means no command is used.)
- *PostCommand*, which is a code from 0 to 255 that is a command to process when the field loses focus. (0 means no command is used.)
- *TurboVision option value*. When 128, characters display as asterisks (*) for secure data entry.
- *Validation type*. This is any validation policy in "Input Validation Data Types" on page 460.
- *Record list*, which is in one of two formats:
  - *recordName,fieldName*, such as Security, DrawerErrorLimit
  - *recordName::blockField, fieldName*, such as Security, PriceOverrideLimit, 1

NoteBook

A *NoteBook* control has these comma-delimited positional parameters for defining a series of panels that each can be selected from tabs that appear in a page list:

- *x coordinate* from 0 to 80
- *y coordinate* from 0 to 24
- *width of control* from 0 to 80
- *height of control* from 0 to 24
- *notebook tabs position*, which is either Left, Right, Top, or Bottom
- *notebook style*, which is either Standard, AutoSize, or Framed
- *tab size*, which is the width of text displayed on the notebook tabs, when there is more than one notebook page for the dialog
- *page list*, which is the list of text entries that appear as tabs

See the *Borland TurboVision for DOS Programmer's Guide* for more information.

PageLabel

A *PageLabel* control specifies up to 14 characters to appear at the top of the dialog panel.

RadioButtons

A *RadioButtons* control has comma-delimited positional parameters:

- *x coordinate* from 0 to 80
- *y coordinate* from 0 to 24
- *width of control* from 0 to 80
- *PreCommand*, which is a code from 0 to 255 that is a command to process when the field gains focus. (0 means no command is used.)
- *PostCommand*, which is a code from 0 to 255 that is a command to process when the field loses focus. (0 means no command is used.)
- *TurboVision option value*
- *textlist*, comma-delimited text in double quotes that appears beside each radio button

```
BCType =  RadioButtons,38,6,36,7,0,206,0,"Scheduled Modify               ",\
"Scheduled Modify / Restore     ",\
"Scheduled Report               ",\
"Operator  Modify               ",\
"Operator  Modify / Restore     ",\
"Operator  Report               "
```

StaticLine

A *StaticLine* control has comma-delimited positional parameters for defining an entire line of static text:

- *x coordinate* from 0 to 80
- *y coordinate* from 0 to 24

StaticText

A *StaticText* control has comma-delimited positional parameters for defining an area of static text:

- *x coordinate* from 0 to 80
- *y coordinate* from 0 to 24
- *width of control* from 0 to 80
- *TurboVision option value*
- *text* to be displayed

Input Validation Data Types

These are the reserved data types, which are often used in a section titled InputValidation:

INTEGER
Field value must be an integer

NUMERIC
Field must contain only numeric characters

RANGE
Field value must be within the validated range

DATE
Field value must be a date

VALUESET
Field value must be one of the specified values

AMOUNT
Field value must be a monetary value within the specified range

DECIMAL
Field value is assumed to be decimal and must be within the specified range

STRFTIME
Field value must be a date

ALPHANUMERIC
Field must contain only alphanumeric characters

ITEMCODE
Field value must be an item code defined in the Item Record file

ALPHABET
Field must contain only alphabetic characters

STRING_RANGE
Field must contain only numerics and string lengths will be enforced for the beginning and ending range.

You can change the reserved data type to one of those listed above. If you change it to a range, you must add minimum and maximum limits.

Tiered Overlay Files

There is a specific method for changing personalization and other `.ini` file parameter values. It involves using a *tiered overlay* of `.ini` files that do not directly change a base `.ini` file. Tiered overlay files can be used with `.ini` files that are used with code written in the C or C++ language (see “Method for C/C++ Files” on page 461).

Method for C/C++ Files

The tiered overlay method of processing changes to `.ini` files accommodates conditional changes to `.ini` files (at build time) as well as customer changes or changes resulting from the distribution of maintenance. Up to ten tiered overlay files can exist for each `.ini` file. The unique file name for each overlay file is created as follows:

file name

The logical name (minus the surrounding brackets) of the base `.ini` file.

extension

A unique value from the following ranges:

.us0 through .us3

Toshiba-issued maintenance

.us4 through .us6

Toshiba Global Commerce Solutions Business Partner modifications

.us7 through .us9

User modifications

Examples of possible overlay file names are `acereini.us5` for the `acereini.ini` file and `acedeini.us8` for the `acedscre.ini` file.

When a request is made to access an `.ini` file, the base `.ini` file is accessed first. SurePOS ACE next attempts to open a series of files with the same name but with sequentially named extensions, from `us0` to `us9`. For example, after processing values in the base `.ini` file `aceglrsr.ini`, SurePOS ACE attempts to process related files `aceglrsr.us0`, `aceglrsr.us1`, and so on to `aceglrsr.us9`.

All Toshiba Global Commerce Solutions, Business Partner, and user-defined modifications are made in the sequential tiered overlay files that contain additional, replacement, and displacement entries for the base `.ini` file. SurePOS ACE applies these files in sequence, beginning with any Toshiba-issued maintenance, including Business Partner modifications, and finishing with your modifications.

Each applied tiered overlay file *overlays* the combination of information from preceding files. Although these overlay files are overlaid in sequence (from `us0` to `us9`), it is not necessary for all preceding files to exist to use a higher order file. For example, `aceglrsr.us9` can exist without any preceding tiered overlay files and overlays the base as expected.

The tiered overlay method uses the same syntax as base `.ini` files with certain extensions. Syntax extensions overlay (change) or delete previously defined entries, or add new ones.

Overlay

To overlay previously defined data, identify the section and the entry name in the tiered overlay file, using the same syntax as in the base `.ini` file. The syntax appears in Figure 4:

```
[ExistingSectionName]
ExistingEntryName = Value
```

Figure 4. Syntax of Entry Changes for Tiered Overlay File

Delete

To delete a previously defined section and all of its entries, identify the section name and use the /D: parameter with the entry name. The syntax appears in Figure 5:

```
/D:[ExistingSectionName]
```

Figure 5. Syntax of Section Deletions for Tiered Overlay File

To delete a previously defined entry within a section but leave the rest of the section, identify the section name and use the /D: parameter with the entry name. The syntax appears in Figure 6:

```
[ExistingSectionName]
/D:[ExistingEntryName]
```

Figure 6. Syntax of Entry Deletions for Tiered Overlay File

Note: When deleting an entry, do not include assignment data as in the original entry.

If you attempt to delete or overlay a section or entry that does not have a match in the base or composite `.ini` file, SurePOS ACE does not perform any action for the entry and does not provide any notification that the attempt failed. Such a failed operation might occur if you specify an entry name that another tiered overlay file has already deleted, or if you misspell a section name.

Add

To add a section and an entry to the base `.ini` file, use the /A: parameter with the new section name and with the entry name. The syntax appears in Figure 7:

```
/A:[NewSectionName]
/A:[NewEntryName = Value]
```

Figure 7. Syntax of Additions for Tiered Overlay File

For information about adding a customer format to a tiered overlay file for an output format `.ini` file, see "ACEFMTDS (Output Formatting for 2x20 Display)" on page 490.

If you attempt to add a section that already exists in the composite `.ini` file, there is no effect. However, an attempt to add an entry that already exists in a section overlays the data for the entry with data in the new entry.

A tiered overlay file is automatically combined with the base `.ini` file to form a composite `.ini` file. The existence of the tiered overlay file is transparent to SurePOS ACE functions that use the composite `.ini` file.

Using Tiered Overlays

TurboVision Resource .ini Files

TurboVision resource `.ini` files define references and strings used by SurePOS ACE code. All resource files are *reserved*.

Personalization Options .ini Files

The ACEPERSL module interprets personalization options `.ini` files as described in "Using ACEPERSL to Maintain Personalization Options" on page 572. ACEPERSL also supports maintaining user interface changes to companion .dat files.

Table 107 lists the options `.ini` files and their companion .dat files.

Table 107. Companion Options .ini and .dat Files

Options .ini File	Options DAT File	Description
`optnparm.ini`	eamoptns.dat	SurePOS ACE base personalization options
`acecpntr.ini`	acecate5.dat	SurePOS ACE coupon translation options
`nlsparms.ini`	nlsoptns.dat	National language support (NLS) personalization options
`apsparms.ini`	apsoptns.dat	SurePOS ACE EPS personalization options
`optnse2a.ini`	N/A	Maps 4680-4690 Supermarket Application options to SurePOS ACE options
`optnseam.ini`	N/A	Base listing of 4680-4690 Supermarket Application options

Options `.ini` files are described in:

- "OPTNPARM (Options)" on page 534
- "NLSPARMS (NLS Options)" on page 533
- "ACECPNTR (Coupon Translation Options)" on page 474
- "APSPARMS (SurePOS ACE EPS Personalization Options)" on page 529

Output Formatting .ini Files

The `.ini` files for output formatting are used by the ACEBLDFL utility to generate the associated .dat files. The command to generate the .dat files is:

```
make -f makefile.gen outputformatfiles
```

The output formatting `.ini` files and their companion .dat files are:

Table 108. Companion .ini and DAT Files for Output Formatting

.ini File	DAT File	Description
`acefmtds.ini`	acefmtds.dat	Output formatting for the display
`acefmtfs.ini`	acefmtfs.dat	Output formatting for the fullscreen display
`acefmtgr.ini`	acefmtfs.dat	Output formatting for gift receipts
`acefmtpr.ini`	acefmtpr.dat	Output formatting for the printer
`gstfmtds.ini`	gstfmtds.dat	Output formatting for the GST display
`gstfmtpr.ini`	gstfmtpr.dat	Output formatting for the GST printer

Output formatting `.ini` files are described in:

- "ACEFMTDS (Output Formatting for 2x20 Display)" on page 490

- “ACEFMTFS (Output Formatting for Fullscreen Display)” on page 496
- “ACEFMTPR (Output Formatting for Printer)” on page 496
- “GSTFMTDS (Output Formatting for GST Display)” on page 530
- “GSTFMTPR (Output Formatting for GST Printer)” on page 530

Tiered Overlay .ini File Example

The following example shows a tiered overlay `.ini` file that deletes sections and entries in the `optnparm.ini` file.

```
;Stubs (no options in Personalization associated with these functions)
;Remove Age Entry
;Remove Compare Price
;Remove Department Report
;Remove Tender Listing

;Remove APS
[MainMenu]
/D:APS
```

```
;Remove Catalina Marketing Options
;Remove CatalinaMarketingActive
[Options.Store.Marketing]
/D:K.Ctrl02
/D:K.Ctrl13
;Remove CatalinaMarketingPipe
/D:K.Ctrl03
/D:K.Ctrl12
```

```
;Remove Check Printing
[Print]
/D:Check Printing

;Remove Enhanced Printing
[Extended Print]
/D:Post Print
```

```
;Remove Multi Tier Coupons
[Options.Coupon]
Ctrl01 = Notebook, 1,1,75,8, Left,AutoSize,15, \
                Acceptance, Acceptance 2, Acceptance 3, \
                UPC Value Codes 1, UPC Value Codes 2, Taxability, \
                Tracking, Multiplying, Multiplying 2, Preferred, \
                Delay/Replay, Overrides

CtrlX3 = Button, 18,20,29, 0, ~O~K,     -10
CtrlX4 = Button, 47,20,29, 0, ~C~ancel, -11
```

```
;Remove Summary Journal
[Options.Store]
Ctrl01 = Notebook,1,1,74,25, Left,AutoSize,18,\
             Name&Address, Foodstamps/Tax, \
             Accounting, Exception Logging,\
             Suspend/Retrieve, Rounding, Devices, Elec. Shelf Label,\
             Selective Item Rp, IRI Shopper, Marketing,\
             Term. Productivity, Item Maintenance, Mgr Audit Trail,\
             Copyright Display

CtrlX3 = Button,    21,19,27, 0, &tilde;ÕK,     -10
CtrlX4 = Button,    48,19,27, 0, &tilde;C&tilde;ancel, -11
```

```
;If Coupon Tiering enabled, remove certain Coupon Mulitiplying options
;Remove 'Manufacturer Coupon Multiplier' static/edit text fields
```

```
[Options.Coupon.Multiplying]
/D:F.Ctrl02
/D:F.Ctrl22
```

```
;Remove 'Multiplied coupon value limit' and
;'Multiplied coupon count limit per transaction' static/edit text fields
[Options.Coupon.Multiplying 2]
/D:G.Ctrl03
/D:G.Ctrl08
/D:G.Ctrl22
/D:G.Ctrl26
```

```
[Options.User Option]
Ctrl01 = Notebook, 1,1,74,23, Left,AutoSize,10, Integers, Flags, Debug

[Options.User Option.Debug]
;**** TRANSLATORS **** MAXLENGTH=QUOTES
E.Label  = PageLabel, "Debug     "

E.Ctrl01 = StaticText, 1, 1,15, 0,"Memory Debugging Variables"
E.Ctrl02 = StaticText, 1, 2,1, 0,"Enable Memory Debugging :"
E.Ctrl03 = StaticText, 2, 4,1, 0,"Memory Threshold Value: "
E.Ctrl04 = StaticText, 2, 5,1, 0,"Write Debug Information In TLog: "
E.Ctrl05 = StaticText, 2, 6,1, 0,"Write Debug Information In SJ: "
```

Chapter 4. .ini File Descriptions

Significant fields of each `.ini` file are described in this section of the book.

ACEACRSR (Accounting Resources)

`aceacrsr.ini` is the accounting program (ACEACCNT) resources file. The file controls the menu entries for the Accounting user interface and closing accounting program messages. The file also specifies messages that are displayed in the accounting user interface during various accounting processes, such as reconciliation, closings, system status reporting, miscellaneous groups maintenance, account selection, automated reconciliation, miscellaneous transaction processing, and accounting summarization.

`aceacrsr.ini` is a *reserved* file that you should not modify.

ACEADRSR (ADDMI Batch Separator Resource)

`aceadrsr.ini` is an Alternate Delayed Data Maintenance Interface (ADDMI) program (ACEACCNT) resources file. The file specifies string values for status messages for batches.

This file is a *reserved* file that you should not modify.

ACEBINLG (BIN Report Log)

`acebinlg.ini` controls how the BIN File Rebuild application (acebirbl) rolls the acebinlg.dat file.

This file is a *modifiable* file.

Options Section

This section of the `INI` file specifies the file names that are used for the BIN report log file and the maximum size of the log file.

LOGNAME

Modifiable. The list of file names to be used when saving the log file if the size of the log file exceeds the maximum size specified on the SIZE option. Any number of files (comma delimited) may be listed on the option. When the current log file exceeds the maximum size, the log files are rolled (the last file in the list is deleted, the next to last file is renamed to the last file name. This rolling process continues until the current file is renamed to the second file name in the list and a new empty file, which is specified by the <ACEBINLG> logical name, is created to begin collecting the BIN report data).

The default value for this option specifies two log file names - a current file (acebinlg.dat) and a previous file (acebinlp.dat).

SIZE

Modifiable. The maximum size of the log.

Source Code

The following is source code for the `Options` section of the `acebinlg.ini` file.

```
[Options]
LOGNAME=ACEBINLG.DAT,ACEBINLP.DAT
SIZE=32000
```

ACEBNRSR (ACEBIRBL Resources)

`acebnrsr.ini` is the resources file for the acebirbl utility. Entries in the file control the text that is used in background messages related to the utility.

This file is a *reserved* file that you should not modify.

ACEBORSR (Main Menu Resources)

`aceborsr.ini` is the Main Menu program (ACEBORUN) resources file. Entries in the file control the text that appears on the Main Menu and its submenus for Manager Procedures and User Procedures. The file also includes entries for text that appears when an operator signs on to the Main Menu.

This file is a *modifiable* file.

Menu Section

This section of the INI file defines string values for items on the SurePOS ACE Main Menu, including text that appears beside each button.

BaseId

Reserved. A TurboVision control that becomes an enum for these menu strings.

Title

Modifiable. Name that appears at the top of the panel.

textWithinDoubleQuotes

Modifiable. Translatable strings that appear on the menu.

&tilde;*characterWithinTildes* &tilde;

Modifiable. Action characters that serve as shortcuts for menu navigation.

Button*n*

Modifiable. Text that appears on Buttons can contain up to 23 characters.

Comment*n*

Modifiable. Comments can contain up to 90 characters of text on two lines, with each being 45 characters. The first line ends at the end of the last full word before position 46. If a word overlaps position 46, SurePOS ACE moves the whole word to line 2. Ending double quotes should be at the end of the text on shorter lines and not at position 46.

Source Code

The following is partial source code for this section. Refer to the `aceborsr.ini` source file for the complete code.

```
[Menu]
BaseId=3000

Title = "SUREPOS ACE Main Menu"

Button3 = "&tilde;B&tilde;atch Data Maintenance "
Button4 = "&tilde;D&tilde;ata Maintenance       "

ExitButton = "     E&tilde;x&tilde;it     "

Comment1 = "...open/close store. Perform store operations"
Comment2 = "...produce in-store reports"
Comment3 = "...batch jobs of maintenance data"
Comment4 = "...add, change, or delete files records"
```

Mngr_Procs Section

This section of the INI file defines string values for items on the Manager Procedures Menu, including text that appears beside each button.

Field descriptions for this section are the same as those in "Menu Section" on page 468, except for the following:

BaseId

Reserved. A TurboVision control that becomes an enum for these menu strings.

Button*n*

Modifiable. Text that appears on Buttons can contain up to 23 characters.

Comment*n*

Modifiable. Comments can contain up to 60 characters of text on two lines, with each being 30 characters. The first line ends at the end of the last full word before position 31. If a word overlaps position 31, SurePOS ACE moves the whole word to line 2. Ending double quotes should be at the end of the text on shorter lines and not at position 31.

Source Code

The following is partial source code for this section. Refer to the aceborsr.ini source file for the complete code.

```
[Mngr_Procs]
BaseId=3050

Title = "Manager Procedures Menu"

Button1 = "   &tilde;T&tilde;erminal Monitor         "
Button2 = "   &tilde;S&tilde;elective Item Report    "

Comment1 = "...monitor terminals operation"
Comment2 = "...print selective items"
```

User_Procs Section

This section of the INI file defines string values for items that you can include on the User Procedures Menu, including text that appears beside each button.

BaseId

Reserved. A TurboVision control that becomes an enum for these menu strings.

Title

Name that appears at the top of the panel. The title line can be up to 60 characters between double quotes. The appearance is better if you put the ending quotes at the end of the text without extra spaces.

textWithinDoubleQuotes

Translatable strings that appear on the menu.

&tilde;*characterWithinTildes* &tilde;

Action characters that serve as shortcuts for menu navigation.

Button1-9

You can define buttons before adding procedure names using `Process 1-9`. SurePOS ACE requires a corresponding process before it will display the button. Button text is shown offset from the left in the source for appearance.

There is another factor influencing whether a button is displayed. An operator must be authorized (in Data Maintenance) to use User Procedure 1-8 for those buttons to appear on the menu that the operator sees. The button for User Procedure 9 always appears on the menu because operator authorization is not required to use it.

Comment1-9

Comments can contain up to 60 characters of text on two lines, with each being 30 characters. The first line ends at the end of the last full word before position 31. If a word overlaps position 31, SurePOS ACE moves the whole word to line 2. Ending double quotes should be at the end of the text on shorter lines and not at position 31.

Each comment displays next to the corresponding button. For example, comment1 displays next to button1. You can define comments before adding procedure names. SurePOS ACE requires a corresponding process before it will display the comment.

Process1-9

A process is a user-defined executable program. If a user process is only defined for button1, only button1 and comment1 (if it has been defined) appear on the panel. The name of a user process must be added to the right of the equals sign to define a user procedure.

To test the appearance of buttons and comments that you define, use a dummy name for the procedure, such as myprog, (without quotes). The program need not exist to see the button and comment.

Processes are assumed to be in the adx_ipgm directory. Do not use file name extensions when defining a process. For example, if the process is adx_ipgm \ myprog.386, define it as `Process1 = myprog`

Source Code

The following is partial source code for the User_Procs section. If this were an example of all the source code, only button1 and button2 would appear on the interface, because process3 is not defined. Refer to the `aceborsr.ini` source file for the complete code.

```
[User_Procs]
BaseId=3100

Title = "User Procedures Menu"
;
Button1 = "    User Procedure &tilde;1&tilde;    "
Button2 = "    User Procedure &tilde;2&tilde;    "
Button3 = "    User Procedure &tilde;3&tilde;    "

Comment1 = "...comment for User Procedure 1    "
Comment2 = "...comment for User Procedure 2    "
Comment3 = "...comment for User Procedure 3    "
Comment4 = "...comment for User Procedure 4    "

Process1 =
Process2 =
```

SignOn Section

This section of the INI file defines string values for items on the SurePOS ACE SignOn panel, including text that appears on buttons.

BaseId

Reserved. A TurboVision control that becomes an enum for these menu strings.

Title

Name that appears at the top of the password pop-up.

textWithinDoubleQuotes

Translatable strings that appear on the menu.

&tilde;*characterWithinTildes* &tilde;

Action characters that serve as shortcuts for menu navigation.

OperatorID, Password, NewPassword, and VerifyPassword

Variables that define string values for text that appears as labels beside entry fields.

ChangeButton

Variable that defines a string value for the button that operators must push to change a valid password.

Error

Variable that defines an operator message that appears when an operator attempts to sign on with an invalid password.

Source Code

The following is source code for the SignOn section of the `aceborsr.ini` file.

```
[SignOn]
BaseId=3200

Title  ="SUREPOS ACE SignOn"
OperatorID     =  "Operator ID:    "
Password       =  "Password:       "
ChangeButton   =  " C&tilde;h&tilde;ange "
NewPassword    =  "New Password:   "
VerifyPassword =  "Verify Password:"
Error          =  "Error signing on."
```

ChangePassword Section

This section of the `.ini` file defines the title of the Change Password pop-up.

BaseId

A TurboVision control that becomes an enum for these menu strings.

Title

Name that appears at the top of the password pop-up.

textWithinDoubleQuotes

Translatable strings that appear on the menu.

Source Code

The following is source code for the ChangePassword section of the `aceborsr.ini` file.

```
[ChangePassword]
BaseId=3240

Title  ="Change Password"
```

Error Section

This section of the `.ini` file defines string values for various errors that might occur.

BaseId

A TurboVision control that becomes an enum for these strings.

textWithinDoubleQuotes

Translatable strings that appear on the menu.

%variableName

Dynamically substituted variables. In this case, %1 in the `PasswordWarning` can be up to a 3 digit number. In `ErrorLoadingStub`, %1 can be up to an 8-digit number. In `BoProgNotFound`, %1 can be up to a 10-character file name.

Source Code

The following is partial source code for the Error section. Refer to the `aceborsr.ini` source file for the complete code.

```
[Error]
BaseId=3250

ProgNoFound = "Program Not Found                                           "
PasswordExpired = "The password has expired.
     "
;%1 can be  up to 3 digit number
PasswordWarning = "The password expires in %1 days.
     "
```

ACECDRSR (ACECNVDL Resources)

`acecdrsr.ini` is the resources file for the acecnvdl utility. Entries in the file control the text that is used in the log entries that the utility creates.

This file is a *reserved* file that you should not modify.

ACECFRSR.INI

`acecfrsr.ini` is the resources file for the acefirbl utility. Entries in the file control the text that is used in the log entries that the utility.

This file is a *reserved* file that you should not modify.

ACECFRSR (ACEFIRBL Resources)

`acecfrsr.ini` is the resources file for the acefirbl utility. Entries in the file control the text that is used in background messages related to the utility.

This file is a *reserved* file that you should not modify.

ACECLSLG (Close Log Rollover Settings)

The `adx_idt1\aceclslg.ini` file has two parameters that control how SurePOS ACE rolls the aceclslg.dat file to a new period, creating a previous period version and optionally, old versions of the file. One parameter, `LOGNAME`, identifies the names of the previous and any number of optional old period files. The other parameter, `SIZE`, specifies the threshold size of the aceclslg.dat file. The threshold is the maximum size, which triggers SurePOS ACE to roll the file by renaming it to the previous period file name, which is the second name in the `LOGNAME` list. You can add other names to the list. Positions three through the end of the list are old versions that SurePOS ACE will also roll.

The default setting for the `SIZE` parameter is 32000 bytes. You can modify the setting. The default `LOGNAME` list includes aceclslg.dat and aceclslp.dat. You can add the names of old period files to the list. When SurePOS ACE rolls the files, it deletes the last file named in the list, renames the next-to-the-last file to the last file name, renames the third-to-last to the second-to-last, and so on until it renames aceclslg.dat to aceclslp.dat. At that point, it creates a new aceclslg.dat file. You can use a text editor to view these distributed files.

See “ACECLSLG (Close Log)” on page 54 for information about the log file.

Modifying ACECLSLG.INI

You must *not* modify the `aceclslg.ini` file itself. Each field in `aceclslg.ini` is *reserved*. All changes should be made through user delta files, which are described in "Tiered Overlay Files" on page 460.

Note: Store the overlay files for `aceclslg.ini` in ADX.ID1.

ACECMISC (Miscellaneous Resources)

This miscellaneous resources file contains messages that are displayed during suspend/retrieve processing.

Modifying ACECMISC.INI

You must *not* modify the `acecmisc.ini` file itself. Each field in `acecmisc.ini` is *reserved*.

User extensions that you create might require that you modify this file to reflect your code changes. All changes should be made through user delta files, which are described in "Tiered Overlay Files" on page 460.

ACECPNTR (Coupon Translation Options)

The `acecpntr.ini` file is an options file like `optnparm.ini`. It contains options and menu specifications for coupon translation.

This file is a *reserved* file that you should not modify.

ACECPYRT (Copyright Information Resources)

To comply with legal requirements, the `acecpyrt.ini` file contains copyright statements. Depending on a personalization option, the copyright statements can display each time you sign on to SurePOS ACE.

Because copyright statements in this file must appear verbatim, this is a *reserved* file that you should not modify.

ACECSRSR (Global Resources)

This is the global resources file. It contains entries that control text that is displayed during recovery processing and auto report processing, and also database IDs that are used in *ISTLogProcessorControlStorage* processing.

This is a *reserved* file that you should not modify.

ACEDDESC (Department Descriptors)

`aceddesc.ini` contains department descriptors for SurePOS ACE.

This file is a *reserved* file that you should not modify. You should use the Descriptors Editor in the Personalization interface to manage department descriptors.

ACEDDMI (Delayed Data Maintenance)

The `aceddmi.ini` file includes user interface and batch control parameters.

This file is a *reserved* file that you should not modify. However, user extensions you make might require you to modify this file to reflect your code changes.

Modifying aceddmi.ini

Modify `aceddmi.ini` with user delta `.ini` files as described in "Tiered Overlay Files" on page 460.

To change string values, change only text between double quotes unless otherwise noted. It is important to not move the right double quote with certain exceptions, such as for titles. Unless specifically noted in the file, assume that the maximum length for any message or descriptor is the number of characters between double quotes. Some ending double quotes are past column 100 so if you do not see one when editing your user delta file, scroll to the right. In general, where text in double quotes ends with a series of periods, you can replace them with text.

Security Feature

Use a user delta file for `aceddmi.ini` to disable fields on the Item Data Maintenance interface that are displayed when you use the Item Data Editor in Batch Data Maintenance. Disabled fields appear on the interface and the user can view the data, but the data cannot be changed.

For each field that you want to disable, create a user delta file entry to turn off the TurboVision options flag (that is, set the value to -1) for the field. The options flag is the last field before the descriptor for radio buttons and check boxes, and the last field before the qualifier field for input lines. For example, the following examples set the option field to -1:

```
ItemCkBoxItemMovement    = CheckBoxes,50, 1,5,1,0,72,-1,Keep Item Movement
ItemTareRadio=RadioButtons,1,8,20,4,0,85,-1,"None ",\
"Tare ID ",\
"Tare Weight ",\
"Tare Percent "
ItemDealQtyInput=InputLine,50,8,2,61,0,-1,INTONLY
```

INIControl Section

See "INIControl Section Parameters" on page 452 for a description of these parameters.

Source Code

```
[INIControl]
Menu=MainMenu
RestoreMenus=TRUE
Validate=InputValidation
```

MainMenu Section

This section identifies the submenu that displays when you select Batch Data Maintenance on the Main Menu. It also specifies an action key to select the entry. Characters bracketed by &tilde; characters are highlighted and become action keys. Both tildes (&tilde;) are required, before and after the action character.

Source Code

```
[MainMenu]
Delayed Data Maintenance Batches=Submenu," ",0
```

Delayed Data Maintenance Batches Section

This section identifies menu selections that define dialogs for the most part. (One entry, `Exit`, defines a process.) See "Dialog" on page 454 and "Process" on page 454 for more information.

Source Code

Partial code for the Delayed Data Maintenance Batches section is included here. Refer to the `aceddmi.ini` source file for the complete code.

```
[Delayed Data Maintenance Batches]
Batch Status Display      =Dialog, "&tilde;B&tilde;atch Status Display     ",0,0,78,22,600,601,1000
Batch Control Screen      =Dialog, "&tilde;B&tilde;atch Control Screen     ",0,0,79,23,500,501,1100
Control Record Editor     =Dialog, "&tilde;C&tilde;ontrol Record Editor    ",0,0,78,23,200,201,1200
```

Appendix Section

This section identifies two other sections that define dialogs. See "Dialog" on page 454 for more information about the control.

Source Code

```
[Appendix]
IDM=Dialog, "Item Data Maintenance    ",0,0,78,22,0,0
ODM=Dialog, "Operator Data Maintenance",0,0,78,23,0,0,80
```

Delayed Data Maintenance Batches.Execution Manager Section

This is a dialog definition. See “Dialog Controls” on page 455 for an explanation of each control.

This is partial code for the Delayed Data Maintenance Batches.Execution Manager section. Refer to the `aceddmi.ini` source file for the complete code.

```
[Delayed Data Maintenance Batches.Execution Manager]
EBBatchList = ListBox,1,3,74,12,266,267,0,"<empty>             "
EBStatusStatic= StaticText,4,16,30,4,"           Execution Counter :"
EBStatus= InputLine,36,16,14,0,0,4,N
EBMarkButton=       Button,2,18,12,37,"  &tilde;M&tilde;ark      ",520
EBClearMarkButton= Button,14,18,19,37,"  &tilde;C&tilde;lear Mark       ",521
```

Delayed Data Maintenance Batches.Control Record Editor Section

This is a dialog definition. See “Dialog Controls” on page 455 for an explanation of each control.

Source Code

This is partial code for the Delayed Data Maintenance Batches.Control Record Editor section. Refer to the `aceddmi.ini` source file for the complete code.

```
[Delayed Data Maintenance Batches.Control Record Editor]
; ********** Batch ***************
BCIDStatic= StaticText,3,1,14,4,"Batch ID      "
BCID= InputLine,18,1,9,202,203,4,N
BCIDCombo=Combo,80,1,1,1,BCID, ,1
; ********** Restore Batch ***************
BCIDRestoreStatic= StaticText,3,3,14,4,"Restore ID    "
BCIDRestore= InputLine,18,3,6,204,205,4,N
; **********  Batch Status ***************
BCStatusStatic= StaticText,1,5,22,0,"Batch Status            "
BCCreateButton= Button,1,20,15,54,"Create &tilde;B&tilde;atch   ",240
BCSaveButton=  Button,16,20,13,54,"&tilde;S&tilde;ave Batch   ",241
```

InputValidation Section

This section specifies the rules used for validating data entered for the fields defined in the `aceddmi.ini` file. Refer to “Input Validation Data Types” on page 460 for information about each type of validation.

Source Code

```
[InputValidation]

R0_9       = RANGE,0,9     , NULL
```

```
R0_99      = RANGE,0,99    , NULL
R0_999     = RANGE,0,999   , NULL
R0_99999   = RANGE,0,99999 , NULL
R0_9999999 = RANGE,0,9999999, NULL
R1_999     = RANGE,1,999   , NULL
R0_49      = RANGE,0,49    , NULL
R0_50      = RANGE,0,50,     NULL
R0_32767   = RANGE,0,32767 , NULL
R0_8       = RANGE,0,8     , NULL
INTONLY    = INTEGER, +, NULL
CENTS      = AMOUNT, 0, 9999999999, NULL
```

Additional Sections

There are many other sections in the `aceddmi.ini` file that are not described here. Refer to the `aceddmi.ini` source file for the other sections.

ACEDDRSR (Delayed Data Maintenance Resources)

This is the resources file for delayed data maintenance that includes entries for messages that can occur during batch creation processing.

This file is a *reserved* file that you should not modify.

ACEDERSR (Descriptor Editor Resources)

This is the resources file for the Descriptor Editor. Entries in the file control text that appears in the user interface and messages that can occur during descriptor editing. It provides default text for various buttons, titles, and messages.

This is a *reserved* file that you should not modify.

ACEDMP (Application Dump Control)

The `acedmp.ini` file contains settings to enable controller storage dumps for specified applications. The file's contents only apply to SurePOS ACE applications and to applications built using the SurePOS ACE development environment.

Unhandled kernel and supervisor error codes, which have a high-order word of x'8000', result in an application abend. The `acedmp.ini` file specifies which applications may cause the controller to dump on an abend when the controller dump flag is off. When an application abend occurs, if the application is enabled to dump the controller, these activities occur:

1. If defined in the `acedmp.ini` file, an optional status message is displayed (on the console for console applications via 'cout' and on the background status for background applications).
2. A `X475 STORAGE DUMP PENDING` application event is logged.
3. The application pauses for the delay period specified in the `acedmp.ini` file.
4. A controller dump is initiated.

These are the keywords for sections in the `acedmp.ini` file:

DumpOnException

Indicates whether or not to dump the controller when an exception occurs. Valid values are `Y` or `N`.

Note: If the controller application dump flag for 4690 OS is enabled, then even if the application-specific dump flag is disabled, the operating system will initiate a dump of the controller when the application ends abnormally. This keyword has no effect on the 4690 controller dump flag; it enables dumping the controller for specific applications when the controller dump flag is OFF. This keyword does not have an impact on the communications dump flag.

DumpDelay

Specifies the number of seconds to delay between after the program exception occurs before initiating a dump. Valid settings are 1 to 64800 (64800 = 18 hours). Values outside this range or nonvalid values (such as alphabetic characters) will cause the delay to be set to the default of 60 seconds.

DumpMessage

Specifies the message text to display on the console or as a background status message while the `DumpDelay` period is active. Only the first 46 bytes will be displayed on the background screen.

If you define a message for a background application, the message will be displayed in the background application's message area. If you cancel the background application by pressing F8 while the message is displayed, the dump will not occur.

If you define a message for a foreground application, the message will be written to stdout via cout. The screen position of the message will depend on the location of the cursor on the screen, which is not necessarily the same position as the cursor on the TurboVision UI. Also, the description on the 4690 Window Control screen will be updated with the message text. You cannot use Control+C to cancel the application while the message is displayed. However, you can use the 4690 Window Control screen to cancel the process.

Most programs that are built using the SurePOS ACE development environment already include the necessary `lib`s to support this new function. For customer executables that do not support INI files, you might need to modify their make files to include ACECOMMN$L so that the necessary objects are available for linking.

For more information on extending dump support for customer executables, see “Extending Dump Support” on page 634.

Modifying acedmp.ini

If you need to modify the file, use overlay `.ini` files as described in “Tiered Overlay Files” on page 460. Because the default `acedmp.ini` file only contains a process section for ACECSMLL, you will need to specify most user changes as an add function for both the section and the keyword.

Globals Section

The Globals section defines the default settings to be used for all SurePOS ACE application processes unless overridden in a process section.

Source Code

This is the source code for the Globals section of the `acedmp.ini` file.

```
[Globals]
DumpOnException=N
DumpDelay=60
DumpMessage="Exception caught. Storage dump pending.       "
```

Sections for Processes

In general, the name of a process section is the same name as either the logical name or the program name less the extension, depending on whether the process is started using the logical name or the program name. The name of a process section is always specified in lowercase characters and contains no more than eight characters. If the logical name contains more than eight characters, only the first eight characters should be used.

If there is not a direct match between a logical name and the program name, as occurs with the SurePOS ACE Scheduler and the SurePOS ACE EPS Host Link Handler, the process name may be different than expected.

- For the Scheduler, the logical name ACESCHDLR (ace:aceschdl.386) uses process name `aceschdl` and would be specified as section [aceschdl].
- For the Host Link Handler, the logical name ACEAPSHCH (ace:aceapshl.386) uses process name `aceapshc` and would be specified as section [aceapshc]. If started using the program name ace:aceapshl.386, the section name would be [aceapshl].

The values specified in a process section will override values specified in the Globals section. When a keyword is not present in a process section, the default setting from the Globals section is used.

Source Code

This is the source code for the only process section that SurePOS ACE provides in the `acedmp.ini` file.

```
[acecsmll]
DumpOnException=N
```

ACEDMRSR (Data Maintenance Resources)

`acedmrsr.ini` is the resources file for Data Maintenance. It includes entries for messages that can occur during maintenance of item data records, operator authorization records, tender verification records, customer records, and targeted coupon messages.

It also contains report formatting definitions for the Item Detail Report and user interface definitions for the Customer Data Maintenance dialogs.

This file is a *reserved* file that you should not modify.

ACEDSCRE (Descriptor Editor)

The `acedscre.ini` file contains the following sections that are used by the SurePOS ACE Descriptor Editor, which is available from the Miscellaneous pull-down of the Personalization menu:

- Configuration settings for the editor
- Category lists of descriptors that belong to specific reports and report areas
- Category lists of descriptors that belong to specific areas of the sales application
- Category lists of descriptors for specific departments

The `acedscre.ini` file is a *reserved* file that you should not modify. However, user extensions you make might require you to modify this file to reflect your code changes.

Modifying acedscre.ini

Modify `acedscre.ini` with overlay `.ini` files as described in "Tiered Overlay Files" on page 460. The logical name for the `.ini` file is <ACEDEINI>, so the file name of any overlay files must be acedeini.

You can change the value on each option in the Options section, but the option names are *reserved*. Because the entries in the ReportLists, SalesLists, and DepartmentLists sections are only used to determine what descriptors are displayed in the Descriptor Editor, you can change the option names and/or their values.

Options Section

AutoSave

This option defines whether to save edits as they are made or require the operator to press a button to save their edits. If this option is set to Y, all changes are saved immediately, which means that there is no need for a Save button, so it is removed from the interface for the Descriptor Editor.

AllowBXXXEdit

This option determines whether the operator can edit `Bxxx` identifiers, which appear at the beginning of many descriptors, with the Descriptor Editor. If this option is set to N, the operator can edit only the body of the descriptor. If this option is set to Y, the operator can edit the Bxxx identifier and the body of the descriptor.

If this option is set to N and there is a space character following the `Bxxx`, the space is also protected. If the descriptor initially has no space following the `Bxxx`, the operator can edit all characters following the `Bxxx`. However, if the operator saves a descriptor with a space immediately after the `Bxxx`, that operator will not be able to edit the space again.

```
[Options]
AutoSave=N
AllowBXXXEdit=N
```

ReportLists Section

This section defines lists of descriptors that belong to specific reports and report areas such as Department Subtotals. An operator can select a list after selecting the Lists push button in the Report Descriptors Editor in Personalization. Multiple lists can be selected by selecting the push button multiple times and selecting a different list each time.

```
[ReportLists]
Common=1-6,91-92
Terminal Productivity=140-160
Tender Listing=960-987
Store Totals Recap=53,1106
System Status=63,430-451,1067,4311-4316
Operator Sales=7,25,52,1105
Operator Performance=7,25,54,620-740,1000-1039,1041-1048,3301-4300
Over Short=7,8,25-28,55,1060-1068
Miscellaneous Transactions Recap=61,500-505,840-960
Item Movement=10-11,21-22,60,410-424
Audit Trail=510-545,551-599
Cash Drawer Position=7-8,25-28,57,400-407
Cash Report=7-8,25-26,51,1100-1104,1107-1148,1150-1166
Department Analysis=58-59,600-614,617-618,620-840,1050-1059,3301-4300
Transaction Log=1651,1653-1712,1862-1986
Exception Log=510-539,542-545,551-599,1551-1650,1652-1861,2847-2848
Department Subtotals=620-840,3301-4300
```

SalesLists Section

This section defines lists of descriptors that belong to specific areas of the terminal sales application. These lists are not mutually exclusive. A descriptor or range of descriptors can appear in more than one list. This lets you include a descriptor such as `B064 COUPON MUST MATCH PREVIOUS SALE` in both an *Operator Guidances* list and a *Coupon Descriptors* list.

An operator can select a list after selecting the Lists push button in the Report Descriptors Editor in Personalization. Multiple lists can be selected by selecting the push button multiple times and selecting a different list each time.

```
[SalesLists]
Operator Guidances=1-120,132,140-152,169,525-529
Authorization Guidances=300-332
Preferred Customer=600-614
Suspend/Retrieve=671-689,773,776,778,780,783,785-786,4500-4519
Check Printing=864-865,870,872-874,876
Password=877,878,1039
Tender Names=1001-1030
Sales Prompts=1031-1038,1040-1046
External Authorization=3000-3018
NOSALE-TOTAL Help=4000-4032
Override Descriptions=1047-1054
Coupon Descriptors=1,64-66,69,609-613,1100,1108-1109,1081,1112,1170-1171
Volume Discount=1110-1111,1191-1192
```

DepartmentLists Section

This section defines lists of department descriptors for grouping departments. A descriptor or range of descriptors can appear in more than one list.

An operator can select a list after selecting the Lists push button in the Department Descriptors Editor in Personalization. Multiple lists can be selected by selecting the push button multiple times and selecting a different list each time.

Because each store can assign its own set of departments to the department descriptors, it is difficult to provide meaningful default lists of descriptors. Therefore, there is no DepartmentLists section in the `acedscre.ini` file supplied with SurePOS ACE. This is an example of how to code the section:

```
[DepartmentLists]
Default Departments=1,101,201,301,401,501,601,701,801,901
Meat and Poultry=101-199
```

ACEDTMNI (Data Maintenance)

The `acedtmni.ini` file defines user interface controls and validation policies for the Data Maintenance program.

This file is a *reserved* file that you should not modify. However, user extensions you make might require you to modify this file to reflect your code changes.

Modifying acedtmni.ini

Modify `acedtmni.ini` with user delta `.ini` files as described in "Tiered Overlay Files" on page 460.

To change string values, change only text between double quotes unless otherwise noted. It is important that you do not move the right double quote with certain exceptions, such as for titles. Unless specifically noted in the file, assume that the maximum length for any message or descriptor is the number of characters between double quotes. Some ending double quotes are past column 100 so if you do not see one when editing your user delta file, scroll to the right. In general, where text in double quotes ends with a series of periods, you can replace them with text.

If you rename either default value for the *ID* option in EPS -> Value Cards -> Card Plan personalization, you must overlay the `acedtmni.ini` file with a corresponding change so that Data Maintenance recognizes the new value. For example, if you rename ID 61 to 59, you must include the following in an overlay file:

```
[Data Maintenance.Item Data Maintenance.Value Card]
/a:ValueCardTypeCombo  =   \
Combo, 25, 1, 0, 64, ValueCardTypeInput, "00 Not a Card",\
                                         "59 Gift Card",\
                                         "62 Phone Card", \
                                         "63 Blackhawk Card",\
                                         "71 Lottery"
```

Security Feature

Use a user delta file for `acedtmni.ini` to disable fields on the Item Data Maintenance interface. Disabled fields appear on the interface and the user can view the data, but the data cannot be changed.

For each field that you want to disable, create a user delta file entry to turn off the TurboVision options flag (that is, set the value to -1) for the field. The options flag is the last field before the descriptor for radio buttons and check boxes, and the last field before the qualifier field for input lines. For example, the following examples set the option field to -1:

```
ItemCkBoxItemMovement    = CheckBoxes,50, 1,5,1,0,72,-1,Keep Item Movement
ItemTareRadio=RadioButtons,1,8,20,4,0,85,-1,"None ",\
"Tare ID ",\
"Tare Weight ",\
"Tare Percent "
ItemDealQtyInput=InputLine,52,8,2,61,0,-1,INTONLY
```

INIControl Section

See “INIControl Section Parameters” on page 452 for information about parameter specifications in this section.

Source Code

```
[INIControl]
Menu=MainMenu
RestoreMenus=TRUE
Validate=InputValidation
```

MainMenu Section

This is a dialog definition. See “Menu Controls” on page 453 for an explanation of each control.

Source Code

```
[MainMenu]
Data Maintenance = Submenu, "&tilde;D&tilde;ata Maintenance", 0
```

Data Maintenance Section

This section defines the Data Maintenance pull-down menu. The width of the pull-down menu window expands to that of the longest menu item. You can expand space between double quotes to a maximum of 50 characters. These menu items are also panel titles that appear with the active notebook title.

Source Code

This is a dialog definition. See “Dialog Controls” on page 455 for an explanation of each control.

The following is partial source code for this section. Refer to the `acedtmni.ini` source file for the complete code.

```
[Data Maintenance]
Tender Verification Maintenance=Dialog, "&tilde;T&tilde;ender Verification Maintenance",\
0,0,77,21,100,101,1200
Exit                           =Process,"E&tilde;x&tilde;it                           ",\
-1,1520
```

Data Maintenance.Item Data Maintenance Section

This section defines the notebook pages for Item Data Maintenance.

Source Code

This is a dialog definition. See “Dialog Controls” on page 455 for an explanation of each control.

```
[Data Maintenance.Item Data Maintenance]
; DIALOG CONTROL BUTTONS
ItemCodeStatic = StaticText,18,1,14,0,"Item Code     "
ItemCodeInput  = InputLine,18,2,14,54,55,4,X
ItemDescriptionStatic = StaticText,38,1,18,0,"Description       "
ItemDescriptionInput  =  InputLine,38,2,18,57,0,4,X
ItemDepartmentStatic  =  StaticText,66,1,26,0,"Dept.   "
ItemDepartmentInput   =  InputLine,66,2,3,0,81,4,INTONLY
;   NOTEBOOK
ItemNotebook=Notebook,1,3,75,17,Left,Standard,14, Pricing Data, \
Coupon,        Item Type,     Item Controls, \
Misc / Coupon, Taxes,         User Data,     \
Misc.Coupons,  Optional
ItemPrintButton     = Button,29,21,12,54," &tilde;P&tilde;rint   ",95
ItemLookupButton     = Button,3,21,12,54," &tilde;L&tilde;ookup  ",90
ItemSaveButton      = Button,16,21,12,54,"  &tilde;S&tilde;ave  ",91
ItemEraseButton     = Button,42,21,12,54," &tilde;E&tilde;rase   ",92
ItemHostPriceButton = Button,54,21,21,54," Show &tilde;H&tilde;ost Pricing ",93
```

Data Maintenance.Item Data Maintenance.Pricing Data Section

This is a dialog definition. See “Dialog Controls” on page 455 for an explanation of each control.

Source Code

This is partial code for the section. Refer to the `acedtmni.ini` source file for the complete code.

```
[Data Maintenance.Item Data Maintenance.Pricing Data]
Label1 = PageLabel, "Pricing Data  "
ItemPMButtonsStatic= StaticText,1,1,18,0,"Pricing Method    ",R
ItemPMButtonsInput= RadioButtons,1,2,23,9,52,52,12,"0- Split Package  ",  \
"1- Base  Plus One ",  "2- Group Threshold",  \
"3- Group Adjusted ",  "4- Unit  Adjusted ",  \
"5- Alias Pricing  "
ItemUnitPriceStatic=StaticText,25,2,5,0,"Unit Price       ",R
ItemUnitPriceInput= InputLine,44,2,10,58,56,4,CENTS8
```

Data Maintenance.Item Data Maintenance.Coupon Section

This is a dialog definition. See "Dialog Controls" on page 455 for an explanation of each control.

Source Code

This is partial code for the Data Maintenance.Item Data Maintenance.Coupon section. Refer to the `acedtmni.ini` source file for the complete code.

```
[Data Maintenance.Item Data Maintenance.Coupon]
Label2 = PageLabel, "Coupon        "
CouponTypeStatic=StaticText,1,1,30,0,"Coupon Type                  "
CouponTypeRadio=RadioButtons,1,2,30,6,0,80,12,"Fixed Value              ",\
"Net Value               ","Percent Off Link        ",\
"Percent Off Validation  ","Multiple Items          ", \
"Fake Coupon             "
CouponAmountStatic=StaticText,32,2,10,0,"Amount    "
CouponAmountInput=InputLine,44,2,6,78,79,4,CENTS
MatchingStatic=StaticText,1,9,38,0,"Matching Items Required               "
```

InputValidation Section

This section specifies validation rules used for validating numeric data entered for input line fields defined in the `acedtmni.ini` file. Refer to "Input Validation Data Types" on page 460 for information about each type of validation.

When an operator selects Data Maintenance, SurePOS ACE creates validations for each input line field as it initializes the interface. It uses the validation for an input field when an operator enters data into that field.

Existing validation rules in the `InputValidation` section are *reserved* specifications. Data Maintenance uses them to verify specific types and lengths of data that an operator enters in input line fields.

The second and third parameters of most validation rules define the range of values accepted by the rule. In the case of the `INTONLY` rule, there is no range, but rather a plus sign (+) to specify that only positive values are acceptable:

```
INTONLY    = INTEGER, +, NULL
```

The association between a validation rule and an input line field is specified in the definition of the input field. For example, `INTONLY` defines the rule for validating the `TenderIDInput` input field in this defintion:

```
TenderIDInput=  InputLine,25,2,2,102,103,4,INTONLY
```

Source Code

```
[InputValidation]
R0_9       = RANGE,0,9     , NULL
R0_99      = RANGE,0,99    , NULL
R0_999     = RANGE,0,999   , NULL
R0_99999   = RANGE,0,99999 , NULL
R0_9999999 = RANGE,0,9999999, NULL
R1_999     = RANGE,1,999   , NULL
R0_49      = RANGE,0,49    , NULL
R0_50      = RANGE,0,50,     NULL
R0_32767   = RANGE,0,32767 , NULL
R0_8       = RANGE,0,8     , NULL
INTONLY    = INTEGER, +, NULL
CENTS      = AMOUNT, 0, 9999999999, NULL
CENTS8     = AMOUNT, 0, 99999999, NULL
CENTS5     = AMOUNT, 0, 99999, NULL
CENTS3     = AMOUNT, 0, 999, NULL
CENTS4     = AMOUNT, 0, 9999, NULL
DECML1_999 = DECIMAL, 1, 999, NULL
CENTS94999 = AMOUNT, 0, 94999, NULL
```

Data Maintenance.Tender Verification Maintenance Section

This is a dialog definition. See “Dialog Controls” on page 455 for an explanation of each control.

Source Code

This is partial code for the Data Maintenance.Tender Verification Maintenance section. Refer to the `acedtmni.ini` source file for the complete code.

```
[Data Maintenance.Tender Verification Maintenance]
TenderIDStatic= StaticText,24,1,15,0,"Tender ID      "
TenderIDInput=  InputLine,25,2,2,102,103,4,INTONLY
TenderAccountStatic= StaticText,44,1,28,0,"Customer Account Number     "
TenderAccountInput= InputLine,44,2,24,104,105,4,INTONLY
TenderNotebook=Notebook,1,3,75,16,Left,Standard,14,Account Status,\
Account Limits
TenderSaveButton= Button,42,19,15,54,"    &tilde;S&tilde;ave     ",141
TenderEraseButton= Button,60,19,15,54,"    &tilde;E&tilde;rase    ",142
```

Data Maintenance.Tender Verification Maintenance.Account Status Section

This is a dialog definition. See “Dialog Controls” on page 455 for an explanation of each control.

Source Code

```
[Data Maintenance.Tender Verification Maintenance.Account Status]
Label1 = PageLabel, "Account Status"
TenderStatButtonsStatic= StaticText,1,1,40,0,"Status of Account                 "
TenderStatButtonsInput= RadioButtons,1,2,52,8,110,111,4, \
"Status Code 50 : Accept                      ",\
"Status Code 55 : Reject - risk 1             ",\
"Status Code 56 : Reject - risk 2             ",\
"Status Code 57 : Reject - risk 3             ",\
"Status Code 58 : Reject - risk 4             ",\
"Status Code 60 : Accept - no fee             ",\
"Status Code 70 : User defined                "
```

Data Maintenance.Tender Verification Maintenance.Account Limits Section

This is a dialog definition. See "Dialog Controls" on page 455 for an explanation of each control.

Source Code

```
[Data Maintenance.Tender Verification Maintenance.Account Limits]
Label2 = PageLabel, "Account Limits"
TenderAmtStatic= StaticText,41,1,15,0,"Limit Amount   "
TenderMaxStatic= StaticText,1,2,39,0,"For ONE check/tender transaction       "
TenderMax1Input = InputLine,41,2,10,106,107,4,CENTS8
TenderMaxAllStatic= StaticText,1,4,39,0,"For ALL check/tender transactions      "
TenderMaxAllInput=InputLine,41,4,10,108,109,4,CENTS8
```

Data Maintenance.Operator Authorization Records Section

This section defines the notebook pages for Item Data Maintenance.

Source Code

This is a dialog definition. See "Dialog Controls" on page 455 for an explanation of each control.

This is partial code for the Data Maintenance.Operator Authorization Records section. Refer to the `acedtmni.ini` source file for the complete code.

```
[Data Maintenance.Operator Authorization Records]
OperIDStatic= StaticText,26,1,12,0,"Operator ID"
OperIDInput= InputLine,26,2,9,154,155,4,N
OperatorNotebook=Notebook,2,3,74,18,Left,Standard,21, Password ,\
Terminal Procedures,Terminal Functions,Operations,\
Accounting Reports,Sales Reports,Listing Reports,\
Performance Reports,Data Maint Reports,Misc Reports,\
User Reports,Delayed Data Maint,Data Maint,\
Personalization,User Procedures,Electronic Journal
```

```
OperatorLookupButton= Button,26,21,12,54,"  &tilde;L&tilde;ookup    ",190
OperatorSaveButton= Button,37,21,12,54,"   &tilde;S&tilde;ave     ",191
```

Data Maintenance.Operator Authorization Records.Password Section

This is a dialog definition. See “Dialog Controls” on page 455 for an explanation of each control.

Source Code

```
[Data Maintenance.Operator Authorization Records.Password]
Label1 = PageLabel, "Password             "
OperPassStatic= StaticText,2,2,24,8,"Operator Password       "
OperPassInput= InputLine,26,2,8,0,193,140,INTONLY
OperExpireInput= CheckBoxes,2,5,35,1,0,0,12,"Expire Password? "
```

Data Maintenance.Operator Authorization Records.Terminal Procedures Section

This is a dialog definition. See “Dialog Controls” on page 455 for an explanation of each control.

Source Code

```
[Data Maintenance.Operator Authorization Records.Terminal Procedures]
Label2 = PageLabel, "Terminal Procedures  "
OP0= CheckBoxes,1,1,44,16,0,0,12,"Sales Transaction                     ",\
"Tender Cashing                       ",\
"Tender Exchange                      ",\
"Loan  Transaction                    ",\
"Pickup Transaction                   ",\
"Tender Listing                       ",\
"Price Verify \ Change                ",\
"Training Mode                        ",\
"Terminal Transfer                    ",\
"Terminal Monitor                     ",\
"Tender Count                         ",\
"Return Item                          ",\
"WIC Transaction                      ",\
"Department Totals                    ",\
"User Non-Sales 1                     ",\
"User Non-Sales 2                     "
```

Additional Sections

There are many other sections in the `acedtmni.ini` file that are not described here. Refer to the `acedtmni.ini` source file for the other sections.

ACEEJRSR (Electronic Journal Resources)

This `.ini` file contains definitions for background messages used by ejrnlapp, for static text used in the ejfsldlg and ejfvwdlg dialogs, for static text displayed when About is selected on the Help pull-down menu, and for buttons used in the two dialogs.

This file is a *reserved* file that you should not modify.

ACEESRSR (ACEESCLM Resources)

`aceesrsr.ini` is the resources file for aceesclm, the shelf label utility. Entries in the file control the text that is used in background messages related to the utility.

This file is a *reserved* file that you should not modify.

ACEEVRSR (Electronic Voucher Authorization Resources)

The `aceevrsr.ini` file identifies string values that appear in the electronic voucher user interface as menu and dialog titles, field labels, button labels, error messages, and status messages.

This file is a *reserved* file that you should not modify.

ACEFMTDS (Output Formatting for 2x20 Display)

The `acefmtds.ini` file defines output formatting for the 2x20 display.

Modifying acefmtds.ini

All fields in the `acefmtds.ini` file are *modifiable* with the following exception: section names and the *Id* and *FormatType* fields are *reserved*. Modify `acefmtds.ini` with tiered overlay files (see "Method for C/C++ Files" on page 461) instead of modifying the base file. In addition to tiered overlay files, C++ extensions are required if you want to add new output fields or new output formats. See the sections that follow for more information.

Adding a New Customer Output Format

Customers can add up to 30 formats to the output format `.ini` files using tiered file overlays. If you use a tiered overlay file to add a new customer format, the syntax specifies that the ID is a customer ID. See Figure 8.

```
/A:[MyNewFormat]
/A:Id           = Customer, 20
```

Figure 8. Syntax of Adding a New Customer Format for Tiered Overlay File

C++ extensions are required when you add a customer output format. To implement a new output format:

1. Modify the tiered overlay file for the output format INI file to add the new format.
2. Rebuild the output format files by issuing the make -f makefile.gen outputFormatFiles command.
3. Modify your code to call your new format, using *CustomerOutputFormatFactory_xxx* (where *xxx* is the name of the new format that is defined as a section in your overlay file (for example, *MyNewFormat* in Figure 8).
4. Rebuild SurePOS ACE.

Note:

The # sign is used on an output format if the definition of the output line would exceed the maximum length supported by the output device (2 x 20, full screen display or printer).

Adding a New Customer Output Field

C++ extensions are also required when you add a customer output format field. To implement a new output field:

1. Subclass the appropriate output formatter (see Table 109).
2. Implement a new *getFunc* routine for the subclassed object. Model the routine after the base routine. "Switch" on functype.
   - Add cases for your new field types for this output format.
   - The Default case should call the parent *getFunc* routing, and pass the same parameters as the base routine.
3. Modify the tiered overlay file for the output format `.ini` file to add the new field.
4. Rebuild the output format files by issuing the make -f makefile.gen outputFormatFiles command.
5. Rebuild SurePOS ACE.

Format and Field Types for Output Format INI Files

Table 109 lists the format types, and Table 110 through Table 116 list the field types that you can use in the output format `.ini` files. The types are case sensitive.

New format types and field types may be added via maintenance, enhancement, and extension activities, so these lists may change as time passes.

Table 109. Possible Format Types Used in Output Format .ini Files

Format Type	Description	File
T	Transaction Formatter	`TNSCTFMT.CPP`
N	Tender Entry Formatter	`TNDRFRMT.CPP`
I	Item Entry Formatter	`ADITMFMT.CPP`
W	Workstation Formatter	`WRKSTFMT.CPP`
D	Discount Entry Formatter	`DSCNTFMT.CPP`
A	Accounting Tender Entry Formatter	`ACCTNFMT.CPP`
E	EM ID Formatter	`EMIDFRMT.CPP`

Format Type	Description	File
P	Price Verify Formatter	`PRCVRFMT.CPP`
0-4	Reserved for Toshiba Global Commerce Solutions Business Partners	
5-9	Reserved for customers	

Table 110. Transaction Formatter Field Types

Field Type	Description
A	Sales transaction amount due before taxes
B	Sales transaction amount due after taxes
D	Tax due after exemptions applied
E	Completed transaction total
F	Foodstamp balance allowed
G	Foodstamp total amount
H	VAT header for sales transaction
I	Number of items sold in this transaction
J	Transaction total (with coupons applied)
K	Sales transaction amount due after taxes, but before this tender was added
M	Total taxes before this tender was added
N	Transaction number for this transaction
n	Node ID for a locally suspended or retrieved transaction
R	VAT description for sales transaction
r	Original transaction number of retrieved transaction
S	Transaction total description
T	Transaction total
U	Foreign balance
V	VAT table for the sales transaction
W	Workstation/terminal ID for this transaction
w	Original workstation/terminal ID of retrieved transaction
X	Total taxes for the sales transaction
Y	Completed transaction foreign total
0-4	Reserved for Toshiba Global Commerce Solutions Business Partners
5-9	Reserved for customers

Table 111. Tender Entry Formatter Field Types

Field Type	Description
A	Account number associated with this tender

Field Type	Description
B	Sales transaction balance due after this tender is applied
C	Change returned after this tender is applied
D	Tender description
E	Expiration date associated with this tender
F	Amount of the fee charged for accepting this tender
G	Customer name
M	Manual tender indicator
N	Name of the tender (same as description)
O	Override indicator
S	Short tender description
T	Amount tendered in this entry
U	Exchange rate
0,2-4	Reserved for Toshiba Global Commerce Solutions Business Partners
1	Modifier for the print line
5-9	Reserved for customers

Table 112. Item Entry Formatter Field Types

Field Type	Description
B	Deal price
C	Coupon type descriptor
D	Item description
E	Normal sales price
F	Fuel Volume
H	Threshold
I	LQD price
M	Manual weight indicator
N	Pricing method
O	Override indicator
P	Price
Q	Quantity
R	Repeat count
S	S amount for pricing method 2 items
T	Taxability indicator
U	Unit price
V	Velocity code

Field Type	Description
W	Weight
X	Points multiply percentage
Y	Item type
Z	Deal quantity
b	Points-only item - bonus/redeemed modifier
c	UPC
e	Fuel Unit Price
f	Manual fuel indicator
i	Points-only item - base item points value
n	Points do not apply to the item
p	Points-only item - points
q	Coupon net value amount
v	Coupon value
x	Extended net value (net value × quantity or weight)
0-4	Reserved for Toshiba Global Commerce Solutions Business Partners
5-9	Reserved for customers

Table 113. Workstation Formatter Field Types

Field Type	Description
D	Date
M	Time in military format
N	Transaction number
O	Operator ID
S	Store ID
T	Time
W	Terminal ID
0-4	Reserved for Toshiba Global Commerce Solutions Business Partners
5-9	Reserved for customers

Table 114. Discount Entry Formatter Field Types

Field Type	Description
A	Discount amount
D	Discount group description
E	Total tax exempt amount
I	Tax indicator
O	Override indicator

Field Type	Description
P	Discount group percentage
T	Total amount available for discount
V	Void indicator
0-4	Reserved for Toshiba Global Commerce Solutions Business Partners
5-9	Reserved for customers

Table 115. Accounting Tender Entry Formatter Field Types

Field Type	Description
A	Tender amount
D	Tender description
O	Override indicator
Q	Tender quantity
T	Total tender amount
S	Shorter accounting transaction description
X	Short accounting transaction description
0-4	Reserved for Toshiba Global Commerce Solutions Business Partners
5-9	Reserved for customers

Table 116. EM ID Formatter Field Types

Field Type	Description
I	EM customer ID
N	EM customer name
W	EM customer welcome message (depends on the customer ID)
0-4	Reserved for Toshiba Global Commerce Solutions Business Partners
5-9	Reserved for customers

Table 117. Price Verify Formatter Field Types

Field Type	Description
V	Coupon value
R	Number of items required for the coupon
p	Coupon percentage
d	Item description
i	Item price
0-4	Reserved for Toshiba Global Commerce Solutions Business Partners
5-9	Reserved for customers

ACEFMTFS (Output Formatting for Fullscreen Display)

The `acefmtfs.ini` file defines the output formatting for the fullscreen display. See "Format and Field Types for Output Format INI Files" on page 491 for the different types of formats and fields that you can define.

Modifying acefmtfs.ini

All fields in the acefmtfs.ini file are *modifiable* with the following exception: section names and the *Id* and *FormatType* fields are *reserved*. Modify `acefmtfs.ini` with tiered overlay files (see "Method for C/C++ Files" on page 461) instead of modifying the base file.

In addition to tiered overlay files, C++ extensions are required if you want to add new output fields or new output formats. See "Adding a New Customer Output Format" on page 490 and "Adding a New Customer Output Field" on page 491 for more information.

ACEFMTGR (Output Formatting for Gift Receipts)

The `acefmtgr.ini` file defines the output formatting for gift receipts.

Gift receipts use sections `Fuel`, `Weight`, `SingleQuantity`, and `MultipleQuantityOneLine`. Fields `D`, `F`, `I`, `Q`, `W`, and `c` can be specified in these sections. See Table 112 for more information about these fields.

Modifying acefmtgr.ini

All fields in the `acefmtgr.ini` file are *modifiable* with the following exception: section names and the *Id* and *FormatType* fields are *reserved*. Modify `acefmtgr.ini` with tiered overlay files (see "Method for C/C++ Files" on page 461) instead of modifying the base file.

In addition to tiered overlay files, C++ extensions are required if you want to add new output fields or new output formats. See "Adding a New Customer Output Format" on page 490 and "Adding a New Customer Output Field" on page 491 for more information.

ACEFMTPR (Output Formatting for Printer)

The `acefmtpr.ini` file defines the output formatting for the printer. See "Format and Field Types for Output Format INI Files" on page 491 for the different types of formats and fields that you can define.

Modifying acefmtpr.ini

All fields in the `acefmtpr.ini` file are *modifiable* with the following exception: section names and the *Id* and *FormatType* fields are *reserved*. Modify `acefmtpr.ini` with tiered overlay files (see "Method for C/C++ Files" on page 461) instead of modifying the base file.

In addition to tiered overlay files, C++ extensions are required if you want to add new output fields or new output formats. See "Adding a New Customer Output Format" on page 490 and "Adding a New Customer Output Field" on page 491 for more information.

ACEGIPC (GIPC)

The source file name for this `.ini` file is `gipc.ini`. During the generation of the installation diskettes, the file is renamed to `acegipc.ini`.

`acegipc.ini` is the initialization file for the guaranteed interprocess communication (GIPC) program.

Modifying acegipc.ini

The `acegipc.ini` file is a *reserved* file that you should not modify. If you need to modify the file, use overlay `.ini` files as described in "Tiered Overlay Files" on page 460. The file name of any overlay files must be `gipc_ini`, not `acegipc.ini`.

Use extreme caution when modifying these parameters. If you are not sure of what you are changing, *do not* change it.

The `X331 UNABLE TO WRITE TO GIPC` message sometimes occurs when your store options are set to capture check images. In that case, you can avoid the error by creating a `gipc_ini.usx` overlay file with a `MaxReadLength` parameter value greater than the default value.

Constants Section

MaxReadTime

Maximum number of milliseconds the spooler waits for a message.

MaxReadLength

Maximum length of a GIPC message.

WaitAckTime

Number of milliseconds to wait for an Acknowledgement (Ack) response to a sent message.

WaitSendTime

Number of milliseconds to wait after an Ack response to send the next message.

WaitResendTime

Number of milliseconds to wait after a send failure before retrying the send.

DefaultExpiryDays

Number of days to keep a message (0 = infinite; never delete).

MinimumPercentRecoverable

Minimum percent of recoverable space before a reorganization is done.

MinimumHoursBetweenReorganizations

Minimum number of hours after a reorganization before another is done.

MaximumRecordsToCopy

Maximum number of records to read from the spool during each iteration when reorganizing.

BacklogThreshold

Number of spooled messages required before GIPC is considered backogged.

Source Code

```
[Constants]
MaxReadTime=15000
MaxReadLength=40000
WaitAckTime=30000
WaitSendTime=150
WaitResendTime=60000
DefaultExpiryDays=0
MinimumPercentRecoverable=10
MinimumHoursBetweenReorganizations=1
MaximumRecordsToCopy=50
BacklogThreshold=5000
```

ACEGLRSR (Global Generic Interface Text)

The `aceglrsr.ini` file contains global resources used by the SurePOS ACE interface. These resources include labels for push buttons, text for menu bar items, and error messages. Because the Manager's Audit Trail report can be accessed from both the Accounting and the Reports functions, column headers and other strings used in the report are included in this `.ini` file.

This file is a *reserved* file that you should not modify.

ACEGPRSR (GIPC Resources)

The `acegprsr.ini` file defines background GIPC messages.

This file is a *reserved* file that you should not modify.

ACEHCRSR (HCH Messages)

The `acehcrsr.ini` file defines source messages for HCH processing.

This file is a *reserved* file that you should not modify.

ACEHPRSR (HPH Messages)

This `.ini` file defines source message text for SurePOS ACE EPS HPH operational messages.

This file is a *reserved* file that you should not modify.

ACEICRSR (Item Code Migration Utility Resources)

This `.ini` file defines source message text for ACEICRBL, the GTIN item code migration utility.

This file is a *reserved* file that you should not modify.

ACEJRRSR (ACE Transaction Search Tool Resources)

The `acejrrsr.ini` file defines interface text for the ACE Transaction Search Tool.

This file is a *reserved* file that you should not modify. However, your store might prefer different output than the default file provides.

Modifying acejrrsr.ini

Modifications to `acejrrsr.ini` should be made with tiered overlay files (see "Method for C/C++ Files" on page 461), instead of by modifying the base file.

The header of the Cash Receipt in the Detail Results view is a resource that you might want to change. This is the default text for the header:

```
======= Welcome to SUREPOS ACE =======
```

To modify the text:

- Create an overlay file for the `DetailHeader` field.
- Specify new header text that obeys the width restrictions of the Cash Receipt display. For the best results, your new text should be the same length as the default text.

Source Code

This is partial code for the `.ini` file. Refer to the `acejrrsr.ini` source file for the complete code.

```
[EJReport]
BaseId=19000
;***************************************************************************
; Menu and SubMenu Items
;***************************************************************************
ReportTitle     = "~T~Log Journal Report"
ManagersReport    = "~M~anagers Journal          "
JournalReport     = "~J~ournal Report            "
;***************************************************************************
; Report Selection Dialog
;***************************************************************************
EJReportTitle     = "Journal Report Optional Filters"
EJMgrReportTitle  = "Managers Journal Report Optional Filters"
;***************************************************************************
; Report Summary and Detail Results
;***************************************************************************
SummaryResults       = "Journal Report Summary Results "
DetailResults        = "Journal Report Detail Results "
SearchResultsHeader  = "Term  Trx  Operator  Date     Time    Gross +    Gross -
TrxType"
DetailHeader         = "======= Welcome to SUREPOS ACE ======="
;***************************************************************************
; Search Maintenance
;***************************************************************************
SearchListTitle   = "Journal Report Search List "
MgrSrchListTitle  = "Managers Journal Report Search List "
SearchListHeader  = "Search Name      String Type    Search Description"
ManagersRpt       = "Manager's Journal"
```

ACEKFRSR (Keyed File Rebuild Dialogs)

The `acekfrsr.ini` file defines string values that appear in the Item File Rebuild Utility as dialog titles, field descriptions, messages, and button text.

This file is a *reserved* file that you should not modify.

ACEMTOPS (Miscellaneous Transactions Groups)

`acemtops.ini` provides values used to create the Miscellaneous Transaction Subtotal Descriptors file (acemstld.dat) and the Miscellaneous Transaction Lists file (acemsctl.dat) if they do not already exist.

This file is a *reserved* file that you should not modify. Use the Accounting user interface, which is available from the SurePOS ACE Main Menu, to maintain miscellaneous accounts.

ACEMTRSR (Matrix Keyboard Editor Resources)

The `acemtrsr.ini` file contains notebook and button descriptors and messages used during matrix keyboard personalization.

This file is a *reserved* file that you should not modify.

ACEOMRSR (Operator Data Maintenance Resources)

This file contains descriptors used for a printed report. The format of the report provides a column of descriptors on the left and a column of flags on the right. The flags are either Y or N. The flag column is placed close to the descriptors for ease of reading.

This file is a *reserved* file that you should not modify.

ACEOPTLG (Personalization Options Changed Log)

`ACEOPTLG.INI` controls the logging mechanism for changes made to Personalization options. This file is a *reserved* file. You can modify `aceoptlg.ini` through the tiered overlay method, which is described in "Tiered Overlay Files" on page 460.

To apply the tiered overlay files, use Load Controller Storage.

Options Section

This section of the INI file specifies the file names that are used for the logging mechanism log file and its backup, as well as the maximum size of the log and whether logging is enabled for the different scenarios where Personalization options can change.

You can modify these fields using your tiered overlay file:

LogUIUpdate

Enables logging for Personalization changes performed using the GUI. The valid values for this option are Y (logging enabled) and N (logging not enabled).

LogFileUpdate

Enables logging for Personalization changes performed using the /U parameter. The valid values for this option are Y (logging enabled) and N (logging not enabled).

LogDIFUpdate

Enables logging for Personalization changes performed using the DIF XML Update Options request. The valid values for this option are Y (logging enabled) and N (logging not enabled).

LOGNAME

The list of the names to be used when saving the log file, if the size of the log file exceeds the maximum size specified on the SIZE option. Any number of files (comma delimited with no spaces) may be listed on the option. When the current log file exceeds the maximum size, the log files are rolled:

1. The last file in the list is deleted.
2. The next to the last file is renamed to the last file name.

This rolling process continues until the current file is renamed to the second file name in the list and a new empty file, which is specified by the `<ACEOPTLG>` logical name, is created to begin collecting the Personalization changes log data.

The default value for this option specifies two log file names - a current file (`aceoptlg.dat`) and a previous file (`aceoptlp.dat`).

SIZE

The maximum size of the log. The default value for this option is 5242880 bytes (5MB).

Source Code

The following is source code for the Options section of the `aceoptlg.ini` file.

```
[Options]
LogUIUpdate=N
LogFileUpdate=N
LogDIFUpdate=N
LOGNAME=ACEOPTLG.DAT,ACEOPTLP.DAT
SIZE=5242880
```

ACEPNFMT (Panel Diary Format)

This is the Transaction Summary Log (TLog) string format`.ini` file used by the panel diary processing module, ACEPANEL. It specifies the formats of fields in TLog records X'00'-X'11' that are to be processed by ACEPANEL. These are the TLog strings with preferred customer information.

This file is a *reserved* file that you should not modify.

ACERDESC (Report Descriptors)

`acerdesc.ini` contains descriptors for the SurePOS ACE reports.

This file is a *reserved* file that you should not modify.

SurePOS ACE uses descriptor `.ini` files to dynamically generate messages and descriptors. Default messages and descriptors, which are contained in a .dat file identified in the Defaults section of the file, are merged with a set of delta messages and descriptors defined in the `.ini` file. The resulting messages and descriptors are stored in the descriptor data file identified in the Output section of the `.ini` file.

ACERDISK (RAM Disk Parameters)

`acerdisk.ini` contains parameters to initialize the SurePOS ACE RAM disk for use by virtual session terminals.

You can change the value on each option in this section, but the option names are *reserved*.

SurePOS ACE uses the `acerdisk.ini` file to list what files are to be loaded into the Q (or T) RAM disk and used during initialization. This file also specifies how often the files should be checked to see if any have changed and need to be recopied from the hard drive to the RAM disk. Overlay files are supported in the normal format of overlay files.

Note: The parameters in `acerdsk.ini` can only be used for virtual sessions.

ACEREINI (Report Engine)

This section describes the report engine (ACETVREL) INI file, `acereini.ini`, which contains the first direct interface between the report engine and the BOB code that it runs. It defines BOB reports that are available to the report engine.

Instructions for the ReportMenu section show how the report engine uses `acereini.ini` to build the report outline that ACETVREL displays on initialization. Information in other sections defines individual reports.

This file is opened from the report engine under the logical file name ACEREINI. You must define this logical name in your system before the report engine can run correctly.

Modifying acereini.ini

Modify `acereini.ini` with user delta `.ini` files as described in "Tiered Overlay Files" on page 460.

Modify the `acereini.ini` file to change the grouping of reports that appear on the Reports menu, or to include your own BOB code as source for reports you write. You can also select:

- Where to file filed reports and how to name them
- The naming convention for automatic report scripts
- Printer identifiers for all printers in the system
- System printer characteristics
- Default colors for displayed reports
- An authorization procedure for each report

To use your own BOB report file in place of one that exists for a specific report:

1. Copy the existing BOB file to a new file with a new name.
2. Modify the file with your own additions, deletions, and changes.
3. In the section of the `acereini.ini` file that defines the report ("Additional Sections" on page 509), remove the old BOB file name from the `Sources` parameter.
4. Add the name of your BOB file to the `Sources` parameter in the same section of the `acereini.ini` file.

Related Files

Resource strings (text on buttons and in menus, and other text) are read from a file referred to through the logical file name ACERERSR. The default version of this file is named `acerersr.ini`.

FiledReports Section

The FiledReports section describes where filed reports are written and the default file naming convention to use.

DefaultName

Modifiable. The name to use as the file name for any report that is not assigned a file name by the report requestor.

FileExtension

Modifiable. The default extension to use for a filed report file.

Source Code

```
[FiledReports]
DefaultName=filed
FileExtension=rpt
```

Printers Section

The Printers section describes printers in the system. An entry with just a &tilde; refers to a printer named in a section of the same name.

Default

Modifiable. Specifies which printer should be used if the user does not make a specific selection. This printer is highlighted in the printer selection dialog on start up. The name in the Default entry must exactly match a section name that defines the printer.

System Printer

Modifiable. The &tilde; refers to a section named System Printer, which is the next section described.

Source Code

```
[Printers]
Default=System Printer
System Printer=&tilde;
```

System Printer Section

This section describes properties of the system printer. Create a similar section for each printer you name in the Printers section.

Name

Modifiable. Specifies a text description of the printer that can be displayed to the user of the Reports application. If this entry is not present, the name of the printer section is used.

DeviceName

Modifiable. Specifies the name of the printer device. For example, prn is the name of the system printer on a 4690 system.

PageLength

Modifiable. Specifies the length of the printer page in lines.

PageWidth

Modifiable. Specifies the width of the printer page in characters.

PageEjectString

Modifiable. Optional, but recommended. Specifies the string that can be used on this printer to eject the page. For many printers, this string is ♀.

CompressionOnString

Modifiable. The string to turn on compressed print for this printer - necessary for the Department Totals Report.

CompressionOffString

Modifiable. The string to turn off compressed print for this printer.

Source Code

```
[System Printer]
Name=System Printer
DeviceName=prn
PageLength=59
PageWidth=80
PageEjectString=\012
CompressionOnString=\015
CompressionOffString=\018
```

Colors Section

The Colors section indicates colors to use for the report background and report text.

ReportBackground

Modifiable. Specifies one of these colors for the background:

0	Black
1	Dark blue
2	Dark green
3	Teal
4	Red
5	Magenta
6	Brown
7	Gray

ReportForeground

Modifiable. Specifies one of these colors for the background:

0	Black
1	Dark blue
2	Dark green
3	Teal
4	Red
5	Magenta
6	Brown
7	Gray
8	Black
9	Light blue
10	Light green
11	Cyan
12	Light red (pink)
13	Light magenta
14	Yellow
15	White

Source Code

```
[Colors]
ReportBackground=7
ReportForeground=0
```

Reports Section

This section is an expansion of the ReportsMenu section, where it was established as an expandable sublist of reports.

All fields

Modifiable. Each field is preceded by a &tilde;, which means that each is a sublist defined by another section further in the INI file. You can add your own entries or delete ones that are there by default. However, keep in mind that you must add another, matching section when you add an entry to this section.

For example, there is an `Accounting` section in the `.ini` file. Accounting Reports is what appears on the Reports menu. It is the first expandable entry listed under All Reports.

Source Code

```
[Reports]
~Accounting=       "Accounting Reports"
~Sales=            "Sales Reports"
~Listings=         "Listing and Log Reports"
~Performance=      "Performance Reports"
~Data Maintenance= "Data Maintenance Reports"
~Security=         "Security Reports"
~EM=               "Electronic Marketing Reports"
```

Accounting Section

This section is an expansion of the Reports section, where it was established as an expandable sublist of reports. It is a category of reports.

All fields

Modifiable. The left side of each field specifies the name of another section in this INI file that further defines the specific report. The right side of each field appears on the Reports menu.

For example, there is an Operator/Terminal Cash section further in the INI file. Operator/Terminal Cash Report appears on the Reports menu, under All Reports and Accounting Reports.

Source Code

```
[Accounting]
Operator/Terminal Cash=          "Operator/Terminal Cash Report"
Office Cash=                     "Office Cash Report"
Cash Drawer Position=            "Cash Drawer Position Report"
Over/Short=                      "Over/Short Report"
Store Totals Recap=              "Store Totals Recap Report"
Coupon Multiplication Limit=     "Coupon Multiplication Limit Report"
Miscellaneous Transaction Recap= "Miscellaneous Transaction Recap Report"
Tender Recap=                    "CHK/MISC Tender - Store Recap Report"
```

Reports in SQL Section

The SQL section defines other sections that provide reports for use during debug, development, and migration. These reports appear under the heading Developer Tools on the Reports menu after you rename the `acereini.usx` overlay file to a user overlay, such as `acereini.us9`, to activate the sections. The report defined for each section is:

SQLQ

SQL Query Report - Issues SQL queries against the tables and columns that are defined in the `sapath.rvt` file.

SQLU

SQL Update Report - Issues SQL updates against the tables and columns that are defined in the `sapath.rvt` file.

ItemLayout

Item Record Layout - Issues a report that contains the offset, length, and SQL data type of all fields in the `sapath.ddf` file. If the field is on an uneven boundary, then the offset and length are shown in bytes with a decimal position representing the bits.

All the fields in the SQLQ, SQLU, and ItemLayout sections are *reserved*.

Source Code

```
[SQLQ]
ProcedureId=219
Execute=executeSQLQReport
Prepare=prepareSQLQReport
Sources=sqlqrpt.bob

[SQLU]
ProcedureId=219
Execute=executeSQLUReport
Prepare=prepareSQLUReport
Sources=sqlurpt.bob

[ItemLayout]
ProcedureId=219
Execute=executeItemRecordLayoutReport
Prepare=prepareItemDataReport
Sources=itmdtrpt.bob
```

Supported SQL Data Types

Table 118 shows the SQL data types that have been tested and are supported for the Item Data Layout Report.

Table 118. SQL Data Types for Item Data Layout Report

SQL Data Type	Description
0x00	Text-To_Char
0x01 (1)	Text-To_VARChar
0x02 (2)	Packed_To_Char
0x03 (3)	Bit_To_Bit
0x06 (6)	UChar_To_SmallInt
0x09 (9)	UShort_To_Integer
0xA (10)	Long_To_Integer
0xC (12)	Long_To_Decimal
0x17 (23)	Packed_To_SmallInt
0x18 (24)	Packed_To_Integer

SQL Data Type	Description
0x19 (25)	Packed_To_Decimal
0x1A (26)	Packed_To_TimeStamp
0x1B (27)	Packed_To_Date

Reports in WIC Section

The WIC section defines other sections that provide reports that contain information about WIC EBT transactions. These reports appear under the heading WIC EBT Reports on the Reports menu after you rename the `acereini.usw` overlay file to a user file name, such as `acereini.us9`, to activate the sections. The report defined for each section is:

HotCard
Hot Card Report

MaxPrice
Max Price Report

AuthorizedItems
Authorized Items Report

Reconciliation
Reconciliation Report

Refer to the *SurePOS ACE: Planning and Installation Guide* for detailed descriptions of the contents of the WIC EBT reports.

All the fields in the `HotCard`, `MaxPrice`, `AuthorizedItems`, and `Reconciliation` sections are *reserved*.

Source Code

```
[HotCard]
ProcedureId=300
Execute = executeHotCardReport
Prepare = prepareHotCardReport
Sources = wehotcrd.bob

[MaxPrice]
ProcedureId=300
Execute = executeMaxPriceReport
Prepare = prepareMaxPriceReport
Sources = weupcmax.bob

[AuthorizedItems]
ProcedureId=300
Execute = executeAuthorizedItemsReport
Prepare = prepareAuthorizedItemsReport
Sources = weupcath.bob

[Reconciliation]
ProcedureId=300
Execute = executeWICEBTRecon
```

```
Prepare = prepareWICEBTRecon
Sources = wereconc.bob
```

Other Report Category Sections

These sections are similar to the Accounting section:

- Sales
- Listings
- Performance
- Data Maintenance
- Security
- EM

Each section is an expansion of the Reports section, where each was established as an expandable sublist of reports. Each section lists the reports that appear under it. Each report that is named has a corresponding section in this `.ini` file.

All the fields in each section are *Modifiable*.

Additional Sections

The rest of the sections in `acereini.ini` file are expansions of menu sections, where they were established as sections that define reports.

By default, these section names are:

- Cash Drawer Position
- Coupon Multiplication Limit
- Department Totals
- Department Variance
- Exception Log
- Hourly Department Totals
- Item Data Detail
- Item Data Summary
- Item Movement
- Item Sales Exception
- Miscellaneous Transaction Recap
- Negative Sales
- Office Cash
- Operator Authorization
- Operator Performance
- Operator Sales
- Operator/Terminal Cash
- Over/Short
- Period Points Close Summary
- Preferred Customer Audit Log
- Preferred Customer Exception Log
- Refund
- Store Totals Recap
- Tender Listing

- Tender Verification
- Terminal Productivity
- Transaction Log
- Void

Each section contains these fields:

ProcedureId

Modifiable. Specifies an integer that is the procedure ID of the report. This ID is used to check whether an operator is authorized to execute the report. If no procedure ID is specified in this INI file, all operators can run the report.

Sources

Modifiable. Specifies the source files that are required to run the report. The files that are listed in this entry are loaded the first time the report is executed.

Prepare

Reserved. Specifies the name of the BOB function that is called to prepare the report. This function gathers any parameters required for the report.

Execute

Reserved. Specifies the name of the BOB function that is called to execute the report.

ACERERSR (Report Engine Resources)

`acerersr.ini` defines text for the Reports pull-down menu, and messages issued during report processing or issued while printing filed reports, processing auto reports, or viewing the report error log.

This file is a *reserved* file and you should not modify it.

ACERFRSR (Loyalty Program Resources)

The `acerfrsr.ini` file contains messages used while transferring points activity.

This file is a *reserved* file that you should not modify.

ACERLRSR (Reporting List Editor Resources)

The `acerlrsr.ini` file contains resources for the Reporting List Editor function in Personalization.

This file is a *reserved* file that you should not modify.

ACESCHDL (Delayed Data Maintenance Scheduler)

The `aceschdl.ini` file controls DDM batch scheduling. You can also use it to disable or enable some functions and to schedule your own programs.

The scheduler `.ini` file contains control information used by the scheduler to control processing and housekeeping chores. Some settings in this file were previously in 4680-4690 Supermarket Application hidden options.

Constants Section

All fields are *modifiable.*

Frequency

Duration in minutes of the interval that the scheduler waits before processing triggers. This is its idle cycle length.

Queued Event ID

This is a unique ID of the pipe to which queued batch events are sent.

GIPC Read Pipe

Specifies the ID of the GIPC Spooler read pipe, which is identified in the `gipc.ini` file.

Addmi Autostart

This is a Boolean toggle that enables or disables the scheduler being able to automatically start ADDMI processing (`ACEADDML.EXE`):

0	The scheduler can automatically start ADDMI processing.
1	The scheduler cannot automatically start ADDMI processing.

AddmiWaitForImmediate

This is a Boolean toggle that enables or disables normal ADDMI processing where immediate batches wait for the scheduler to execute before processing the next batch:

0	ADDMI processes all immediate batches without waiting for the scheduler to execute..
1	Normal ADDMI processing where batches are processed one at a time.

ExecuteBatchesInSequence

This is a Boolean toggle that enables or disables normal scheduler execution where batches can be executed in random order. This random order is especially evident when enhanced ADDMI processing is enabled (AddmiWaitForImmediate = 0) since immediate batches are no longer processed one at a time:

0	Normal Scheduler batch execution which could be in random order.
1	Scheduler executes batches in batch id and sequence order.

DelayBetweenBatches

The number of seconds to delay between the execution of ADDMI batches.

In Alternate-Master configurations, make sure that the value is large enough to allow for file distribution. In most environments, 5 seconds would allow enough time for all files to distribute.

Retention Days

The number of days that DDM retains source data and control records after completion.

Lookback Days

The maximum age in days of a valid batch. Older batches will not be executed and they will not be marked complete.

Batch Records

The number of keyed records that the scheduler creates when it creates eamdmctl.dat at startup.

DDMBatchRecordCount

The number of DDM records to be read into memory before execution of the batch begins. If memory is limited and the DDM batch is large, this parameter enables DDM to load a batch in phases. A partial batch containing the specified number of records is loaded, executed, and its memory freed before the next partial batch is loaded.

If you use a tiered overlay file to delete this variable from `aceschdl.ini` or the variable is set to zero or a negative number, the entire DDM batch will be read into memory and the batch will be executed in a single pass.

The valid values for this variable are 1-2,147,483,647. The larger the value that is specified, the more memory that will be used to build a partial batch, but the fewer the number of partial batches that will have to be built. Because a file is opened and closed for each partial batch, you should specify a value of 1000 or greater to avoid performance problems.

Note: SurePOS ACE does not support executing multiple batches from a single source file. This variable only provides a method of sequentially processing parts of a single batch.

DDMProcessWaits

When processing batches on the background screen, process waits to delay the application so that the user has an opportunity to view each of the messages.

Set this parameter to Y to process delays. Setting this parameter to any other value will cause the application to avoid processing the delays.

DDMUpdateThreshold

When processing batches, this value sets the number of items to process per batch before posting a status update to the background screen.

The valid values for this variable are numbers in the range 1-2,147,483,647.

WIC Support

This value controls whether WIC EBT support is enabled or disabled.

Set the value of this variable to 0 to disable WIC EBT support.

Set the value of this variable to 1 to enable WIC EBT support. When support is enabled, you can process the following list of files by starting acewerbl in the background:

idyjjj.hcl

Hot Card List Files

idyjjj.apl

UPC/PLU (Approved Product) List Files

idyjjj.exx

Error Files

idyjjj.sxx

Reconciliation Files

In the file name formats, *id* is the state abbreviation, *y* is the last digit of the year, *jjj* is the Julian day of the year (must be < 366), and *xx* is a sequential number.

BIN Support

This value controls whether BIN file support is enabled or disabled.

Set this variable to 0 to disable BIN file support.

Set this variable to 1 to enable BIN support and process the <ACEBNINP> file, if the file exists.

Lookback Hours

The number of hours within the same day to look back for an Unattended Close to start. For example, if the close is scheduled to start any time within the 2:00 hour (02:00-02:59), and the default of 3 lookback hours is in effect, then a missed close may be processed by the scheduler any time between 02:00 and 05:59 when the scheduler is restarted. This means that a missed 2:00 close and a missed 02:59 close each can be initiated though lookback processing until 05:59.

In addition to the generic use of this value, it can be used to start Unattended Closes targeted to the 2 am hour on the first day of daylight savings time, to handle adjustment of the 4690 OS clock ahead one hour.

Coupon Fraud Support

This value controls whether coupon fraud file support is enabled or disabled.

Set this variable to 0 to disable coupon fraud file support.

Set this variable to 1 to enable coupon fraud support and process the <ACECPNFR> file, if the file exists.

Reprint Receipt Support = 0

This value controls whether Reprint Receipt File Cleanup support is enabled or disabled.

Set to 0 to disable scheduling Reprint Receipt File Cleanup Support (default).

Set to 1 to enable Reprint Receipt File Cleanup Support. When enabled, ACERECLN.386 will be kicked off at the specified hour of the current day to create clean receipt files and to remove old receipt files for all terminals.

Reprint Clean Hour = 0

The receipt clean hour ranges from 0 to 23 i.e., 12 a.m. to 11 p.m. and defaults to 0 (midnight). If a value other than 0 to 23 is specified, a default value of 0 will be used.

Source Code

```
[Constants]
Frequency=10
Queued Event Id=K
GIPC Read Pipe=Q
Addmi Autostart= 0
DelayBetweenBatches = 30
Retention Days= 30
Lookback Days= 30
Batch Records= 400
DDMBatchRecordCount=1000
DDMProcessWaits= Y
DDMUpdateThreshold= 10
WIC Support = 0
BIN Support = 0
Lookback Hours = 3
Coupon Fraud Support = 0
```

```
Reprint Receipt Support = 0
Receipt Clean Hour = 0
```

ACESDESC (Sales Descriptors)

`acesdesc.ini` contains sales descriptors for SurePOS ACE.

This file is a *reserved* file that you should not modify.

SurePOS ACE uses descriptor `.ini` files to dynamically generate messages and descriptors. Default messages and descriptors, which are contained in a .dat file identified in the Defaults section of the file, are merged with a set of delta messages and descriptors defined in the `.ini` file. The resulting messages and descriptors are stored in the descriptor data file identified in the Output section of the `.ini` file.

ACESLCRD (Customer Reporting and Maintenance Workbench)

`aceslcrd.ini` provides the user interface definitions for the Customer Reporting and Maintenance Workbench, which is available by selecting Manager Procedures -> Selective Reports -> Selective Customer Report from the SurePOS ACE Main Menu.

The Customer Reporting and Maintenance Workbench lets you specify search criteria for selecting customer data (from eamfbact.dat and eamfbcus.dat) to examine in a variety of views. You can also use the Maintenance tab to select from a variety of both batch and interactive processing options for performing maintenance on the customer records you have selected.

This file also defines the Customer Reporting and Maintenance Workbench report formats and data validation policies.

ACESLIRD (Item Reporting and Maintenance Workbench)

The `aceslird.ini` file provides parameters for the Item Reporting and Maintenance Workbench, which is available by selecting Manager Procedures -> Selective Reports -> Selective Item Report from the SurePOS ACE Main Menu.

The Item Reporting and Maintenance Workbench lets you specify search criteria for selecting item data (from eamitemr.dat and eamimov*.dat) to examine in a variety of views, such as a view of the flag fields in the item record. You can also use the Maintenance tab to select from a variety of both batch and interactive processing options for performing maintenance on the item records you have selected.

Modifying aceslird.ini

Most of the fields in the `aceslird.ini` file are *modifiable with caution*. Some are *modifiable* and a few are *reserved*.

Views Section

This section of the INI file identifies other sections that further define views that are available in the Item Reporting and Maintenance Workbench.

Adding an entry to this section adds a view, which is a notebook page accessible through the Next view tab at the bottom of the workbench. Any view you add here must be further defined in a section of its own.

ViewNormal

Modifiable with caution. Specifies the name of another section in this INI file that defines the first view of the record.

ViewFlags

Modifiable with caution. Specifies the name of another section in this INI file that includes all flag fields in the record except for user flags.

ViewMisc

Modifiable with caution. Specifies the name of another section in this INI file that includes data in these eamitemr fields: LINKEDTO, FAMILYNU (current and previous), and any alias item number in the unused SALEQUAN and SALPRIC.

ViewUser

Modifiable with caution. Specifies the name of another section in this INI file that includes user-defined data in these eamitemr fields: USEREXIT1, USEREXIT2, or any of the user flag bits in INDICAT1A.

ViewMovement

Modifiable with caution. Specifies the name of another section in this INI file that includes data in these eamimov* fields: ITEMCODE, NUMITEMS, AMTSALE, and WEIGHT.

ViewHostPrice

Modifiable with caution.

ViewExceptions

Modifiable with caution.

ViewExtended

Modifiable with caution.

ViewClubs

Modifiable with caution. Must be added using an overlay file if you want to display secondary points club information. The overlay file should include:

```
[Views]
/A:ViewClubs=&tilde;
```

ViewExtra2

Modifiable with caution. Must be added using an overlay file if you want to display data from the coupon co-pay amount field in the Item Record file. The overlay file should include:

```
[Views]
/A:ViewExtra2=&tilde;
```

ViewFuel

Modifiable with caution. Must be added using an overlay file if you want to display data from the fuel attribute field in the Item Record file. The overlay file should include:

```
[Views]
/A:ViewFuel=&tilde;
```

ViewEAS

Modifiable with caution. Must be added using an overlay file if you want to display data from the EAS Type field in the Item Record file. The overlay file should include:

```
[Views]
/A:ViewEAS=&tilde;
```

ViewDualP

Modifiable with caution. Must be added using an overlay file if you want to display alternate pricing data from the Item Record file. The overlay file should include:

```
[Views]
/A:ViewDualP=&tilde;
```

ViewSCS

Modifiable with caution. Must be added using an overlay file if you want to display search results for the Self Checkout information. The overlay file should include:

```
[Views]
/A:ViewSCS=&tilde;
```

ViewItemSubType

Modifiable with caution. Must be added using an overlay file if you want to display data from the Item Sub Type field in the Item Record file. The overlay file should include:

```
[Views]
/A:ViewItemSubType=&tilde;
```

Source Code

```
[Views]
ViewNormal=&tilde;
ViewFlags=&tilde;
;ViewFuel=&tilde;
ViewMisc=&tilde;
ViewUser=&tilde;
ViewMovement=&tilde;
ViewMovementLong=&tilde;
ViewHostPrice=&tilde;
ViewExceptions=&tilde;
ViewExtended=&tilde;
ViewExtra1=&tilde;
;ViewExtra2=&tilde;
;ViewClubs=&tilde;
;ViewEAS=&tilde;
;ViewDualP=&tilde;
;ViewSCS=~
;ViewItemSubType
```

ViewNormal Section

This defines a view of the most commonly referred-to item data.

Name

Modifiable. Specifies the name of the view. Tildes (&tilde;) delineate the highlighted character should the name appear on a pull-down menu at some time in the future.

Header

Modifiable. Specifies the header that appears at the top of the view. A backward slash (\) is a continuation character that identifies the following line as a continuation.

Format

Modifiable. Specifies what appears on each data line of the view. A percent sign (%) identifies a variable that must be defined in itmqrynm.hpp.

A second percent sign is the start of a C printf-style format specifier with minimum field width and precision modifiers. For example, `%ITEMCODE%12.14s` displays the item code as a character string that has a minimum of 12 characters and a maximum of 14 characters.

Spaces between fields are interpreted as spaces in the output.

Source Code

```
[ViewNormal]
Name="~N~ormal"
Header=\
"Itemcode     Description        Dpt IT PM Price 1   Price 2   Qty  MG"
Format=%ITEMCODE%14.14s %DESCRIPTION%18.18s %PRIMARYDEPARTMENT%3.3s \
%ITEMTYPEDESC%2.2s %PRIMARYPRICEMETHOD%2.2s \
%PRIMARYPRICE1%11.11s %PRIMARYPRICE2%9.9s %PRIMARYDEALQTY%4.4s %PRIMARYMPGROUP%2.2s
```

ViewUser

This section uses the Name, Header, and Format parameters described in "ViewNormal Section" on page 516.

If you selected the Use promotion code for coupon validation option in Options -> Coupon -> Acceptance, user exit field 1 in the item record contains the promotion code. This view shows the value of user exit field 1 with a heading of User 1. If you want the report to show Promo or something similar for that field, you must modify the Header parameter in this section.

Source Code

```
[ViewUser]
Name=~U~ser
Header=\
Itemcode     Description        User 1  User 2    U1 U2 U3 U4
Format=%ITEMCODE%14.14s %DESCRIPTION%18.18s %USERINT1%6.6s  %USERINT2%6.6s
  \
%USERFLAG1%2.2s %USERFLAG2%2.2s %USERFLAG3%2.2s %USERFLAG4%2.2s
```

Other Views Sections

The other sections identified in the Views section ("Views Section" on page 514) also use the Name, Header, and Format variables described in "ViewNormal Section" on page 516. Refer to the `aceslird.ini` source file for the fields that are included on each variable.

Reports Section

This section describes the format that is used when a view is printed or filed.

ActiveView

Modifiable with caution. Specifies that the ActiveView section describes the headers and trailers that are used to report the current (active) displayed view.

PrintNormal

Modifiable with caution. Specifies that the PrintNormal section describes a report format.

PrintCompressed

Modifiable with caution. Specifies that the PrintCompressed section describes a report format.

PrintMovement

Modifiable with caution. Specifies that the PrintMovement section describes a report format.

Source Code

```
[Reports]
ActiveView=&tilde;
PrintNormal=&tilde;
PrintCompressed=&tilde;
PrintMovement=&tilde;
```

PrintNormal Section

This is a two-line per item report of all selected items.

Name

Modifiable. Not used at this time.

Description

Modifiable. Specifies the name that appears on the Print/File Report menu as the name of the selectable report.

PageLength

Modifiable. Specifies the length of the report page in number of lines.

PageWidth

Modifiable. Specifies the width of the report page in number of characters.

PageHeader1-3

Modifiable. Specify the first through third lines that print at the top of each page of the report beginning on page two.

ReportHeader1-7

Modifiable. Specify the first through seventh lines that print at the top of page one of the report.

Line1-2

Modifiable. Specify the first and second lines that print for each item listed in the report.

ReportTrailer1-4

Modifiable. Specify the first through fourth trailer lines that print on the last page of the report.

PageTrailer1-2

Modifiable. Specifies the first and second footer lines for all pages of the report except the last page.

```
[PrintNormal]
Name=Standard Report
Description=Default Report
PageLength=60
PageWidth=80
PageHeader1=\
"Selective Item Record Report:"
PageHeader2=
PageHeader3=\
"Itemcode     Dpt IT PM =======Pricing======  Qty  MG Fam User   Link PR WR
QA QR"
ReportHeader1=\
Selective Item Record Report:
ReportHeader2=
ReportHeader3=Printed by %OPERATORID%s (%OPERATORNAME%s) at %TIME%s on %DATE%s
ReportHeader4=
ReportHeader5=%REPORT_COMMENTS%
ReportHeader6=
ReportHeader7=\
"Itemcode     Dpt IT PM =======Pricing======  Qty  MG Fam User   Link PR WR
QA QR"
Line1=\
%ITEMCODE%14.14s %PRIMARYDEPARTMENT%3.3s \
%ITEMTYPEDESC%2.2s %PRIMARYPRICEMETHOD%1.1s \
%PRIMARYPRICE1%11.11s %PRIMARYPRICE2%11.11s %PRIMARYDEALQTY%4.4s %PRIMARYMPGROUP%2.2s \
%FAMILYONE%3.3s %USERINT1%6.6s %LINKEDCODE%4.4s \
%PRICEREQD%2.2s %WEIGHTREQD%2.2s %QUANTITYOK%2.2s %QUANTITYREQD%2.2s
Line2=\
- %DESCRIPTION%18.18s    %ALIASCODE%14.14s                 %FAMILYTWO%3.3s\
%USERINT2%6.6s
ReportTrailer1=
ReportTrailer2=%REPORTED_ITEMCOUNT%6.6s ITEMS REPORTED OF %SUM_ITEMCOUNT%6.6s ITEMS FOUND
ReportTrailer3=
ReportTrailer4="%DATETIME%s - Page %PAGE%4.4s"
PageTrailer1=
PageTrailer2="%DATETIME%s - Page %PAGE%4.4s"
```

PrintCompressed Section

This is a one-line per item report of all selected items. Data is compressed in the print line to be able to show all data for each item on one line.

Name

Modifiable. Not used at this time.

Description

Modifiable. Specifies the name that appears on the Print/File Report menu as the name of the selectable report.

PageLength

Modifiable. Specifies the length of the report page in number of lines.

PageWidth

Modifiable. Specifies the width of the report page in number of characters.

Initialization

Modifiable. The string to turn on compressed print for this report.

Termination

Modifiable. The string to turn off compressed print for this report.

PageHeader1-3

Modifiable. Specify the first through third lines that print at the top of each page of the report beginning on page two.

ReportHeader1-7

Modifiable. Specify the first through seventh lines that print at the top of page one of the report.

Line1

Modifiable. Specifies the first line that prints for each item listed in the report.

ReportTrailer1-4

Modifiable. Specify the first through fourth trailer lines that print on the last page of the report.

PageTrailer1-2

Modifiable. Specify the first and second footer lines for all pages but the last.

```
[PrintCompressed]
Name=Compressed Report
Description=Compressed Report
PageLength=60
PageWidth=132
Initialization=\015
Termination=\018
PageHeader1=\
"Selective Item Record Report:"
PageHeader2=
PageHeader3=\
Itemcode     Description        Dpt IT PM =======Pricing======  Qty  Alias
Code   MG Fam1&2  User1  User2  Link PR WR QA QR

ReportHeader1=\
"Selective Item Record Report:"
ReportHeader2=
ReportHeader3=Printed by %OPERATORID%s (%OPERATORNAME%s) at %TIME%s on %DATE%s
ReportHeader4=
ReportHeader5=%REPORT_COMMENTS%
ReportHeader6=
ReportHeader7=\
"Itemcode     Description        Dpt IT PM =======Pricing======  Qty  Alias
Code   MG Fam1&2  User1  User2  Link PR WR QA QR RS"
;
Line1=\
%ITEMCODE%14.14s %DESCRIPTION%18.18s %DEPARTMENT%3.3s \
%ITEMTYPEDESC%2.2s %PRIMARYPRICEMETHOD%1.1s \
%PRIMARYPRICE1%11.11s%PRIMARYPRICE2%11.11s %PRIMARYDEALQTY%4.4s %ALIASCODE%14.14s %PRIMARYMPGROUP
%2.2s\
%FAMILYONE%3.3s %FAMILYTWO%3.3s %USERINT1%6.6s %USERINT2%6.6s %LINKEDCODE%4.4s\
%PRICEREQD%2.2s %WEIGHTREQD%2.2s %QUANTITYOK%2.2s %QUANTITYREQD%2.2s %RESTRICTION%2.2s
;
ReportTrailer1=
ReportTrailer2=%REPORTED_ITEMCOUNT%6.6s ITEMS REPORTED OF %SUM_ITEMCOUNT%6.6s ITEMS FOUND
ReportTrailer3=
ReportTrailer4=%DATETIME%s - Page %PAGE%4.4s
PageTrailer1=
PageTrailer2=%DATETIME%s - Page %PAGE%4.4s
```

PrintMovement Section

Name

Modifiable. Not used at this time.

Description

Modifiable. Specifies the name that appears on the Print/File Report menu as the name of the selectable report.

PageLength

Modifiable. Specifies the length of the report page in number of lines.

PageWidth

Modifiable. Specifies the width of the report page in number of characters.

PageHeader1-8

Modifiable. Specify the first through eighth lines that print at the top of each page of the report.

Line1-2

Modifiable. Specify the first and second lines that print for each item listed in the report.

PageTrailer1-2

Modifiable. Specify the first and second footer lines for all pages of the report.

UserOption1

Modifiable.

```
[PrintMovement]
Name=Item Movement Report
Description=Item Movement Report
PageLength=60
PageWidth=80
```

```
;2345678901234567890123456789012345678901234567890123456789012345678901234567890
PageHeader1=\
"Selective Item Record: Item Movement Report"
PageHeader2=
PageHeader3=\
"Printed by %OPERATORID%s (%OPERATORNAME%s) at %TIME%s on %DATE%s"
PageHeader4=
PageHeader5="%REPORT_COMMENTS%"
PageHeader6=
PageHeader7=\
"Itemcode     Description        Dpt IT       Sales     Quantity    Weight"
PageHeader8=\
-------------------------------------------------------------------------------
Line1=\
%ITEMCODE%14.14s %DESCRIPTION%18.18s %PRIMARYDEPARTMENT%3.3s %ITEMTYPEDESC%2.2s
Line2=: Current Short                        \
%SALESAMOUNT%14.14s%SALESQUANTITY%10.10s%SALESWEIGHT%12.12s
Line3=: Old Short                            \
%OSALESAMOUNT%14.14s%OSALESQUANTITY%10.10s%OSALESWEIGHT%12.12s
Line4=: Current Long                         \
%LSALESAMOUNT%14.14s%LSALESQUANTITY%10.10s%LSALESWEIGHT%12.12s
Line5=: Old Long                             \
%LOSALESAMOUNT%14.14s%LOSALESQUANTITY%10.10s%LOSALESWEIGHT%12.12s
PageTrailer1=\
-------------------------------------------------------------------------------
PageTrailer2=%DATETIME%s - Page %PAGE%4.4s
UserOption1=ItemMovementNeeded
```

ExtendedData Section

This section is for defining views and reports on data from the extended portion of the item record file, eamitemr.dat. You may define views and reports for data that you include in the extended portion. Follow the methods used to define views and reports for the first 46 bytes of the record.

For each view that you define, be sure to list it also under the Views section. This will cause the view to be included in the ring of views that can be seen in the Item Reporting and Maintenance Workbench.

For each report that you define, be sure to list it also under the Reports section. This will cause the report to be listed as a possible format when you print a report in the Item Reporting and Maintenance Workbench.

If you want to be able to query the coupon co-pay amount in the Item Reporting and Maintenance Workbench, you must use a tiered overlay file to add an uncommented *CouponCopay=~* line.

If you want to be able to query the fuel attribute in the Item Reporting and Maintenance Workbench, you must use a tiered overlay file to add an uncommented *IN_FuelItem=~* line.

Source Code

```
[ExtendedData]
;Category=~
Commodity=~
Subcommodity=~
ReportingCode=~
MSINumber=~
AlternateDiscountByDept=~
BigCrossPromo=~
```

```
InStoreAddedItem=~
InStorePriceChangeFlag=~
HostSalePrice=~
HostQuantity=~
HostLQDDeal=~
HostLQDLimit=~
HostPricingMethod=~
HostPriceReasonCode=~
HostPriceOperatorID=~
HostPriceChangeDateTime=~
LARGELINKEDTO=~
ValueCardType=~
ValueCardState=~
SecondaryPointsClub01=~
SecondaryPointsClub02=~
SecondaryPointsClub03=~
CouponLevel=~
EASType=~
;Coupon CoPay=~
;IN_FuelItem=~
```

Commodity Section (and Other Extended Data Sections)

All sections defined in the ExtendedData section use the same variables.

Prompt

Modifiable. Text that will appear for the field or flag on the search criteria notebook page in the Item Reporting and Maintenance Workbench.

PageNo

Modifiable. Number of the notebook page on which the field or flag will appear when specifying search criteria in the Item Reporting and Maintenance Workbench.

InputType

Reserved. Specifies the type of input:

0	CheckBox
1	Boolean
2	InputLine
3	Range

InputWidth

Modifiable. Number of characters in the field.

Offset

Modifiable. Offset of field into extended data area. For example, specify 3 for a field that is at offset 49 in a 101-byte item record.

Length

Modifiable. Number of bytes used by the field.

DataType

Modifiable. Specifies the 4690 data type:

0	ASCII

1	Packed String
2	Packed Integer
3	Byte
4	Integer
5	Long
6	BitField

BitTestValue

Modifiable. This variable is only used for flags. It is therefore not specified in the Commodity section.

ExplainSearch

Modifiable. Specifies text and variables that describe which items are being searched.

Source Code

This is the source code for the Commodity section. The format of all sections that refer to data that only occurs in extended item records is the same. Refer to the `aceslird.ini` source file for the values given to each variable in the different sections.

```
[Commodity]
Prompt=Commodity Number
PageNo=12
InputType=3
InputWidth=3
TableName=ItemData
ExplainSearch=Commodity Number from %s to %s
```

Validation Section

This section describes variables that can be used as criteria during a search to limit (or select) the number of items that appear in the view or report.

Do *not* change the names of variables (left side of the equals mark in each line). You can add variables that exist in itmqrynm.hpp to this section. You can also change the data edit you perform on the variable. An example of this type of change is shown below.

Most of these fields appear as range fields in the Add Items Query (Add tab) panel, the Refine Items Query (Refine tab), and Remove Items Query (Exclude tab) panels. (Some are values that are checked to make sure they are integers.)

These variables are defined in the itmqrynm.hpp file. As mentioned above, the left side of each line (left of the equals sign) is *reserved*. The right side is *modifiable with caution*. If you change the right side of a line, specify a value defined in "ValidationPolicy Section" on page 525.

You can use other variables defined in the itmqrynm.hpp file. Most, if not all of them, however, are already used in the Item Reporting and Maintenance Workbench as either range fields or flag fields.

Although flag fields are used extensively in the interface, they are not listed in this section.

Source Code

This is partial code for the Validation section. Refer to the `aceslird.ini` source file for the complete code.

```
[Validation]
ADVERTISEDPRICE    = AMOUNT
ALIASCODE          = ITEMCODE
COST               = AMOUNT
DATEOFLASTSALE     = RDOLS
DEPARTMENT         = R1_999
FAMILYONE          = R0_999
FAMILYTWO          = R0_999
ITEMCODE           = ITEMCODE
ITEMTYPE           = R0_7
LINKEDCODE         = R0_9999
MIXANDMATCH        = R0_99
PRICE1             = AMOUNT
PRICE2             = AMOUNT
PRICINGMETHOD      = R0_5
```

ValidationPolicy Section

These are ranges and other data types that are defined for the Item Reporting and Maintenance Workbench.

Existing validation policy names are *reserved*. You can add validation policies. To do so, you must define them in one of these formats:

```
policy_name = reserved_data_type, initial_value

policy_name = RANGE, low_value, high_value, initial_value
```

Reserved data types include:

INTEGER
Field value must be numeric

ITEMCODE
Field value must be an item code

RANGE
Field value must be within the validated range

STRFTIME
Field value must be a date and time

You can change the reserved data type to one of those listed above. If you change it to a range, you must add minimum and maximum limits.

Source Code

This is partial code for the ValidationPolicy section. Refer to the `aceslird.ini` source file for the complete code.

```
[ValidationPolicy]
AMOUNT  = INTEGER,        NULL
DATE    = STRFTIME,       NULL
PERCENT = INTEGER,        NULL
R0_5    = RANGE, 0,    5, NULL
R0_7    = RANGE, 0,    7, NULL
R0_99   = RANGE, 0,   99, NULL
R0_999  = RANGE, 0,  999, NULL
R1_999  = RANGE, 1,  999, NULL
R0_9999 = RANGE, 0, 9999, NULL
RDOLS   = RANGE,100,1299, NULL
ITEMCODE= ITEMCODE,       NULL
```

ACESLRSR (Selective Item Report Resources)

The `aceslrsr.ini` file provides resources for the Selective Item Report and the Selective Customer Report. The reports use the Item Reporting and Maintenance Workbench and the Customer Reporting and Maintenance Workbench, respectively.

This file contains dialog titles, push button text, text for notebook page tabs, character strings used in the printed and filed reports, and both message prompts and error messages.

This file is a *reserved* file that you should not modify.

ACESQL (SQL Engine)

The SQL Engine opens the `acesql.ini` file using the logical file name SQLDATA. This logical name must be defined in your system before the SQL Engine can run correctly.

This file is a *reserved* file that you should not modify. Modify `acesql.ini` with tiered overlay files (see "Method for C/C++ Files" on page 461) instead of modifying the base file.

SQLData Section

This section identifies the name of the dictionary file.

Source Code

```
[SQLData]
Data Dictionary=<ADX_IPGM:>sapath.ddf
```

ACETIRLG (ACETIRBL Log Configuration)

`acetirlg.ini` controls how the Terminal Item Record File Rebuild application (acetirbl) rolls the acetirlg.dat file.

This file is a *modifiable* file.

Options Section

This section of the INI file specifies the file names that are used for the ACETIRBL log file and the maximum size of the log file.

LOGNAME

Modifiable. The list of file names to be used when saving the log file if the size of the log file exceeds the maximum size specified on the SIZE option. Any number of files (comma delimited) may be listed on the option. When the current log file exceeds the maximum size, the log files are rolled (the last file in the list is deleted, the next to last file is renamed to the last file name. This rolling process continues until the current file is renamed to the second file name in the list and a new empty file, which is specified by the <ACETIRLG> logical name, is created to begin collecting the ACETIRBL log data).

The default value for this option specifies two log file names - a current file (acetirlg.dat) and a previous file (acetirlp.dat).

SIZE

Modifiable. The maximum size of the log.

Source Code

The following is source code for the `Options` section of the `acetirlg.ini` file.

```
[Options]
LOGNAME=ACETIRLG.DAT,ACETIRLP.DAT
SIZE=32000
```

ACETIRNI (ACETIRBL Constants)

`acetirni.ini` contains keyword-value pairs used to influence program behavior of the ACETIRBL utility.

This file is a *reserved* file that you should not modify.

ACETIRSR (ACETIRBL Resources)

`acetirsr.ini` is the resources file for the ACETIRBL utility. Entries in the file control the text that is used in background or console messages related to the utility.

This file is a *reserved* file that you should not modify.

ACETMRSR (Terminal Monitor Resources)

The `acetmrsr.ini` file provides resources for the Terminal Monitor user interface, which is available from the Managers Procedures menu of the Main Menu.

This file is a *reserved* file that you should not modify.

ACETVRSR (Library Resources for TurboVision)

The `acetvrsr.ini` file provides resources to TurboVision library functions.

This file is a *reserved* file that you should not modify.

ACETXRSR (Tax Plans Resources)

The `acetxrsr.ini` file provides resources for the Tax Plans editing interface, which you access by selecting Tax Plans on the Miscellaneous pull-down menu of the Personalization menu.

This file is a *reserved* file that you should not modify.

ACEUBRSR (Resources for Unattended Close Program)

`aceubrsr.ini` contains entries that control the text that is displayed during unattended close processing.

This file is a *reserved* file that you should not modify.

ACEUPRSR (ACEUBATP Resources)

`aceuprsr.ini` is the resources file for the aceubatp utility. Entries in the file control the text that is used in background messages related to the utility.

This file is a *reserved* file that you should not modify.

ACEVERSR (Generic Resources for TurboVision View Engine)

The `aceversr.ini` file controls generic resources for the TurboVision view engine.

This file is a *reserved* file that you should not modify.

ACEWERSR (ACEWERBL Resources)

`acewersr.ini` is the resources file for the acewerbl utility. Entries in the file control the text that is used in background messages related to the utility.

This file is a *reserved* file that you should not modify.

ACEXFRSR (Loyalty Program Points Transfer Resources)

The `acexfrsr.ini` file provides resources for transferring preferred customer points activity.

This file is a *reserved* file that you should not modify.

ADXCSOZF (Background Error Messages)

The `adxcsozf.ini` file contains background error messages for the controller operator display.

This file is a *reserved* file that you should not modify. However, user extensions you make might require you to modify this file to reflect your code changes.

About adxcsozf.ini

`adxcsozf.ini` is an input file that the SurePOS ACE error message module, `ACECSOZF.EXE`, uses to create the <ADXCSOZF> data file, adxcsozf.dat, and the error message include file, acebmsgs.hpp.

Each line in this file has two parts separated by a comma. The first part is a direct block offset into this file and the message text to be displayed. The second part is the entry in acebmsgs.hpp that maps the message to a system variable.

The *X* messages in the file are logged by SurePOS ACE to the application event log. The maximum length for each message is 111 characters including the block offset, `Xnnn`, at the beginning of each line.

Make changes to this file through the user delta `.ini` file process described in "Tiered Overlay Files" on page 460. To apply your tiered overlay files, run the ACECSOZF program instead of ACECNVDL.

APSGRANT (SurePOS ACE EPS Operator Options Authorization Grant)

The `apsgrant.ini` file is similar in structure to the one described in "OPTGRANT (Operator Options Authorization Grant)" on page 533. Authorizations in the file apply only to SurePOS ACE EPS option values that appear on the EPS pull-down of the Personalization user interface.

The `apsgrant.ini` file is created by the /G parameter of the ACEPERSL program, which is described in "ACEPERSL, the Personalization Module" on page 574.

The `apsgrant.ini` file identifies itself, the .dat file from which it was made, and its creation date and time in comments at the top of the file.

The `apsgrant.ini` file has sections that correspond to those in the `apsparms.ini` file. Like `apsparms.ini`, each section lists individual options. However, instead of a default value for each option, these entries specify nine individual values, which each grant or deny personalization access to one of the nine operator levels. A value of 1 grants access, which is the default for each of the nine authorizations for each option.

APSPARMS (SurePOS ACE EPS Personalization Options)

The `apsparms.ini` file is an options file like `optnparm.ini`. The file contains options and menu specifications for the Toshiba Global Commerce Solutions SurePOS ACE Electronic Payment Support (SurePOS ACE EPS).

This file is a *reserved* file that you should not modify.

CPNGRANT (Coupon Translation Operator Options Authorization Grant)

The `cpngrant.ini` file is similar in structure to the one described in "OPTGRANT (Operator Options Authorization Grant)" on page 533. Authorizations in the file apply only to coupon translation option values that appear on the Translation pull-down of the Personalization user interface.

The `cpngrant.ini` file is created by the /G parameter of the ACEPERSL program, which is described in "ACEPERSL, the Personalization Module" on page 574.

The `cpngrant.ini` file identifies itself, the .dat file from which it was made, and its creation date and time in comments at the top of the file.

The `cpngrant.ini` file has sections that correspond to those in the `acecpntr.ini` file. Like `acecpntr.ini`, each section lists individual options. However, instead of a default value for each option, these entries specify nine individual values, which each grant or deny personalization access to one of the nine operator levels. A value of 1 grants access, which is the default for each of the nine authorizations for each option.

FDSLN (Logical Names)

The `fdsln.ini` file contains logical name definitions for SurePOS ACE.

This file is a *reserved* file that you should not modify.

GSTFMTDS (Output Formatting for GST Display)

The `gstfmtds.ini` file defines the output formatting for the GST display. See "Format and Field Types for Output Format INI Files" on page 491 for the different types of formats and fields that you can define.

Modifying gstfmtds.ini

All fields in the `gstfmtds.ini` file are *modifiable* with the following exception: section names and the *Id* and *FormatType* fields are *reserved*. Modify `gstfmtds.ini` with tiered overlay files (see "Method for C/C++ Files" on page 461) instead of modifying the base file.

In addition to tiered overlay files, C++ extensions are required if you want to add new output fields or new output formats. See "Adding a New Customer Output Format" on page 490 and "Adding a New Customer Output Field" on page 491 for more information.

GSTFMTPR (Output Formatting for GST Printer)

The `gstfmtpr.ini` file defines the output formatting for the GST printer. See "Format and Field Types for Output Format INI Files" on page 491 for the different types of formats and fields that you can define.

Modifying gstfmtpr.ini

All fields in the `gstfmtpr.ini` file are *modifiable* with the following exception: section names and the *Id* and *FormatType* fields are *reserved*. Modify `gstfmtpr.ini` with tiered overlay files (see "Method for C/C++ Files" on page 461) instead of modifying the base file.

In addition to tiered overlay files, C++ extensions are required if you want to add new output fields or new output formats. See "Adding a New Customer Output Format" on page 490 and "Adding a New Customer Output Field" on page 491 for more information.

IOP (Toolkit I/O Processor Simulator)

The `iop.ini` file is a simulation file used to replicate the 4690 OS environment in a Windows development environment. Other `.ini` files used to simulate the POS environment in a Windows development environment are described in "SCNNRWND (Toolkit Scanner Window Simulator)" on page 546 and "MSRWNDW (Toolkit Magnetic Stripe Reader Window Simulator)" on page 532.

This `.ini` file defines a pseudo input state table (IST) to simulate 4690 key stem definitions in a Windows development environment. It should exactly match the IST, eams@000, of the 4690 system you are targeting. If you use the IST Utility to change IST key stem definitions, you should change this file to keep it in sync with your actual definitions.

The Input Sequence Table Utility is described in the *Toshiba Global Commerce Solutions 4690 Operating System: Programming Guide*. It is available from the 4690 OS Installation and Update Aids screen.

You can modify the keys and ALPHA sections in the `iop.ini` file. Other sections in the file are *reserved* and you should not modify them.

The ALPHA section provides the key mappings to support alphabetic characters during entry of a driver's license number or a state abbreviation. The settings in this section must match the alpha key codes that are specified in Options -> Lookup -> Alpha Key Codes personalization (refer to the *SurePOS ACE: Planning and Installation Guide* for detailed information about the options). The values in the default `iop.ini` file match the default personalization values. For Windows developers, support for alphabetic characters is only enabled using the GUI keyboard and a mouse. The PC keyboard cannot be used to enter alphabetic characters.

Keys Section

This section assigns 4690 key labels (button text) and key codes.

The format of each variable is Button*xx*=button_text,function_code.

Button*xx*

Specifies the name of a button.

button_text

Specifies the label of the 4690 key stem.

function_code

Specifies the 4690 key code. A value of zero specifies that the button is unused.

Source Code

This is partial code for the keys section. Refer to the `iop.ini` source file for the complete listing.

```
[keys]
Button1=Sign On/Off,61
Button2=Override,79
Button3=Groc-Tax,206
Button4=FS/No FS,76
Button5=Price,74
Button6=Frz Foods,207
Button7=Tax/No,77
Button8=Tare,71
Button9=Liquor,208
Button10=Clear,73
```

Clear Section (and Other State Definition Sections)

All sections that define states use the same variables.

State Name

Modifiable. The title of each section is the state name. State names are for convenience only. They are not used during the simulation.

Settings

Modifiable. Specifies a string of comma-delimited values that indicate:

- The state code

- Whether the state begins an input sequence
- Whether the state displays data
- Whether the state notifies when there is an error
- Whether a scanner is allowed
- The next state to enter when there is an error

Code1-nnn

Modifiable. Specifies a string of comma-delimited values that indicate:

- The function code
- Whether it is a motor key
- The minimum data size
- The maximum data size
- The next state to enter when there is an error
- Whether data follows the key

Note: The similar 4690 OS flag specifies whether data precedes the keystroke, which is just the opposite of this flag. If data does precede the keystroke, the 4690 OS flag should be Yes and this flag should be No. This flag value should be just the opposite of the setting for the 4690 OS flag.

When the next state to enter on error is a zero, an error causes a lock.

Values of T and Y are equivalent. Characters must be uppercase.

Values of F and N are equivalent. Characters must be uppercase.

Unexpected results occur when a state is defined without a function code.

Function code definitions indicate whether data follows the function. In 4690 OS, there is a flag to indicate whether data precedes the function. Therefore, these values are just the opposite of those in a 4690 state table definition.

Source Code

The source code for the Clear and SIGNON sections is shown here. The format of all sections that define states is the same. Refer to the `iop.ini` source file for the values given to each variable in the different sections.

```
[Clear]
Settings=1,Y,N,N,N,1
Code1=73,Y,0,25,0,N

[SIGNON]
Settings=2,Y,Y,N,N,0
Code1=78,N,1,9,3,N
```

MSRWNDW (Toolkit Magnetic Stripe Reader Window Simulator)

The `msrwndw.ini` file is used to simulate the POS environment in a Windows development environment. This `.ini` file specifies card data for several representative card types, which allows the simulator to test appropriate MSR processing in a development environment.

This is a *modifiable* file that you can change to represent any card data you accept in your stores. Track 2 is the only track used during the simulation. The track 2 data format is:

```
Track2=cardNumber=expirationDate
```

The expiration date is in the format YYMM, which is not checked during the simulation.

Source Code

The following is partial source code for the INI file. Refer to the msrwndw.ini source file for the complete code.

```
[Good Visa Card 1]
Track1=B4503300111209^Shakespeare/Will A^2106000000
Track2=4503300111209=21060000000000

[Good Gift Card 1]
Track1=B6006490900001235^Shakespeare/Gilbert^2106000000
Track2=6006490900001235=21060000000000

[Good Master Card 1]
Track1=B5199999999999991^Shakespeare/Edmund^2106000000
Track2=5199999999999991=21060000000000

[Expired Visa Card 1]
Track1=B4503300111209^Shakespeare/Will A^9712000000
Track2=4503300111209=97120000000000
```

NLSGRANT (NLS Operator Options Authorization Grant)

The nlsgrant.ini file is similar in structure to the one described in "OPTGRANT (Operator Options Authorization Grant)" on page 533. Authorizations in the file apply only to NLS option values that appear on the NLS pull-down of the Personalization user interface.

The nlsgrant.ini file is created by the /G parameter of the ACEPERSL program, which is described in "ACEPERSL, the Personalization Module" on page 574.

The nlsgrant.ini file identifies itself, the .dat file from which it was made, and its creation date and time in comments at the top of the file.

The nlsgrant.ini file has sections that correspond to those in the nlsoptns.ini file. Like nlsoptns.ini, each section lists individual options. However, instead of a default value for each option, these entries specify nine individual values, which each grant or deny personalization access to one of the nine operator levels. A value of 1 grants access, which is the default for each of the nine authorizations for each option.

NLSPARMS (NLS Options)

The nlsparms.ini file is an options file like optnparm.ini. It contains options and menu specifications for national language support (NLS) personalization, which is available from the Personalization user interface.

This file is a *reserved* file that you should not modify.

OPTGRANT (Operator Options Authorization Grant)

The optgrant.ini file is created by the /G parameter of the ACEPERSL program, which is described in "ACEPERSL, the Personalization Module" on page 574.

The `optgrant.ini` file identifies itself, the DAT file from which it was made, and its creation date and time in comments at the top of the file.

```
//**********************************************...*
// OPERATOR OPTIONS AUTHORIZATION PARAMETER VALUES
//
// FROM: <$AMOPTNS>
// DATE: 04/30/98 07:02pm
//**********************************************...*
```

The `optgrant.ini` file has sections that correspond to those in the options INI file, `optnparm.ini`.

Like `optnparm.ini`, each section lists individual options. However, instead of a default value for each option, these entries specify nine individual values, which each grant or deny personalization access to one of the nine operator levels. A value of 1 grants access, which is the default for each of the nine authorizations for each option.

In the following example of a partial `Store` section in `optgrant.ini`, all options except Name are authorized for all nine operator levels. The Name option is authorized for all levels except level 2.

```
[Store]
                                    Name =   1,Array,Boolean,9,1,0,1,1,1,1,1,1,1
                                Division =   2,Array,Boolean,9,1,1,1,1,1,1,1,1,1
                                  Number =   3,Array,Boolean,9,1,1,1,1,1,1,1,1,1
                                Address1 =   4,Array,Boolean,9,1,1,1,1,1,1,1,1,1
                                Address2 =   5,Array,Boolean,9,1,1,1,1,1,1,1,1,1
                                Address3 =   6,Array,Boolean,9,1,1,1,1,1,1,1,1,1
                                   State =   7,Array,Boolean,9,1,1,1,1,1,1,1,1,1
                                     Zip =   8,Array,Boolean,9,1,1,1,1,1,1,1,1,1
```

OPTNPARM (Options)

The `optnparm.ini` file contains base personalization options for SurePOS ACE. It also contains links to all other files that contain base personalization options.

You can define entire options panels with any number of options by modifying `optnparm.ini` through the tiered overlay method, which is described in "Tiered Overlay Files" on page 460. Toshiba Global Commerce Solutions-defined windows, submenus, dialogs and notebooks are *reserved* and you should not change them. Toshiba Global Commerce Solutions reserves the right to make changes to existing GUI components in `optnparm.ini` in future or maintenance releases of SurePOS ACE.

You can modify these sections and fields in your tiered overlay file:

- `CustomerMenu` entry in the `MainMenu` section - modify to identify your GUI menu definitions
- `CustomerRecord1` through `CustomerRecord10` sections - modify to define new options
- `LinkedINIandOptionsFiles` section - modify to identify a separate options file that contains your new definitions
- `InputValidation` section - modify to add new validation ranges

Note: Although you can use tiered overlay files `optnparm.us7`, `optnparm.us8`, or `optnparm.us9`, the remainder of this description refers to `optnparm.us9`.

If you define all changes in `optnparm.us9`, your changes can be terminal-specific, which means you can apply the options to selected terminals instead of only being able to apply them to all terminals.

Overview of optnparm.ini

The file can be divided into two sets of parameters: those sections that describe options data values, and those sections that describe TurboVision menus and panels for the Personalization user interface. The sections that describe TurboVision menus and panels are the MainMenu section, the sections that describe submenus that were identified in the MainMenu section, and the sections that define dialogs that are opened from the submenus. All other sections describe options data values.

Related Files

The `optnparm.ini` file can contain links to other files of personalization option settings and menus that you create or that Toshiba Global Commerce Solutions or a Toshiba Global Commerce Solutions Business Partner have created. (For example, if you install the SurePOS ACE EPS feature, there is a link to the `apsparms.ini` file within the `optnparm.ini` file.) Linked files can contain your changes or personalization changes for optional PRPQs, features, or functions. "LinkedINIandOptionFiles Section" on page 537 describes how to link options files.

SurePOS ACE uses option settings to dynamically create the .hpp include file, for including option values in objects, and the companion eamoptns.dat file. Other related files are described in "FileNames Section" on page 535.

INIControl Section

Reserved. See "INIControl Section Parameters" on page 452 for a description of these parameters.

FileNames Section

This is a *reserved* section in `optnparm.ini` and in `optnparm.us9`. Certain fields are *modifiable* in your user options file that you identified as a linked options file.

OptionsFile

Reserved. This specifies the file name of the eamoptns.dat file.

StoreReadFile

Modifiable in linked options file. This specifies the logical local file name, $AMOPTNS, of the data storage associated with options specified in this INI file.

StoreWriteFile

Modifiable in linked options file. This specifies the logical write-to file name on the file server, EAMOPTNS, that contains values of all options specified in this INI file.

TerminalReadFile

Reserved; This specifies the prefix of the logical local file name, $AMO, of the data storage that SurePOS ACE uses to load individual terminal options files. It replaces that part of the file name specified in the `TerminalWriteFile` parameter. SurePOS ACE can use these option values to restore prior values during terminal personalization.

TerminalWriteFile

Reserved; This specifies the prefix of the file name on the file server, EAMO, used for individual terminal options files, which prepend this value to a unique terminal identifier.

GroupReadFile

Reserved; This specifies the prefix of the logical local file name, $AMOG, of the data storage that SurePOS ACE uses to load terminal group options files. It replaces that part of the file name specified in the GroupWriteFile parameter. SurePOS ACE can use these option values to restore prior values during terminal group personalization.

GroupWriteFile

Reserved; This specifies the prefix of the file server, EAMOG, used for terminal group options files, that prepend this value to a unique terminal group identifier.

hppFile

Modifiable in linked options file. This specifies the name of the include file that you can build after changing option values when using a separate linked options file. Run ACEPERSL with the /P parameter from the command line and select Generate HPP from the File pull-down menu or run ACEPERSL with the /H parameter from the command line. The include file is dynamically built using options specified in this `.ini` file and in other files that it includes. See "Using ACEPERSL to Maintain Personalization Options" on page 572 for more information.

dumpFile

Reserved. Specifies the name of a flat ASCII file that lists all options and their values. When you select File -> Dump options, the current option settings are placed in the file, overwriting any previous contents.

grantFile

Reserved. Specifies the name of the grant file that contains operator options authorizations. The file is created when you run ACEPERSL with the /G parameter from the command line. See "Using ACEPERSL to Maintain Personalization Options" on page 572 for more information.

saveINIFile

Reserved. Specifies the name of the `.ini` file to which ACEPERSL saves current options settings (from the .dat file) when you use the /S parameter. See "Using ACEPERSL to Maintain Personalization Options" on page 572 for more information.

authorizationFile

Reserved. Specifies the name of the file, `optlevel.ini`, that defines the security level for drop-down personalization menu options. It is a higher level method of specifying operator authorization than specifying individual option authorizations through either using F9 AuthChg in Personalization, or using the /A parameter on the ACEPERSL command. See "ACEPERSL, the Personalization Module" on page 574 for information about the /A parameter.

authorizations

Reserved. Specifies the logical name, <ACEAUTHZ>, of the authorization data file (*.dat).

LinkedINIandOptionFiles Section

Modifiable in optnparm.us9 to include the name of an external file of user options. Entries in this section describe external files that each act as an addendum to this `.ini` file.

APS

Reserved. Specifies the name of another file containing options that can be accessed from the Personalization menu action bar.

Translation

Reserved. Specifies the name of another file containing options that can be accessed from the Personalization menu action bar.

NLS

Reserved. Specifies the name of another file containing options that can be accessed from the Personalization menu action bar.

MainMenu Section

Modifiable in ***optnparm.us9***. You can modify this section in `optnparm.us9` to identify a `Customer` submenu that defines GUI controls for options that you define in `CustomerRecord1-10`. Toshiba Global Commerce Solutions-defined entries are *reserved* and you should not modify them.

Each entry defines the name of a pull-down menu on the Personalization menu action bar. Each entry in this section is also the name of another section within this `.ini` file.

The syntax for each entry in this section is:

```
name = Submenu, menu_bar_option, help_id
```

The syntax parameters are:

name

Name of another section in this `.ini` file.

Submenu

Specifies that the entry defines a pull-down menu on the Personalization menu action bar. This field is a constant.

menu_bar_option

Specifies the text that appears on the action bar and designates an action character by placing a tilde (&tilde;) before and after the character. The text must be specified between double quotes.

help_id

Specifies the topic number of the associated help panel in the acepehlp.txt file.

File Section

Reserved. This section is *reserved* and you should not modify it. Entries in the section are explained to help you define your `Customer` section, if you are using one.

The `File` section defines a pull-down menu section that was specified in the `MainMenu` section. Menu sections are composed of menu option sections, which each appear on the menu. Each menu option can be one of these choices:

- *Submenu*. See “Submenu” on page 455 for a description of this type of control. This is the type of menu created by the File section. It appears on the Personalization menu action bar as the File pull-down menu because it is a submenu of the MainMenu section.
- *NewLine*. See “NewLine” on page 454 for a description of this type of control. This occurs twice within the File section. All 15 characters of the menu option are line characters for each new line. So, each draws a line across the File pull-down menu.
- *Process*. See “Process” on page 454 for a description of this type of control. This occurs several times in the File section.
- *Dialog*. See “Dialog” on page 454 for a description of this type of control.

Actions defined in this section include:

Open terminal

Reserved. Specifies there is a File.Open Terminal process that defines a dialog entered by the Open terminal/group menu option.

Close terminal

Reserved. Specifies the process to call to close an open terminal-specific or group options file when the Close terminal/group menu option is selected.

Load terminal

Reserved. Specifies there is a File.Load Terminal section that defines a process entered by the Load terminal/group menu option.

Erase terminal

Reserved. Specifies there is a File.Erase Terminal section that defines a process entered by the Erase terminal/group menu option.

TerminalGroup

Reserved. Specifies there is a File.TerminalGroup section that defines a process entered by the Terminal groups menu option.

Save

Reserved. Specifies the procedure to call to save the options files.

Reset

Reserved. Specifies the procedure to call to rebuild all options that are currently in `optnparm.ini` and its linked options files. Because this parameter is defined as *Protected* in the default `INI` file, Reset does not appear on the default pull-down menu. Refer to “ACEPERSL, the Personalization Module” on page 574 for information on using the 4690 command line to access this menu option.

Generate HPP

Reserved. Specifies the procedure to call to dynamically build a new include file, which is incoptns.hpp by default. Because this parameter is defined as *Protected* in the default `INI` file, Generate HPP does not appear on the default pull-down menu. Refer to “ACEPERSL, the Personalization Module” on page 574 for information on using the 4690 command line to access this menu option.

Dump options

Reserved. Specifies the procedure to call to dump all options to files.

Exit

Reserved. Specifies the procedure to call to exit the Personalization menu.

Control Section

Reserved.

File0 Section

Reserved. This section specifies the names of records that are each defined in another section of the specified name.

Each of these fields is *reserved*. Each field defines the name of another section in this `.ini` file that defines a record in the companion .dat

file. Here are descriptions of some representative fields:

Store

Identifies a section of this `.ini` file that defines a record containing option values. Parameters in this specification are positional.

The first parameter, 1, is a unique index value for locating this record definition in the storage area reserved for `optnparm.ini` personalization values.

The second parameter specifies that the area of storage the Store section defines is a `Record` of options values.

See "Store Section" on page 540 for more information.

LookupKey

Identifies another section of this `.ini` file that defines a record containing option values. Its parameters are positional, with exactly the same meanings as `Store` and other `Record` definitions in this section.

See "LookupKey.List Section" on page 543 for more information.

Source Code

This is partial code for the File0 section. Refer to the `optnparm.ini` source file for the complete code.

```
[File0]
Control                    = 4096, Record
Store                      =  1, Record
Security                   =  2, Record
Overrides                  =  3, Record
ItemLimits                 =  4, Record
TransactionLimits          =  5, Record
LookupKey                  =  6, Record
Discounts                  =  7, Record
Weight                     =  8, Record
Tare                       =  9, Record
```

```
RestrictedSales             = 10, Record
Tender                      = 11, Record
```

Store Section

Reserved section. Each field in the Store section is reserved.

The Store section is a representative record section that defines data fields in the personalization Store record, which was specified in the File0 section. Record sections are composed of fieldName definition statements, which each define a field that appears on the user interface.

Figure 9 shows the generic syntax of a data field (fieldName) statement.

```
fieldName = index, fieldType
```

Figure 9. Syntax of Data Field Statement

Syntax parameters are described in the sections that follow.

fieldName

The `fieldName` field represents the variable that stores the personalization option being defined. The `fieldName` statement uses the format shown in Figure 10 for Int, UInt, Long, ULong, and Boolean field types. Any one of these field types is referred to as a *simple field type* (as opposed to strings, arrays, and sections).

```
fieldName = index, simpleFieldType, defaultValue
```

Figure 10. Syntax of Data Field Statement with Default Value

These are some examples of this format in the Store section:

```
Number                    =  3, ULong,           "1   "
MaximumTerminals          = 11, Int,              50
DiskSpaceWarningC         = 13, ULong,            20
CatalinaMarketingStatus   = 23, Boolean,          Y
```

Quotation marks around a default value indicate the maximum number of digits allowed for the value.

index

The *index* is a unique sequential integer value that is an index into field values contained in the area of storage identified by the section name. Index *99* is reserved for `Record` definitions for the *DeleteFlag* field.

fieldType

The fieldType identifies the data type of the field being defined. Depending on the type of field, the syntax for defining the field name definition statement varies slightly.

These are valid values of fieldType:

- Int
- UInt
- Long
- ULong
- Boolean
- String
- Array
- Sections

The field name statement uses the format shown in Figure 11 for the String field type.

String

Defining a `String` field type is shown in Figure 11.

```
fieldName = index, STRING, length, defaultValue
```

Figure 11. Syntax of Data Field Statement (Data Type=String)

These are some examples of this format in the Store section:

```
Name                    =  1, String,20,        "Store Name";
State                   =  7, String, 3,        "ST"
CatalinaMarketingPipe   = 25, String, 10,        S
```

Quotation marks around the default value indicate the maximum length.

Array

The field name statement uses the format shown in Figure 12 for an Array field type:

```
fieldName = index, ARRAY, simpleFieldType, arraySize, defaultValueList
```

Figure 12. Syntax of Data Field Statement (Field Type=Array)

This is an example of this format in the Store section:

```
ToggleTaxes             = 26, Array, Boolean, 8, Y,Y,Y,Y,Y,Y,Y,Y
```

Sections

The `Sections` field type is another example of a compound field type. Unlike the `Array` field type, a section in the `.ini` file can have only one field of type `SECTIONS`. The fieldName statement uses the format shown in Figure 13.

```
fieldName = number_of_sections, SECTIONS, defaultSectionList
```

Figure 13. Syntax of Data Field Statement (Field Type = SECTIONS)

Note: The *number_of_sections* cannot exceed 100.

The SECTIONS field type is similar to an ARRAY, but is an array of structures. Each structure is composed of multiple field types. Because a SECTION is made up of more than one field, it takes more sections in the INI file to describe it.

For example, the record LookupKey has a field called List which is a SECTIONS field type. Another section, LookupKey.List must describe the new field. (In this case, this field is made up of 7 UInts, 1 String, and 1 Boolean. Every SECTIONS template must also include a Boolean data field called DeleteFlag with an index of 99.)

Source Code

```
[Store]
Name                     =  1, String, 20,       "Store Name          "
Number                   =  3, ULong,            "1   "
State                    =  7, String, 3,        "ST"
Zip                      =  8, String, 10,       "ZIP       "
MaximumTerminals         = 11, Int,               50
DiskSpaceWarningC        = 13, ULong,             20
KeepRingAndTenderTimes   = 24, Boolean,           Y
CatalinaMarketingPipe    = 25, String, 10,        S
ToggleTaxes              = 26, Array, Boolean, 4, Y,Y,Y,Y
```

LookupKey Section

Reserved. The LookupKey section is a representative list record section that defines an array of compound data types.

Names specified in the LookupKey section point to other sections that have names that are derived by concatenating LookupKey to the names. For instance, the List field of LookupKey defines the names Department1 to Department9. The INI file section, LookupKey.List defines default values for all LookupKey.List.Department*n* sections. The LookupKey.List.Department1 section defines values for the Department1 element.

Each field is *reserved*. See Figure 9 for a description of the generic syntax of a data field specification.

Source Code

```
[LookupKey]
WeightKeyCodes       =  1, Array, UInt, 20, 1,2,3,4,5,6,7,8,9,0,0,0,0,0,0,0,0,0,0,0
MatrixKeyboardActive =  2, Boolean,         0
MatrixKeyboardId     =  3, UInt,            0
UseAutoRoundUp       =  4, Boolean,         0
ItemCode1            =  5, String,          14, ""
ItemCode2            =  6, String,          14, ""
ItemCode3            =  7, String,          14, ""
ItemCode4            =  8, String,          14, ""
ItemCode5            =  9, String,          14, ""
ItemCode6            = 10, String,          14, ""
ItemCode7            = 11, String,          14, ""
ItemCode8            = 12, String,          14, ""
ItemCode9            = 13, String,          14, ""
ItemCode10           = 14, String,          14, ""
NumberOfItemcodes    = 15, UInt,            10
```

```
AlphaKeyCodes        = 16, Array, UInt, 26, 61,206,76,74,207,77,71,208,72,75,\
                                            205,204,203,93,94,202,92,65,67,201,\
                                            90,66,62,91,200,81
List = 40, Sections, Department1, Department2, Department3, \
                     Department4, Department5, Department6, \
                     Department7, Department8, Department9
```

LookupKey.List Section

Reserved. The LookupKey.List section is a representative list record section that defines an array of compound data types. It defines default values for all elements in the list.

Each successive level of granularity inherits all preceding definitions. The most granular level, such as `LookupKey.List.Department1`, must have the `Id` field (with a unique value) but need not have any other values. Values not specified are taken from the higher-level template sections.

A list record section is the parent definition of all data field definitions that can then be used by child list data sections.

List record sections are composed of field name definition statements just as regular record sections are. There are several differences however.

The rules that govern list records are:

- The maximum number of entries in the list must be 99 or fewer.
- The list record section must define all data fields. You cannot use data fields in list data sections that are not defined in the parent.
- The first field name definition statement must define a UInt data field named Id, which is a keyword.
- The last field name definition statement in the list record section must define a Boolean field named `DeleteFlag`, with an index value of 99 and no default value.
- Default data values in the list record section apply to each list data section if there is no default.
- You cannot display more than one list in the same TurboVision dialog because there would be conflicting ID values. You must use a separate dialog for each list.

Each field is *reserved*. See Figure 9 for a description of the generic syntax of a data field specification.

Source Code

```
[LookupKey.List]
Id                          =  1, UInt
LookupKey                   =  2, String, 17,""
; For the following options the values are 0=none, 1=Opr, 2= Mgr, 3=NotAllowed
TaxNoTax                    = 11, UInt,   0
FSNoFS                      = 12, UInt,   0
Refund                      = 13, UInt,   0
StoreCoupon                 = 14, UInt,   0
ManufacturerCoupon          = 15, UInt,   0
ValidateWeightPrice         = 16, UInt,   0
DeleteFlag                  = 99, Boolean, 0
```

LookupKey.List.Department<n> Sections

Reserved. The LookupKey.List.Department1 section is a representative list data section that defines compound data values for an element of the list defined by the LookupKey.List parent list record section.

Each list data section defines values for that set of values in the list identified by the *Id* field.

Each field is *reserved*. Data field specifications include only the name of the field, as defined in the parent list record section, and default values for the ID.

```
Id                          =  1
<fieldName>                =  <defaultValue>
```

Figure 14. LookupKey.List.DepartmentN Data Field Definition Statements

Id

A reserved field name that must be the first data definition statement.

fieldName

A field name that must match one defined in the parent list record section.

defaultValue

A data value that is within the limits specified by the data type in the parent list record section.

Source Code

```
[LookupKey.List.Department1]
Id                   = 1
LookupKey            = 999901
TaxNoTax             = 0
FSNoFS               = 0
Refund               = 0
StoreCoupon          = 0
ManufacturerCoupon   = 0
ValidateWeightPrice= 0
DeleteFlag           = 0
```

InputValidation Section

`Modifiable in optnparm.us9` to add new validation ranges. This section specifies the rules used for validating data that is entered for the fields defined in the `optnparm.ini` file. Refer to "Input Validation Data Types" on page 460 for information about each type of validation.

Modification Example - Variety Key Codes

Support for a key code that you use to specify a tender variety depends on the *Variety Key Code* option in personalization. Input validation for the default values of 0, 97, 98, and 99 for the option uses the following code in the `optnparm.ini` file:

```
[InputValidation]
R97_99 = RANGE, 97, 99
RMAP1= RANGE, 0, 0
RMAP2= RANGE, 0, 0
RMAP3= VALUESET, 0, 0
M5 = MULTIPLEPOLICY, OR, R97_99, R0_0, RMAP1, RMAP2, RMAP3
```

If you want to add additional values for the option, you must create a file (usually `optnparm.us9`) that overlays this code.

If you are not using matrix keyboards, values in the range 130-190 are recommended for additional values for the option. You should add sequential values to simplify the range checking. For example, if you want to add validation for key codes 130-154, you would add the following lines to the overlay file:

```
[InputValidation]
RMAP1=RANGE, 130, 154
```

If you want to add validation for key codes 130-154, 170-180, 185, and 190, you would add the following lines to the overlay file:

```
[InputValidation]
RMAP1= RANGE, 130, 154
RMAP2= RANGE, 170, 180
RMAP3= VALUESET,185,190
```

To use the key codes for which you have defined validation rules, you must add the key codes to your keyboard and also to the adx_ipgm\eams@000 input state table, if they do not already exist. You can use the function code range 91 - 99 as the default values for the key codes that you add to the input state table. Refer to the *4690 Operating System: Programming Guide* for information about using the Input Sequence Table Utility.

Source Code

This is partial code for the InputValidation section. Representative types of validation are shown. Refer to the `optnparm.ini` source file for the complete code.

```
[InputValidation]
N = INTEGER
NA = INTEGER, ARRAY
P = INTEGER, +
PA = INTEGER, +, ARRAY
AN = ALPHANUMERIC
R1_20 = RANGE, 1, 20
R1_80 = RANGE, 1, 80
R1_4A= RANGE, 1, 4, ARRAY
R1_5 = RANGE, 1, 5
R1_5A= RANGE, 1, 5, ARRAY
R999_999 = RANGE, -999, 999
VS6_8 = VALUESET, 6, 8
VS1_10_100_1000 = VALUESET, 1, 10, 100, 1000
DT = STRFTIME
NN = NUMERIC
ITEM = ITEMCODE
AB = ALPHABET
CD = COMPAREDIALOG, 0, Ctrl22
```

```
MP = MULTIPLEPOLICY, AND, R1_4, CD
FV = FRANKFORMATVAR
```

Additional Sections

There are many other sections in the `optnparm.ini` file that are not described here. Refer to the `optnparm.ini` source file for the other sections.

SCNNRWND (Toolkit Scanner Window Simulator)

The `scnnrwnd.ini` file is used to simulate the POS environment in a Windows development environment. This `.ini` file specifies several representative items and their codes that might be scanned, which allows the simulator to test appropriate processing of scanned items.

All fields in the `scnnrwnd.ini` file are *modifiable.*

Source Code

In this `.ini` file, each section name is an item code description. The item code is specified on the Code parameter in the section.

```
[Item Number 1]
Code=000000000001

[Item Number 2]
Code=000000000002

[Item Number 3]
Code=000000000003

 [Item Number 4]
Code=000000000004

[Item Number 5]
Code=000000000005

[Item Number 6]
Code=000000000006

[Item Number 7]
Code=000000000007

[Item Number 8]
Code=000000000008

 [Item Number 9]
Code=000000000009
```

Chapter 5. Migrating to and Enabling GTIN

Migrating to GTIN Support

Note: Version 3 of SurePOS ACE is the first version with GTIN support.

Use this procedure to migrate to GTIN support:

1. These migration paths are supported that allow you to migrate to a version that supports GTIN:
   - New installation of a version of SurePOS ACE that supports GTIN.
   - Migration from a version that does not support GTIN to a version of SurePOS ACE that supports GTIN.
   - Migration from SurePOS ACE V1R5 to a version of SurePOS ACE that supports GTIN.

   The Item Movement List file (aceitmls if you migrate from a previous version of SurePOS ACE and eamilist if you migrate from Supermarket Application) is automatically converted to support 14-digit item codes when you migrate to a version of SurePOS ACE that supports GTIN. During migration, a new record containing the value 000099999999999999 is added to the beginning of the Item Movement List file to indicate that the file supports 14-digit item codes. When that record is in the file, migration of the file will not be re-attempted on subsequent installations. The new record will not be displayed on the interface when the file is read, because a movement list ID of zero is not valid.
2. Verify that the new state table, label format table, and GS1 DataBar/GTIN bar code personalization options are installed. These files are not overlaid during migration.

   These are the changes to the label format table:
   - Modified format 26 - GS1 DataBar-14 by adding an GS1 DataBar14 label format definition that contains one field, length 15, no check digit, function code 80
   - Modified format 27 - GS1 DataBar Expanded by adding an GS1 DataBarEXPAN label format definition that contains one field, length 80, function code 80

   These are the changes to the input state table:
   - MAIN state - modified the field length that is accepted by the ENTER key (80) from 20 to 80
   - PR/CHG state - modified the field length accepted by the ENTER key (80) from 13 to 80
3. If you are using a noncustomized `sapath` file, then verify that the new `sapath` definitions are installed. You can verify that the new definitions are installed by using the GCONF32 utility to view the `sapath.rvt` layout. If there is a file layout for EAMITM14, then the new definitions are installed.

   If you are using a customized `sapath` file, then you must do the following:
   1. Use the GCONF32 utility to copy the following file layout from the `sap169.rvt` file to your `sapath.rvt` file:
      - aceitmls
      - eamexcp?.dat
      - eamtran?.dat
      - eamitm14
      - <ACN_M14:>eamimo??.dat
   2. Generate a new `sapath.ddf` file from the modified `sapath.rvt` file.

Enabling GTIN Support

Use this procedure to enable GTIN support after migrating to a version of SurePOS ACE that supports GTIN:

1. Configure your scanners to enable GS1 DataBar-14 and GS1 DataBar Expanded bar codes.
2. Convert the eamitemr.dat file to support 14-digit item codes using one of the following methods:
   - Use Personalization.
     1. Select Item File Rebuild on the Misc menu in Personalization.
     2. Select Build New.
     3. Type 7 for the new key length. (Type the existing values for Record Length and Number of Records.)
     4. Select Rebuild. The ACEICRBL utility starts in the background.
     5. Select Cancel to close the panel.
     6. After the utility finishes processing the file, select Refresh.
     7. Select Activate New.
   - Use the command line.
     1. Start the ACEICRBL utility (see “Migrating the Item Record File” on page 549 for the syntax) and change the key length from 6 to 7.
     2. Rename the output file (eamitemr.tmp) to eamitemr.dat.
   - Copy adx_ipgm\gtnitemr.dat to adx_idt1\eamitemr.dat - this can only be used if a clean install was done. adx_ipgm\gtnitemr.dat is a sample 14-digit Item Record file that is shipped with a vision of SurePOS ACE that supports GTIN.
3. Re-IPL all terminals so that the sapath layout for 14-digit item codes will be used.
4. Modify host programs to use the new batch header (X'FFFFFFFFFF14') to indicate support for 14-digit item codes in ADDMI batches.

Before you add 14-digit item codes to the Item Record file, you should run a Long Period Close. The close processing will create new long and short Item Movement files that support 14-digit item codes. After you enable GTIN and before you perform a Long Period Close, the two leftmost digits of each item code will be truncated when data is written to the Item Movement file. If the two leftmost digits are both zero, item movement will be tracked correctly. If either digit is a nonzero number, then item movement will be tracked incorrectly.

Disabling GTIN Support

Use this procedure to disable GTIN support:

1. Convert the eamitemr.dat file to remove support for 14-digit item codes using either of the following methods:
   - Use Personalization.
     1. Select Item File Rebuild on the Misc menu in Personalization.
     2. Select Build New.
     3. Type 6 for the new key length. (Type the existing values for Record Length and Number of Records.)
     4. Select Rebuild. The ACEICRBL utility starts in the background.
     5. Select Cancel to close the panel.
     6. After the utility finishes processing the file, select Refresh.
     7. Select Activate New.
   - Use the command line.
     1. Start the ACEICRBL utility (see “Migrating the Item Record File” on page 549 for the syntax) and change the key length from 7 to 6.

   2. Rename the output file (eamitemr.tmp) to eamitemr.dat.
2. Re-IPL all terminals so that the sapath layout for 12-digit item codes will be used.
3. Use the following steps to determine which batches must be migrated:
   a. On the SurePOS ACE Main Menu, select Batch Data Maintenance.
   b. Select Filter Batches.
   c. Select Migration Required.
   d. Select OK.
   e. For each batch that is displayed, select the batch and then type M.
   f. Select Start Actions.

Returning to a Pre-GTIN Level of SurePOS ACE

If you cancel ASM, your data files will be restored to their state before the installation was performed. Any changes that were applied to the files will be lost. To preserve any changes that were applied to the Item Record file (eamitemr.dat), Item Movement List file (aceitmls.dat) or the Item Movement files (eamimo??.dat), perform the following steps:

1. Follow the steps in "Disabling GTIN Support " on page 548.
2. To preserve changes to the Item Record file, type the following command:

```
copy c:\adx_idt1\eamitemr.dat c:\adx_ibul
```

3. To preserve changes to the Item Movement files, perform these steps:
   a. Backup your current copy of the eamimo??.dat file.
   b. Convert each Item Movement file from a 7-byte key to a 6-byte key by typing the following commands (conversion truncates the two leftmost digits of the item code):

```
acekfrbl c:\adx_idt4\eamimove.dat c:\adx_idt4\eamimove.tmp -IK7 -K6 -IL23 -L22
acekfrbl c:\adx_idt4\eamimovo.dat c:\adx_idt4\eamimovo.tmp -IK7 -K6 -IL23 -L22
acekfrbl c:\adx_idt4\eamimolc.dat c:\adx_idt4\eamimolc.tmp -IK7 -K6 -IL23 -L22
acekfrbl c:\adx_idt4\eamimolp.dat c:\adx_idt4\eamimolp.tmp -IK7 -K6 -IL23 -L22
```

   c. Stop Checkout Support.
   d. Rename each .tmp file to a .dat file. For example, rename eamimove.tmp to eamimove.dat.
4. To preserve changes to the Item Movement List file, type the following command:

```
aceicrbl c:\adx_idt1\aceitmls.dat c:\adx_ibul\aceitmls.dat -tDD -IK4 -K4 -L16 -IL18
```

5. Cancel maintenance. All files in c:\adx_ibul should be restored successfully.

Using ACEICRBL to Migrate EAMITEMR and ACEITMLS

With support for 14-digit (GTIN) item codes, SurePOS ACE provides the ACEICRBL utility program to be used to migrate the eamitemr and aceitmls files to the GTIN layout.

Note: If TOF is active, the ACETIRBL program is automatically started when ACEICRBL (target = EAMITEMP) is used to rebuild an Item Record file.

Migrating the Item Record File

The syntax of the utility is `aceicrbl source destination -IKx -Ky`. These are the parameters:

source

Specifies the path and file name for the file to be migrated.

destination
: Specifies the path and file name to be used for the output file.

-IKx
: Specifies the key length (`x`) of the file to be migrated. Valid values for `x` are 6 or 7.

-Ky
: Specifies the key length (`y`) of the output file. Valid values for `y` are 6 or 7.

When you migrate from a 6-byte key to a 7-byte key, ACEICRBL prepends a byte containing 00 to the item code, alias item code, and large linked item code fields. To convert the eamitemr file from 12-digit item codes to 14-digit item codes, enter the following command:

```
ACEICRBL c:\adx_idt1\eamitemr.dat c:\adx_idt1\eamitemr.tmp -1K6 -K7
```

When you migrate from a 7-byte key to a 6-byte key, ACEICRBL truncates the two leftmost digits of the item code, alias item code, and large linked item code fields. To convert the eamitemr file from 14-digit item codes to 12-digit item codes, enter the following command:

```
ACEICRBL c:\adx_idt1\eamitemr.dat c:\adx_idt1\eamitemr.tmp -1K7 -K6
```

Migrating the Item Movement Lists File

The syntax of the utility is `aceicrbl source destination -tDD -IK4 -K4 -ILx -Ly`. These are the parameters:

source
: Specifies the path and file name for the file to be migrated.

destination
: Specifies the path and file name to be used for the output file.

-tDD
: Specifies that the file to be migrated and the output file are direct files.

-IK4
: Specifies the key length of the file to be migrated. Only a value of 4 is valid for the key length.

-K4
: Specifies the key length of the output file. Only a value of 4 is valid for the key length.

-ILx
: Specifies the length (`x`) of each record in the file to be migrated. Valid values for `x` are 16 or 18.

-Ly
: Specifies the length (`y`) of each record in the output file. Valid values for `y` are 16 or 18.

To convert the ACEITMLS file from 12-digit item codes to 14-digit item codes, enter the following command:

```
ACEICRBL c:\adx_idt1\aceitmls.dat c:\adx_idt1\aceitmls.tmp -tDD -IK4 -K4 -L18 -IL16
```

To convert the ACEITMLS file from 14-digit item codes to 12-digit item codes, enter the following command:

```
ACEICRBL c:\adx_idt1\aceitmls.dat c:\adx_idt1\aceitmls.tmp -tDD -IK4 -K4 -L16 -IL18
```

Appendix A. Setting Up the Development Environment

The Toshiba Global Commerce Solutions SurePOS Application Client/Server Environment for 4690 OS Developer's Toolkit lets Toshiba Global Commerce Solutions developers and service organizations, your developers, and Toshiba Global Commerce Solutions Business Partners extend SurePOS ACE, which includes rebuilding SurePOS ACE executables with third party extensions. The Toolkit supports both controller and terminal code.

To add user function, you can use these five methods to extend SurePOS ACE code:

- Personalization
- BOB code for reports and some SurePOS ACE EPS functions
- C++ extension techniques supported by SurePOS ACE:
  - Policies
  - Events
  - Inheritance
  - Java extension techniques to modify the ACE PINPad Controller. See section (Installing/ Building PPCTRL.JAR (PIN Pad Controller) using Toolkit) for complete details

The Toolkit helps you use the three C++ extension methods. Before using the Toolkit to extend SurePOS ACE objects, you should thoroughly review predefined SurePOS ACE personalization options, which control SurePOS ACE behavior without you having to write any code. Make sure there is not an option that controls the behavior that you intend to change. The Toolkit is not needed for personalization because personalization does not require rebuilding SurePOS ACE. See "Using ACEPERSL to Maintain Personalization Options" on page 572 for more information.

BOB is a language like C++ that is interpreted by the SurePOS ACE BOB engine. You need only a text editor to write BOB reports and to extend a SurePOS ACE EPS function written in BOB. The BOB engine compiles and interprets any BOB code at run time. You do not need the Toolkit to work with BOB code.

You do need the Toolkit for the three C++ methods of extending SurePOS ACE, which involve writing, testing, and debugging code, and linking executables.

Windows is the development environments for the 4690 OS version of SurePOS ACE. You do not build SurePOS ACE on a 4690 OS platform. The Toolkit provides both DOS and Windows tools for building SurePOS ACE on Windows platforms. These components are included in the Toolkit:

- SurePOS ACE header files and object files
- SurePOS ACE EPS header files and object files
- Build tools
- 4690-OS-specific build tools
- POS device and environment simulators, as described in "IOP (Toolkit I/O Processor Simulator)" on page 530, "MSRWNDW (Toolkit Magnetic Stripe Reader Window Simulator)" on page 532, and "SCNNRWND (Toolkit Scanner Window Simulator)" on page 546

In addition to tools shipped with the Toolkit, you must have the C++ compiler and the makefile processor described in "Requirements" on page 554.

The Toolkit is available only in English but works with any national language version of SurePOS ACE to produce objects for that language.

Requirements

Software Requirements

The Toolkit has the capability to produce SurePOS ACE executables for Windows and for 4690 OS. Windows executables are solely for development and debugging.

The Toolkit has the following software prerequisites:

- Windows operating system
- IBM Visual Age C++ (VA C++) compiler, Version 3.6.5 Fixpak 2, including the runtime files. The compiler and a debugger, which can also remotely debug SurePOS ACE applications on 4690 OS, are included in the IBM Development Environment for OS4690. For information about obtaining a copy of this compiler, use the Knowledgebase Search on the *www.ibm.com/support/docview.wss?uid=pos1R1002981* Web site and search on "VA C++".

  The runtime files are available in the 4690opt\vadevenv.zip file on the 4690 OS CD. You need PKUNZIP to open the file.
- Object REXX to run the command files that are used in generating 4690 installation packages for CD-ROM. An open-source version of Object REXX is available from the http://www.oorexx.org web site. Customers who have purchased the IBM version of Object REXX do not need to install the open-source version.

Hardware Requirements

These are the minimum requirements for a machine capable of compiling and linking SurePOS ACE:

- 1 GHz processor is recommended
- 256 MB of RAM (512 MB is recommended)
- 6 GB of free space on your hard drive (in NTFS partitions)
- SVGA display

The minimum hardware requirements for a system running Windows 7, 8, or 10 are the minimum hardware requirements set by Microsoft.

Refer to the documentation of the prerequisite software products for additional requirements.

Off-the-shelf tools, which are often used in a Windows environment, might cause additional requirements.

Installation and Setup

Installing the SurePOS ACE Toolkit involves two separate tasks:

- Installing all prerequisite software (see “Installing Prerequisite Software” on page 554)
- Installing the Toolkit (see “Installing the SurePOS ACE Toolkit” on page 555)

Installing Prerequisite Software

This section is a brief overview of what software and fixes you should install. It is presented in the same order that you should follow as you install the software.

1. Install the Windows XP , Windows 7, 8, or 10 operating system.

2. Install the IBM Visual Age 3.6.5 C++ compiler. For information about how to obtain a copy of this compiler, use the Knowledgebase search found at www.toshibacommerce.com/support Web site to search for "VA C++".
3. Install the IBM Visual Age C++ runtime files. Unzip the 4690opt\vadevenv.zip file from the 4690 OS CD into the directory on your development system that is specified on the `ACE_VA_RUNTIMES` environment variable.
4. Install Object REXX. After installation, ensure that the ibmcxxw\bin directory precedes the objrexx directory in the PATH environment variable. If the order of directories in the PATH environment variable is incorrect, you will get an error similar to this example (the offset might be different) when you link the ACE executables:

   ```
   C:\ibmcxxw\lib\user32.lib : fatal error LNK1101:
   invalid object module at file offset 0x0043666a+0x1a
   ```

   Verify that NLSPATH and INCLUDE variables are set correctly and that INCLUDE variable is set before the Object REXX.

   Note: The Toolkit on Windows 7, 8, or 10 must be run by a user that is an Administrator.

After installing all prerequisite software, you are ready to install the SurePOS ACE Toolkit.

Installing the SurePOS ACE Toolkit

Installing the SurePOS ACE Toolkit accomplishes two goals:

- Copying all files that you need for development from the product CD to your hard disk
- Setting up your environment so you can start development

Opus Make Version 6.12 for Windows is included in the Toolkit. It is used for rebuilding SurePOS ACE. When the environment variables are set using either aceflat.bat or acent.bat, the PATH is set to point to the Opus Make directory location in the Toolkit. Toolkit users do not need to do anything else for Opus Make to work.

The Setup program queries you during installation to determine your preferences for these options:

- Destination for the SurePOS ACE Toolkit (where all header and object files are placed)
- Destination for the SurePOS ACE adx* directories (where all .dat and `.ini` files will be placed by you, not by the build process)
- Whether Object REXX has been installed (creation of 4690 load images requires that Object REXX be installed. The Toolkit will install without Object REXX being installed.)

Note: If you are going to run any of the SurePOS ACE applications, for example ACEPERSL from the command line in a Windows environment, set the Command Prompt window size to 24x80. If the window size is larger than 24x80, then you could encounter these problems:

- SurePOS ACE panels might not seem to work in a Windows environment because the panel will display in a part of the Command Prompt window that has scrolled out of view.
- SurePOS ACE panels that display well in a Windows environment might be too large to display correctly in a 4690 OS environment.

Running the Setup Program

Starting the SurePOS ACE installation is simple:

1. Open a DOS command window.

2. Change to the root directory on the drive of your CD-ROM.
3. Type setup, then press Enter.

Logical names (specified in `fdsln.ini`) help SurePOS ACE locate important files for generating files at compile time. For the same reason, logical names must be consistent with the location of SurePOS ACE adx_xxxx directories.

If you allow setup to replace your logical names file (fdsln.ln), then setup makes your logical names consistent with the location of the SurePOS ACE adx_xxxx directories. The adx_xxxx directories contain files that are typically found on a SurePOS ACE controller. They include log files, INI files, HLP files and BOB files. Many are accessed exclusively through logical names. Setup backs up the Logical Names file as fdsln.bak and replaces it with the modified fdsln.ln, which reflects the location of the SurePOS ACE adx_xxxx directories.

Setup issues a warning if it does not find Object REXX. You can continue installing the Toolkit without Object REXX. However, it is a prerequisite for using the Toolkit. You might be able to compile SurePOS ACE without Object REXX, but all scripts for simplifying build tasks (such as building CDs) require Object REXX.

Setup automatically creates and updates the environment variables in the `aceflat.bat` file for 4690 OS targets and in the `acent.bat` file for Windows targets. If you specify that the Windows registry should be updated, the setup program adds registry settings for the POS simulator under the *\HKEY_LOCAL_MACHINE\SOFTWARE\vrtlcshr.ini\Virtual Cash Register* entry. If you are running on Windows 7, 8, or 10 at the end of a CD installation, you might encounter a pop-up screen telling you that this program might not have installed correctly. You may ignore this message and click on "This program installed correctly."

Note: Parallel installations of the same version of SurePOS ACE are not supported.

Environment Variables

The variables that are described in Table 119, except for the customer variables, must be set in the environment for a build to proceed correctly. You can set the environment variables for a specific build by running either `aceflat.bat` for 4690 OS target builds or `acent.bat` for Windows target builds.

Table 119. Environment Variables

Name	Description
ACE_BOB	The root directory of the BOB installation to be used by SurePOS ACE.
ACE_BOBLIB	This is the full path name of the BOB library.
ACE_CHECK_LIBS	If set to NO, will prevent executable makefiles from trying to recursively build libraries.
ACECOMPHDRS	A space-delimited or semicolon-delimited list of directories to find non base-SurePOS ACE header files (TurboVision, roguewave, bob, and SQL). TurboVision actually requires two entries because some files include a TurboVision header using the syntax `tv.h` while other files use the syntax `tvision\tv.h`.
ACE_CUSTOMER_EXES	A space-delimited or semicolon-delimited list of executables (without the exe suffix) that you have written and want to be included in a full rebuild of SurePOS ACE.

Name	Description
ACE_CUSTOMER_HELPFILES	A space-delimited list of text files for online help (without the txt suffix) that you have written and want to be included in a full rebuild of SurePOS ACE.
ACE_CUSTOMER_LIBS	This specifies a space-delimited or semicolon-delimited list of library files that you have written. Do *not* use the .lib suffix.
ACE_CUSTOMER_INI	This specifies a file which, if it exists, is included by `make.ini` during the build process.
ACE_DEFAULT_SRC_ROOT	This specifies the top level directory where default SurePOS ACE source is installed. It is assumed that the source is in subdirectories of the directory, which are created during installation of the toolkit.
ACE_DEFAULT_TARGET_ROOT_NT	For builds for a Windows target system, this specifies the top level directory where default SurePOS ACE object files (subdirectory OBJ), libraries (subdirectory LIB), and executables (subdirectory BIN) reside. Targets that do not exist in the local build tree directories are picked up here. (See `ACE_WORKING_TARGET_ROOTS_NT`.)
ACE_DEFAULT_TARGET_ROOT_FLAT_4690	For builds for a 32-bit 4690 OS target, this specifies the top level directory where default SurePOS ACE object files (subdirectory OBJ), libraries (subdirectory LIB), and executables (subdirectory BIN) reside. Targets that do not exist in the local build tree directories are picked up here. (See `ACE_WORKING_TARGET_ROOTS_FLAT_4690`.)
ACE_OPTIONAL_MAKEFILE_GEN_TARGETS	Specifies SurePOS ACE DAT files to include in the build.
ACE_OPTIONAL_FCN_EXCLUSIONS	Specifies optional SurePOS ACE features to be excluded from the build. See "Optional Function (Toshiba and Customer)" on page 563 for more information.
ACE_OS4690_DIR	Specifies the directory to which you copied os4690.zip from the 4690 system.
ACE_PROCESS_CUSTOMER_INIS	This specifies a space-delimited or semicolon-delimited list of INI files to be processed for .USX file overlays. Customers use this variable if they have added their own INI files to the build process.
ACE_PSK_DIR	Specifies the directory where you installed the PSK.
ACE_ROGUE	The root directory of the RogueWave installation to be used by SurePOS ACE.
ACE_RWLIB	This is the full path name of the RogueWave library.
ACE_SQL	The root directory of the SQL installation to be used by SurePOS ACE.
ACE_SQLLIB	This is the full path name of the SQL library.
ACE_SWING_DIR	Specifies the directory to which you copied swingall.jar from the 4690 system.
ACE_TARGET_OS	This specifies the target OS for executables. Valid values are NT or 4690OS.

Name	Description
ACE_TERMINAL_EXES	A space-delimited or semicolon-delimited list of terminal executables to build when building all of the executables via makefile.ace.
ACE_TV	The root directory of the TurboVision installation to be used by SurePOS ACE.
ACE_TVLIB	This is the fully-qualified path name of the TurboVision library.
ACE_VA_RUNTIMES	The root directory for a 32-bit 4690 target build only, under which all 32-bit 4690 OS runtime files and libraries are stored.
ACE_WORKING_DATA	A space-delimited or semicolon-delimited list of directories containing local changes to .ini or .dat files. The first directory in this list receives any DAT files generated during the build process.
ACE_WORKING_HDRS	A space-delimited or semicolon-delimited list of directories containing local header changes. The first directory in this list receives any headers generated during the build process.
ACE_WORKING_SRC	A space-delimited or semicolon-delimited list of directories containing local source changes. These directories are searched first during the build process.
ACE_WORKING_TARGET_ROOTS_NT	This is similar to `ACE_DEFAULT_TARGET_ROOT_NT` except that this variable can consist of a single directory, or a space-delimited or semicolon-delimited list of directories that can each contain the subdirectories OBJ, LIB, and BIN. However, make uses the first directory (either the single directory or the first in a list) to store locally created targets in the OBJ, LIB, and BIN subdirectories. Make creates these subdirectories if they do not already exist.
ACE_WORKING_TARGET_ROOTS_FLAT_4690	This is the same as `ACE_WORKING_TARGET_ROOTS_NT` except that it is for 32-bit 4690 OS targets.
VERSION	Specifies whether the target build has tracing capability or debug information. If set to DEBUG, the target build has debug information only. If set to TRACING, the target build has debug information and tracing capability (for the Windows Simulator only). If set to RELEASE, the target build has neither capability. This environment variable is set to TRACING for 32-bit 4690 OS targets and is set to DEBUG for Windows targets.
WIN32LIBS	The directory where win32 libraries exist. This is for Windows builds only.

Some environment variables have default values that vary with the target operating system. All such default values assume that SurePOS ACE is installed in the directory E:\ACE, which is arbitrary. You can install SurePOS ACE in any directory that uses eight-character names with a three-character extension.

Note: If the default value for variable X is `\BIN`, the value of X is the value of variable Y concatenated with `\BIN`.

Variables not listed in Table 120 do not have default values. If you do not specify valid values for the variables that do not have defaults, Opus Make exits with an error.

Table 120. Environment Variables with Default Values

	Target Operating System	
Name	Windows	4690 OS
ACE_BOB	%ACE_DEFAULT_SRC_ROOT%\bob	%ACE_DEFAULT_SRC_ROOT%\bob
ACE_BOBLIB	%ACE_BOB%\nt\bob_nt.lib	%ACE_BOB%\flexos\bob468.lib
ACE_CHECK_LIBS	YES	YES
ACE_ROGUE	%ACE_DEFAULT_SRC_ROOT%\rogue	%ACE_DEFAULT_SRC_ROOT%\rogue
ACE_RWLIB	%ACE_ROGUE%\lib\vtlmt.lib	%ACE_ROGUE%\lib\b46txtlh.lib
ACE_SQL	%ACE_DEFAULT_SRC_ROOT%	%ACE_DEFAULT_SRC_ROOT%
ACE_SQLLIB	%ACE_SQL%\sqlnt.lib	%ACE_SQL%\sql468.lib
ACE_TARGET_OS	Default value is NT.	
ACE_TERMINAL_EXES	acetsfna acetsfsl	acetsfna acetsfsl
ACE_TV	%ACE_DEFAULT_SRC_ROOT%\tvision	%ACE_DEFAULT_SRC_ROOT% \tvision
ACE_TVLIB	%ACE_TV%\nt\tvnt.lib	%ACE_TV%\flexos\tvflexos.lib
WIN32LIBS	%CXXMAIN%\lib	N/A

Generating Installation Packages for a 4690 System

This section details the process for generating 4690 installation packages.

Note: The JAR executable command must be in your Windows `PATH` environment variable to properly execute the installation commands. This executable is located in the \bin subdirectory of your Java Development Kit directory (i.e. C:\jdk1.1.8\bin).

To generate 4690 installation packages:

1. Create your code changes and any needed makefiles. See "Structure and Modification" on page 560.
2. Set the source and target variables. See "Building Against a Default Image" on page 565.
3. Modify the preppkg.cmd file, which is a sample command file to build a SurePOS ACE installation package that is included in the Toolkit. You should change the first code line to `base = "NO"` and modify the variable declarations as needed to match your environment. See "Setting the Variables Used by the preppkg Program" on page 566 for more information.
4. Create the add2pkg3.bld file, which is an ASCII file that describes what files should be added to or removed from the base build. See "Using the add2pkg3.bld File" on page 568 for more information about the add2pkg3.bld file.
5. Enter `Rexx preppkg` to build a package that can be installed from CD-ROM.
6. The preppkg program updates the maintenance version information.

   The d:\sp_ace\ace\install\sa_instl\adxeabgf.dat file (where d:\sp_ace is the directory in which the Toolkit is installed) is updated with the PTF number (e-fix number) as it changes in report module level.

The GA, or CSD maintenance level number, and PTF numbers are set in these files, and become part of d:\sp_ace\ace\install\asm_proc\maintlv1.bld, the main product catalog file.

The default package number is 3 for the base product.

There is usually no need to rebuild the SurePOS ACE EPS installation package.

As the `preppkg.cmd` file is running you may see the error message, "The system cannot find the file specified" before this message, "Finished generating the 4690 disks". You can ignore this error message.

Structure and Modification

The SurePOS ACE Toolkit is set up for each library and executable to have its own makefile of the same name. For instance, the executable `acepersl.exe` has a makefile called acepersl.mak. The one exception to this is that aceddmfl.mak is the makefile for both `aceddmbl.exe` and `aceddmml.exe`.

These makefiles reside in the make directory. Each makefile contains only information that is specific to its target. For the most part, this is a list of object files needed to build the target. An executable also has a list of libraries that it needs. If more than one executable has a large number of objects (or libraries) in common, these objects are assigned to a macro defined in `make.ini`. This is an initialization file used by Opus Make.

In addition to defining macros common to more than one file, `make.ini` specifies all default build rules. It also defines built-in macros that Opus Make uses for building targets. (srcpath.c is an example of such a macro.) You should become familiar with Opus Make syntax and semantics before you change these macros. The `make.ini` file includes:

- The `customer.ini` file as described in "Optional Function (Toshiba and Customer)" on page 563
- Toshiba optional function macros
- OS-specific sections:
  - Built-in macro defines
  - Compiler, linker, and lib flags
  - OS-specific build rules
  - OS-specific macros
- General build rules
- General macro definitions

An example makefile is shown in Figure 15.

```
############################################################################
#           Licensed Materials - Property of Toshiba                       #
#                "Restricted Materials of Toshiba"                         #
#          (C) Copyright Toshiba Global Commerce Solutions, Inc. 1997, 1998#
############################################################################

# Include dependency file if it exists
%if %file(aceborun.$(DEPEND_SUFFIX))
%include aceborun.$(DEPEND_SUFFIX)
%endif

# aceborun: Back-Office Executable (fires off the other back-office programs)

OBJS_TERMNOAPS       = epstrmnp$O  2xnoepap$O  epsathrp$O            \
                       tndrvlpf$O

OBJS_ACEBORUN        = aceborun$O  bocmmndf$O  datentry$O  atsgnona$O \
                       tvsgnond$O  tinputur$O  saoprstr$O  rsrclbrr$O \
                       isrsrclb$O  file$O      bomdlfct$O  tvapplct$O \
                       tvchpwd$O   iscopyrt$O  $(OBJS_TERMNOAPS)
```

```
LIBS_ACEBORUN = acecommn$L  acebuscm$L  acebusns$L \
                acestrgi$L  acestrsa$L  acebusst$L \
                $(LIB_COMMON_C)                    \
                $(ACE_TVLIB)                           \
                $(CUSTOMER_LIBS)                   \
                $(SYSTEM_LIBS_C)


aceborun.exe .REREAD : $(ACEBORUN_OPTIONAL_FCN) $(OBJS_ACEBORUN) $(LIBS_ACEBORUN)
              %do linkCommand

dependencies:
    %do dependCommand DEP_OBJS="$(ACEBORUN_OPTIONAL_FCN) $(OBJS_ACEBORUN)"
CURRENT_MAKE_FILE=$(INPUTFILE)


.BEFORE:
    %do libCheck LIBS="$(LIBS_ACEBORUN)"
```

Figure 15. aceborun.mak Example Makefile

The first part of the makefile sets macros that specify objects and libraries used to build the executable. The last part specifies two rules that build the executable and dependencies.

This is the syntax for building `aceborun.exe`:

```
make -f aceborun.mak aceborun.exe
```

The `ACEBORUN_OPTIONAL_FCN` macro, which is mentioned in the dependencies section, is not set inside this makefile. Macros used in makefiles can be set in `make.ini`, `or in the environment. Opus Make treats environment variables similarly to macros defined in a makefile. The `ACEBORUN_OPTIONAL_FCN` macro is set in the environment. Macros ending in `_OPTIONAL_FCN` contain a list of objects that comprise Toshiba or customer optional functions. See "Optional Function (Toshiba and Customer)" on page 563 for more information about these macros.

This is the syntax for building dependencies:

```
make -f aceborun.mak dependencies
```

The aceborun.mak file is rebuilt only if objects that it directly depends upon change or if the dependent libraries change. The aceborun.mak makefile depends directly on OBJS_ACEBORUN and the `ACEBORUN_OPTIONAL_FCN` macro because it lists them in the `dependencies` section. Opus Make processes the special `.BEFORE` target before it processes any other targets. SurePOS ACE uses this target to recursively make dependent libraries. If you know that libraries are up to date, you can set the environment variable `ACE_CHECK_LIBS` to No to avoid recursive makes. Recursive makes are avoided if dependencies only are being generated.

Some makefiles kick off the overall build process.

bldace.mak

The toplevel makefile used to start everything. It recursively calls make on makefile.gen and makefile.ace. It sets up dependencies, build and link libs, and executables.

makefile.gen

Builds all of generated header and .dat files. Even though its contents are checked during the building of each executable or library, makefile.gen is explicitly called because it generates .dat files, which are not involved in the building of executables or libraries.

makefile.ace

makefile.ace first builds all SurePOS ACE libraries and executables. To rebuild all libraries, executables, or dependencies, use makefile.ace.

Here is the syntax for building SurePOS ACE:

- To build all dependencies:

```
make -f makefile.ace dependencies
```

- To build all libraries:

```
make -f makefile.ace all_libs
```

- To build all executables and libraries:

```
make -f makefile.ace all
```

  If you are building for a Windows target, you can also build all executables:

```
make -f -makefile.ace all_exes
```

Adding an Executable or Library

Adding a new executable or library to the build system requires two steps:

1. Write the makefile to build the target as described in "Writing the Makefile to Build the Target" on page 562.
2. Add code to recursively make dependent libraries as described in "Recursively Making Dependent Libraries" on page 563.

Writing the Makefile to Build the Target

The SurePOS ACE convention is to name the makefile as `base_exe_name.mak`. If the executable is called `hello.exe`, the makefile is called hello.mak.

The makefile specifies objects and libraries needed to make the executable, or specifies just objects if the target is a library. It also provides a rule to build the target and provides a rule to build dependencies.

For an executable, the build rule specifies what objects and libraries the executable is dependent upon and performs a %do linkCommand to build the executable. For example, the makefile for the `hello.exe` executable specifies:

```
HELLO_OBJS = ...
HELLO_LIBS = ...

hello.exe : $(HELLO_OBJS) $(HELLO_LIBS)
    %do linkCommand
```

A library build rule should specify dependent objects only. For example, this is what a build rule for the bye.lib library might include:

```
BYE_OBJS = bye.obj tata.obj toodles.obj hastalavista.obj

bye.lib : $(BYE_OBJS)
```

For the dependencies rule, a %do dependCommand is performed. For the command to work correctly, certain variables must be passed to it:

- Each target should set a variable `DEP_OBJS` to its immediately dependent objects. (For example, executables should not include objects from libraries.) The list should be enclosed in quotes.
- A variable `CURRENT_MAKE_FILE` must be set to the value of the built-in `INPUTFILE\` macro. See the *Opus Make User's Guide* for more information about passing variables via %do.
- To use dependencies correctly, each makefile should include a file called *base_file_name*.$ (DEPEND_SUFFIX). The value for DEPEND_SUFFIX depends on the target operating system. It is *Ndp* for Windows and *F4d* for 4690 OS.

  For example, the makefile acepersl.mak, when compiling the target for 4690 OS, includes an `ACEPERSL.F4D` for dependency information.

Recursively Making Dependent Libraries

Each executable makefile should have the special .BEFORE target and use it to recursively make libraries that it links with. This ensures that libraries are up to date before the makefile attempts to link the executable.

Similar to the %do dependCommand (see "Writing the Makefile to Build the Target" on page 562), another rule is called by %do libCheck. You must set the variable `LIBS` to `LIBS_BASE_EXE`, which is the list of libraries that this executable requires in order to link successfully.

Optional Function (Toshiba and Customer)

There are two types of optional function in SurePOS ACE, Toshiba-supplied function and user-created function.

The first type is included with the SurePOS ACE product. All Toshiba-supplied optional functions are automatically included in builds. To exclude any of the functions, you must set the `ACE_OPTIONAL_FCN_EXCLUSIONS` environment variable to a semicolon-delimited list that contains the functions to be excluded. These are the values that you can include in the list:

ageentry
Customer age entry logging

apsinq
SurePOS ACE EPS balance inquiry feature

catalina
Catalina Marketing feature

checkprint
Check printing

compareprice
Comparison pricing feature

couponmult
Coupon multiplication feature

dptreport
Department report totals

dumpex
: Application dump control feature

ej
: Electronic journal feature

enhanceprint
: Enhanced printing feature (clean receipt, for example)

foreignmoney
: Foreign currency support

ramdisk
: Drop RAM disk during initialization

selfcheckout
: Self-checkout enhancements for the Toshiba Global Commerce Solutions Self Checkout System

tlstreport
: Tender Listing reports

For example, to turn off check printing and the enhanced printing feature, `ACE_OPTIONAL_FCN_EXCLUSIONS` has the value checkprint;enhanceprint.

The second type of optional function is function that you add. This function is included in SurePOS ACE through the use of environment variables. To let you modify existing executables, each executable depends on objects defined in a macro with the name *base_executable_name_OPTIONAL_FCN*. For `hello.exe`, this variable is `HELLO_OPTIONAL_FCN`. An example within SurePOS ACE code is `CTLNMKON.CPP` for optionally including the Catalina Marketing function.

For many base SurePOS ACE executables, these variables are not defined. They are null strings. Each terminal executable uses its own variable, `executableName_OPTIONAL_FCN`.

There are several terminal configurations for SurePOS ACE, which differ only by which optional function is included. These diverge from the typical makefile structure slightly. These terminal configurations are supported through makefiles:

Table 121. SurePOS ACE Terminal Configurations

ACE_TERMINAL_EXES	SurePOS ACE EPS	Full-Screen Option	Store Integrator GUI
acetsfna	No	Yes	No
acetsfsl	Yes	Yes	No
jsifts10	Yes	No	Yes

The makefile for each terminal sets appropriate optional functions and includes a base terminal makefile called terminal.mak. Rules for building the target and its dependencies are in this file.

The `make.ini` file includes a file called `customer.ini`. You are responsible for creating and maintaining your own `customer.ini` file. As its name implies, `customer.ini` contains all customer-specific macro definitions and build rules. You can define your own optional functions in this file. For instance, if you have extended the checkout support executable (`acecsmll.exe`), `customer.ini` might contain this line:

```
ACECSMLL_OPTIONAL_FCN= customer_specific.obj
```

The point of interest in the code architecture is that refreshes from Toshiba do not overwrite the `customer.ini` file.

Building Against a Default Image

In a networked environment that is either tight on local disk space or that has more than one version of a SurePOS ACE build active at a time, it is advantageous for developers to have only their changes on their local system and use a *default* location to pick up source or targets they have not changed. The SurePOS ACE build process supports this setup using the environment variables `ACE_WORKING_TARGET_ROOTS_NT`, `ACE_DEFAULT_TARGET_ROOT_NT`, `ACE_WORKING_SRC`, `ACE_WORKING_HDRS`, and `ACE_WORKING_DATA`. Essentially, all of the `*WORKING*` variables point to local changes, while `*DEFAULT*` variables point to source and targets that should be used if they do not exist locally.

Target Variables

The `ACE_WORKING_TARGET_ROOTS_NT` variable consists of a list of directories. Each directory may or may not have the subdirectories bin, obj, and lib. Note that the first directory in the list must contain all three subdirectories or the build process will attempt to create the subdirectories. `ACE_DEFAULT_TARGET_ROOT_NT` consists of a single directory which may or may not have the subdirectories bin, obj, and lib. See the earlier descriptions of these variables for more information.

Example:

- ACE_WORKING_TARGET_ROOTS_NT=f:\dir1 f:\dir2 e:\dir3
- ACE_DEFAULT_TARGET_ROOT_NT=n:\dir4

The following directories exist:

- F:\dir1\obj
- F:\dir1\lib
- F:\dir1\bin
- F:\dir2\lib
- E:\dir3\obj
- N:\dir4\bin
- N:\dir4\obj
- N:\dir4\lib

In this example, the build process uses the following directories, in this order, to search for object files: f:\dir1\obj e:\dir3\obj n:\dir4\obj. Also, object files generated during a build are placed into f:\dir1\obj.

For library files, the search order would be f:\dir1\lib f:\dir2\lib n:\dir4\lib. Again any library files would be placed into f:\dir1\bin.

Lastly, for the executables, the search order would be f:\dir1\bin n:\dir4\bin. Any executable generated would be placed into f:\dir1\bin.

Source Variables

Environment variables assume that the source directory tree created during installation does not need to be replicated at the local level.

ACE_DEFAULT_SRC_ROOT Variable

The ACE_DEFAULT_SRC_ROOT environment variable points to the directory (presumably on the LAN) where this source directory tree exists. Internally, makefiles use the directory pointed to by this variable to come up with a set of directories that are searched for source, headers, and other files.

Example:

```
ACE_DEFAULT_SRC_ROOT=f:\ace
```

Makefiles search directories `f:\ace\person f:\ace\storimpl f:\ace\ui`

ACE_WORKING_SRC Variable

The ACE_WORKING_SRC environment variable contains a list of directories that are directly searched for source, such as .c, .cpp, and .asm files.

Example:

```
ACE_WORKING_SRC=f:\localdir1;f:\localdir2
```

Makefiles will search directory f:\localdir1 for source files first, then f:\localdir2, and then all of the directories calculated from the ACE_DEFAULT_SRC_ROOT environment variable.

ACE_WORKING_HDRS and ACE_WORKING_DATA Variables

The ACE_WORKING_HDRS environment variable is used for header (.h, .hpp, .inl, .cc) files and the ACE_WORKING_DATA environment variable is used for data (.hlp, .dat, .ini) files. Both environment variables consist of lists of directories. Any header files that are generated during a build are put into the first directory specified in the ACE_WORKING_HDRS environment variable. Any data files that are generated are put into the first directory specified in the ACE_WORKING_DATA environment variable.

Setting the Variables Used by the preppkg Program

The sample preppkg.cmd program sets variables that are needed to generate a 4690 OS installation package. During installation of the Toolkit, the variables used by the preppkg program are modified based on the directory structure in your system. If you change the location of files after the Toolkit is installed, you must modify the variables described in this section.

Table 122 describes the variables that are used or set for a base package. If your development environment is set up as described in Figure 16, then the table also gives the value for each variable.

```
The toolkit is installed in d:\hobbes
Business partner code is installed in d:\nrsch1
Customer-specific code is installed in d:\geh1
The customer-specified release name is ge.h1
```

Figure 16. Example of Development Environment

Table 122. Variables Used by preppkg.cmd

Variable	Value	Description
pkgBuild	d:\hobbes\ace\pkgbld	Target directory of the installation package
primeDir	d:\hobbes\ace	Top-level directory of the base SurePOS ACE files
buildRoot	d:\hobbes	Top level of the location of the base targets
baseSource	(null)	Not used
baseRoot	(null)	Not used
primeDir	d:\hobbes\ace	Top of the base SurePOS ACE directory tree
release	ace.hobbes	Release name

The values listed in Table 122 are overridden by the customer-specific values that are specified next in the preppkg.cmd file. If your files are outside the base SurePOS ACE file tree, then you would specify the variables as listed in Table 123.

Table 123. Customer Variables Used for preppkg.cmd Example

Variable	Value	Description
pkgBuild	d:\geh1\ace\pkgbld	Target directory of the installation package
primeDir	d:\geh1\ace	Top-level directory of the customer-specific SurePOS ACE files
buildRoot	d:\geh1	Top level of the location of the customer targets
baseSource	d:\nrsch1\ace d:\nemo\ace	Top levels of files for business partner and base builds. Used as a search path if files are not found in the directory specified by the buildRoot variable.
baseRoot	d:\nrsch1 d:\nemo	Top levels of targets for business partner and base builds. Used as a search path if files are not found in the directory specified by the buildRoot variable.
primeDir	d:\geh1\ace	Top of the customer-level SurePOS ACE directory tree

Variable	Value	Description
release	ge.h1	Release name
bldNumbr	B012	Used as the efix number in reporting the modification level
bildTlkt	N	Placeholder only (not used in package generation)
aceLevel	h012	Placeholder only (not used in package generation)
lvlAbbrv	h012	Placeholder only (not used in package generation)
nversion	V1R5	Version and release used in ASM
versonid	geh1	Placeholder only (not used in package generation)
newproc	Y	Adds the directory specified on the baseSource variable to the search path
spclNrsc	Y	Placeholder only (not used in package generation)
bldPrefix	B	Placeholder only (not used in package generation)
ACEPtf	UR00000	PTF used by report module level
APSPtf	UR00000	PTF used by report module level

Using the add2pkg3.bld File

The add2pkg3.bld file is an ASCII file that contains three sections, in the following order:

1. Files to be added to uncompressed files
2. Files to be added to compressed files
3. Files to be removed from the build

In each of the three sections of the file, each line refers to only one file to be added or removed. You can use multiple lines in any of the sections.

```
* Add to uncompressed files
*
Y N N I R A README.ABC   README.ABC   ROOT     NA Y NONE

* Add to the compressed files
*
Y Y Y I R F OPTNPARM.US4 OPTNPARM.US4 ROOT     CC Y NONE
Y Y Y I R F ACESDESC.US4 ACESDELT.US4 ROOT     CC Y NONE
Y Y Y I R F ACERDESC.US4 ACERDELT.US4 ROOT     CC Y NONE
Y Y Y I R A ACEREINI.US4 ACEREINI.US4 ROOT     CC Y NONE
Y Y Y I R A ACEDTMNI.US4 ACEDTMNI.US4 ROOT     CC Y NONE
Y Y Y I R A ACEDDMI.US4  ACEDDMI.US4  ROOT     CC Y NONE
Y Y Y I R A ACESLIRD.US4 ACESLIRD.US4 ROOT     CC Y NONE

* Remove the following files from the base
ACETSNAL.386
```

Figure 17. Example of add2pkg3.bld Format

In the third section, each line contains the name of one file to be removed. In the first two sections, each line contains six positional flag fields followed by file descriptor information. Each line contains these fields (these names do not exist outside this document, they are used for reference only):

Flag1
: Specifies whether to copy the file to CD. Valid values are `Y` and `N`. `N` indicates that the build process itself uses the file.

Flag2
: Specifies whether the file may be included on maintenance CDs in the future. Valid values are `Y` and `N`.

Flag3
: Specifies whether to include an entry in the product control file. Valid values are `Y` and `N`.

Flag4
: Indicates the module type. The Flag4 value prefixed to pgm specifies which adx_* directory to place the file in.

Flag5
: Specifies the type of maintenance being performed on the module. Valid values are `A` (add), `R` (replace), or `D` (delete).

Flag6
: Specifies how the maintenance will be applied to the module. Valid values are `A` (apply normally) or `F` (force).

File1
: Specifies the name of the file on the build directory.

File2
: Specifies the name of the file on the CD.

Location1
: Specifies the location on the CD to place this file

Distr
: Specifies the distribution attributes for the file. Valid values are `CC`(compound on close), `LL` (local), `CU` (compound on update), or `NA` (??)

TransFlag
: Specifies whether the file is translatable. Valid values are `Y` or `N`.

Special
: Specifies whether special processing is needed. `NONE` indicates no special processing is needed.

Installing and building the PPCTRL.JAR (PIN Pad Controller) using Toolkit

Software Requirements

The Toolkit has the following software prerequisites:

- JavaPOS – Drivers that support IBM and Toshiba POS specific peripherals such as 2x20 displays and receipt printers on Windows.
- Apache Ant™ - A Java library and command-line tool that helps build the software.
- OS4690.zip - contains 4690 OS specific Java classes.
- COMM4690.JAR - contains 4690 specific communications classes.

Installing and building

Installing and building PPCTRL.JAR (PIN Pad Controller) using Toolkit nvolves the following tasks:

- Installing all prerequisite software (see)
- Silent installation and uninstalling: see
- Install the latest Apache Ant™ Builder from the web (see)
- Copy OS4690.ZIP on Windows (see
- Building and executing PPCTRL (PIN Pad Controller) using Apache Ant

Installing Prerequisite Software

Do the following to install the Toshiba JavaPOS/OPOS on Windows:

1. Download the appropriate installable file(s) from: www.toshibacommerce.com/support.
    1. Under Hardware, select UPOS (OPOS/JavaPOS), and click on the right arrow.
    2. Download the Windows 64-bit w/SDK driver package.
2. Run the Setup.exe and follow the directions on each panel.
    1. During installation, select JavaPOS Development Support (not OPOS).
    2. Be sure to select Install as System JVM and take defaults for other prompts.
3. After the installation is complete, you must restart your system for the configuration changes to take effect.

Silent installing and uninstalling

For silent installation and how to uninstall, refer to Toshiba UnifiedPOS User's Guide XXXXXXXX:

Installing the latest version of Apache Ant™

Use the following steps to install the latest version of the Apache Ant™ build file:

1. Download the appropriate installable file(s) from: http://www.ant.apache.org/bindownload.cgi
2. Add system variables:
    1. Set ANT_HOME to point to where you installed Apache Ant. For example, c:\apach-ant-1.9.4
    2. Set JAVA_HOME to point to where your JVM isinstalled if not set already.
3. Add system path variable. For example, `c:\apach-ant-1.9.4\bin` to point to bin directory where you installed Apache Ant.
4. Install the EPS source from the Toolkit. (PIN Pad Controller source files can be obtained from the EPS source CD.) Ensure you install the EPS source in the same directory that you installed your ACE source.

Copy the OS4690.Zip and COMM4690.JAR on Windows

Copy the OS4690.Zip from your 4690 controller to the directory where you installed the toolkit under the directory `<toolkit install directory>\ace\ppctrl\lib`. The zip file resides on `C:\OS4690S\JAVA\LIB` of your 4690 controller.

Copy COMM4690.JAR from your 4690 controller to the directory where you installed the toolkit under the directory`<toolkit install directory>\ace\ppctrl\lib`. The jar file resides on C:\JAVA of your 4690 controller.

Building and executing PPCTRL (PIN Pad Controller) using Apache Ant

Apache Ant uses build.xml to build the PIN Pad Controller jar file (`PPCTRL.JAR`). Do the following:

1. You must rename the build.tk file to build.xml to replace what comes with the toolkit. Next, the build.xml file compiles the Java programs and creates a `PPCTRL.JAR` file in the **`<toolkit install directory>\ace\ppctrl\bin\jar`** directory where you installed the toolkit.
2. To compile, type the command ant jar from the **`<toolkit install directory>\ace\ppctrl`** directory.
3. To compile and execute, type the command ant run from **`<toolkit install directory>\ace\ppctrl`** directory.

First, The build.xml initially installed with the package is used by Toshiba developers. . XXXXXXXXX

XXXX

Executing PPCTRL (PIN Pad Controller) using Remote GUI or POSBC

If you are using Remote GUI on Windows to run PPCTRL (PIN Pad Controller), you will need to update the PPINSWINS.ZIP with the updated `PPCTRL.JAR` file and rerun the `PPINSWIN.BAT` file.

Building the 4690 Installation

To create a 4690 Installation Package with the new `ppctrl.jar`, replace the `ppctrl.jar` in the directory **`<toolkit install directory>\javadist\`** before running the `preppkg.cmd`.

Starting Simulation of Terminals on Windows

You can simulate one or more terminal sessions at a time. This allows you to debug functions like terminal transfer and terminal monitor in the Windows environment.

Each terminal session is started separately from the command line. This is the syntax of the command to start a session:

```
terminal_application [terminal_number]
```

You can specify any terminal application name that is supported by SurePOS ACE. If you do not specify a terminal number, then the default terminal number, which is 1, is used.

Miscellaneous Programs and Utilities

This section describes these programs and utilities:

- "Using ACEPERSL to Maintain Personalization Options" on page 572
- "Using DIF to Maintain Personalization Options" on page 586
- "Using ACECNVDL to Maintain Descriptors" on page 590
- "Using ACECOPYR to Modify File Keys" on page 593
- "Using ARC to Compile Application Resource Files" on page 594
- "Using ACEKFRBL to Resize the Customer Activity File" on page 598
- "Command Line Options for the Selective Item Report" on page 598
- "Using ACEUBATP for Batch Maintenance of Keyed Files" on page 599
- "Using ACEWERBL to Convert WIC EBT Files" on page 606
- "Using ACERECLN to Clean Up and Create Reprint Receipt Files" on page 607
- "Using ACEREPRN to Search Reprint Receipt Files" on page 608

Using ACEPERSL to Maintain Personalization Options

The Personalization module is instantiated as a user interface from the SurePOS ACE Main Menu but you can also instantiate it from the command line, using the module name ACEPERSL. When instantiated as a user interface, you must have an operator ID and password to use it. When instantiated from the command line, all parameters except /A and /G require an operator ID and password. The *SurePOS ACE: Planning and Installation Guide* describes what personalization is, how an operator can perform it through the user interface, and the meaning of each of its hundreds of options.

Relationship between Personalization INI and DAT Files

There is a circular relationship among personalization source INI files, personalization data (DAT) files, the Personalization user interface, and the ACEPERSL command. For example, if the eamoptns.dat does not exist, SurePOS ACE creates the file at the first invocation of ACEPERSL. SurePOS ACE uses the default values in `optnparm.ini` to build the eamoptns.dat file.

Whenever an operator uses the Personalization user interface and saves changed option values, ACEPERSL saves the changes only to eamoptns.dat and the other options DAT files. It does not update `optnparm.ini` or the other options INI files, which makes those INI files backlevel. Nothing updates the `optnparm.ini` file or the other options INI files; they are an intact record of default option values shipped by Toshiba Global Commerce Solutions. All current level changes are maintained by the system in eamoptns.dat and the other data files.

Not only can ACEPERSL create eamoptns.dat from `optnparm.ini`, but it can also create a personalization INI file from eamoptns.dat with the `/S` option. The INI file that it creates contains the most recent changes from eamoptns.dat. The newly created INI file does not affect the default option values in `optnparm.ini`.

ACEPERSL supports the use of tiered overlay files to maintain INI files. Whenever ACEPERSL rebuilds eamoptns.dat, it applies any data in the overlay files to `optnparm.ini`. See “Tiered Overlay Files” on page 460 for more information about how SurePOS ACE applies user delta INI files.

Table 124. Options INI Files and their Companion DAT Files

Options INI File	Delta INI File	Options DAT File	Description
`OPTNPARM.INI`	`EAMOPTDL.INI`	EAMOPTNS.DAT	SurePOS ACE base personalization options
`NLSPARMS.INI`	`NLSOPTDL.INI`	NLSOPTNS.DAT	National language support (NLS) personalization options
`APSPARMS.INI`	`APSOPTDL.INI`	APSOPTNS.DAT	SurePOS ACE EPS personalization options
`ACECPNTR.INI`	Not applicable	ACECATE5.DAT	Coupon translation options

Authorizing Operators to Perform Personalization

There is a significantly different type of circular relationship among the files and interfaces involved in authorizing operators to perform personalization. Although it is a similar scenario on the surface, with INI files, DAT files, the Personalization user interface, and the ACEPERSL command, there is a completely different system for generating and synchronizing the authorization data. As it was earlier, the new installation scenario is useful for showing the relationships.

Using authorization to perform base personalization as an example, if the operator authorization data file aceauthz.dat does not exist, then the first thing that happens is that ACEPERSL creates the file. So far, this scenario seems similar to that for personalization values. The major difference, however, is that *there is no INI file from which ACEPERSL can create the ACEAUTHZ.DAT file*; ACEPERSL simply authorizes all nine operator levels to change each personalization option value and creates the aceauthz.dat file accordingly.

Assume an operator uses the Personalization F9 user interface and saves changed authorization values. ACEPERSL saves the changes to the aceauthz.dat file only. It does not update an INI file because there is no INI file (yet).

Suppose that you intend to implement an intricate plan of operator authorization. So far, the only interface to changing authorization data is through the user interface, which requires you to press F9 at each field and specify a Boolean value for each of nine authorization levels for that field. If your authorization plan requires you to set authorization levels for 300 options and you must change only 4 of the 9 level flags, you still have a daunting task ahead of you to change 1200 flag settings.

ACEPERSL lets you automate the task by dumping a *grant* file of existing authorization values that you can edit and reapply. To apply your changed grant file to the aceauthz.dat file, you simply issue the ACEPERSL command with the /A parameter. The grant file can then be batch processed and applied to aceauthz.dat.

Table 125. Files Involved in Authorizing Operators to Personalize

Default Options Authorization Data File	Default Grant File Name	Options File Reference	Grant File Reference
ACEAUTHZ.DAT	`OPTGRANT.INI`	"OPTNPARM (Options)" on page 534	"OPTGRANT (Operator Options Authorization Grant)" on page 533
NLSAUTHZ.DAT	`NLSGRANT.INI`	"NLSPARMS (NLS Options)" on page 533	"NLSGRANT (NLS Operator Options Authorization Grant)" on page 533
CPNAUTHZ.DAT	`CPNGRANT.INI`	"ACECPNTR (Coupon Translation Options)" on page 474	"CPNGRANT (Coupon Translation Operator Options Authorization Grant)" on page 529
APSAUTHZ.DAT	`APSGRANT.INI`	"APSPARMS (SurePOS ACE EPS Personalization Options)" on page 529	"APSGRANT (SurePOS ACE EPS Operator Options Authorization Grant)" on page 529

ACEPERSL, the Personalization Module

To efficiently maintain option values and authorization values, you must use ACEPERSL to generate INI files from DAT files, to generate DAT files from INI files, and to generate HPP include files.

The ACEPERSL program has this syntax:
Personalization Program Syntax
ACEPERSL *userID password* [{ /A *additionalParameter* /C /G | /H *additionalParameter* | /O *additionalParameter* | /I *additionalParameter* /P | /T *additionalParameter* /TG *additionalParameter* /D | /L *additionalParameter* /LG *additionalParameter* | /N *additionalParameter* /S | /U *additionalParameter* | /XSD *additionalParameter* }]

ACEPERSL accepts these command line parameters:

/A (or -a)

Reads operator options authorization values in the grant-type INI file identified by the additional parameter and updates the appropriate DAT file.

/C (or -c)

Specifies that the terminal-specific or terminal group options file may be created, if it does not exist when used in conjunction with the /T or /TG, and /U parameters.

/D (or -d)

Dumps options to text files that are specified by the `dumpFile` parameter in the `FileNames` section in each options INI file.

/G (or -g)

Creates all grant INI files that have current operator options authorization values.

/H (or -h)

Creates an options include file using the name specified by the `hppFile` field in the `FileNames` section of the options INI file that is identified by the additional parameter.

/I (or -i)

Replaces all options values in eamoptns.dat with values in a user INI file identified in the additional parameter.

/L (or -l)

Loads options in all terminals. Optionally, specifies the terminal or terminals for which options are to be loaded.

/LG (or -lg)

Specifies the terminal groups for which options are to be loaded

/N (or -n)

Replaces individual options values in eamoptns.dat with values from an INI file that is identified by the additional parameter. The /N functionality is almost the same as /U, the difference being options won't be saved to disk if there is an error. The options changes are only put into effect if all of the options changes are successful. When preceding with /T or /TG, the atomicity of this new option is on a terminal/terminal group basis. If a terminal number range is specified and a specific terminal number options file can't be updated, for example, because the target option file doesn't exist, the loop will continue through all the range, reporting the results for each terminal. If compound errors are found when performing an atomic update, ACEPERSL's return code will be associated with the last error encountered while performing the update. There will be no precedence given to any error code over another error code when performing an atomic update.

/O (or -o)

Generates an output options data (DAT) file from the INI file specified by the /I or /U parameter.

If this parameter is used, it must precede the /I or /U parameter.

/P (or -p)

Puts ACEPERSL in protected mode, which makes protected functions available for use.

/S (or -s)

Saves the current DAT file settings to an INI file. For a terminal-specific or terminal group options file, the INI file contains both structure information and the unique options for the terminal.

/T (or -t)

Specifies the terminal or terminals for which options are to be dumped, updated, loaded, or saved. For the dump, update, and save commands, if the options file for a specified terminal does not exist, then an error message is displayed and processing continues with the options file for the next terminal that is specified.

If this parameter is used, it must be specified before the /D, /L, /S, or /U parameter on the command line.

/TG (or -tg)

Specifies the terminal group or terminal groups for which options are to be dumped, updated, loaded, or saved. For the dump, update, and save commands, if the options file for a specified terminal group does not exist, then an error message is displayed and processing continues with the options file for the next terminal group that is specified.

If this parameter is used, it must be specified before the /D, /L, /S, or /U parameter on the command line.

/U (or -u)

Replaces individual options values in eamoptns.dat with values from an INI file that is identified by the additional parameter.

If the /T parameter precedes this parameter, then option values in eamop*xxx*.dat, where *xxx* is the terminal number, are replaced. If the /TG parameter precedes this parameter, then option values in eamopg*xx*.dat, where *xx* is the terminal group number, are replaced. The base options file (eamoptns.dat) should exist before you update a terminal-specific options file.

/XSD (or -XSD)

Generates an XSD schema which defines the messaging formats used for XML Personalization Management. The first parameter is the output file and the remaining parameters are the parameter files that are included in the formats. The parameter files should appear in the following order to generate the schema properly: optnparm.ini apsparms.ini nlsparms.ini acecpntr.ini. For example, acepersl /xsdacepersl.xsd optnparm.ini apsparms.ini nlsparms.ini acecpntr.ini.

These are the detailed descriptions of the parameters:

/AupdateFromThisGrantFile

Reads authorization values from a grant INI file and writes them to the default options authorization DAT file identified in the `FileNames` section of the grant INI file. Any data that ACEPERSL writes to the authorization DAT file overwrites data in the DAT file and appears in the user interface. Entries in your INI file must be laid out like those in grant files mentioned in the description of the /G parameter.

For example:

```
acepersl /Ac:\ADX_IPGM\OPTGRANT.INI
```

This command reads the `optgrant.ini` file, which is a grant file of the base options file that was generated with the /G parameter in a previous command. ACEPERSL then writes any specified options in `optgrant.ini` to aceauthz.dat, replacing just the values specified in `optgrant.ini`.

/C

Specifies that the terminal-specific options file may be created, if it does not exist when used in conjunction with the /T or /TG, and /U parameters.

/D

Creates dump text files specified by the `dumpFile` parameter in the `FileNames` section in each options INI file:

OPTNPARM.INI

EAMOPTNS.TXT

NLSPARMS.INI

NLSOPTNS.TXT

ACECPNTR.INI

ACECATE5.TXT

APSPARMS.INI

APSOPTNS.TXT

acecate5.txt is stored in the adx_idt1 directory. The other dump files are stored in the adx_ipgm directory.

If this parameter is specified after the /T parameter on the command line, then the dump file will contain all options from the eamoptns.dat file. For options that are specifically included in the terminal-specific options file (eamop*xxx*.dat), the dump file will contain the terminal-specific value. For options that are specifically included in the terminal group options file (eamopgxx.dat), the dump file will contain the terminal group unique value.

/G

Creates each grant file specified on the `grantFile` attribute of the `FileNames` section in each options INI file. A grant file contains an entry for each option specified in the companion DAT file. (Companion DAT files are identified in Table 124.)

To illustrate how this works, if you issue the command `ACEPERSL /G`, then ACEPERSL creates all grant files.

/H*updateUsingDataFromThisINIFile*

Creates an options include file using the name specified by the `hppFile` field in the `FileNames` section of the options INI file that is identified by the additional parameter. INCOPTNS.HPP is the default.

/I*updateUserDeltaINIFileSpec*

You can cause SurePOS ACE to use option values in an INI file of your own instead of those in `optnparm.ini`. There are two parameters that let you instantiate values from your own file, the /U (update) option and the /I (initialize) option.

When changing only some option values, use the /U option. When replacing all option values, use the /I option.

To initialize all option values, create an INI file that *must* specify values for all personalization options because it is a replacement for option values in optnparm.ini, which SurePOS ACE ignores. To cause SurePOS ACE to use your INI file as this replacement, start ACEPERSL with the /I option and your INI file specification. Specify the fully-qualified file name after the /I without any spaces between the two. For example:

```
acepersl 99999999 99999999 /Ic:\allopts.ini /G
```

Operator 99999999 issues this command to replace aceauthz.dat with the values in `allopts.ini`.

Another method for automatically including your INI file values is to name the file one of the three reserved names, `optnparm.us7`, `optnparm.us8` or `optnparm.us9`. SurePOS ACE automatically includes INI files with US0 through US9 extensions as described in "Tiered Overlay Files" on page 460.

/L*terminalNumber(s)*

Loads eamoptns.dat values in all terminals. Optionally, specifies the terminal or terminals for which options are to be loaded.

The additional parameter for this /L parameter must contain terminal numbers in the range 1-999. Leading zeroes are ignored (for example: 001 is processed the same as 1). The terminal numbers can be specified with any of the following formats:

- One terminal number.
- Multiple terminal numbers separated by commas and without spaces (for example: 1,3).
- A range of terminal numbers with the first number less than the second. The beginning and ending numbers are separated by a hyphen and the range contains no spaces. For example, 2-35 for terminals 2 through 35 is allowed; 35-2 for the same terminals is not valid.

/LG*groupNumber(s)*

Specifies the terminal group or terminal groups for which options are to be loaded. The additional parameter for this /LG parameter must contain group numbers in the range 1-99. Leading zeroes are ignored (for example: 01 is processed the same as 1). The group numbers can be specified with any of the following formats:

- One group number.
- Multiple group numbers separated by commas and without spaces (for example: 1,3).
- A range of group numbers with the first number less than the second. The beginning and ending numbers are separated by a hyphen and the range contains no spaces. For example, 2-35 for groups 2 through 35 is allowed; 35-2 for the same groups is not valid.

/M

Used during migration to sync up the tender shadow list with the main list, that is, Tender2 ID definitions should match Tender ID definitions. Currently Tender2 is the only shadow list base ACE supports. IDs can get out of sync if users update/delete only from Tender, but not Tender2. Updates via the /U option to add/delete new tender IDs, should be in tandem with updates to the Tender2 shadow list.

/P

Puts acepersl in protected mode, which makes some functions visible (such as Generate HPP).

To make new options that you have created into part of the Personalization user interface code, start ACEPERSL with the /P option. When the Personalization menu appears, select File and Generate HPP. Replace the current HPP files with the HPP files that were just generated. When you restart personalization, your new options will appear.

/O*filespec-of-output-file*

The name of the output options DAT file that overrides the output options file name specified in the INI file. If used, this parameter must precede the /I or /U parameter.

/S

Saves all current option settings in DAT files (including those made from the user interface) to INI files that are specified by each `saveINIFile` parameter in the INI files identified in Table 124. By default, the `saveINIFile` settings identify files that do not exist so that you do not overlay the original default values.

If this parameter is specified after the /Tparameter on the command line, then the INI file will contain only those options that are specifically included in the terminal-specific options file. The INI file name will be `eamop`*`xxx`*`.ini`, where *xxx* is the terminal number (padded with leading zeroes, if necessary).

If this parameter is specified after the /TG parameter on the command line, then the INI file will contain only those options that are specifically included in the terminal group options file. The INI file name will be `eamopgxxxx.ini`, where *xx* is the group number (padded with leading zeroes, if necessary).

/T*terminalNumber(s)*

Specifies the terminals for which the terminal options are to be dumped, updated, loaded, or saved.

- For dumped options, the output file is named eamop*xxx*.txt, where *xxx* is the terminal number that was specified on the command.
- For saved options, the output file is named `eamopxxx.ini`, where *xxx* is the terminal number that was specified on the command.

The additional parameter for this /T parameter must contain terminal numbers in the range 1-999. Leading zeroes are ignored (for example: 001 is processed the same as 1). The terminal numbers can be specified with any of the following formats:

- One terminal number.
- Multiple terminal numbers separated by commas and without spaces (for example: 1,3).
- A range of terminal numbers with the first number less than the second. The beginning and ending numbers are separated by a hyphen and the range contains no spaces. For example, 2-35 for terminals 2 through 35 is allowed; 35-2 for the same terminals is not valid.

/TGgroupNumber(s)

Specifies the terminal groups for which the group options are to be dumped, updated, loaded, or saved.

- For dumped options, the output file is named eamopg*xx*.txt, where *xx* is the terminal group number that was specified on the command.
- For saved options, the output file is named `eamopgxx.ini`, where *xx* is the terminal group number that was specified on the command.

The additional parameter for this /TG parameter must contain terminal group numbers in the range 1-99. Leading zeroes are ignored (for example: 01 is processed the same as 1). The terminal group numbers can be specified with any of the following formats:

- One terminal number.
- Multiple terminal group numbers separated by commas and without spaces (for example: 1,3).
- A range of terminal group numbers with the first number less than the second. The beginning and ending numbers are separated by a hyphen and the range contains no spaces. For example, 2-35 for terminal groups 2 through 35 is allowed; 35-2 for the same terminal groups is not valid.

/U*updateUserDeltaINIFileSpec*

Appends options values in an INI file that you create to default options values. Entries in your INI file must be laid out like entries in the options INI files. If you add and/or delete a Tender list element, then the corresponding change should also be made to the Tender2 list which is considered a shadow list. The shadow list provides additional field definitions for the original list. Currently Tenders is the only list that has a shadow list.

After creating your INI file, start acepersl with the /U option and your INI file specification to update the option. Specify the fully-qualified file name after the U without any spaces between the two. For example:

```
acepersl 99999999 99999999 /Uc:\sngleopt.ini
```

Operator 99999999 issues this command to overwrite values in eamoptns.dat with those in `sngleopt.ini`.

Another method for automatically including values from your INI files is to name such a file using one of the three reserved names, `optnparm.us7`, `optnparm.us8` or `optnparm.us9`. SurePOS ACE automatically includes INI files with US0 through US9 extensions, as described in "Tiered Overlay Files" on page 460, when it is generating DAT files for the first time (or regenerating them because of a /U or /I parameter).

If this parameter is specified after the /T parameter on the command line, then the options values in the file specified by updateUserDeltaINIFileSpec will be applied to the specified terminal-specific options files. If this parameter is specified after the /TG parameter on the command line, then the options values in the file specified by updateUserDeltaINIFileSpec will be applied to the specified terminal group options files.

(none)

Without a parameter, ACEPERSL starts by reading logical name <OPTNPARM>, which is by default the `optnparm.ini` file. ACEPERSL uses the `FileNames` section in `optnparm.ini` to determine the DAT file from which to read the current option values. It also uses the `LinkedINIandOptionFiles` section to do the same for linked files, such as those for NLS options, SurePOS ACE EPS options, and coupon translation options.

Return Codes from ACEPERSL

These are the return codes that are sent from the ACEPERSL application:

Return Code	Meaning
0	No errors found.
1	Not used.
2	Not used.
3	Error writing to target file.
4	Error opening source file.
5	Incompatible command line options were used.
6	Invalid terminal(s) or terminal group(s).
7	Error opening base INI file.
8	INI file does not have a root file.
9	Error(s) found while parsing option INI file. At least one option was updated.
10	Another instance of ACEPERSL is running.
11	An invalid option was used.
12	Error(s) found while parsing option INI file. No options were updated.
13	Error reading options from options file.

Examples of ACEPERSL /U

This section contains examples of the following activities that you can perform with the ACEPERSL /U command:

- Modifying an option
- Adding an item to a drop-down list
- Deleting an option from a terminal-specific options file
- Deleting an option from a list ID in a terminal-specific options file
- Deleting a list ID from a terminal-specific options file
- Adding or updating options in a terminal-specific options file for list IDs that exist in the base options file

Modifying an Option

Figure 19 shows an INI file that modifies the exchange rate for a foreign currency.

```
[INIControl]
File         = OPT, FileNames, File0

[FileNames]
OptionsFile       = EAMOPTNS
StoreReadFile     = <$AMOPTNS>
StoreWriteFile    = <EAMOPTNS>
TerminalReadFile  = <$AMO:>
TerminalWriteFile = <EAMO:>
hppFile           = INCOPTNS.HPP

[File0]
ForeignCurrency            = 40, Record

[ForeignCurrency]
ExchangeRateValue          = 3, ULong, 801
```

Figure 19. Modifying One Option with an INI File

The `INIControl` section and the `FileNames` section define the database file where options are stored. These same lines appear in `optnparm.ini`. The entry after [File0] indicates which section is to be modified (`ForeignCurrency`). The entry after [ForeignCurrency] indicates the variable to be changed (`ExchangeRateValue`), and its new value (`801`). The other parameters on the `ExchangeRateValue` statement are *reserved* and cannot be changed.

Adding an Item to a Drop-down List

Figure 20 shows an INI file that adds two items to the list of user-defined messages in Loyalty -> Messages -> User-Defined personalization.

```
[INIControl]
File         = OPT, FileNames, File0

[FileNames]
OptionsFile       = EAMOPTNS
StoreReadFile     = <$AMOPTNS>
StoreWriteFile    = <EAMOPTNS>
TerminalReadFile  = <$AMO:>
TerminalWriteFile = <EAMO:>
hppFile           = INCOPTNS.HPP

[File0]
LoyaltyMessages = 79, RECORD
```

```
[Control]
OptionsChainFileName = 1, STRING, 12, ""

[LoyaltyMessages]
List = 650, SECTIONS, AddOne, AddTwo

[LoyaltyMessages.List.AddOne]
Id = 100
DeleteFlag = 0
LoyaltyTypeFlash = 0
LoyaltyTypeTone = 0
LoyaltyTypeReusable = 1
LoyaltyMinimumTime = 0
LoyaltyContinuation = 0
LoyaltyDisplay = "New Message Added for ID 100"
LoyaltyPrint1 = "Loyalty Print Line 1 for New 100"
LoyaltyPrint2 = "Loyalty Print Line 2 for New 100"
LoyaltyPrint3 = "Loyalty Print Line 3 for New 100"
LoyaltyPrintFeeds = 1, 2, 0
LoyaltyPrintFontId = 0, 0, 0

[LoyaltyMessages.List.AddTwo]
Id = 101
DeleteFlag = 0
LoyaltyTypeFlash = 0
LoyaltyTypeTone = 0
LoyaltyTypeReusable = 1
LoyaltyMinimumTime = 0
LoyaltyContinuation = 0
LoyaltyDisplay = "New Message Added for ID 101"
LoyaltyPrint1 = "Loyalty Print Line 1 for New 101"
LoyaltyPrint2 = "Loyalty Print Line 2 for New 101"
LoyaltyPrint3 = "Loyalty Print Line 3 for New 101"
LoyaltyPrintFeeds = 1, 2, 0
LoyaltyPrintFontId = 0, 0, 0
```

Figure 20. Adding Entries to a List with an INI File

Note: The `INIControl` section and the `FileNames` section define the database file where options are stored. These same lines appear in `optnparm.ini`.

The entry after [File0] indicates which section is to be modified (`LoyaltyMessages`).

The entry after `[LoyaltyMessages]` indicates the messages will be in sections (`SECTIONS`) labeled `[LoyaltyMessages.List.xxx]`, where `xxx` is `AddOne` for the first section and `AddTwo` for the second section.

The lines after `[LoyaltyMessages.List.AddOne]` and `[LoyaltyMessages.List.AddTwo]` define the messages that are being added.

`DeleteFlag = 0` is required if the `Id` previously existed, was deleted, and is now being added again.

Deleting an Option from a Terminal-specific Options File

To delete an option from a terminal-specific options file, you update the options file with an INI file in which the option to be deleted has a type of *DELETE* instead of its usual type. For example, the following update would delete the *Store.Name* option:

```
[Store]
Name = 1, DELETE
```

If the option that you are trying to delete does not exist in the terminal options file, then an error message will be issued and processing will continue with the next option.

Deleting an Option from a List ID in a Terminal-specific Options File

To delete an option from a list ID in a terminal-specific options file, use syntax similar to that for deleting a single option. The DELETE parameter with a comma separator must be appended to each list ID option to be deleted. For example, the following update would delete the *StoreDepartments.List.1.Name* and *StoreDepartments.List.1.DeptNum* options.

```
[StoreDepartments]
List = 99, Sections, 1

[StoreDepartments.List]
Id = 1, UInt
Name = 2, String, 15, ""
DeptNum = 3, UInt, 1

StoreDepartments.List.1
Id = 4
Name = "Deli", DELETE
DeptNum = 5, DELETE
```

Deleting a List ID from a Terminal-specific Options File

To delete an entire list ID from a terminal-specific options file, you specify a value of *Y* or *1* for the DeleteFlag variable (the same syntax as you use to update base options). For example, the following update would delete ID 4 from *StoreDepartments.List.1*:

```
StoreDepartments.List.1
Id=4
DeleteFlag = Y
```

If you want to reverse the deletion of a list ID without overriding the base option, use an update like the following:

```
StoreDepartments.List.1
Id=4
DeleteFlag = Y, DELETE
```

You cannot mark delete on all the entries that are specific to a terminal because the terminal specific options file would be in an invalid empty state. If all the terminal specific options need to be deleted, perform one of the following:

1. Erase the ADX_IPGM/EAMOPxxx file for the terminal.
2. Use the option "Erase terminal" in the Personalization User Interface to erase the terminal.

Adding or Updating Options in a Terminal-specific Options File for Existing List IDs

Adding or updating options for list IDs works similar to that of normal options for those list IDs that already exist in the base options file. For example, in the following code sample, the option field *Name* for the 4th list ID of the *StoreDepartments* list has its value changed to *Deli*.

```
[StoreDepartments]
List = 99, Sections, 1

[StoreDepartments.List]
Id = 1, UInt
Name = 2, String, 15, ""
DeptNum = 3, UInt, 1
```

```
StoreDepartments.List.1
Id = 4
Name = "Deli"
```

If the ID that is specified in the update file does not exist in the base options file, then all option fields for that ID will be created in the terminal-specific options file. Options that are not explicitly given values will use their default values. If you use the preceding example, the option fields *Name* and *DeptNum* will be created in the terminal-specific options file for the 4th list ID of the StoreDepartments list. The *DeptNum* field will be initialized with the value of 1, while the *Name* will be given the value of Deli.

Adding New Personalization Options Using a Tiered Overlay File

You can add personalization options to SurePOS ACE without having to write code to extend objects. To add options, create user delta files (`optnparm.us7`, optnparm.us8 or `optnparm.us9`) as described in "Tiered Overlay Files" on page 460.

Adding a Set of Options in a New Personalization Panel

You can use `optnparm.us7` through `optnparm.us9` to add an entire new panel of options to the personalization user interface. This example adds new options for printing duplicate receipts.

Adding a New Section of Options in OPTNPARM.INI

The `File0` section of `optnparm.ini` defines the organization of options in the file by identifying sections where they are defined. Among the sections it specifies are 10 that are reserved for your use. They are `CustomerRecord1` through `CustomerRecord10`.

```
[File0]
 .
 .
 .
CustomerRecord1            = 66, Record
CustomerRecord2            = 67, Record
CustomerRecord3            = 68, Record
CustomerRecord4            = 69, Record
CustomerRecord5            = 70, Record
CustomerRecord6            = 71, Record
CustomerRecord7            = 72, Record
CustomerRecord8            = 73, Record
CustomerRecord9            = 74, Record
CustomerRecord10           = 75, Record
```

Figure 21. Defining a New Options Section in OPTNPARM.INI

Defining Storage Variables for Options on the New Page

To define storage areas (variables) for new options and to define their default values, you can add entries to `optnparm.us` through `optnparm.us9` in the `CustomerRecord1` section.

```
[CustomerRecord1]
Duplicates                       = 1, UInt,         0
AutoPrintDup                     = 2, Boolean,      1
```

Figure 22. Adding Variables for the New Options

Adding the New Options Page to a Pull-Down Menu

To create a new pull-down menu on the Personalization menu, you add the menu to the `MainMenu` section. For example, to create a pull-down menu named `CustomerMenu`, you would add the `CustomerMenu` entry to the `MainMenu` section.

After creating the CustomerMenu entry on the Personalization menu, you must decide where to add a new notebook page by locating it under the CustomerMenu menu item in `optnparm.us9`. In this example, Receipt is added as one option that appears under the CustomerMenu section and identifies the `CustomerRecord1` data section. The &tilde;R&tilde; highlights the R on `Receipt` to make it the selection character for the hot key.

```
[CustomerMenu]
~R~eceipt          = Dialog, 1, 1, 79, 23, 0, 0, 42, \
                                        CustomerRecord1
```

Figure 23. Adding an Entry to a Personalization Menu

Defining a Screen Layout for New Options

You can define a screen layout by specifying lines for a new section header (`CustomerMenu.Receipt`).

```
;************************************************************
;
;  RECEIPT DIALOG
;
;************************************************************

[CustomerMenu.Receipt]

A.Ctrl01 = StaticText, 1,5,44, 0,Number of Duplicate Copies ....:
A.Ctrl02 = StaticText, 1,7,44, 0,Automatic Duplicate Print .....:

A.Ctrl06 = InputLine,46,5,5,  0,0,0,  CustomerRecord1,Duplicates
A.Ctrl07 = CheckBox, 46,7,5,  0,0,0,  CustomerRecord1,AutoPrintDup
```

Figure 24. Defining the Layout of a New Personalization Screen Panel

Generating a New HPP File for Personalization

To generate a new HPP file, you must start ACEPERSL with the /P option and choose Generate HPP from the File pulldown, or you can run ACEPERSL with the /H option.

After generating the new HPP file, replace the current INCOPTNS.HPP file with the new one.

Operator Options Authorization User INI Files

User delta INI files for operator options authorization are different than those for personalization.

To ensure that you have migrated all operator initiated maintenance performed through the user interface, use the /G parameter of the ACEPERSL command as described in "ACEPERSL, the Personalization Module" on page 574 to create these grant files:

- "OPTGRANT (Operator Options Authorization Grant)" on page 533
- "NLSGRANT (NLS Operator Options Authorization Grant)" on page 533
- "CPNGRANT (Coupon Translation Operator Options Authorization Grant)" on page 529
- "APSGRANT (SurePOS ACE EPS Operator Options Authorization Grant)" on page 529

Edit a grant file to make the changes you intend. (You can coy and rename the file if you like.) You must leave in and not change the `INIControl` and `Files` sections. You can delete any other values that you are not changing and have a file of changes only. After making your changes to existing values, use the /A parameter of the ACEPERSL command as described in "ACEPERSL, the Personalization Module" on page 574 to append values in the edited grant file to the DAT file.

Repeat this procedure for other grant files, if necessary.

If you are editing grant files because you intend to add new menus or options to personalization, you must wait until you have actually added those options as described in "Adding New Personalization Options Using a Tiered Overlay File" on page 584. After an option appears in the user interface, it has the default authorization of allowing all operator levels to change it. Once an option appears in the user interface, it will also appear when you create a grant file, where you can edit it.

Using DIF to Maintain Personalization Options

Store options can be managed from the store enterprise level by using DIF. The stores can send XML messages which are received by DIF. In return, DIF will interface with ACE and execute ACEPERSL actions based on the XML message received. Communications between ACE and the store enterprise should conform to a generated schema. See Appendix J, Personalization Management via DIF API on page 719.

Overview of Function

The following functionality is provided:

- Generate a schema based on parameters files
- Allow the enterprise to query the values of all options
- Send an event from ACE to the enterprise whenever an option is updated
- Allow the enterprise to update options
- Allow the enterprise to indicate that terminals should load options

Schema Generation

ACEPERSL constructs a schema which describes an XML message set by reading parameter files and overlay files. The schema can be used for options management. To generate a schema, use acepersl /xsdOutputFile parametersFile1 ... parametersFileN where the parameter files are listed in the following order (optnparm.ini apsparms.ini nlsparms.ini acecpntr.ini).

An example of generating a schema is:

```
acepersl /xsdacepersl.xsd optnparm.ini apsparms.ini nlsparms.ini acecpntr.ini
```

Any overlay files will be automatically included by ACEPERSL as long as they reside in the same directory as the base parameters file.

The schema will perform range validation based on the input validators in the parameters file. XML standards do not allow element names to start with a number. Any sections or options which start with a number will have an "n" prepended to them.

This solution will not use the schema for validation in production for performance reasons. The schema will be shipped in ADX_IPGM:ACEPERSL.XSD. It is mainly provided as a tool to use during testing.

DIF is not used for schema generation.

Options Query

The store enterprise can send an XML message querying the status of all options. DIF will send the store enterprise an XML message response containing the current values of all options. This includes global store options, terminal/group specific options, EPS options, NLS options, and coupon translation options.

See Appendix J, Personalization Management via DIF API on page 719 for examples of the message format.

Options Changed Event

Any time an options file is saved to disk, ACE will send an OptionsChangedEvent to the retailer via DIF. The event will be sent regardless of how the options file is updated: GUI, command line, or XML.

See Appendix J, Personalization Management via DIF API on page 719 for examples of the message format.

Update Options Request

The store enterprise can send a message to update any option from the following four options files: EAMOPTNS, APSOPTNS, NLSOPTNS, and ACECATE5. Terminal/group specific options may be updated, but only one file may be updated per request. If all options are updated successfully, the changes are saved to disk and an OptionsChangedEvent will be generated containing details of all changed options in XML format. If there is an error updating an option, no options changes are saved to disk. DIF will send the store enterprise an UpdateOptionsResult indicating whether the update occurred successfully.

See Appendix J, Personalization Management via DIF API on page 719 for examples of the message format.

Loading Terminal Options

The store enterprise can send a message to cause terminals to load options. All terminals, specific terminals, or specific terminal groups may be told to load options. The DIF actor will receive the request and run ACEPERSL /L or ACEPERSL /LG (for terminal groups) with the appropriate parameters.

DIF will use the return code from ACEPERSL /L (or ACEPERSL /LG) to construct a reply.

See Appendix J, Personalization Management via DIF API on page 719 for examples of the message format.

XML Personalization Management Setup and Personalization

The setup of this solution involves configuring both ACE and DIF. New personalization options have been added to ACE to enable certain parts of the solution. Changes have been made to the acedif.pro configuration file that is distributed with ACE. Additional changes are necessary to the customizable file difuser.pro.

Schema Generation Setup

No DIF configuration is necessary.

Options Changed Personalization Setup

New entries have been added to the Misc->Store Integrator-> Events and EventCom pages in ACE Personalization. These options are modeled after the store close event and indicate the host and port that DIF is running on. "Enabled" indicates whether the event should be sent. It defaults to disabled; i.e., don't send the event. "Critical" indicates whether ACE should continue trying to send an OptionsChangedEvent indefinitely. Note that if "Critical" is enabled, ACE Personalization will be unavailable until the OptionsChangedEvent is sent successfully. This may prevent changes from being saved. The timeout, port, and host fields allow configuration of where and how the message is sent. The host field should specify the controller where DIF is running.

DIF Properties File Configuration

The acedif.pro file is included with the ACE installation. Acedif.pro contains the base configurations for options changed event, options query, update options request, and load terminal options. However, to use the acedif.pro configurations, the difuser.pro file must be modified (or created if it does not already exist).

Difuser.pro should be modified to configure basic microbroker/Websphere properties, as well as communications properties.

Difuser.pro should contain the following line to enable the new options support for queries, updates and loads:

```
nextFile=acedif
```

The following services should be added to the container.services property in difuser.pro as shown in the following example:

```
container.services = SoepsClientService ACEOptLoadService ACEOptUpdService
 ACEOptQryService
```

The maximum message size for DIF TCP/IP communication should be specified in difuser.pro so that large Options Changed Events and Options Query Results will be consumed by DIF with no error:

```
container.tcpip.message.length=5000000
```

Also, see the *Data Integration Facility: User's Guide* to specify the micro broker maximum message size as 5000 KB in difuser.pro so large messages can be handled without error.

Table 126 is a list of message flows used to configure DIF so that the request/result messages are handled by the corrrect DIF service or actor:

Table 126. Message flows

DIF V2R3 Queue/ Topic	DIF V3R1 Queue/ Topic	DIF Service/Actor	Message Type
RespOptnsLoad	LoadFromActor	ACEOptLoadService	<LoadOptionResult>
ReqOptnsLoad	LoadToActor	ACEOptLoadService	<LoadOptionsRequest>
RespOptnsUpd	UpdateFromActor	ACEOptUpdService	<UpdateOptionsResult>
ReqOptnsUpd	UpdateToActor	ACEOptUpdService	<UpdateOptionsRequest >
RespOptnsQry	QueryFromActor	ACEOptQryService	<OptionsQueryResult>
ReqOptnsQry	QueryToActor	ACEOptQryService	<OptionsQueryRequest>
OptnsChg	ChangeFromActor	ACEOptnsChangedActor	<OptionsChangedEvent >

For example, the RespOptnsLoad queue or topic is where the ACEOptLoadService will place <LoadOptionResult> messages. The ReqOptnsLoad queue or topic is where the ACEOptLoadService will receive <LoadOptionsRequest> messages.

The ACEOptUpdService will place <UpdateOptionsResult> messages in the RespOptnsUpd queue/topic and receive <UpdateOptionsRequest> messages via the ReqOptnsUpd queue/topic.

The ACEOptQryService will place <OptionsQueryResult> messages in the RespOptnsQry queue/topic and receive <OptionsQueryResult> messages via the ReqOptnsQry queue/topic.

The ACEOptnsChangedActor will place <OptionsChangedEvent> messages in the OptnsChg queue or topic.

If using DIF V3R1 or above, edit acedif.pro and comment out the section titled "DIF V2R3 ACE OPTIONS CONFIGURATION" by prepending all lines with a # symbol. Uncomment all lines under "DIF V3R1 ACE OPTIONS CONFIGURATION" by removing the first # symbol on each line.

If using DIF V3R1 or above, edit DIFUSER.XML, and integrate the following lines inside the JmsConfiguration beans tag:

```
<!-- ACE Services -->
  <!-- Options query -->
  <bean id="ACEOptQryService"
class="com.tgcs.retail.di.net.service.jms.spring.ServiceMsgInbound">
  </bean>
    <jms:listener-container
      container-type="default"
      destination-type="queue"
      connection-factory="connectionFactory"
      acknowledge="auto">
    <jms:listener destination="QueryToActor" ref="ACEOptQryService"/>
  </jms:listener-container>

  <!-- Options load -->
  <bean id="ACEOptLoadService"
class="com.tgcs.retail.di.net.service.jms.spring.ServiceMsgInbound">
  </bean>
    <jms:listener-container
      container-type="default"
      destination-type="queue"
      connection-factory="connectionFactory"
      acknowledge="auto">
    <jms:listener destination="LoadToActor" ref="ACEOptLoadService"/>
  </jms:listener-container>

    <!-- Options update -->
  <bean id="ACEOptUpdService"
class="com.tgcs.retail.di.net.service.jms.spring.ServiceMsgInbound">
  </bean>
    <jms:listener-container
      container-type="default"
      destination-type="queue"
      connection-factory="connectionFactory"
      acknowledge="auto">
    <jms:listener destination="UpdateToActor" ref="ACEOptUpdService"/>
  </jms:listener-container>
```

Using ACECNVDL to Maintain Descriptors

SurePOS ACE provides the ACECNVDL program to process INI files and migrate DAT files (from existing SurePOS ACE messages and descriptors you might have customized) to create new SurePOS ACE sales, department, and report descriptor data files.

ACECNVDL builds these DAT files:

- acesdesc.dat from:
  - `acesdesc.ini`
  - Toshiba-delta sales descriptors in `acesdesc.us0` through `acesdesc.us3`
  - Toshiba Global Commerce Solutions-Business Partner-delta sales descriptors in `acesdesc.us4` through `acesdesc.us6`
  - User delta sales descriptors in `acesdesc.us7`, `acesdesc.us8`, and `acesdesc.us9`
  - eamsdesc.dat
- aceddesc.dat from:
  - `aceddesc.ini`
  - Toshiba-delta sales descriptors in `aceddesc.us0` through `aceddesc.us3`
  - Toshiba Global Commerce Solutions-Business-Partner delta sales descriptors in `aceddesc.us4` through `aceddesc.us6`
  - User delta sales descriptors in `aceddesc.us7`, `aceddesc.us8`, and `aceddesc.us9`

Note: You can enter existing department descriptors in `eamrdesc.ini` into a user delta file.

- acerdesc.dat from:
  - `acerdesc.ini`
  - Toshiba-delta sales descriptors in `acerdesc.us0` through `acerdesc.us3`
  - Toshiba Global Commerce Solutions-Business-Partner delta sales descriptors in `acerdesc.us4` through `acerdesc.us6`
  - User delta sales descriptors in `acerdesc.us7`, `acerdesc.us8`, and `acerdesc.us9`
  - eamrdesc.dat

ACECNVDL also can build user delta INI files from these DAT files:

- acesdesc.dat
- aceddesc.dat
- acerdesc.dat

ACECNVDL Syntax

The ACECNVDL program has this syntax:
Descriptor Generation Program Syntax
ACECNVDL [{ -? | -d *inputDATFileName* -h | -i *sourceINIFileName* -n | -o *outputDATFileName* | -s *savedINIFileName* | -u *updateSavedINIFileName* }]

```
acecnvdl options
```

where *options* are:

-?

Displays help for the ACECNVDL command.

-d*filespec-of-default-input-descriptor-file*

Identifies the name of the default descriptor DAT file that is merged into the output DAT file. No specification uses eamsdesc.dat as the default input descriptor file.

-h

Creates a sales descriptor include file using the name identified by the `GenerateHPPName` field in the `Output` section of the `acesdesc.ini` file. SLSDSCSA.HPP is the default. Copy the resulting HPP file to the `INCLUDE` directory.

-i*filespec-of-input-INI-file*

Identifies the name of the input INI file that is used to supply values for the output DAT file. `acesdesc.ini` is the default.

-n

Causes ACECNVDL to use only descriptors provided in INI files to create the DAT file. It does not use the file identified by the -d parameter.

-o*filespec-of-output-file*

The name of the output DAT file. For example, acesdesc.dat is an output DAT file.

-s

Identifies the name of the temporary save file for saving descriptor DAT file values to INI files.

-ufilespec-of-update-saved-INI-file

Identifies the name of a user delta INI file that ACECNVDL applies when building the DAT file.

When using the ACECNVDL command, you can also use a slash (/) in addition to a hyphen (-) to identify a parameter. You can also type the parameter as either a lowercase or uppercase letter.

Examples of Using ACECNVDL

Saving Existing SurePOS ACE Descriptors (from a DAT File) to a Temporary INI File

To save existing SurePOS ACE descriptors (from a DAT file) to a temporary INI file that you can edit to create a user delta INI file, use this syntax:

```
acecnvdl -i<input INI filespec> -o<output DAT filespec> -s<save temp INI filespec>
```

Parameter meanings are:

-i	The input SurePOS ACE INI descriptor file. SurePOS ACE compares entries in this file to those in the output DAT file, to determine which entries in the DAT file are not in this INI file, and therefore, must be in the temporary INI file that it is creating.
-o	The input SurePOS ACE DAT descriptor file that SurePOS ACE uses as source to produce the temporary INI file.
-s	The temporary INI file to build from existing data in the DAT file.

To create a temporary INI file from existing DAT descriptor files, issue these commands:

- Creating a temporary INI file from the Sales Descriptors (DAT) file:

  ```
  acecnvdl -iacesdesc.ini -oacesdesc.dat -sacestemp.ini
  ```

- Creating a temporary INI file from the Report Descriptors (DAT) file:

  ```
  acecnvdl -iacerdesc.ini -oacerdesc.dat -sacertemp.ini
  ```

- Creating a temporary INI file from the Department Descriptors (DAT) file:

  ```
  acecnvdl -iaceddesc.ini -oaceddesc.dat -sacedtemp.ini
  ```

Applying Toshiba Global Commerce Solutions, Business Partner, and User Delta INI Files to Base INI Descriptor Files

To apply Toshiba Global Commerce Solutions, Business Partner, and your delta INI files to the base INI descriptor files as described in "Tiered Overlay Files" on page 460, use this syntax:

```
acecnvdl -i<input INI file> -o<output DAT file> -u<user delta INI file> -n
```

Parameter meanings are:

-i	The input SurePOS ACE INI descriptor file.
-o	The output SurePOS ACE DAT descriptor file.
-u	The user delta INI file to use as additional input for constructing the DAT file.
-n	Use only descriptors provided in SurePOS ACE INI files to create the DAT file.

To create a DAT file from more than one INI file, issue these commands:

- Creating a DAT file from the base Sales Descriptors INI file and a delta INI file:

  ```
  acecnvdl -iacesdesc.ini -oacesdesc.dat -uacestemp.ini -n
  ```

- Creating a DAT file from the base Report Descriptors INI file and a delta INI file:

  ```
  acecnvdl -iacerdesc.ini -oacerdesc.dat -uacertemp.ini -n
  ```

- Creating a DAT file from the base Department Descriptors INI file and a delta INI file:

  ```
  acecnvdl -iaceddesc.ini -oaceddesc.dat -uacedtemp.ini -n
  ```

Creating a Customized HPP File

To create a customized HPP from a customized Sales Descriptors INI file, use this syntax:

```
acecnvdl -i<custom INI file> -h
```

Parameter meanings are:

-i	Identifies the input Sales Descriptors INI file, which is a customized file.
-h	Builds the new HPP file using the name defined in the `Output` sections of the INI file.

To create a new HPP file that objects can include to use new sales descriptors after you have created the temporary INI file `acestemp.ini`, issue this command:

```
acecnvdl -iacestemp.ini -h
```

Using ACECOPYR to Modify File Keys

The ACECOPYR utility program allows you to modify the key in individual records in a file. Invoke the utility once for each record for which you want to modify the key.

The syntax of the utility is:

```
acecopyr keyedFileName -[Move|Copy|Exists]oldKey -[Add|Replace|Force]newKey -i
```

where:

Move (Shortcut = m)

Copies the record with the old key to a record with the new key, and deletes the old record.

Copy (Shortcut = c)

Copies the record with the old key to a record with the new key, and keeps the old record.

Exists (Shortcut=x)

Checks for the existence of a record with the old key. No data is changed. The second parameter (*newKey*) is not allowed.

Add (Shortcut = a)

Copies the record with the old key to a record with the new key only if a record with the new key does not already exist.

Replace (Shortcut = r)

Copies the record with the old key to a record with the new key only if a record with the new key already exists.

Force (Shortcut = f)

Copies the record with the old key to a record with the new key regardless of whether a record with the new key already exists. Use this when you want to create the record and do not know whether a record already exists.

i

Specifies that the old and new keys are integer values, not packed decimal. For example, you should specify this parameter when running the utility against the acedptv* files; you should not use the parameter when running the utility against the ITEMREC file.

SurePOS ACE might invoke the utility to migrate files during installation (see the README file for information about whether that will occur). If the utility was automatically invoked, you can back out the maintenance, by invoking the utility, and then running ASM without performing a close.

Using ARC to Compile Application Resource Files

SurePOS ACE has a resource compiler, `ARC.EXE`, to manage machine-readable information for the controller application. Like many SurePOS ACE applications, it uses an INI file.

TurboVision Resource Files

All TurboVision-based executables isolate translatable character strings from object code with INI files and the *ISINIFileResourceLibrary* class. With this structure:

- You can easily use symbolic resource IDs in place of hardcoded character strings.
- The translator can easily identify strings that require translation.
- Your users can easily switch languages by simply copying a few files.

Programming Setup

In the main program of each executable, entries similar to the following statements are required to initialize the environment to use resources.

```
//************************************************************************
// Include the symbolic resource id names
//************************************************************************
#include ISTVGlobalResources_rch
#include ISTVACEACRSR_rch

//************************************************************************
// Include the required default resource definitions
//************************************************************************
#include                        // Global
#include                        // Accounting example

//************************************************************************
// Create the singleton object
//************************************************************************
ISINIFileResourceLibrary::createInstance();

//************************************************************************
// Read the default resource definitions from the structure defined in the
// RDH include files
//************************************************************************
ISINIFileResourceLibrary::instance().read(GlobalResourceDefaults); // Global

                                                    // Accounting example
ISINIFileResourceLibrary::instance().read(MainResourceDefaults); // Main section
ISINIFileResourceLibrary::instance().read(CntnrResourceDefaults); // Cntnr section
 .                                                  //   ...
 .
 .
//************************************************************************
// Read the resource definitions defined in the INI files
//************************************************************************
ISINIFileResourceLibrary::instance().read("");   // Global
ISINIFileResourceLibrary::instance().read("");   // Accounting example
```

Getting a Resource

The ISGET_RESOURCE macro returns the value of a resource. It returns a "CHAR *" for the specified resource ID. The following sample is from the `aceaccnt.cpp` file:

```
theTransMenu = new TSubMenu(ISGET_RESOURCE(Main_mnu_Accnt),0,hcNoContext);
```

Translating and Changing a Language

The INI files contain text that must be translated. Except for the keywords BaseID and Prefix, text that follows an equal-sign character (=) must be translated. Text can be in double quotes, but the quotes are not required. However, the use of double quotes around translatable strings makes it much easier to use existing third-party translation tools.

To activate another language in the SurePOS ACE runtime environment, you must copy translated INI files to the base SurePOS ACE INI file names, such as `aceacrsr.ini`.

SurePOS ACE *RSR.INI Resource Files

These INI files define references and strings used by SurePOS ACE code. Strings defined in these files should be translated into the user's native language.

See Chapter 4, .ini File Descriptions on page 467 for a description of each resource `INI` file.

The following excerpt is from the `INI` file used by the accounting support program, `ACEACCNT.CPP`:

```
;*****************************************************************************
;
;       ACEACCNT resource INI file
;
;*****************************************************************************


;*****************************************************************************
; ACEACCNT      -       Main
;*****************************************************************************
[Main]
BaseId=500

mnu_Accnt  = "~A~ccounts"

mnu_Curr   = ~C~urrent  Period
mnu_Prev   = ~P~revious Period
mnu_Rfsh   = ~R~efresh
mnu_Accn   = ~S~elect account
 ...
;*****************************************************************************
; TVACCNMV      -       Cntnr
;*****************************************************************************
[Cntnr]
BaseId=600

Title = Account Management View

err_NoOffice    = Cannot do transaction on Office account
err_NoneOffice  = Office account not found
err_NotRcncld   = Cannot open reconciled account
 ...
```

*RSR.RCH Files

These files are produced by the ARC utility from the corresponding INI file. They contain enumerators used by the code for resources defined in the `*rsr.ini` files.

Enumerators are created for each section defined in the INI file. Names consist of the value specified for the PREFIX keyword (or the section name if the PREFIX keyword is not specified) concatenated to the resource name.

Enumerator values for each section start with the value specified for the BASEID keyword in each section.

The following excerpt is from the RCH file used for Accounting Support (`ACEACCNT.CPP`):

```
//*****************************************************************************
// ACEACRSR.rch written by resource compiler
// ACEACRSR.INI [ACEACRSR.INI]
// 04/08/97 14:42:31
//*****************************************************************************


#ifndef ACEACRSR_RCH
#define ACEACRSR_RCH

enum
```

```
{
  Main_BaseId = 500,
  Main_mnu_Accnt,
  Main_mnu_Curr,
  Main_mnu_Prev,
  Main_mnu_Rfsh,
  Main_mnu_Accn,
 ...
 };

enum
{
  Cntnr_BaseId = 600,
  Cntnr_Title,
  Cntnr_err_NoOffice,
  Cntnr_err_NoneOffice,
  Cntnr_err_NotRcncld,
 ...
```

*RSR.RDH Files

The ARC utility produces these files from the corresponding INI file. The files contain information used to provide defaults.

The following excerpt is from the RDH file used for Accounting Support (`aceaccnt.exe`):

```
//****************************************************************************
// ACEACRSR.rdh written by resource compiler
// ACEACRSR.INI [ACEACRSR.INI]
// 04/08/97 14:42:31
//****************************************************************************
#ifndef ACEACRSR_RDH
#define ACEACRSR_RDH

#include ISINIFileResourceLibrary_i
#include ISTVGlobalResources_rch

static ISINIFileResourceLibrary::ISResourceDefinition MainResourceDefaults[] =
{
  { Main_mnu_Accnt, "~A~ccounts" },
  { Main_mnu_Curr, "~C~urrent  Period" },
  { Main_mnu_Prev, "~P~revious Period" },
  { Main_mnu_Rfsh, "~R~efresh" },
  { Main_mnu_Accn, "~S~elect account" },
  .
  .
  .
};

#include ISINIFileResourceLibrary_i
#include ISTVGlobalResources_rch

static ISINIFileResourceLibrary::ISResourceDefinition CntnrResourceDefaults[] =
{
  { Cntnr_Title, "Account Management View" },
  { Cntnr_err_NoOffice, "Cannot do transaction on Office account" },
  { Cntnr_err_NoneOffice, "Office account not found" },
  { Cntnr_err_NotRcncld, "Cannot open reconciled account" },
  { Cntnr_err_IsRcncld, "The account is reconciled" },
 ...
 };
```

Using ACEKFRBL to Resize the Customer Activity File

SurePOS ACE provides the ACEKFRBL program to resize the Preferred Customer Activity File, eamfbact.dat.

ACEKFRBL Syntax

The ACEKFRBL program has this syntax:
Keyed File Rebuild Program Syntax
ACEKFRBL *Input file name Output file name* [{ -C *Customer name offset in input file* | -L *New record length* | -N *New number of records* }]

Personalization Options for ACEKFRBL

To use ACEKFRBL, change the following options in Loyalty -> Activity personalization to the values needed for the new file, then start the program.

- `Size of activity file records`
- `Number of bytes before customer name`

Example of Using ACEKFRBL

Assume that your existing eamfbact.dat file has 101-byte records and each record contains a 17-byte customer name. The customer name is at offset 84. (See "EAMFBACT (Preferred Customer Activity File)" on page 206 for considerations that go into determining the size of the records and the offset of the customer name.) You want to use a 20-byte customer name and expand the record size to 120 bytes. You do not want to change the number of customer records in the file.

To convert the file:

1. Set the following options in Loyalty -> Activity personalization to the specified value, then save the options.
   - `Size of activity file records`=120
   - `Number of bytes before customer name`=100
2. Rename the existing file to eamfbact.sav, which will be the input file.
3. Issue the following command:

```
ACEKFRBL EAMFBACT.SAV EAMFBACT.DAT -C84 -L120
```

Command Line Options for the Selective Item Report

The ACESLIRL program has this syntax:
Selective Item Report Program Syntax
ACESLIRL *userID password* [{ /D *additionalParameter* /D }]

ACESLIRL accepts these command line parameters:

/D (or /d or -D or -d)

Traces Selective Item Report (SIR) output to the default aceslirl.trc file if used without the optional file name. For example, operator 1 uses the following command to trace SIR output to aceslirl.trc:

```
C:\adx_ipgm\>aceslirl 1 1 -d
```

If you specify a trace file name, all output is recorded to the file you specify (in the current directory). For example, operator 99999999 uses the following command to trace SIR output to the trace.dat file:

```
C:\adx_ipgm\>acelsirl 99999999 99999999 -dtrace.dat
```

Using ACEUBATP for Batch Maintenance of Keyed Files

BATCH is a background utility program that applies maintenance updates specified in the eambatch.dat file to any keyed files that are specified in the input data. The program is started by sending a START USER PROGRAM command for ACEUBATP from the host. ACEUBATP can optionally be started automatically once a day after a designated time.

Using the BATCH program for keyed file maintenance provides these primary benefits:

- Changes for many keyed file records can be transmitted in one sequential file to reduce transmission time and host processing.
- Multiple batches of changes for multiple keyed files can be transmitted in one file to the store controller.
- Selected fields in a keyed file can be modified, leaving other keyed record data unchanged.
- Groups of similar records in a keyed file can be changed or reported as a group with one input record.
- The number of records in a keyed file can be reported with a single request.
- Targeted coupons can be added, deleted, or reported to or from the Customer Activity file, regardless of their location in the record.

Functions and Limitations

All keyed file data provided to the BATCH program must be a subset of the keyed file data as it exists in the store controller file. There is no translation of data fields. The BATCH program processes all input data in the order in which it appears in the input file. All maintenance is applied immediately, so there is not a concept of delayed maintenance. The execution status of all jobs processed by the BATCH program is recorded in a special status file (eambstat.dat) as well as in the Exception Log, and all errors are recorded in the Delayed Data Maintenance Error Log and possibly in the System Error Log. The batch status file is intended for host processing and the Delayed Data Maintenance Error log and Exception log can be reported in the store. A unique logical name is used to access the Delayed Maintenance Error Log so that these errors can be logged to a different file for host retrieval.

For a given batch, you can make the following selections:

- Select whether a full record is to be treated as a replacement or as an addition.
- Select whether an add is allowed to replace or whether a replace is allowed to add. This provides the 'Add if not present' and 'Replace only if present' functions as well as the 'Add or replace' function.
- Select whether an error should be logged for either a replace record that does an add or for an add record that does a replace.

- Select whether to add the record with partial data and set all other fields to X'00' when a partial replace is requested for a specific record and the record is not on file.

All report requests processed by the BATCH program cause output report data to be appended to the batch report file (eambrept.dat). A report header record is followed by the reported keyed file records, which are packed sequentially.

There is no limit to the number of records that can be altered or reported with a single request to the BATCH program.

Batch Interface Definition

The BATCH program is named ACEUBATP and resides in the adx_ipgm directory. It can be started from the background application screen or by a START USER PROGRAM command from the host. The input file for the BATCH program is named eambatch.dat and resides in the adx_idt1 directory. It is loaded from the host using the name /BATCH. The records in the input file are created by the host and are processed solely by the BATCH program.

See "EAMBATCH (Batch Maintenance File)" on page 119 for the format of the input data. The report file used by the BATCH program is named eambrept.dat and is located in the adx_idt4 directory. You can dump the report file to the host using the name /BREPT. The status file for the BATCH program is named eambstat.dat and is located in the adx_idt4 directory. You can dump the status file to the host using the name /BSTAT.

When the BATCH program starts processing, it immediately renames eambatch.dat to eambinpr.dat, which allows the input file to be reloaded immediately after the BATCH program is started. The eambinpr.dat file is processed first if one exists, which provides for recovery in the event that the processing of a file is interrupted.

If input data that is not valid is found, the input file is saved in eamberrd.dat. Only the most recent input file with data that is not valid is saved.

The BATCH program logs selected errors to the Delayed Maintenance Error File (eamdmerr.dat). See "EAMDMERR (Delayed Data Maintenance Error File)" on page 151 for the file format.

The BATCH program creates the following error codes:

- 07 - Record to be added already exists; update discarded.
- 09 - Record to be added already exists; record replaced.
- 24 - Record to be replaced not found; record added.
- 25 - Record to be deleted not found.
- 35 - Record to be reported not found.
- 92 - Record did not have free space to add targeted coupons.
- 93 - File open error; batch discarded.
- 96 - Batch not valid; batch discarded.
- 97 - Batch not valid; record discarded.
- 99 - Record to be modified not found, update discarded.

Using ACEBIRBL to Rebuild BIN Files

ACEBIRBL is a utility program that reads a retailer-provided input file and creates a keyed BIN file. Only one instance of the program can be running at a time. A record is written to the acebinlg.dat file every time that ACEBIRBL starts and ends.

Starting the Program from the Command Line

This is the syntax of the ACEBIRBL command:

```
acebirbl input_file output_file
```

Note: You can enter `acebirbl -?` to display help text.

These are the parameters for the ACEBIRBL command:

input_file
The name of the ASCII input file, which is preceded by either `-I` or `-i`.

output_file
The name of the output file, which is preceded by either `-O` or `-o`.

If no input file is specified on the command, the file specified by the <acebninp>logical name is used. If no output file is specified on the command, the file specified by the <acebinfl> logical name is used.

Starting the Program from the Scheduler

To start ACEBIRBL from the Scheduler, you must enable the `Bin Support` option in the `aceschdl.ini` file (see "ACESCHDL (Delayed Data Maintenance Scheduler)" on page 510).

If the option is enabled and the files specified by the <acebninp> and <acebnint> logical names exist, then the Scheduler will start ACEBIRBL with no parameters.

Input File

See "ACEBNINP (ACEBIRBL Input File)" on page 46 for a description of the input file.

Output File

See "ACEBINFL (BIN File)" on page 43 for a description of the output file.

Processing of Files

These are the steps in processing the BIN file:

1. ACEBIRBL renames the input file to the file name that is specified by the <acebnint> logical name.
2. The renamed file is opened and the records are read and processed.
3. If the header Action Code is `U` (update) then the current BIN file (acebinfl.dat) is copied to the temporary file (acebntmp.dat).
4. The BIN record is written to the temporary file that is specified by the <acebntmp> logical name (acebntmp.dat is the default file name).
5. After the input file has been completely read and processed, the temporary file is renamed to the file name that is specified by the <acebinfl> logical name. If the program was run from

the command line and an output file name was specified, the temporary file is renamed to that file name.

6. The original file is renamed with one of these extensions:

.asv	The input file was read successfully with no errors.
.abd	One or more errors occurs while processing the input file.

If you want to retrieve a previous keyed BIN file (acebinfl.dat), this is the process:

- If errors occurred each time ACEBIRBL was run since the file to be retrieved was created, then rename the input file with the .asv extension and run ACEBIRBL again.
- If there have been successful runs of ACEBIRBL since the file to be retrieved was created, then you must recreate or retrieve the original ASCII input file and run ACEBIRBL again.

Using ACEFIRBL to Rebuild Coupon Fraud Files

ACEFIRBL is a utility program that reads a retailer-provided input file and creates keyed coupon fraud files for both basic and GS1 coupons. Only one instance of the program can be running at a time. A record is written to the coupon fraud log file (acefirlg.dat) file every time that ACEFIRBL starts and ends.

Starting the Program from the Command Line

This is the syntax of the ACEFIRBL command:

```
acefirbl command_line_switch1 command_line_switch2 ...
```

These are the command line switches for the ACEFIRBL command that are not case-sensitive:

-I input_file_name

The name of the ASCII input file. If no input file is specified, <ACECPNFR> is used as the default.

-Obas basic_output_file_name

The name of the generated keyed file for basic fraudulent coupons. If not specified <ACECFBAS> is used as the default.

-Ogs1 GS1_output_file_name

The name of the generated keyed file for GS1 Databar fraudulent coupons. If not specified, <ACECFGS1> is used as the default.

-ForceReload

If specified, sets all terminals to reload the fraudulent coupon files. This provides a mechanism for initiating a reload of the fraudulent coupon files without other processing.

Note: You can enter acefirbl -? to display help text.

Starting the Program from the Scheduler

To start ACEFIRBL from the Scheduler, you must enable the `Coupon Fraud Support` option in the `aceschdl.ini` file (see “ACESCHDL (Delayed Data Maintenance Scheduler)” on page 510).

If the option is enabled and the files specified by the <acecpnfr> and <acecpnft> logical names exist, then the Scheduler will start ACEFIRBL with no parameters.

Input File

See “ACECPNFR (ACEFIRBL Input File)” on page 73 for a description of the input file.

Output File

See “ACECFBAS (Coupon Fraud Basic File)” on page 52 and “ACECFGS1 (Coupon Fraud GS1 File)” on page 52 for a description of the output file.

Processing of Files

Following are the steps to process the Coupon Fraud files:

1. ACEFIRBL renames the input file to the file name that is specified by the <acecpnft> logical name.
2. The renamed file is opened and the records are read and processed.
3. The Basic and GS1 output records are written to temporary files that are specified by the <acecbast> and <acecgs1t> logical names where `acebast.dat` and `acegs1t.dat` are the default file names.
4. After the input file has been completely read and processed, the temporary files are renamed to the file names specified by the <acecfbas> and <acecfgs1> logical names. If the program was run from the command line and output file names were specified, the temporary files are renamed to those file names.
5. The original file is renamed with one of these extensions:

.asv	The input file was read successfully with no errors.
.abd	One or more errors occurred while processing the input file.

If you want to retrieve previous keyed coupon fraud files (`acecfbas.dat` or `acecfgs1.dat`), do the following:

- If errors occurred each time ACEFIRBL was run since the file to be retrieved was created, then rename the input file with the `.asv` extension and run ACEFIRBL again.
- If there have been successful runs of ACEFIRBL since the file to be retrieved was created, then you must recreate or retrieve the original ASCII input file and run ACEFIRBL again.

Using ACETIRBL to Rebuild Terminal Item Record Files

ACETIRBL is a utility program that is used to build the terminal Item Record file. The terminal Item Record file, whether current or pending, contains the source item records that were selected for inclusion based on both item movement history and item code ranges specified in options.

You can start the program from the command line. The program is also automatically started when an Item Record file is rebuilt using either ACEKFRBL or ACEICRBL (target = EAMITEMP) if TOF is active.

Starting the Program from the Command Line

This is the syntax of the ACETIRBL command:

```
acetirbl command_line_switch1 command_line_switch2 ...
```

These are the command line switches for the ACETIRBL command:

-Activate
: If specified, activates a pending terminal Item Record file. This switch is used when personalization activates a pending Item Record file.

-ForceReload
: If specified, sets all terminals to reload the terminal Item Record file. This provides a mechanism for initiating a reload of the terminal Item Record file without other processing.

-NoBackup
: If specified, terminal Item Record files are not backed up during activation.

-NoPending
: If specified, terminal Item Record files are not saved as pending during a restore.

-NoPreload
: If specified, do not run the preload processes; the customer intends to run the processes later. (The default value specifies that the preload processes will be run.)

-NoReload
: If specified, do not set terminals to reload the terminal Item Record file. (The default value sets all terminals to reload the terminal Item Record file.)

-Restore
: If specified, restores a backup terminal Item Record file. This switch is used when personalization restores a backup Item Record file.

Return Codes from ACETIRBL

These are the return codes that are sent from the ACETIRBL utility:

Return Code	Meaning
0	Successful processing of all stages.

Return Code	Meaning
1	Unable to build record selection list.
2	Unable to build terminal Item Record file.
3	Unable to compress terminal Item Record file.
4	Legacy RAM disk preload (ADXTRM0L) failed.
5	Loadshrink (ADXRTCCL) failed.
6	Unable to update store record in Terminal Offline Status file.
7	Unable to notify all terminals to reload terminal Item Record file.
8	Rename of files failed.
9	Distribution error.
10	Build of preload bundle (ADXPLDBL) failed.
11	Activation of preload bundle (ADXPLDDS) failed.
12	Unable to execute due to an invalid command attribute.
13	Initialization error (failure to access acetirni.ini or acetirlg.ini).

Processing of Files

These are the steps in rebuilding the terminal Item Record file:

Note: ACETIRBL uses approximately 6MB per 100,000 unique items contained in the processed Item Movement files.

1. Delete work files, if they exist.
2. Determine the records to include based on item movement history.
3. Allocate and load the terminal Item Record file, selecting items from the Item Record file. First select items from the item code ranges specified in personalization, then select from ordered item movement.
4. Perform a reasonability check on the number of records actually selected for inclusion in the file. If the count of records contained in the new terminal Item Record file does not exceed the threshold value defined in the `acetirni.ini` file, then log an `X510 LOW RECORD COUNT PROCESSED BY ACETIRBL` application event as a warning.
5. If production terminal Item Record files (trmitemr.dat and zipitemr.dat) are being built, then mark the Terminal Offline Status file to indicate that a new terminal Item Record file is being built.
6. Create a compressed version of the terminal Item Record file.
7. Rename the target terminal Item Record file and the target compressed file.
8. Rename the temporary files to the target names.
9. Set the file distribution characteristic of the target terminal Item Record file and the compressed file.
10. If a production terminal Item Record file is being built or activated:
    - Run the preload processes.
    - Update the store record of the Terminal Offline Status file with the new date and time stamp.

11. If a production terminal Item Record file is being built and reload is required, mark the terminals for reload.

Using ACEWERBL to Convert WIC EBT Files

ACEWERBL is a utility program that performs these functions:

- Converts sequential Approved Product List and Hot Card List files from a WIC EBT agency to keyed files that contain only the information that SurePOS ACE requires to process WIC EBT transactions.
- Processes Reconciliation and Error files from a WIC EBT agency and updates claim information in the acewercl.dat file.

There are no parameters for the ACEWERBL command. The utility processes any files that have a valid file extension and that are located in the adx_idt1 directory.

Input Files

The input file names must have the format `AAYJJJ.ext`. These are the components of the file name:

AA	The two-character agency identifier.
Y	The last digit of the year. The value must be 0-9.
JJJ	The Julian day of the year. The value must be 1-366.
ext	A file extension that indicates the type of file. These are the valid file extensions:

APL - Approved Product List. These files are processed second.
A?? - Reconciliation file. `??` is replaced by a sequence number. These files are processed last.
E?? - Error file. `??` is replaced by a sequence number. These files are processed third.
HCL - Hot Card List. These files are processed first.

Output Files

Table 127 lists the input and output files for the utility program.

Table 127. Input and Output Files for ACEWERBL Utility

Input File	Output File
UPC/PLU List (Approved Product List) from WIC EBT agency	ACEWE??.APL (see "ACEWE??.APL (WIC EBT Approved Product List File)" on page 95)
Hot Card List from WIC EBT agency	ACEWE??.HCL (see "ACEWE??.HCL (WIC EBT Hot Card List File)" on page 97)
Reconciliation File from WIC EBT agency	Updates ACEWERCL.DAT (see "ACEWERCL.DAT (WIC EBT Claim/ Reconciliation Status File)" on page 93)

Input File	Output File
Error File from WIC EBT agency	Updates ACEWERCL.DAT (see "ACEWERCL.DAT (WIC EBT Claim/ Reconciliation Status File)" on page 93)

Post Processing of Files

After a file has been processed, the original file is backed up in the adx_idt1 directory. Based on the results of the conversion, the original file is renamed with the extensions shown in Table 128. Also, the file names for the Error and Reconciliation files are modified - the two-character sequence number, indicated by ?? in the format for the original file name, is appended to the file name. For example, the Reconciliation file TX4053.A00 is renamed to TX405300.asv if the file is successfully converted.

Table 128. File Extensions after ACEWERBL Executed

Type of File	Successful Conversion	Conversion Errors
Approved Card List	.asv	.abd
Error file	.esv	.ebd
Hot Card List	.hsv	.hbd
Reconciliation file	.asv	.abd

Using ACERECLN to Clean Up and Create Reprint Receipt Files

ACERECLN (Receipt Clean Utility) is a utility program that creates new reprint receipt files for today and tomorrow, and erases any old reprint receipt files stored longer than the Personalization option: Number of days to save receipts. It will be kicked off by the Scheduler at a default time of 12:00 a.m., which can be changed by the retailer. The Number of days to save receipts option must have a value at the store level > 0 for ACERECLN to do any work. This task will first create the new empty files for the day, and then it will erase any old files. If the process is terminated and restarted, it will determine what has already been done and continue until its processing is complete. The ACE Receipt Clean Utility is not impacted by a store closing, and should be scheduled at a different time than the store close runs since both result in processing and distributing files (best to spread out for performance reasons).

Starting the Program from the Command Line

To run the ACERECLN program from the command line, type ACERECLN. For help on running the program, type ACERECLN -?.

Starting the Program from the Scheduler

The Scheduler file aceschdl.ini has the following new keywords to support this function:

Reprint Receipt Support = 0

This value controls whether Reprint Receipt File Cleanup support is enabled or disabled.

Set to 0 to disable the scheduling Reprint Receipt File Cleanup Support (default).

Set to 1 to enable the Reprint Receipt File Cleanup Support. When enabled, ACERECLN.386 will be kicked off at the specified hour of the current day to create empty receipt files and to remove old receipt files for all terminals.

Receipt Clean Hour = 0

The receipt clean hour ranges from 0 to 23, i.e., 12:00 a.m. to 11:00 p.m. and defaults to 0 (midnight). If a value other than 0 to 23 is specified, a default value of 0 will be used.

Following is the directory/file structure for the files created:

- C:\EJ\R – the directory to store the receipt files. This task will create the directory structure if it does not exist.
- RMDDTTT – the Reprint Sales Receipt file, where M=month (1-9, A=10, B=11, C=12), DD=day, and TTT=terminal number. A file will be created for each terminal that has a record in the EAMTERMS.dat file. Only terminals that have the Number of days to save receipts option, with a value > 0, will actually write to the file.
- RMDDIDX – The Reprint Sales Receipt Index file, where M=month (1-9, A=10, B=11, C=12), and DD=day. Each receipt stored in the Reprint Sales Receipt file will have an entry in this file and all the terminals will write to the same index file for a specific day.

Using ACEREPRN to Search Reprint Receipt Files

The reprint server program ACEREPRN is used to perform the search logic to find stored sales transaction receipts that match the search criteria sent from Store Integrator GUI through the ACE terminal to the back office application. At a high level, this program will serve requests to search for stored sales transaction receipts and return matching results to the terminals via TCP/IP. The communications settings are defined in Personalization in Print->SalesReceipt->Reprint Receipt. The results will be limited by one of the search attributes passed in an XML message to this program that indicates the maximum number of matching entries to return.

ACEREPRN should be defined as a background task on the acting file server to start at IPL when searching and reprinting sales transaction receipts in use.

Using SAPATH.RVT

The `sapath.rvt` file contains the definitions of SurePOS ACE data files in the form of tables. Each table defines the name, data type, offset, length, and, if necessary for bit field definitions, the mask for each field in the file. Both fixed and optional fields are defined.

The `sapath.ddf` file is constructed from the `sapath.rvt` file by the DataMiner configuration tool (GCONF32). The `sapath.ddf` file, which is platform-independent, uses logical names to locate files.

Beginning with the V1R4 release, SurePOS ACE uses a data dictionary to access fields in the Item Record File. The data dictionary is a class, an instance of which will provide all definition information for all the fields of a given SurePOS ACE data file.

If you want to modify the layout of a data file, you should not modify the `sapath.ddf` file. You should use a file overlay for the `acesql.ini` file to specify your own `sapath.ddf` file as a delta to the base file.

Appendix B. The BOB Programming Language

Writing a Report for the Engine

This section describes how to write BOB reports that use the environment available to execute BOB reports, which was just described. To introduce a new report to the report engine, you must perform these steps:

1. Define your report to the engine in <ACEREINI>. To add a new report, you must create a new section for your report with an *Execute*= entry and a *Sources*= entry, and also with optional *Prepare*= and *ProcedureId*= entries.
2. Write a *prepare* function if one is necessary for your report.
3. Write an *execute* function along with any BOB classes you require.

SQL Use in ACETVREL

Structured Query Language (SQL) is a language for interacting with table-based databases. This section introduces how SurePOS ACE uses SQL to report data from its database objects but does not provide an exhaustive overview of SQL. Such information is generally available. The SQL Engine that the report engine uses is *DataMiner*.

The SQL Engine permits BOB reports to view accounting and other store data as a series of tables rather than files. It separates the logical use of file data from storage mechanisms and permits specific queries through SQL *select* statements.

The SQL Engine provides data mapping of keyed, sequential, and direct files through a data dictionary, which is the `sapath.ddf` file. The DataMiner configuration tool (GCONF32) builds this binary data dictionary file. The data dictionary contains definitions of tables, which describe the data format of the tables and what data files they are in. The `sapath.ddf` file uses logical names to locate files. The `sapath.ddf` file is platform-independent.

Special column names that the SQL engine generates for you are:

file_version

The SQL Engine data definition tool permits file specification with the use of wildcard characters. For example, it is possible to specify that all files named eamacc??.dat use the same table for data file mappings.

If a table is defined for a series of files using wildcard characters, it automatically has a string *file_version* column. You can use the *file_version* column to narrow a query to a subset of all files that use the table. This *virtual column* contains the value of the wildcard characters. For example, the table for EAMACCTC.DAT contains *TC* in the *file_version* column.

header_offset

For tables in sequential files, it is possible to specify that a given record is a *header* record. An example of this is the Transaction Log (TLOG) file. There is a header record (record type X'00') that begins every transaction in the file. In the DataMiner configuration tool, this is referred to as the *separator record* for the file.

All tables that exist in a sequential file that contains a separator record have a *header_offset* column. It contains the offset into the file of the separator record associated with any given row of the table. For example, the *header_offset* column of a *TransLogItemEntryTable* row (the TLOG item entry string), would contain the offset in the file of the header record for this item entry string.

record_offset

This column exists in the same files that the header_offset does. Its value specifies the offset for the row within the table, rather than the offset of its associated header string. The separator record does not have a *record_offset* because it would be the same as the *header_offset*. For a row of the *TransLogItemEntryTable*, the *header_offset* is the offset in the file of the header string, while the *record_offset* is the offset in the file of the item entry string itself.

An Introduction to SQL

Reports primarily use the SQL Select statement, which has this general syntax:

```
SELECT <list of columns, comma-delimited>
FROM <list of tables, comma-delimited>
WHERE <selection criterion>
ORDER BY <sort specification>
```

where:

list of columns, comma-delimited

Columns (fields) within a row (record) of the table (record type) that you are examining. This specification can use scalar functions AVG(), SUM(), MAX(), MIN(). It can also contain basic formulas.

list of tables, comma-delimited

The From clause specifies the table to select rows from. You can list more than one table to *join* data from them. Refer to any book about SQL to learn about joins. The SurePOS ACESQL Engine supports left outer joins, but not right or full outer joins.

selection criteria

In the WHERE clause, specify criterion that narrows the query result to the targeted rows. This clause commonly uses column names. An example is: `Where amount >= 5.00`. You can use formulas and sub-queries in this clause.

sort specification

The Order By clause permits you to specify a result column by which to sort your output rows. You can specify DESC or ASC to sort in descending or ascending order. You can specify more than one sort column.

To retrieve data, you must execute an SQL query. Generally, SQL queries are first prepared and then executed. Validation of the query occurs during the preparation step. SurePOS ACE SQL classes prepare a query if the query is not already prepared. If the prepare of a query fails, it is usually because there is an error in its coding. Such errors might include specifying a table or column that does not exist, or an error in syntax such as a missing comma.

The sample query, `SELECT * FROM Operator WHERE OperatorId = `555'` selects data from every column because the asterisk (*) is a special character denoting all columns. It selects data from the Operator table. It selects only those rows where the OperatorId column contains a value equal to 555. Because the Operator table is the only table located in the Operator Authorization file (EAMOPERA), this query looks through the entire file and retrieves all authorization information for operator 555.

The sample query, `SELECT ITEMNAME FROM ItemData WHERE ITEMCODE = ?` selects item description data that is in the ITEMNAME column. It selects data from the Item Record file, which contains the ItemData table. It selects only those rows of data (items) where the item code is a value that is specified dynamically at execution.

SQL queries permit the use of *variables*, which are called *parameters*. The query can be prepared for execution before the value of such variables is known. The variables can be added as parameters later, before the query executes. In fact, the query can be prepared once and executed many times, each time with different values for its parameters. Issuing one prepare and multiple executions rather than many of both usually results in performance gains.

The report engine provides a class, *ISSQLQuery* [RESQLQRY.HPP/CPP] that encapsulates SQL queries and permits them to be prepared, executed, and their resulting data retrieved. This class is available in the BOB environment for reports to use during execution. Reports use them through a mechanism called *exporting*.

BOB Language Extensions: Additional Classes Available To Help Write Reports

In addition to the BOB language, there are these classes for you to use when writing Bob code for the report engine. The classes are of two types:

- Classes written in BOB language
- Classes written in C++ but *exported* to the Bob execution environment

This section describes how to export a C++ class or function to BOB. It also shows what C++ classes and functions have been exported already and are available for your use.

Exporting Classes

Exporting a class to the BOB environment is the process of making a class written in C++ available for use by BOB code. It is accomplished by creating a new class, a subclass of *BobCPPClass* [bobclass], whose function members refer to some C++ class.

An example of an exported class is the *ISGraphicText* class, which is defined in module [grphctxt.hpp/cpp]. It is a normal C++ class with member functions and data. When BOB code is executing to produce output for a report, it must create *ISGraphicText* objects to draw them on *ISReportDevices*. This is made possible by the introduction of a *BobCPPClass* subclass called *GraphicTextClass*.

The purpose of *GraphicTextClass* is to introduce a new class into the BOB environment when it is constructed. Its methods New and Delete contain instructions for allocating and freeing memory for the object in the BOB memory pool. Its constructor introduces the new class name into the BOB environment and associates functions with the new class name. Figure 28 shows the constructor:

```
//****************************************************************************
// Constructor
//****************************************************************************
GraphicTextClass::GraphicTextClass(void)
  : BobCPPClass("GraphicText", BOBCPPCLASS_HAS_NEW | BOBCPPCLASS_HAS_DELETE)
{
  // Get/set the text
  typedef ISString (ISGraphicText::*PM)(void) const;
  PM getText = &ISGraphicText::text;
  AddFunctionMember(new BobCPPPrimitiveMemberNone<ISGraphicText, ISString>("getText",
getText));

void (ISGraphicText::*setText)(const ISString&) = &ISGraphicText::text;
  AddFunctionMember(new BobCPPPrimitiveMemberOne<ISGraphicText, void,
         const ISString&>("setText", setText));

  // Set the alignment of the text
  void (ISGraphicText::*setAlignment)(UInt, UInt) = &ISGraphicText::setAlignment;
  AddFunctionMember(new BobCPPPrimitiveMemberTwo<ISGraphicText, void, UInt,
          UInt>("setAlignment", setAlignment));
("setAlignment", setAlignment));

// Get the alignment of the text
  AddFunctionMember(new BobCPPPrimitiveMemberNone<ISGraphicText,
        UInt>("getAlignmentIndex", &ISGraphicText::getAlignmentIndex));
  AddFunctionMember(new BobCPPPrimitiveMemberNone<ISGraphicText,
        UInt>("getAlignmentOffset", &ISGraphicText::getAlignmentOffset));

  // Set the font of the text
  void (ISGraphicText::*setFont)(const ISString&) = &ISGraphicText::font;
  AddFunctionMember(new BobCPPPrimitiveMemberOne<ISGraphicText, void,
         const ISString&>("setFont", setFont));
}
```

Figure 28. GRPHCTXC.HPP: Constructor for Introducing a New Class

The member initialization statement for this class introduces a new class into the BOB environment that is known to BOB code as *GraphicText*. Each *AddFunctionMember* request introduces a new member function to the GraphicText class: *getText()*, *setText(const ISString&)*, *setAlignment(void, Uint)*, *getAlignmentIndex()*, *getAlignmentOffset()*, and *setFont(void, const ISString&)*.

The *::AddFunctionMember* method is inherited from the parent *BobCPPClass*. The template classes *BobCPPPrimitiveMemberNone*, *One*, *Two*, and so on, are defined in [bobclass.hpp/cpp].

All *BobCPPClass* subclasses should be singleton classes. Their creation as a single instance generally takes place during construction of the *ISReportFactory* singleton class.

You can choose to make available to the BOB class some or all of the functions of the associated C++ class. You can also choose to add a method to the BOB class that is not a member of the associated C++ class by writing a C++ function (not a member function) and introducing this function to the BOB class. Foe example, the C++ *ISGUIInterface* class is exported to BOB by the *ISGUIInterfaceClass* [gintrfcc.hpp/cpp] as BOB class name *ExternalGUICanvas*.

ExternalGUICanvas is used during the BOB prepare function for a report. It is the canvas to which the BOB prepare function must add *ISGUIControlDefinitions*. The *ISGUIInterface* itself is an abstract class, providing only signatures for the methods to *::addControl* and *::getControlNamed*. Because these functions are virtual, their address is unknown and cannot be provided to the *BobCPPPrimitiveMemberxxx* class constructor. Therefore, the method of adding these members to the BOB *ExternalGUICanvas* class that has been described is not suitable.

To add methods to a virtual class, you must write them as static methods whose address are known and provide them to the *BobCPPPrimitiveMemberxxx* constructor. These static methods call the *addControl* or *getControlNamed* methods of the *ISGUIInterface** object. This provides virtual resolution.

Figure 29 shows one of those static methods:

```
//*****************************************************************************
// Retrieve a control from the interface.
//*****************************************************************************
static void ISGUIInterfaceGetControlNamed(int argc)
{
  ISTRACE_IN(AddControl);
  ISASSERT(argc >= 1);
  CheckArgumentCount(argc, 2);
  StackCheckClass(0, theStringClass);

  // Get a pointer to the interface
  ISGUIInterface* interface
    = (ISGUIInterface*) (((ExternalObject*) StackGetObject(1))->external);

  // Ask the interface for the control being referenced
  ISGUIControlDefinition* control
    = interface->getControlNamed(StringGetData(StackGetHandle(0)));

  // Tidy up the stack
  StackDrop(2);
```

```
   // Set up the return value to BOB
   if (control == 0)
   {
     // The control name was not found return nil
     StackPushNil();
   }
   else
   {
     // Create an external object to hold the pointer to the C++ object.
     ExternalObject* object
       = (ExternalObject*) ISGUIControlClass::instance()->NewSimpleExternal(control);

     // Mark the object as belonging to C++
     HandleMakeInteger(&object->internal, 1);

     // Return the completed external object
     StackPushObject((Object*) object);
   }
 }
```

Figure 29. Static Method for a Virtual Class

Functions such as this must access BOB's stack, which includes *StackPushObject*, *StackDrop*, and *StackCheckClass*, among other objects. These functions and macros are available in bobobjct.h.

Exporting Functions

Exporting a C++ function to be available as a BOB function is quite similar to the exporting of a class. Instead of using the *::AddMemberFunction* method of a *BobCPPClass*, you use the *::AddFunction* method of the *BobCPPInterface* [bobcppin.hpp/cpp]. BobCPPInterface is the parent class of the *ISReportEngineBobInterface* [rebbintr.hpp/cpp].

There are two places where these requests are made within the engine. The first is during the construction of the singleton *BobEnvironmentClass* [bbenvrnc.hpp/cpp]. The second is during the construction of the singleton *ISReportFactory* [rprtfctr.hpp/cpp].

Both methods produce the same abilities; the difference is one of organization. The first is intended to export functions that report code would commonly use. The second method is for things like debugging service functions.

Figure 30 is an example from [`bbenvrnc.cpp`] of the C++ *storeId()* function:

```
static ISString getStoreId()
{
return TheSystem.storeID();
}
```

Figure 30. Exported Function from BBENVRNC.CPP

This function is made available to the BOB code as function *salesDescriptor(Integer)* when the *BobEnvironmentClass* constructor makes the request:

```
bob->AddFunction(new BobCPPPrimitiveFunctionNone<ISString>("storeId", getStoreId));
```

Figure 31. Adding a New BOB Function

Detail: C++ Classes Exported To BOB

Table 129 summarizes C++ functions and classes that are exported and available in the BOB coding environment.

Table 129. C++ Classes Exported to BOB Coding Environment

C++ Hierarchy/Global Function that is Being Exported		BobCPPClass that Does the Exporting	BOB Class as it is Known by BOB Code
[grphctxt]	ISGraphicText ::ISString text() ::void text(const ISString&) ::void setAlignment(UInt, UInt) ::UInt getAlignmentIndex() ::Uint getAlignmentOffset() ::void font(const ISString&)	[grphctxc] GraphicTextClass	GraphicText ::String getText() ::void setText(String) ::void setAlignment(Integer, Integer) ::Integer getAlignmentIndex() ::Integer getAlignmentOffset() ::void setFont(String)
[reprtdvc]	ISReportDevice ::ISSize getPageSize() ::void startReport() ::void endReport() ::void startCompression() ::void endCompression ::void newLine() ::void newPage() ::UInt lineCount() ::UInt pageCount() ::void drawWithOffset(const ISGraphicText& const ISPoint&) ::ISSize size(const ISGraphicText&) ::Boolean makesPermanentRecord() ::Uint getDeviceType()	[rprtdvcc] ReportDeviceClass	ReportOutputDevice ::Pair getPageSize() ::void startReport() ::void endReport() ::void startCompression() ::void endCompression() ::void newLine() ::void newPage() ::Integer lineCount() ::Integer pageCount() ::void DrawWithOffset (GraphicText, ISPair) ::Pair size(GraphicText) ::Integer makesPermanentRecord() ::Integer getDeviceType()
[resqlqry] [resqlqrc]	ISSQLQuery ::void addParameter(const ISSQLValue&) ::void clearParameters() ::Boolean prepare() ::Boolean execute() ::Boolean isValid() ::const ISString& name() ::Boolean fetchNext() ::Boolean hasNamedColumn (const ISCollectableString&) ::ISSQLValue getNamedColumn (const ISCollectableString&) void ISSQLQueryGetAllColumns(int argc)	[resqlqrc] ISSQLQueryClass	SQLQuery ::void addParameter (Integer/String/Time) ::void clearParameters() ::Integer prepare() ::Integer execute() ::Integer isValid() ::String name() ::Integer fetchNext() ::Integer hasColumn (String) ::Integer/String/Time getColumn (String) ::OrderedCollection getAllColumns()

[bbenvrnc]	BobEnvironmentObject [singleton]
	::pushQuery() [arg is on stack]
	::popQuery()
	::Boolean SetVariable(const char*, ObjectHandle*)
		[bbenvrnc] BobEnvironmentClass	Environment
			::pushQuery(SQLQuery)
			::popQuery()
			::Integer setVariable(String, ?)

[guiintrf] [gintrfcc]	ISGUIInterface*** ::Boolean addControl(ISGUIControlDefinition*) via void ISGUIInterfaceAddControl(...) ::ISGUIControlDefinition* getControlNamed(const ISString&)	[gintrfcc] ISGUIInterfaceClass	ExternalGUICanvas ::Integer addControl (ExternalGUIControl) ::ExternalGUIControl getControl (String)

C++ Hierarchy/Global Function that is Being Exported		BobCPPClass that Does the Exporting	BOB Class as it is Known by BOB Code
[gintrfcc]	via void ISGUIInterfaceGetControlNamed(...)		
[guiintrf]	ISGUIControlDefinition ::void setName(const ISString&) ::void setType(UInt) ::void setEntryType(UInt) ::void setMaximumWidth(UInt) ::void setText(const ISString&) ::void setMinimumValue(UInt) ::void setMaximumValue(UInt) ::void setPosition(const ISPoint&) ::void setSpread(const ISPoint&) ::void addEntry(const ISString&) ::void setSelectionIndex(UInt) ::void setState(UInt) ::void setCharAny(Boolean) ::const ISString& getName() ::const ISString& getText() ::UInt getState() ::UInt getSelectionIndex()	[gintrfcc] ISGUIControlClass	ExternalGUIControl ::void setName(String) ::void setType(Integer) ::void setEntryType(Integer) ::void setMaximumWidth(Integer) ::void setText(String) ::void setMinimumValue(UInt) ::void setMaximumValue(UInt) ::void setPosition(Pair) ::void setSpread(Pair) ::void addEntry(String) ::void setSelectionIndex(Integer) ::void setState(Integer) ::void setCharAny(Integer) ::String getName() ::String getText() ::Integer getState() ::Integer getSelectionIndex()

Detail: C++ Functions Exported to BOB

Table 130 summarizes C++ functions and classes that are exported and available in the BOB coding environment.

Table 130. C++ Functions Exported to BOB Coding Environment

C++ Module	C++ Name	Class Exporting	Export Module	BOB Name
bbenvrnc	ISString getSalesDescriptor(UInt)	BobEnvironmentClas s	bbenvrnc	String salesDescriptor(Integer)
	ISString getReportDescriptor(UInt)			String reportDescriptor(Integer)
	ISString getDepartmentDescriptor(UInt)			String departmentDescriptor(Integer)
	ISString storeId()			String storeId
	ULong getPercent(ULong numerator, ULong denominator)			Integer percent(Integer numerator, Integer denominator)
	void getOptionValue(int argc)			Integer optionValue(Integer ID, Integer termID = nil)
	void getOptionValues(int argc)			OrderedCollection optionValues(Integer listID,

C++ Module	C++ Name	Class Exporting	Export Module	BOB Name
				OrderedCollection IDsToRetrieve, OrderedCollection IDsToMatch, OrderedCollection valuesToMatch, Integer termID = nil)
	void getINIFileSectionNames(int argc)			OrderedCollection INIFileSectionNames(String INIFileName)
	void getINIFileValues(int argc)			OrderedCollection INIFileValues(String INIFileName, String sectionName OrderedCollection entryIDsToRetrieve)
	void INIFileEntry(int argc)			OrderedCollection getINIFileEntry(String, String, String)
	void INIFileClose(int argc)			OrderedCollection closeINIFile(String)
	void pack (int argc)			String pack(String stringToPack, Integer stringLengthToPack, 0/1 binaryPack)
	void unpack(int argc)			String unpack(String stringToUnpack, Integer length, 0/1 binaryUnpack)
	void logReportException(int argc)			void logReportException(String reportName, String operTermReportedOn, Integer typeOfReport (display, print, file), Integer accountability reported)
rprtfctr	Boolean loadSourcesForName(const ISCollectableString&)	ISReportFactory	rprtfctr	Integer compileBobFile(String)
	Long gcCount()			Integer gcCount()
	Long gcTime()			Integer gcTime()
	Long segCount()			Integer segCount()

C++ Module	C++ Name	Class Exporting	Export Module	BOB Name
	void bobBreakpoint()			void bobBreakpoint(Integer breakpointID, <anything> variableToView = nil)
	void getLogFiles(int argc)			OrderedCollection getLogFiles(String, Integer)
	ISString createTempFile(ISString, int)			String createTempFile(String, Integer)
	ISString getAutoReportEntry()			String getAutoReportEntry()
	Boolean isAutoReport()			Integer isAutoReport()
	ISString getAutoReportScript()			String getAutoReportScript()
	ISString expandItemCode(ISString)			String expandItemCode(String)
	ISString expandItemMovementCode (ISString, int)			String expandItemMovementCode (String, Integer)
	Boolean isGTINenabled()			Integer isGTINenabled()
	Boolean isItemMovementGTINenabled()			Integer isItemMovementGTINenabled( )
	Boolean isItemDataFieldAvailable(ISStrin g)			Integer isItemDataFieldAvailable(Strin g)
	Boolean isSAPATHFieldAvailable(ISStrin g)			Integer isSAPATHFieldAvailable(Strin g)
	ULong getFileSize(Char*)			unsigned Long getFileSize(char*)
	ISString getDataOffset(ISString)			String getDataOffset(String)
	ISString getDataLength(ISString)			String getDataLength(String)
	ISString getDataType(ISString)			String getDataType(String)
	ISString getSecureCheckID2(ISString,char)			String getSecureCheckID2(String, Byte)

C++ Module	C++ Name	Class Exporting	Export Module	BOB Name
	ISString getSecureAccount(ISString, int)			String getSecureAccount(String, Integer)
	ISString getSecureValueCard(ISString)			String getSecureValueCard(String)

Writing the Prepare Function

Prepare functions follow the syntax shown in Figure 32. The pass argument is an integer indicating whether the call is for preparation of the *guiCanvas* (pass = 1) or for the retrieval and validation of data in the controls within the *guiCanvas* (pass = 2). The guiCanvas argument is an object containing a collection of controls.

```
prepareOperatorTerminalCashReport(pass, guiCanvas ; <local variables>)
```

Figure 32. Prepare Function Syntax

The provided *guiCanvas* is an *ExternalGUICanvas*. The pass parameter is an integer having a value of either 1 or 2. During pass 1, the function should populate the *guiCanvas* with controls. During pass 2, it should retrieve controls from the *guiCanvas* and get their data. Because of the export mechanism in C++ code, pass 1 populates an *ISGUIInterface* with *ISGUIControlDefinition* objects using the *ISGUIInterface::addControl* method. Likewise, pass 2 retrieves controls using the *::getControlNamed* method.

Several BOB classes have already been written to provide mechanisms for populating the canvas:

- *GUIWindow* (GUI.BOB) "Abstract" class representing a window.
- *GUIControl* (GUI.BOB) An abstract class representing a control to be placed onto the *GUIMultiCellCanvas*. It is a wrapper for the *ExternalGUIControl* class. (It sets the type for you and performs other functions.)
    - *GUIEntryField*
    - *GUIRadioButtonGroup*
    - *GUICheckBox*
    - *GUIStaticText*
    - *GUIComboBox*
- *GUIMultiCellCanvas* (GUI.BOB) represents the multi-cell canvas that the prepare pass 1 function populates with controls. It is a wrapper for the *ExternalGUICanvas*. Objects of this class provide an *ExternalGUICanvas* that they make requests of when asked to add or retrieve controls.

Figure 33 demonstrates the use of these classes and of the prepare functions. The primary task during pass 1 is to create, parameterize, and add controls. During pass 2, the primary task is to locate and gather information from controls. Information is stored in global or static variables. (Global variables begin with the $ symbol.)

```
//*******************************************************************
// Prepare the report
//*******************************************************************
prepareOfficeCashReport(pass, guiCanvas ; canvas, control)
{
  canvas = new GUIMultiCellCanvas("");
  canvas->fromExternal(guiCanvas);
  if (pass == 1)
{
    // Pass 1 creates GUI controls for parameter entry
    control = new GUIStaticText("Please enter the following:");
    control->setPosition(new Pair(1, 1));
    control->setSpread(new Pair(2, 1));
    control->addToCanvas(canvas);
    periodExtendedRadioButtonGroup(pass, canvas, 1, 3, "T");
    control = new GUIStaticText(" ");
    control->setPosition(new Pair(1, 4));
    control->addToCanvas(canvas);
  }
  else
  {
    // Pass 2 extracts entered parameters from the interface
    // Get the period
    periodExtendedRadioButtonGroup(pass, canvas, 0, 0, "T");
  }
}
```

Figure 33. Preparing a Report

Writing the Execute Function

Figure 34 is an example of a sample execute function. The given device (*physicalDevice*) is an *ISReportDevice* to which the report is output. It is brief but typical of all *execute* functions, which have these steps:

1. Create a *StandardReportDevice* [stndrdrd.bob] for the given *physicalDevice*.
2. Create an instance of some new *ISSAReport* object that represents the report to be run.
3. Make sure the instance is created without error. If so, ask it to *printOn* the *StandardReportDevice*.

```
//*******************************************************************
// Generate the report
//*******************************************************************
executeOperatorTerminalCashReport(physicalDevice ; device, report)
{
 device = new StandardReportDevice(physicalDevice);
 report = new OperatorTerminalCashReport(device->makesPermanentRecord());
 assert(report->isValid() != 0);
 if (report->isValid() == 0)
 {
   //
   //  Fatal Error:  The report is not valid
   //         ##error("oprtrtcr.bob:  error 005");
   assert(0);
  }
 report->printOn(device);
}
```

Figure 34. BOB Execute Function

All reports are ultimately subclasses of the BOB *ISReport* class [RPRTCMPN.BOB]. *ISReport* has an *ISSAReport* subclass that is the parent for most, if not all, report classes written. All *ISReports* have printOn and *printCompressedOn* methods, and member data called *theBody*. Additionally, *ISSAReport* has several static helper methods to retrieve descriptors and other such tasks.

When any *ISReport* is asked to *::printOn* a given *StandardReportDevice*, it responds by asking its data member *theBody* to *::printUsing* the device. Therefore, the purpose of the ISReport constructor is to set up *theBody* to be ready to respond to *::printUsing* by producing the output of the report.

The body of the report is meant to be set up as a hierarchial set of *ISReportComponents* [RPRTCMPN.BOB]. There are two general types of report components, those that can contain other report components (composites) and those that cannot (atomics). All are subclasses of ISReportComponent.

- When asked to *::printUsing*, atomic report components end up asking the device to output something with a request like *::insertText*, *::newLine*, or *::insertPageNumber*.
- Composite report components (all with *ISRCComposite* [rprtcmpn.bob] as their ultimate parent) respond to the *::printUsing* request by asking their report components to *::printUsing*.

Atomic report components, when constructed, let you specify the column offset in which their output should begin. Column numbering begins in column 0 (not 1).

Table 131. Atomic Report Components

ISRCPageNumber	[rprtcmpn.bob]	Outputs a string containing the page number. Contains a label and a number. You can specify the format for the label and for the number. Each defaults to `Ts`. You can specify the label text. It defaults to `Page:` if you do not specify text.
ISRCNewLine	[rprtcmpn.bob]	Outputs a new line. When you insert report components into other report components (like literal text, or a number), newLines are not generated automatically. The one exception to this is the column canvas. Therefore, you must add a newline yourself to most report components.
ISRCNewPage	[rprtcmpn.bob]	Forces a page eject. You might use this to force two sections to be on separate pages. Unlike the newLine, you need not explicitly insert page ejects into your report format. The formatter ejects the page as appropriate when enough lines have been created to fill a page.
ISRCNull	[column.bob]	A place-holder piece, NULL places nothing at a given column.
ISRCLiteral	[rprtcmpn.bob]	A literal string value. You can specify the format of your string value and the column in which to begin the component.
ISRCEvaluated	[rprtcmpn.bob]	An evaluated field. You specify a beginning column, a format, and a value to place in the field. The value you place in the field can be variable, which means that it can be calculated. In this case, the expression you provide for the calculation should be placed in _{ ... } characters. (This indicates an anonymous function.)
ISRCLabeled	[column.bob]	Puts a string label and an evaluated component onto the report format. The labeled class behaves as would a combination of an *ISRCLiteral* and an *ISRCEvaluated* field.
ISRCYesNo	[rprtcmpn.bob]	Contains a boolean expression. If the expression evaluates to TRUE at the time of formatting, it prints a YesString; otherwise, it prints a NoString.
ISRCDateTime	[rprtcmpn.bob]	Prints the date/time in the column you provide and using the format you provide.

Table 132. ISRCComposite Subclasses

ISRCSection	[rprtcmpn.bob]	This is usually the type to which *theBody* is set. A section can contain a page header and footer, and contents. You can add any type of *ISReportComponent* to an *ISRCSection*.

ISRCPageHeader	[rprtcmpn.bob]	The header text for all pages in a given ISRCSection. No header prints if the report had no contents. The header can have many lines of text.
ISRCPageFooter	[rprtcmpn.bob]	The footer text for all pages in a given *ISRCSection*. Just like the header, no footer prints on the report if there is no content on the page. The footer can have many lines of text.
ISRCTableComponent	[rprtcmpn.bob]	This component conceptually contains an SQL query and a format. When this component is encountered by the formatter, the SQL query is executed and the format is used for outputting report text for each and every item that resulted from the query. When you make a table component, you provide the SQL query to use. Then, you insert other report components as normal to define the format for the query results. Often, an *ISRCColumnCanvas* is created and inserted into the table component. The canvas contains other report components. Its behavior is similar to a multi-cell canvas.
ISRCTableUpdate Component	[rprtcmpn.bob]	A component like an *ISRCTableComponent*, except that its SQL text is for making a table update rather than a table query. This component does not print anything. Its purpose is to update a table, usually to say that some report has been taken.
ISRCParagraph	[rprtcmpn.bob]	Similar to an *ISRCSection*, but without the header and footer. *ISReportComponents* inserted into a paragraph print on the same page.
ISRCIf	[rprtcmpn.bob]	A conditional component that contains an expression and another report component. If the expression yields a TRUE value when evaluated, the report component is used. Otherwise, it is not used.
ISRCIfElse	[rprtcmpn.bob]	Similar to the *ISRCIf*, but containing two report components. One for TRUE and one for FALSE.
ISRCColumnCanvas	[column.bob]	This class does not inherit from ISRCComposite. It does contain other atomic ReportComponents, however. When creating a column canvas, you specify the number of columns you want and the offset column for the first column. The offset column value is not used for the canvas overall; each report component that it contains has its own offset. Like *MultiCellCanvas* objects in the ICLUI libraries, each cell (row,column) contains exactly one atomic report component. Therefore, when adding to column canvas objects, there is no need to explicitly add newLines. However, to leave a cell blank, you must insert an *ISRCNull* object.

Usually, *theBody* of a report is an *ISRCSection* that contains at least three elements: an *ISRCPageHeader*, an *ISRCPageFooter*, and some other component comprising the middle of the report, often an *ISRCSection* or an *ISRCTableComponent*.

Sometimes, because of a fairly complex output requirement, no assembly of these three provided *ISReportComponents* is satisfactory to build theBody for a report. In these cases, some reports introduce their own new component classes. They perform specific formatting for their own reports in their *::printUsing* method. An example of such a class is the *CashTenderReportTenderSectionComponent* and its subclass *OverShortReportTenderSectionComponent* ([cashtndr.bob]).

CashTenderReportTenderSectionComponent produces output based on options values and conditions otherwise not easily represented using the provided atomic and composite report components.

Figure 35 demonstrates how to use the provided report components to generate a report by showing an excerpt from the Miscellaneous Transaction report:

```
//*****************************************************************************
// Miscellaneous Transaction Recap Report Class
//
// STRUCTURE OF THE REPORT BODY:
//
//  theBody
//  |
//  +--- theMainTable - A query that produces exactly 1 row
//       | [mainTable]  (SELECT DateTime FROM MiscellaneousTransactionStore
//       |               WHERE file_version = ...)
//       |
//       +--- accountTable - A table that selects all info. for all accounts
//            | [account]
//            |
//            +--- accountCanvas - Detail line for one miscellaneous account.
//                                 Same format as the
//                                 accountCanvas when the report is done by list.
//*****************************************************************************
class MiscellaneousTransactionRecapReport : ISSAReport
{
 MiscellaneousTransactionRecapReport();
 theColumnHeadingFormat;
}
//*****************************************************************************
// Construct a Report
//*****************************************************************************
MiscellaneousTransactionRecapReport::
MiscellaneousTransactionRecapReport( ; reportByListID, aQuery, c,
                                       sqlText, theMainTable,
                                       accountCanvas,
                                       listTable, acctForListTable,
                                       accountTable)
{
  ISSAReport();
  //
  //  Set up the column heading format for efficiency
  //
  theColumnHeadingFormat = new TextFormat("Ts Jl S30");
```

Figure 35. Segment One: Using Report Components

Text formats, like the one above, are used in construction of several atomic report components to determine how the component is to be presented. Valid text formatting specifications are:

T<x>

Type of the component should come first in a format string so other format specifications are correctly interpreted. Valid types (values for x) are:

d

Date/time

m

Money

n

Integer

s

String (the default)

D<x>

Date. The x is optional and, if provided, specifies a format for the date. If x is provided, it must be of the format [y], where y is the format. The default is no format (""). Possible formatting entries for y:

d

Day of the month (01-31)

H

Hour in 24-hour format (01-24)

I

Hour in 12-hour format (01-12)

m

Month (01-12)

M

Minute (00-59)

p

AM or PM as appropriate

S

Second (00-59)

y

Two-digit year (00-99)

Y

Four-digit year

J<x>

Justification. Valid types (values for x) are:

c

centered

d

decimal (align the decimal points)

l

left (the default)

r

right

S<x>

Size of component (in characters). The value x is specified as a number of characters or a two numbers with a decimal point between them. The latter format specifies the number of places after the decimal point and the number before. Default size is 0.2.

$<x>

Currency symbol. Default is `$'

,<x>

Thousands separator (0 indicates none). Default is `,'

.<x>

Decimal separator (0 indicates none). Default is `.'

Z<x>

Zero money format (the string that prints when a money value is zero). Default is `.00'

[x]

A money or integer format. Can be used when type is n or m. Default money format is [m-]. Default integer format is [n-]. Possible formatting entries:

$

Put a currency symbol at this point.

+

Put a plus sign at this point if the number is >= 0.

-

Put a dash (hyphen) at this point if the number is < 0.

(

Put right parentesis at this point if the number is < 0.

)

Put right parentesis at this point if the number is < 0.

Figure 36 shows how to construct and execute an SQL Query. This report has two different constructions for *theBody*. Only one is shown here. One construction is used when there is a Miscellaneous Transaction Report list defined. The other is used when there is not. The *select* statement of this query makes a selection that results in either 0 or 1 row returned. The ::isValid request returns TRUE if the query has any data and FALSE otherwise. Therefore, if the query is valid, the reporting list exists and an integer is set accordingly.

```
//
//  Determine whether or not to report by list ID -- if list IDs
//  exist, report list by list.  Otherwise, report all
//  accounts on file.
//
reportByListID = 1;
sqlText = "SELECT MIN(recordNumber) FROM ACEMiscellaneousTransactionList";
aQuery  = new SQLQuery("aQuery", sqlText);
aQuery->execute();
if (!aQuery->isValid())
  {
   reportByListID = 0;
  }
```

Figure 36. Segment Two: Using Report Components

Figure 37 shows another excerpt of code from the Miscellaneous Transaction Report that requests data from the query. The example query retrieves the column named DateTime from the miscellaneous transaction store record for the period (file_version) requested. The global variable $PeriodName was set up during prepare pass 2. Once the query is executed, there is a check to see if it has data. It is assumed that there is always data present because the store record should always exist. Therefore, if it does not, the code uses the global *error()* function. The message contained within quotes is displayed to the operator in a message box. The *error()* function ends the execution of the BOB code and returns control to the report engine.

```
//
// Get the current period start date
//
sqlText = "SELECT DateTime "
        + "FROM MiscellaneousTransactionStore T0 "
        + "WHERE file_version = " + $PeriodName;
aQuery = new SQLQuery("aQuery", sqlText);
if (aQuery->execute())
 {
  if (!aQuery->isValid())
    {
      //
      //  Fatal Error:  The query produced no rows of data
      //
      error ("There is no store record in the misc transaction file.
        This file is required for this report. Perhaps the file is missing.");
    }
 }
 else
 {
   ISSAReport::errorAccessingData();
 }
 $PeriodStartDateTime = aQuery->getColumn("DateTime");
```

Figure 37. Segment Two: Using Report Components

If the query executed and had data, the query is currently pointing at the first row of data. You can use the *::getColumn* method to retrieve data in a specific column of query results. The *getColumn* method can also be invoked with an integer value to indicate the column position. The value begins at 1 for the first column.

```
//***************************************************************************
// Create basic components of the report
//***************************************************************************
thePageHeader = new ISRCPageHeader();
thePageFooter = new ISRCPageFooter();
theBody = new ISRCSection(nil);
//***************************************************************************
// Build the page header
//******************************************************************
thePageHeader->insert(new ISRCLiteral(15,
        reportDescriptor(3702))); // MISC TRANSACTIONS RECAP REPORT
thePageHeader->insert(new ISRCLiteral(47,
        reportDescriptor(1005) + storeId()));  // - STORE
thePageHeader->insert(new ISRCNewLine());
thePageHeader->insert(new ISRCNewLine());
thePageHeader->insert(new ISRCLiteral(1, $PeriodHeader));
thePageHeader->insert(new ISRCEvaluated(27,
       theDateTimeFormat, _{ $PeriodStartDateTime }));
thePageHeader->insert(new ISRCLiteral(45,
       reportDescriptor(1002))); // REPORTED AT
thePageHeader->insert(new ISRCDateTime(60, theDateTimeFormat));
thePageHeader->insert(new ISRCNewLine());
thePageHeader->insert(new ISRCNewLine());
thePageHeader->insert(new ISRCLiteral(1,
       reportDescriptor(3706))); // TRANSACTION TYPE
thePageHeader->insert(new ISRCLiteral(54,
       reportDescriptor(3707))); // AMOUNT
thePageHeader->insert(new ISRCNewLine());
thePageHeader->insert(new ISRCNewLine());
//*****************************************************************
// Build the page footer
//*****************************************************************
thePageFooter->insert(new ISRCNewLine());
thePageFooter->insert(new ISRCPageNumber(69));
thePageFooter->insert(new ISRCNewLine());
```

Figure 38. Creating Report Components

This section of code illustrates the constructor setting up the page header and footer. Report components are added to these two composite components by using the *::insert* method and providing a new report component to be added. This shows the user of the global *storeId* and *reportDescriptor* methods. The variable *theDateTimeFormat* is another *TextFormat* that exists in *ISSAReport*, the parent of this class.

The fourth *::insert* request made of *thePageHeader* inserts an *ISRCEvaluatedComponent* at column 27. The *theDateTimeFormat* is provided to format output, and data to output is given as `_{ $PeriodStartDateTime }`. The `_{ }` syntax is an unnamed function, an expression whose value is evaluated later when the report component is asked to *::printUsing*. This syntax appears many times in the creation of atomic report components. In this instance, the need for delayed evaluation might not be clear because once `$PeriodStartDateTime`'s value is set, it does not change during the *::printUsing* of the report. The following examples make use of this property.

```
//*****************************************************************
// Build the report's body
//*****************************************************************
//
//  NOTE:  theMainTable is designed to produce one and only one record.
//
theMainTable = new ISRCTableComponent("mainTable", sqlText);
theMainTable->beforePrintEvaluate(_{ $Total = 0 });
```

Figure 39. Building the Report Body

This example creates an *ISRCTableComponent* whose query will, as before, produce only one record. The next example shows a few things inserted into this table component, the detail component (*accountTable*), and some atomic components that report totals for this report.

```
sqlText = "SELECT * "
        + "FROM MiscellaneousTransactionAccount A "
        + "WHERE A.file_version = " + $PeriodName;
accountTable = new ISRCTableComponent("account", sqlText);
accountTable->beforePrintEvaluate(_{ $Subtotal = 0 });

accountCanvas = new ISRCColumnCanvas(1, 3);
accountTable->insert(accountCanvas);
accountCanvas->insert(0, new ISRCEvaluated(4, "Tn Jl", _{ $account.Account }));
accountCanvas->insert(1, new ISRCEvaluated(18, theColumnHeadingFormat,
                      _{ descriptorForAccount($account.Account) }));
accountCanvas->insert(2, new ISRCEvaluated(51, theMoneyFormat,
      _{ $Temp = signForAccount($account.Account),
         $Temp == "-" ? $Amount = $account.Amount * -1 : $Amount = $account.Amount,
         $Amount }));
accountCanvas->afterPrintEvaluate( _{ $Subtotal += $Amount } );
accountCanvas->afterPrintEvaluate( _{ $Total    += $Amount } );
```

Figure 40. Inserting Data in the Report

The *accountTable* is an ISRCTableComponent whose query selects all columns of all records in the *MiscellaneousTransactionAccount* table (EAMMTRN?.DAT) for the period the operator chose. All *ISReportComponents* maintain two sets of unnamed functions that are evaluated at the very beginning and end of their *::printUsing* method. The *::beforePrintEvaluate* and *::afterPrintEvaluate* methods insert a new expression into these lists. Here, an *ISRCColumnCanvas* report component is created and inserted into the table. A column canvas is sort of like a multi-cell canvas where the cells are aligned into columns. Expressions in use here are of the format:

```
 _{ $account.<columnName> }
```

When an *ISRCTableComponent* is asked to ::printUsing, it executes its SQL query. For each row of results, it asks the report components within it to *::printUsing*. Of course, the optimal solution is for report components within the *ISRCTableComponent* to print data for the *current* row of results. This is where the need for unnamed functions is more evident, in the syntax *$account.<columnName>* returns the value of *<columnName>* of the *current* row of the table named *account*. This was the name given to the *ISRCTableComponent* during its construction. In order for a different value to be output each time the atomic component is asked to *::printUsing* (for each row of the results), this expression must be reevaluated. Unnamed functions provide this ability.

```
theMainTable->insert(accountTable);
theMainTable->insert(new ISRCNewLine());
theMainTable->insert(new ISRCLiteral(   1,
     reportDescriptor(3705))); // TOTAL MISC TRANSACTIONS :
theMainTable->insert(new ISRCEvaluated(49,
     theMoneyTotalFormat, _{ $Total }));
theMainTable->insert(new ISRCLiteral(  60,
     reportDescriptor(2777)));  // **
theMainTable->insert(new ISRCNewLine());
```

Figure 41. Insertion of Report Body Lines

Insertion of the atomic components associated with the total line for the report.

```
//*******************************************************
// Build the outermost report component
//*******************************************************
theBody->insert(thePageHeader);
theBody->insert(thePageFooter);
theBody->insert(theMainTable);
//*******************************************************
// Show that the report is ready to run
//*******************************************************
theValidFlag = 1;
```

Figure 42. Insertion of Total Line for the Report

At the end of the constructor, mark the report as valid.

```
theExceptionReportNameDescriptorId = 3700;
      // "Misc Trans"
theExceptionAccountabilityIndicator = 5;
}
```

Figure 43. Marking a Report as Valid

Miscellaneous Functions

There are several miscellaneous functions that the report engine provides. Most of these reuse or only slightly change other objects in the engine, so there is no need to discuss them in detail.

View Error Log <ACERELOG>

This function is invoked by selecting View Report Error Log from the Reports menu. It generates the TurboVision cmViewLog event. The *ISTVReportEngineApplication* responds by opening an *ISTVReportLogView* [tvrelgvu.hpp/cpp]. This class is an *ISTVView* that contains a TurboVision *TFileViewer*, which permits the operator to view the contents of the ACERELOG file in a scrollable window.

Print Filed Reports

When the user selects Print Filed Reports from the Reports menu, it generates a TurboVision cmPrint event, which results in an *ISTVFileSelectionDialog* [tvfilesd.hpp/cpp] being opened. This dialog displays a list of filed reports (which are files that meet the file specification in

<ACEREINI> for filed reports). It permits the operator to choose a print or erase function for each one and it then takes the specified action.

Creating and Editing Automatic Report Scripts

The engine provides a mechanism to execute a set of reports from a script file that is formatted like an INI file (with sections and entries). To run the engine in this fashion, you can select Execute Auto Report from the Reports menu, or invoke it with an -A parameter followed by the name of the script to run.

Just as printed reports have a file specification in the <ACEREINI> file, the auto report scripts naming convention (the extension) is specified in the `AutoReports` section.

The SurePOS ACE UI provides a way to create and edit automatic report scripts. Access to the function is by selecting Edit Auto Reports on the Reports menu.

Automatic report script files are read as INI files; as such, there is a line length limit of 512 bytes. Lines which exceed this length are discarded by the underlying INI file class. Because these lines are not available for processing, querying controls for lines which exceed the length will result in an empty string. In many cases, this is not a problem; however, in some cases this can dramatically change the meaning of the report being generated and can cause unexpected results. In general, when you manually create or edit an automatic report scripts, you should adhere to screen field length restrictions, to avoid scenarios which can cause unexpected results.

Each section in a script file represents a report to be taken. The section name is not significant, although it must be unique within the script. The section must contain a `ReportName=` entry that specifies the name of the report to be taken as defined in the <ACEREINI> file.

Reports requested through an automatic report script must be output to a printer or to a file and not to a display.

- To specify a file, include the `FileName=` entry and the `OverwriteFile=` entry. The `OverwriteFile` entry is optional and defaults to FALSE.
- To specify a printer, include the `PrinterName=` entry.

Other entries in a report section specify values for various controls that would appear on the report parameter entry canvas if it were being run through the UI. The control name serves as the entry name for these entries.

Executing Automatic Report Scripts

The engine provides a user interface for store personnel to execute an auto report script.

To execute an automatic report script, the operator selects Execute Auto Report from the Reports menu. This generates the cmAutoReportRun command and results in opening the *ISTVExecuteAutoReportScriptSelectionView* [tvarslct.hpp/cpp]. This view shows the list of available automatic report scripts and permits the operator to choose one. When the operator selects one, the engine opens a *TVAutoReportStatusViewer* [tvarstvw.hpp/cpp]. Much as the *ISTVReportScroller* and *ISTVReportStatusViewer* serve to monitor the status of an executing report, so this viewer monitors the status of an executing report script.

Hints and Tips on Writing Reports

Avoid recursive functions whenever you can. Recursive functions can be very expensive in the BOB language environment.

Choose static variables wisely, because variables in the BOB language are garbage-collected. When the garbage collector executes, only variables not in use are freed. Static variables are considered always in use. Once created, they are not destroyed although you can set them to `nil`.

Do not keep instances of objects you do not want, such as *ISSQLQuery* objects. Such objects might hold files open until they are destroyed.

Test report formats by sending output to a printer or file. It is possible for two report components to overlap output on the display. You might uncover a problem that is masked by routing the report to another destination.

Appendix C. Extending SurePOS ACE

Important Note:

The key sequence, NOSALE 95 SIGNON, must not be changed via extension code. This is a U.S. Weights and Measures requirement.

Developing a Pharmacy Interface

SurePOS ACE supplies a default pharmacy interface. The interface operates by sending XML messages between Terminal Sales and a Data Integration Facility (DIF) actor for prescription information requests, and by sending XML messages between Checkout Support and a DIF actor for prescription information updates. The default actor communicates indirectly with a pharmacy application by accessing the adx_idt4:acerxfil.dat file (see "ACERXFIL (Pharmacy File)" on page 84 for the layout of this file).

Methods of Extending the Pharmacy Interface

There are two methods you can use to extend the SurePOS ACE pharmacy interface:

- You can use normal SurePOS ACE methods of subclassing and overriding member functions if you only need to slightly alter the interface. This is true both for SurePOS ACE code and the DIF actor. Javadoc information on the Java classes used to implement the DIF actor can be found in `c:\sp_ace\v4\javadist\dif_doc` (assuming the toolkit was installed in the `c:\sp_ace\v4` directory).
- You can write your own DIF actor if you need to implement an interface to a new pharmacy application. (For detailed information about creating a pharmacy actor, refer to the *Data Integration Facility: User's Guide*.)

If the current functionality in Terminal Sales and Checkout Support is sufficient, then developers only need to correctly interact with SurePOS ACE by observing the current message protocols. These messages have headers that correspond to the DIF message protocol. Messages requesting pharmacy information have a Message-Profile-Id=PharmacyRequest, while messages updating pharmacy information have Message-Profile-Id=PharmacyUpdate. The bodies of the messages are XML and are described in "XML Messages Used by Pharmacy Interface" on page 632. Figure 44 shows when the messages are used. Both Terminal Sales and Checkout Support initiate the request/response communication. The acknowledgment sent to Checkout Support requires no body; it is not parsed by Checkout Support.

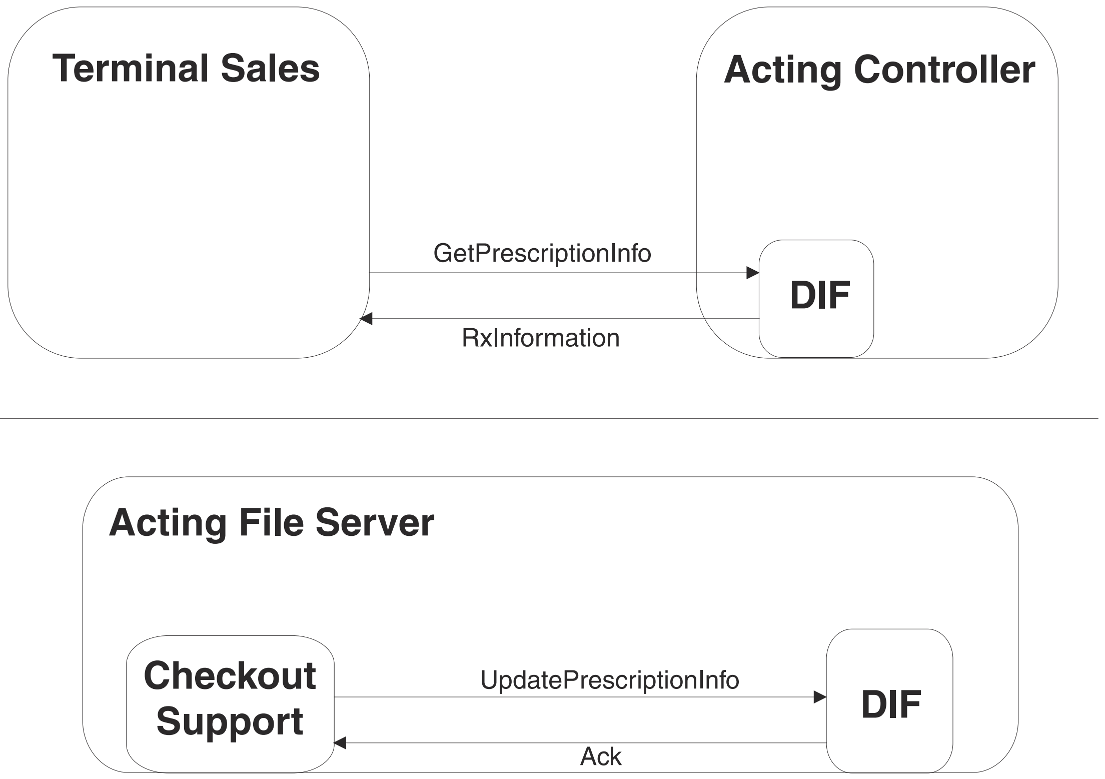


Figure 44. Message Flow for Pharmacy Interface

XML Messages Used by Pharmacy Interface

This section contains information about the XML messages that SurePOS ACE uses to communicate with the pharmacy application.

GetPrescriptionInfo

The Terminal Sales application send this message to the pharmacy application to request information about a prescription. This example requests information about prescription 3000306.

```
<GetPrescriptionInfo>
     <RxTransactionNumber>3000306</RxTransactionNumber>
</GetPrescriptionInfo>
```

RxInformation

The pharmacy application uses this message to send information to the Terminal Sales application. This example sends information about prescription 3000306.

```
<RxInformation>
     <TransactionNumber>000003000306</TransactionNumber>
     <Price>001568</Price>
     <LineItemDiscountableFlag>false</LineItemDiscountableFlag>
     <AlreadySoldFlag>false</AlreadySoldFlag>
</RxInformation>
```

UpdatePrescriptionInfo

Checkout Support sends this message to the pharmacy application to inform it of a prescription sale. This example is for a transaction that contains prescriptions 3000306 and 3000314.

```
<UpdatePrescriptionInfo>
    <RxPrescription>
         <TransactionNumber>3000306</TransactionNumber>
         <Price>1568</Price>
         <TotalPrice>1568</TotalPrice>
         <VoidFlag>false</VoidFlag>
         <TransactionTime>1438</TransactionTime>
         <TransactionDate>050119</TransactionDate>
         <Operator>1</Operator>
         <POSTransactionNumber>3</POSTransactionNumber>
         <Terminal>1</Terminal>
         <Checkpoint>9</Checkpoint>
         <Department>1</Department>
    </RxPrescription>
    <RxPrescription>
         <TransactionNumber>3000314</TransactionNumber>
         <Price>1000</Price>
         <TotalPrice>1000</TotalPrice>
         <VoidFlag>false</VoidFlag>
         <TransactionTime>1438</TransactionTime>
         <TransactionDate>050119</TransactionDate>
         <Operator>1</Operator>
         <POSTransactionNumber>3</POSTransactionNumber>
         <Terminal>1</Terminal>
         <Checkpoint>9</Checkpoint>
         <Department>1</Department>
    </RxPrescription>
</UpdatePrescriptionInfo>
```

Building and Installing Your Extensions

If your extensions require new Java classes, you must construct your own build system to build them. The SurePOS ACE Development Toolkit cannot be configured to build the Java classes for you. Your build system should produce one or more jar files. To include these jar files in the construction of a 4690 installation package:

1. Ensure that the jar files are located in the `c:\sp_ace\v4\javadist` directory (assuming that the SurePOS ACE toolkit was installed in the `c:\sp_ace\v4` directory).
2. Add the jar files to the package list.

You must ensure that your Java classes can be found and loaded by DIF.

1. Create a file called `difuser.bat` that adds your jar files to the logical file name USERJARS (see the *Data Integration Facility: User's Guide* for more information on the `difuser.bat` file). Place the `difuser.bat` file in the `c:\sp_ace\v4\javadist` directory (assuming that the SurePOS ACE toolkit was installed in the c:\sp_ace\v4 directory).
2. Add the `difuser.bat` file to the package list.

If you are creating your own actor, you need to configure DIF properly to route pharmacy messages to your actor and not to the default pharmacy actor that is provided by SurePOS ACE. Refer to the DIF documentation and to the acedif.pro file for detailed information. You should ensure that the difuser.pro file does not include the acedif.pro file in the properties chain for your solution.

After you have created all the new files that you need and have added them to the package, run prepPkg to create the installation image. (See “Generating Installation Packages for a 4690 System” on page 559 for more information about the prepPkg command.)

Extending Dump Support

The ISReloaderListener code has been written to allow Business Partners and customers to extend the code that is used for dumping the controller after an application abends. In most cases, the SurePOS ACE extension will be coded to override the base ISReloaderListener::dumpAllowedNow() method.

Overriding the Base Reloader Listener Class for All .386 Applications

To override the base reloader listener class for all .386 applications:

- Develop extension code, using standard extension techniques.
- Use a static declaration to instantiate the extended singleton.
- Using the `customer.ini` file, append customer objects to the `ACE_TARGET_OS_OPTIONAL_OBJS` variable.

Overriding the Base Reloader Listener Class for A Single .386 Application

To override the base reloader listener class for a single .386 application:

- Develop extension code, using standard extension techniques.
- Use a static declaration to instantiate the extended singleton.
- Using the `customer.ini` file, append customer objects to add the objects to the executable's optional function list *`executableName`*`_OPTIONAL_FCN`.

Removing Dump On Exception for All .386 Applications

To disable dumping the controller after an application abend (in effect, set the `DumpOnException` keyword in all sections of the `acedmp.ini` file to *N*):

- Include `dumpex` in the value of the `ACE_OPTIONAL_FCN_EXCLUSIONS` variable. See “Optional Function (Toshiba and Customer)” on page 563.
- Erase all targets.
- Make flat 4690 targets.

After you have removed dumping on exception, you can enable the function for individual applications by adding `rldrlson$O` to the executable's optional function list `executableName_OPTIONAL_FCN` in the `customer.ini` file.

Enabling New .386 Applications to Support Dump On Exception

As long as the code is enabled (that is, `dumpex` is not excluded), all programs built for 4690 will be enabled for the dump on exception function. To enable the function for your own executables, just add appropriate entries to an `acedmp.usx` overlay file following the same process as you would use for enabling base programs.

Access to Other Options Files During Initialization

The *optionsLoaded()* event is raised with a pointer to an *ISOptions* object. That object contains the options found in the EAMOPTNS.DAT file and, if applicable, EAMOPxxx terminal specific options or EAMOPGxx terminal group options, merged together.

In some cases, it is necessary for extension code to access options contained within other ACE options files, such as the APSOPTNS or NLSOPTNS files. In earlier releases of ACE (prior to version 6), the options file would need to be read by the extension code to access the options values by calling *blRead()*. Each caller to this method would cause a read of the options file from the controller disk. The result is that the APSOPTNS and NLSOPTNS files were read multiple times during initialization, both by the base application and extension code.

Starting with ACE v6, a new method has been implemented to preload the additional options files and make them available to extenders during the options loaded event. Extenders should review any calls to *blRead()* and replace these with calls to *getOptionsObject()*, to obtain a pointer to the preloaded options object. For example:

```
// Change object to pointer
// ISOptions NLSoptions;
ISOptions * NLSoptions;

// Remove call to blread
//result = NLSoptions.blRead("<$LSOPTNS>");
// Add call to getOptionsObject
ISResult readResult =
         ISOptionsStorage::instance().getOptionsObject("<$LSOPTNS>", NLSoptions);

// Change user code to use pointer...
```

Adding Descriptors to the Descriptor Cache during Initialization

A new method has been added for the descriptor cache so that business partners can add descriptors to the cache during initialization. This avoids disk access to read the descriptor file after the point at which the descriptors are loaded into cache.

The new method is *loadMoreSalesDescriptors()* in `tsnlsfct.cpp`. You can override this method. An example of the code to write to add sales descriptors is provided in the method.

Rounding Methods in SurePOS ACE

This section describes the various rounding methods in ACE and when they should be used.

Price Rounding Policy

Use this rounding policy when calculating the prices of multipriced items. Use the ISRoundingPolicy::instance().roundPrice() method for this purpose.

Weight Rounding Policy

Use this rounding policy when calculating the extended price of weighted items. This rounding policy does not apply to the weight of an item nor to the unit price of a weighted item, but is applied to the product of the weight times the unit price. Use the ISRoundingPolicy::instance().roundPrice() method for this purpose. This method will use the weight rounding option for weight items.

Fuel Rounding Policy

Use this rounding policy when calculating the extended price of fuel items. This rounding policy does not apply to the volume of a fuel item nor to the unit price of a fuel item, but is applied to the product of the volume times the unit price. Use the ISRoundingPolicy::instance().roundFuelPrice() method for this purpose.

Discount Rounding Policy

Use this rounding policy when calculating transaction discounts which are associated with the discount key, line item discounts, and department discounts. Use the ISRoundingPolicy::instance().roundDiscount() method for this purpose.

Points Rounding Policy

Use this rounding policy when calculating loyalty points. Use the ISRoundingPolicy::instance().roundPoints() for this purpose.

Coupon Price Rounding

Coupon price rounding always uses conventional rounding. Use the ISRoundingPolicy::instance().roundCouponPrice() method for this purpose.

Currency Rounding

Currency rounding will always use conventional rounding.

Tare Weight Rounding

Tare weight rounding will always use conventional rounding.

GS1 DataBar Application Identifier support in SurePOS ACE

The list of supported Application Identifiers for GS1 DataBar expanded bar codes will be maintained in a protected dictionary in ISItemCodeFormatFactory::theExpandedItemCodeFormats.

This dictionary will be populated by an internal table with the AIs supported by SurePOS ACE as listed in the *Application Identifiers on GS1 DataBar expanded bar codes (non-coupon)* table in the SurePOS ACE: Planning and Installation Guide. If additional AIs need to be considered *in use* in order to be processed by custom code, then a public setter (that is, ISItemCodeFormatFactory::instance().setGS1ExpandedFormatInUse(anAI)) will be available. The parameter is an ISString and should represent the AI key to the dictionary.

After a GS1 DataBar label is scanned, any AI formats that are *in use* and detected on the label will be stored in the ISItemCode object in theExpandedBarCodeFields, that is a protected ordered collection and has the following accessor methods:

```
//Retrieves the field indicated by the AI in theExpandedBarCodeFields collection.
ISExpandedBarCodeField* getExpandedField(ISString anAI) const;
//Returns TRUE if the field indicated by the AI is present in the collection
Boolean hasExpandedField(ISString anAI) const;
```

If custom code is needed to support a quantity and a weight or to support a quantity and extended price in a GS1 DataBar label, then the following methods will need to be overridden:

```
//Return TRUE if the barcode contains an encoded quantity, has no encoded price
 or weight in the same barcode, and the barcode does not begin with "019" that
implies a mandatory price
//or mandatory weight.
Boolean ISItemCode::hasQuantity(void) const
```

```
//Return TRUE if the option 'Quantity Required' is allowed for the item.
This option is ignored and the user will not be prompted to input quantity if
the GS1 barcode contains an encoded price/weight or begins with "019",
 or the 'Wt/Price Required' option is enabled.
Boolean ISAddItemCommand::shouldCheckQuantityRequired(void)
```

Appendix D. Keys on the Terminal Keyboard

This appendix identifies the set of keys that is recognized by the SurePOS ACE terminal application. You must select required keys from this set. Only 37 function keys are available on the keyboard, along with 10 numeric keys, and the asterisk (*) key. Some keys, such as ENTER and CLEAR, often use two function key positions.

This appendix also describes the terminal state table, input data positions, input modifier key positions, and how to define Tax Code keys or redefine the TAX/NO TAX key for use with the Goods and Services Tax (GST) function.

Required Keys

The following keys are required of all users of the application.

10 Numerics
48-57 (System Defined)

SIGN ON/OFF
61

VOID
70

CLEAR
73

Asterisk (*)
78

OVERRIDE
79

ENTER
80

TOTAL
81- The range of 900-999 has been reserved for customers and business partners to use with the TOTAL key.

SUSPEND/RETRIEVE
82

CASH
91

NO SALE
100

Optional Keyboard Functions

You can also assign these keyboard functions:

DISCOUNT
62

DATA ENTRY
63

DEPOSIT
64

MFR COUPON
65

STORE COUPON
66

REFUND
67

TARE
71

WEIGHT
72

PRICE
74

QTY
75

FS/NO FS
76

TAX/NO TAX
77

SUBTOTAL Key
88. SurePOS ACE supports function code 88 for use as a key to subtotal a transaction without replaying delayed coupons. In addition to adding this key to the keyboard layout, you must also add function code 88 to your state tables. You can do this easily by using function code 81 (TOTAL) as a model.

Customer ID Key
89. This key is not included in the default keyboard layouts. If you want to use this key, you must add it to the keyboard layout. If you have migrated from a release of SurePOS ACE earlier than V1R5, you must manually add function code 89 to your state tables.

VERIFY
90

CHECK
92

FOOD STAMP
93

Misc Tender Keys

94-96

Procedures

101-119, where 101 = procedure 1 (Tender Cashing), 102 = procedure 2 (Tender Exchange) and so on up to 119 = procedure 19 (Reserved)

Note: Key 113 is assigned to the WIC transaction procedure. You can define a different key (such as 120) for WIC transactions in Personalization. However, if key 113 is not used for WIC transactions, it cannot be used for anything else.

Unassigned Matrix Keyboard Keys

130-190

Tax Code Keys

191-198, where 191 = Tax Code 1 that toggles Tax Plan 1 taxability when using the Goods and Services Tax (GST), 192 = Tax Code 2 that toggles Tax Plan 2 taxability, 193 = Tax Code 3 for Tax Plan 3, 194 = Tax Code 4 for Tax Plan 4, 195 = Tax Code 5 for Tax Plan 5, 196 = Tax Code 6 for Tax Plan 6, 197 = Tax Code 7 for Tax Plan 7, and 198 = Tax Code 8 for Tax Plan 8.

Assigned Department Keys

200-208, where 200 is Department 1, and so on, up to 208, which is Department 9.

Unassigned Department Keys

209-239, where the code specifies sequential departments, from 209, which is Department 10, up to 239, which is Department 40.

Open Department Key

240. This key allows you to sell multiple departments using only one key on the keyboard, instead of requiring a separate key for each department.

This key is not included in the default keyboard layouts. If you want to use this key, you must add it to the keyboard layout.

Reserved

241. This key is reserved for use with Toshiba Store Integrator

Foreign Currency Key

242. This key is not included in the default keyboard layouts. If you want to use this key, you must add it to the keyboard layout.

Single Tender Key

243. This key is reserved for use as the single tender key if SurePOS ACE EPS has been installed.

Display Reason Codes Key

244. This key is not included in the default keyboard layouts. If you want to use this key, you must add it to the keyboard layout.

This key has the following functions:

- If it is pressed while the prompt for a price-override reason code is displayed, it displays the list of price-override reason codes.
- If it is pressed while the prompt for a refund reason code is displayed, it displays the list of refund reason codes.

- If it is pressed while the prompt for a void reason code is displayed, it displays the list of void reason codes.
- If it is pressed while none of these three prompts is displayed and all reason code functions are active, the list of refund reason codes is displayed. If it is pressed while none of these three prompts is displayed and no reason code functions are active, the `B003 CHECK KEY SEQUENCE` message is displayed.

Display the Velocity Code Chart

245. This key, which is sometimes called the PLU key, is used with acevcode.dat, the Velocity Code Chart file.

Page Up Through the Velocity Code Chart

246. This key is used with acevcode.dat, the Velocity Code Chart file, the Reason Code Lists, and the text full screen receipt window.

Page Down Through the Velocity Code Chart

247. This key is used with acevcode.dat, the Velocity Code Chart file, the Reason Code Lists, and the text full screen receipt window.

Scroll Up Through the Velocity Code Chart

248. This key is used with acevcode.dat, the Velocity Code Chart file.

Scroll Down Through the Velocity Code Chart

249. This key is used with acevcode.dat, the Velocity Code Chart file.

Reserved Key

250

Rain Check Key

251. This key is used with Rain Check entry.

Reserved Key

252

Extended Price Key

253. This key is not included in the default keyboard layouts. If you want to use this key, you must add it to the keyboard layout.

Volume Key

254. This key is not included in the default keyboard layouts. If you want to use this key, you must add it to the keyboard layout.

Checkout Transaction Key

255. This key is not included in the default keyboard layouts. If you want to use this key, you must add it to the keyboard layout.

Terminal State Definition Table

Terminal key sequences are implemented through a state table that defines allowable key sequences to the I/O processor that handles data entry at the terminal. The state table, along with validity checks in the SurePOS ACE code, determines key sequences that are valid at any given time. The table also defines the size of input data fields, whether data is displayed as keyed, and whether input is allowed from the scanner.

The default state table provided by SurePOS ACE is eams@000. It defines standard keying sequences.

The table is on the installation CDs. States defined by the table and function keys allowed in the state are listed in Table 133.

Table 133. Terminal States Table

STATE #/	POSITION:										
NAME:	0	1	2	3	4	5	6	7	8	9	10
1 CLEAR	clear	--	--	--	--	--	--	--	--	--	--
2 SIGNON	--	--	--	"*"	--	--	--	--	--	--	--
3 SIGNON-2	--	--	OVER-RIDE	--	--	SIGNON	--	--	--	--	--
4 SO-PSWD	--	--	--	--	--	--	--	--	--	--	ENTER
5 SPEC-SO	--	--	--	--	--	SIGNON	--	--	--	--	--
6 OVERRIDE	--	CLEAR	--	--	--	--	--	--	--	--	OVER-RIDE; ENTER
9 ENT/CLR	--	--	enter	--	--	--	--	--	--	--	--
10 MAIN	clear	void sus-pend	ENTER tx/notx procedu re Cust ID	"*"	depos it NO-SALE	DEPT PRICE RAIN CHECK DIS-COUNT SIGNON OVERRIDE Unassign Matrix; Kybd Keys	WEIGHT QUANT-ITY total VOLUME	fs/nofs TENDERS	coupons refund	TARE VERIFY Extended Price	DATA; Open Dept. Key; Tax Code Keys; SIF Key
11 ENT/DAT	--	CLEAR	--	--	--	--	--	--	--	--	ENTER
12 STANDALN	--	--	enter	"*"	--	--	--	--	--	--	--
13 ENTER	enter	CLEAR	--	--	--	--	--	--	--	--	--
20 TC/TE	--	void	pro-cedure	"*"	nosale	SIGNON	total	TENDERS	refund	VERIFY	DATA
21 CL/CP/TC	--	void	pro-cedure	"*"	nosale	SIGNON	total QUANT-ITY	TENDERS COUPONS	--	--	DATA
22 T/LIST	--	--	enter pro-cedure	--	nosale	SIGNON	total	tenders	--	--	--
23 TT/TMON	--	--	ENTER pro-cedure	--	nosale	SIGNON	total	--	--	--	--
24 PR/CHG	--	--	ENTER pro-cedure	"*" OVER-RIDE	nosale	SIGNON PRICE dept.	total	--	--	--	--

The following notes apply to the terminal state table in Table 133:

1. Capitalized function key names imply that data is allowed or required with the function key.
2. The CLEAR key can be entered in any state to clear previously entered numerics unless it is specifically defined in the state to behave differently. This global definition of the CLEAR key goes in position 0, which is never passed back to the application.
3. The VOID key can be entered in any state. It is passed to the application in position 1 with no data allowed.
4. The CLEAR state is defined to allow the application to get control when the CLEAR key is pressed.
5. Multiple states are required for sign-on to avoid displaying a password as keyed.
6. States SPEC-SO, SO-PSWD, OVERRIDE, ACCTNO, and ACCT/OVR are used only when prompting for a password, override number, account number, or CLEAR key. ENT/DAT is used when prompting for other data such as price, weight, quantity, or fees.
7. The ENT/CLR state is used only with the external verification prompt to retry or bypass external verification after a timeout.
8. The ENTER state is used only in conjunction with the OVERRIDE state and is required to handle scanned input.
9. The MAIN state handles all input that occurs in or between checkout transactions except for input which is specially prompted for on the display.
10. All non-sale procedures except for operator training use single self-contained states. The following names are used in Table 133.

    TC/TE
    Tender Cashing and Tender Exchange

    CL/CP/TC
    Cashier Loan, Cashier Pickup, and Tender Count

    T/LIST
    Tender Listing

    TT/TMON
    Terminal Transfer and Terminal Monitor

    PR/CHG
    Price Verify/Change
11. The Asterisk (*) key requires data in every state except for STANDALN and CL/CP/TC where data is not allowed.

Function Code and Data Arrays

Function codes and data are passed to the terminal sales application in arrays of ten elements each. Positions of function numbers and data correspond to their positions in the state table.

Input Data Positions

The following keyboard functions are defined with data. Their function numbers appear in the listed positions:

Input Data	Input Position	Associated Function
Item Code	2	ENTER
New Password	2	OVERRIDE

Input Data	Input Position	Associated Function
Terminal #	2	ENTER
Operator #	3	Asterisk (*)
Expiration Date	3	Asterisk (*)
Deal Quantity/Weight	3	Asterisk (*)
Tender Variety	3	Asterisk (*)
Discount Rate	3	Asterisk (*)
Transaction Data	4	NO SALE
Password	5	SIGN ON/OFF
Procedure #	5	SIGN ON/OFF
Price	5	PRICE, Lookup Key, or Rain Check
Discount Group	5	DISCOUNT
Weight	6	WEIGHT
Quantity	6	QTY
Volume	6	VOLUME
Tender Amounts	7	Tender Key or Coupon Key
Customer Account #	9	VERIFY
Tare Value	9	TARE
Extended Price	9	Extended Price
Data Entry	10	DATA ENTRY
Override #	10	OVERRIDE
Department #	10	Open Department Key

Input Modifier Key Position

The following keyboard functions are sometimes defined with no data allowed. Their function numbers appear in the listed positions:

VOID
1

TAX/NO TAX
2

Asterisk (*)
3

NO SALE
4

DEPOSIT
4

Lookup Key
5

FS/NO FS
7

Tender Key
7

REFUND
8

MFR COUPON
8

STORE COUPON
8

OVERRIDE
8

Installing the Tax Exemption and Tax Reversal Functions

The Goods and Services tax (GST) function lets you change the tax status of items during checkout and give customers tax exemptions from individual tax plans. If you do not need these functions, you need not perform these procedures to change the input state table.

If you want to use these functions, there are two methods for entering tax plans during a sales transaction. You can use either or both of these methods:

- Change the TAX/NO TAX key to accept more tax plans than the default tax plan, which it is already defined to work with.

  Changing the key lets a cashier press 1-8 with the TAX/NO TAX key to toggle specific tax plans.
- Add from one to eight TAX CODE keys, each for a specific tax plan.

  Adding TAX CODE keys lets a cashier press one to toggle the appropriate tax plan.

Using TAX CODE Keyboard Functions Only

To use TAX CODE keyboard functions only, perform these procedures:

1. "Adding TAX CODE Keys" on page 649
2. Activate the changes to the adx_ipgm\eams@000 Input State Table.
3. Configure the terminal keyboard. For more information about configuring keyboards and including them in the terminal definition, refer to the *4690 Operating System: User's Guide.*
4. Reload terminal configuration data.
5. Reload controller storage. For more information about loading controller storage, refer to the *4690 Operating System: User's Guide.*

Using the TAX/NO TAX Keyboard Function Only

To use the TAX/NO TAX keyboard function only, perform these procedures:

1. "Changing the TAX/NO TAX Key Definition" on page 647

2. Activate the changes to the adx_ipgm\eams@000 Input State Table.
3. Reload controller storage. For more information about loading controller storage, refer to the *4690 Operating System: User's Guide*.

Using TAX CODE Keyboard Functions and the TAX/NO TAX Keyboard Function

To use TAX CODE keyboard functions and the TAX/NO TAX keyboard function, perform these procedures:

1. “Changing the TAX/NO TAX Key Definition” on page 647
2. “Adding TAX CODE Keys” on page 649
3. Activate the changes to the adx_ipgm\eams@000 Input State Table.
4. Configure the terminal keyboard. For more information about configuring keyboards and including them in the terminal definition, refer to the *4690 Operating System: User's Guide*.
5. Reload terminal configuration data.
6. Reload controller storage. For more information about loading controller storage, refer to the *4690 Operating System: User's Guide*.

Navigating the Input Sequence Table

The *4690 OS: Programming Guide* describes how to update the Input Sequence Table. Here is an overview of how to get to panels that let you make the changes described in this section:

1. Choose option 4, Installation and Update Aids, from the System Main Menu.
2. Select option 3, Change Input Sequence Table Data.
3. Select option 1, Input State.
4. Select option 1, Change a Table from the Input State Table menu, after entering the name of the table, `ADX_IPGM\EAMS@000`.
5. Select option 2, State Definitions, from the Input State Table menu.
6. Select option 1, Change State Definitions, from the States of the Table menu.
7. Type N in response to the prompt, `Change all state definitions`.
8. Type MAIN in response to the prompt, `Type the name of the state to be processed` and press ENTER.
9. Press ENTER to display the Function Codes in the State Main menu, which is where the other procedures begin.

Changing the TAX/NO TAX Key Definition

See “Navigating the Input Sequence Table” on page 647 for information about how to display the Function Codes in the State Main panel.

To change the TAX/NO TAX key, make the specified entries on each of these panels:

1. Function Codes in the State Main

   Type your selection, then press Enter.

   1
2. Press ENTER.
3. Type N in response to the *Change all Function Code definitions* prompt and press ENTER.

4. Type 77 in response to the *Function Code definitions to be processed* prompt and press ENTER.
5. State Main Function Code 77

   Motor Key
   N

   Clear Key
   N

   Notification
   Y

   Message Line 1
   Both lines of this field should be empty.

   Next State
   Current

   Page down to go to the next panel.
6. State Main Function Code 77 / Data Requirement Information

   Requirement
   3

   Message Lines
   Both of these fields should be empty.

   Next State
   Current

   Page down to go to the next panel.
7. State Main Function Code 77 / Data Characteristics of the Function Code

   Keyed Label
   N

   Data Precede
   Y

   Display
   1

   Saved Data
   N

   Page down to go to the next panel.
8. State Main Function Code 77 / Data Length Check Information

   Min. Data Length
   0

   Max. Data Length
   4

   Next State
   Current

Page down to go to the next panel.

9. State Main Function Code 77 / Data Value Check Information

Need Data Value Check

N

Page down to go to the next panel.

10. State Main Function Code 77 / Relative Position Information

Position

10

Mutually Exclusives

All zeroes (0)

Page down to go to the next panel.

11. State Main Function Code 77 / Action for Violation of Mutually Exclusive Positions

Message Lines

Both of these fields should be empty.

Next State

Current

Page down to go to the next panel.

12. State Main Function Code 77 / Manager's Key Requirement Information

Required

N

13. Page up to review your input. Press ENTER to save your changes. Press F3 to return to the Input State Table menu.

Adding TAX CODE Keys

See "Navigating the Input Sequence Table" on page 647 for information about how to display the Function Codes in the State Main panel.

To add TAX CODE keys, make the specified entries on each of these panels:

1. Function Codes in the State Main

```
Type your selection, then press Enter.   2
```

2. Press ENTER.
3. Respond to the *Function code definitions to be processed* prompt with any of these values:

191	TAX CODE 1 key for Tax Plan 1
192	TAX CODE 2 key for Tax Plan 2
193	TAX CODE 3 key for Tax Plan 3
194	TAX CODE 4 key for Tax Plan 4
195	TAX CODE 5 key for Tax Plan 5
196	TAX CODE 6 key for Tax Plan 6
197	TAX CODE 7 key for Tax Plan 7
198	TAX CODE 8 key for Tax Plan 8

191-198	All eight TAX CODE keys for all eight tax plans

4. State Main Function Code 191 (These examples show how to define function code 191. Each function code definition is the same.)

Motor Key	Y
Clear Key	N
Notification	Y
Message Line 1	Both lines of this field should be empty.
Next State	Current

   Page down to go to the next panel.
5. State Main Function Code 191 / Data Requirement Information

Requirement	1
Message Lines	Both of these fields should be empty.
Next State	Current

   Page down to go to the next panel.
6. State Main Function Code 191 / Relative Position Information

Position	10
Mutually Exclusives	All zeroes (0)

   Page down to go to the next panel.
7. State Main Function Code 191 / Action for Violation of Mutually Exclusive Positions

Message Lines	Both of these fields should be empty.
Next State	Current

   Page down to go to the next panel.
8. State Main Function Code 191 / Manager's Key Requirement Information

Required	N

9. Page up to review your input. Press ENTER to save your changes. You can add other TAX CODE keys now or press F3 to return to the Input State Table menu.

Appendix E. The Totals Retention Area

Each terminal keeps necessary information in a battery-protected totals retention area called the *hard totals area* or *non-volatile RAM* (NVRAM). SurePOS ACE references the first 240 bytes in the five records shown below in direct mode. Accessing hard totals is described in detail in the *Toshiba Global Commerce Solutions 4680 BASIC: Language Reference* and the *Toshiba Global Commerce Solutions 4690 Operating System: Programming Guide.*

This layout is for information purposes only. Changing the data in the hard totals area can seriously impact data integrity and normal SurePOS ACE function.

Data object reference	*ISHardTotals* <HARDTTLS.CPP>
Organization	Random
Distribution class	Not applicable
File copies	Current
Record length	56 bytes (sequential mode); 56 bytes (direct mode)

Note: When ACE is initializing, during transaction recovery, it will check to see whether it is running in partial failover mode. If so, it ignores and clears all transaction data in NVRAM and the Totals Save file. If there was an EPS transaction in progress in NVRAM, the EPS tender is reversed with a Time Out Reversal (TOR), so that it is not lost when clearing. The next transaction number to use is stored in the Terminal Storage file, and it will be used to ensure that the next transaction number is correct.

Record 1

Variable	Type	Length	Offset	Description
LastCloseDateTime	PD	5	0	Date and time of last close in format: `YYMMDDHHmm`.
The rest of this record and the first three bytes of Record 2 contains the Sign On Session Checkpoint, which are values at the beginning of this sign-on session.				
NumLoans	INT	2	5	Count of loans to this till.
AmtLoans	INT	4	7	Total amount of loans to this till.
NumPkups	INT	2	11	Number of pickups from this till.
AmtPkups	INT	4	13	Total amount of pickups from this till.
GrossPos	INT	4	17	Gross positive net total for this sign-on session. It is the accumulation of positive sales entries. It does not include totals from voided or training transactions. It is *sales* + *deposits* + *taxes* + *tender fees*.
GrossNeg	INT	4	21	Gross negative net total for this sign-on session. It is the accumulation of negative sales entries. It does not include totals from voided or training transactions. It is *deposit returns* + *item refunds* + *tax refunds* + *discounts* + *cancels of (sales, deposits, item refunds, deposit returns, and tender fees)*.

Variable	Type	Length	Offset	Description
NumTrans	INT	2	25	Number of transactions in this sign-on session.
TillContents	Array	8x4	27	This represents the following eight 4-byte till content fields as of the end of the last transaction:
AmtCash	INT	4	27	Entered tender - cash
AmtCheck	INT	4	31	Entered tender - checks
AmtFoods	INT	4	35	Entered tender - food stamps
AmtMisc1	INT	4	39	Entered tender - miscellaneous tender 1
AmtMisc2	INT	4	43	Entered tender - miscellaneous tender 2
AmtMisc3	INT	4	47	Entered tender - miscellaneous tender 3
AmtManuf	INT	4	51	Entered tender - manufacturer coupons
AmtStore	INT	1	55	First byte of 4-byte AmtStore field, which records the use of store coupons as entered tender. The last three bytes of the field are at the beginning of Record 2.

Record 2

Variable	Type	Length	Offset	Description
AmtStore	INT	3	0	Last three bytes of 4-byte AmtStore field, which records the use of store coupons as entered tender. The first byte of the field is at the end of Record 2.
Reserved	INT	1	3	Reserved.
The rest of this record and the first eight bytes of Record 3 contains the Terminal Status Data File Checkpoint, which are values written successfully to file during the last write from the terminal.				
TransNum	PD	2	4	Transaction number that was last written to file.
NumLoans	INT	2	6	Count of loans to this till that was last written to file.
AmtLoans	INT	4	8	Total amount of loans to this till that was last written to file.
NumPkups	INT	2	12	Count of pickups from this till that was last written to file.
AmtPkups	INT	4	14	Total amount of pickups from this till that was last written to file.
GrossPos	INT	4	18	Gross positive net total for this sign-on session that was last written to file. It is the accumulation of positive sales entries. It does not include totals from voided or training transactions. It is *sales* + *deposits* + *taxes* + *tender fees*.
GrossNeg	INT	4	22	Gross negative net total for this sign-on session that was last written to file. It is the accumulation of negative sales entries. It does not include totals from

Variable	Type	Length	Offset	Description
				voided or training transactions. It is *deposit returns + item refunds + tax refunds + discounts + cancels of (sales, deposits, item refunds, deposit returns, and tender fees)*.
AmtMiscs	INT	4	26	Miscellaneous item entries amount that was last written to file.
NumTrans	INT	2	30	Number of sales transactions that have been written to file.
TillContents	Array	6x4	32	This represents the first six elements of an eight-element 4-byte till content array, which records till contents as of the end of the last transaction. The last two elements are at the beginning of the next record.
AmtCash	INT	4	32	Entered tender - cash
AmtCheck	INT	4	36	Entered tender - checks
AmtFoods	INT	4	40	Entered tender - food stamps
AmtMisc1	INT	4	44	Entered tender - miscellaneous tender 1
AmtMisc2	INT	4	48	Entered tender - miscellaneous tender 2
AmtMisc3	INT	4	52	Entered tender - miscellaneous tender 3

Record 3

Field Name	Type	Length	Decimal Offset	Description
TillContents	Array	2x4	0	This represents the last two elements of an eight-element 4-byte till content array, which records till contents as of the end of the last transaction. The first six elements are at the end of the last record.
AmtManuf	INT	4	0	Entered tender - manufacturer coupons
AmtStore	INT	4	4	Entered tender - store coupons
The rest of this record and all of Record 4 contains the Terminal Status Data Golden Copy, which are values written to NVRAM during the last write from the terminal. These values are written to NVRAM even if the write to file fails.				
Terminal	PD	2	8	The terminal number. The first character of this field indicates whether the record is for the current period or a previous period: **0** Current period **1** Previous period

Field Name	Type	Length	Decimal Offset	Description
Operator	PD	5	10	Operator number of current or most recent user of this terminal.
TransNum	PD	2	15	Transaction number of the current or most recent transaction.
NumLoans	INT	2	17	Number of loans to this till.
AmtLoans	INT	4	19	Amount of loans to this till.
NumPkups	INT	2	23	Number of pickups from this till.
AmtPkups	INT	4	25	Amount of pickups from this till.
The next four fields are net totals for this sign-on session:				
GrossPos	INT	4	29	Gross positive. Accumulation of the positive sales entries. Includes stand-alone totals but no totals from voided or training transactions (sales + deposits + taxes + tender fees).
GrossNeg	INT	4	33	Gross negative. Accumulation of the negative sales entries. Includes no totals from voided or training transactions (deposits returns + item refunds + tax refunds + discounts + cancels of sales, deposits, item refunds, deposit returns, and tender fees).
AmtMiscs	INT	4	37	Amount of miscellaneous item record entries.
NumTrans	INT	2	41	Number of sales transactions.
TillContents	Array	3.5x4	43	This represents the first three and one-half elements of an eight-element 4-byte till contents array, which records till contents that are to be written to file. The last four and one-half elements are at the beginning of the next record.
AmtCash	INT	4	43	Entered tender - cash
AmtCheck	INT	4	47	Entered tender - checks
AmtFoods	INT	4	51	Entered tender - food stamps
AmtMisc1	INT	1	55	The first byte of the four-byte AmtMisc1 field, which records the use of miscellaneous tender 1 as the entered tender. The last three bytes are at the beginning of the next record.

Record 4

Field Name	Type	Length	Decimal Offset	Description
AmtMisc1	INT	3	0	The last three bytes of the four-byte AmtMisc1 field, which records the use of miscellaneous tender 1 as the

Field Name	Type	Length	Decimal Offset	Description
				entered tender. The first byte is at the end of the previous record.
AmtMisc2	INT	4	3	Entered tender - miscellaneous tender 2
AmtMisc3	INT	4	7	Entered tender - miscellaneous tender 3
AmtManuf	INT	4	11	The amount of tendered manufacturer coupons.
AmtStore	INT	4	15	Entered tender - store coupons
The next two fields contain status indicators:				
TranType	PD	1	19	Transaction type in process: **00** Checkout transaction **01** Tender cashing **02** Tender exchange **03** Cashier loan **04** Cashier pickup **05** Tender listing **06** Price verify/change **07** Training session **08** Terminal transfer **09** Terminal monitor **10** Tender count **12** Non-EBT WIC

Field Name	Type	Length	Decimal Offset	Description
				13 Return item transaction **16** Reprint tender receipt **20** EBT balance inquiry **21** Value card balance inquiry **22** WIC EBT balance inquiry **80** Department totals report **99** No transaction in process
Status	INT	2	20	**Bit 7 X'8000'** Reserved2: Reserved. **Bit 6 X'4000'** ForceSignOff: Force a sign-off when the next transaction is not in progress. **Bit 5 X'2000'** ReOpenLog: Re-open the summary log at the next transaction. **Bit 4 X'1000'** ForceTillExchange: Force a till exchange at the next transaction start. **Bit 3 X'0800'** ReloadOptions: Reload options. **Bit 2 X'0400'** ProhibitSignOn: Prohibit sign-on. **Bit 1 X'0200'** ProhibitSignOff: Prohibit sign-off. **Bit 0 X'0100'** ProhibitTransStart: Prohibit transaction start.

Field Name	Type	Length	Decimal Offset	Description
				Bit15 X'0080' ProhibitTransEnd: Prohibit transaction end. **Bit14 X'0040'** TerminalMonitor: Terminal is being monitored. **Bit13 X'0020'** TerminalTransfer: Terminal has been transferred. **Bit12 X'0010'** TerminalSpecialSignOff: Terminal is in special sign-off. **Bit11 X'0008'** TerminalAccountability: Terminal accountability. **Bit10 X'0004'** OperatorSignedOn: Operator signed-on to terminal. **Bit 9 X'0002'** TenderVerification: Out-of-store verification of tenders is active. **Bit 8 X'0001'** Reserved1: Reserved.
UsrExit	INT	2	22	Reserved for user extensions.
Reserved	INT	26	24	Reserved.
VersionStamp	ASCII	4	50	SurePOS ACE version.
Reserved	INT	2	54	Reserved.

Record 5

Field Name	Type	Length	Decimal Offset	Description
Reserved	INT	2	0	Reserved.
Failover	INT	4	2	Reserved for Failover (see Table 134 for the layout).
EPSReserved	ASCII	40	6	Reserved for EPS (see Table 135 for the layout).
TrxStartOffset	Int	4	46	Offset of the next transaction to despool. A value of 0 indicates start despooling at the beginning of the file.

Field Name	Type	Length	Decimal Offset	Description
TrxEndOffset	Int	4	50	Offset of the last transaction to despool. A value of 0 or less than the TrxStartOffset value indicates there are no transactions to despool.
TrxNumStart	PD	2	54	Transaction number of the next transaction to despool. This field and the TrxNumEnd field are used to log the transaction numbers of 1) unrecoverable transactions after a system restarts and 2) transactions not despooled due to processing a store close.

Table 134 shows the fields in the Failover Totals Retention Area, which begins at decimal offset 2 in Record 5.

Table 134. Failover Totals Retention Area

Data Type	Length	Decimal Offset	Description
Integer	1	2	Action being requested: **X'01'** Flush **X'02'** Reset
Integer	2	3	Start of sequential portion of NVRAM (offset from beginning of file).
Integer	1	5	Reserved

Table 135 shows the fields in the EPS Totals Retention Area, which begins at decimal offset 6 in Record 5.

Table 135. EPS Totals Retention Area

Data Type	Length	Decimal Offset	Description
ASCII	3	6	Characters EPS
ASCII	1	9	In-flight flag (set on if an authorization request is in-flight).
Integer	4	10	POS transaction number.
Integer	4	14	Sequence number of the EPS message.
ASCII	12	18	Date and time fields from the request message.
ASCII	16	30	Reserved for future use (will be filled with hex zeros).

Record 6

Field Name	Type	Length	Decimal Offset	Description
TrxNumEnd	PD	2	0	Transaction number of last transaction to despool.

Field Name	Type	Length	Decimal Offset	Description
FlashSize	Int	4	2	Size of the user flash sector at the time of the last load.
EJStartOffset	Int	4	6	Offset of the next EJ data to despool.
EJEndOffset	Int	4	10	Offset of the last EJ data to despool.
EJChecksum	Int	4	14	Checksum of the user data area for the user printer after the last write.
Taxes 1-8	Int	4*8	18	Tax amounts 1-8
IncludedTax 1-8	Int	4*8	50	Included tax amounts 1-8. These amounts are eight 4-byte fields. The first six bytes are at the end of record 6, and the remaining 26 bytes are at the beginning of record 7.

Record 7

Field Name	Type	Length	Decimal Offset	Description
IncludedTax 1-8	Int	4*8	0	Included tax amounts 1-8. These amounts are eight 4-byte fields. The first six bytes are at the end of record 6, and the remaining 26 bytes are at the beginning of record 7.
TaxableAmounts 1-8	Int	4*8	26	Taxable amounts 1-8. These amounts are eight 4-byte fields. The first 30 bytes are at the end of record 7, and the remaining 2 bytes are at the beginning of record 8.

Record 8

Field Name	Type	Length	Decimal Offset	Description
TaxableAmounts 1-8	Int	4*8	0	Taxable amounts 1-8. These amounts are eight 4-byte fields. The first 30 bytes are at the end of record 7, and the remaining 2 bytes are at the beginning of record 8.
Flags	Int	2	2	**X'01'** TOF mode active **X'02'** Log exit and discard details
Till Status Loan Count	Int	2	4	Till loan count
Till Status Pickup Count	Int	2	6	Till pickup count

Field Name	Type	Length	Decimal Offset	Description
Till Status Loan Amount	Int	4	8	Till loan amount
Till Status Pickup Amount	Int	4	12	Till pickup amount
Till Status Gross Positive	Int	4	16	TOF gross positive amount
Till Status Gross Negative	Int	4	20	TOF gross negative amount
Till Status Till Transaction Count	Int	2	24	TOF transaction count
Till Status Tenders 1-8	Int	4*8	26	TOF tenders. These amounts are eight 4-byte fields. The first 30 bytes are at the end of record 8, and the remaining 2 bytes are at the beginning of record 9.

Record 9

Field Name	Type	Length	Decimal Offset	Description
Till Status Tenders 1-8	Int	4*8	0	TOF tenders. These amounts are eight 4-byte fields. The first 30 bytes are at the end of record 8, and the remaining 2 bytes are at the beginning of record 9.
TOFHardTotalsOffsetPriceChange Count	Int	2	2	Offline price change count
LastKnownPrimaryController	ASCII	2	4	The 2-character ID of the last known primary controller for this terminal.
LastKnownBackupController	ASCII	2	6	The 2-character ID of the last known backup controller for this terminal. Note: The terminal must enter backup in order to update this field.
Terminal Number	ASCII	4	8	The number of the terminal that attempted the check tender.
Transaction Number	ASCII	4	12	The transaction number in which the check tender was attempted.
TenderAmount	ASCII	8	16	The amount of the check tender.
MICRLength	ASCII	2	24	The length of the check's MICR.
MICRData	ASCII	50	26	The contents of the check's MICR. The data field is 50 bytes. The first 30 bytes are at the end of Record 9, and the remaining 20 bytes are at the beginning of Record 10.

Record 10

The data that is stored in records 9 and 10 pertains to the last failed check tender, if one exists and if check imaging is enabled. It is used to write a reversal record to the check image file. Upon writing the reversal, the data is cleared. The data stored in records 10 and 11 pertains to the validation of fiscal printers.

Field Name	Type	Length	Decimal Offset	Description
MICRData	ASCII	50	0	The contents of the check's MICR. The data field is 50 bytes. The first 30 bytes are at the end of record 9, and the remaining 20 bytes are at the beginning of record 10.
User	ASCII	10	20	The user data.
Fiscal Validation Signature	ASCII	3	30	The signature for fiscal validation records.
Fiscal Printer Serial Number	INT	32	33	An encrypted serial number used to validate a Fiscal Printer. The first 23 bytes are at the end of Record 10, and the remaining 9 bytes are at the beginning of Record 11.

Record 11

The following data is stored in Record 11.

Field Name	Type	Length	Decimal Offset	Description
Fiscal Printer Serial Number	INT	32	0	An encrypted serial number used to validate a Fiscal Printer. The first 23 bytes are at the end of Record 10, and the remaining 9 bytes are at the beginning of Record 11.
Fiscal Printer Information	INT	16	9	Additional encrypted information that is used to validate a Fiscal Printer.

Appendix F. Heap Enhancements for Debugging and Problem Analysis

SurePOS ACE has been enhanced to make it easier to detect memory leaks and/or corruption in the SurePOS ACE heap. Memory leaks can be subtle, where the leak is so slow it takes several weeks of running in a store before the program runs out of memory. This enhancement provides the capability for leaks and corruption to be discovered with a lot less trouble than before.

The code for this heap checking enhancement is always linked into SurePOS ACE to make it easier to debug problems by enabling the heap checking support for a particular program or terminal. The code does not affect the operation of SurePOS ACE when checking is disabled. The heap checker is actually a *wrapper* around the current heap. It does not replace the heap implementation with custom code. Instead, it replaces the default `new` and `delete` operators with functions that call allocation routines in the heap checker code. The heap checker code increases the size of the allocation requested by the caller and then calls underlying allocation functions. When the memory block has been allocated, the heap checker code places its own information in the beginning of the memory block (caller and other information) and then returns to the caller, a pointer to rest of the memory block. If the heap checker is disabled, the original heap implementation is called.

Most of SurePOS ACE uses the `new` and `delete` operators for memory allocation. So, replacing these functions with custom functions takes care of most memory blocks allocated by SurePOS ACE. However, `malloc` and `free` are used in several places in SurePOS ACE (mostly by the RogueWave code). Unfortunately, it is difficult to replace `malloc` and `free` with custom functions because the C runtime uses these functions to allocate memory for its own uses (including code that runs during runtime initialization). Because the heap checking code also uses C runtime functions (for opening files, etc.), it might indirectly cause memory to be allocated via `malloc`. To handle this, there would have to be additional code to direct memory operations either to the true heap or to the heap checker.

Instead of replacing `malloc` and `free`, this enhancement uses a simpler method, which is to ensure that any time SurePOS ACE source code calls `malloc` or `free`, the heap checker code is used instead. Something like this is already being done for the 4690 OS target code. The C header files stdlib.h and malloc.h contain compiler macros used to redirect calls from `malloc` and `free` to `_is_malloc` and `_is_free`. To ensure proper tracking of the use of these functions, this enhancement simply replaced these functions (which call the runtime heap manager) with new versions that call the heap checker code. This enhancement does something similar for target builds in a Windows development environment. The file swami.h, which should be included by all SurePOS ACE code, now includes btmalloc.h that contains compiler macros to redirect `malloc` and `free` to functions in the heap checker code. In cases where the heap checker code is not linked in, there are default implementations of the replacement functions that call the original `malloc` and `free` functions.

The SurePOS ACE heap debugger works under Toshiba Global Commerce Solutions 4690 OS and under Windows.

Functionality

The heap checker provides some benefit without requiring you to change your code. However, to debug specific leakage problems might require you to add code that calls the heap checker API. Through changes in the configuration file (and without additional coding) the heap checker has this functionality:

- The basic code can be enabled or disabled. When the heap checker is enabled, a flag can be set to dump the contents of the heap on termination (which can be sort of useful for seeing what memory is still allocated).
- Memory wiping can be enabled or disabled. When it is enabled, newly allocated memory is wiped with a special value before being returned to the application. Memory is also wiped when an application frees it. This helps catch cases where program code accesses memory that is not allocated or makes assumptions about the contents of newly allocated memory.
- Deferred memory freeing can be enabled. This is useful in cases where memory blocks are being overwritten after they have been freed.
- Corruption checking can be individually enabled or disabled. When corruption checking is enabled, the heap will log errors when a caller corrupts the heap by changing memory beyond the end of their allocated region. The heap checker can also be configured to occasionally check the entire heap for corruption.
- The name, number, and location of heap log files and other minor configuration values can be specified.

You can use the following additional capabilities by writing some additional code:

- *Setting or adjusting heap checkpoints.* The checkpoint is basically a global number variable. Every time memory is allocated, the current checkpoint value is recorded with the memory block. By setting the checkpoint to a particular value, or incrementing it at the beginning of every transaction, you can use the checkpoint value to identify why a particular memory block was allocated.
- *Dumping all memory blocks that were allocated within a specific checkpoint range to the log file.* This is useful if, for example, you initially set the checkpoint to 1 and then increment it after the end of every transaction. After a few transactions, you would expect that all the memory allocated during the first transaction should be freed. In this case, you could dump all memory allocated during checkpoint 1 to see if you have a memory leak.
- *Adding user-defined messages to the log file.*
- *Fetching or setting the frequency of automatic heap checking.* You can also check the heap for corruption.
- *Dumping the contents of all allocated memory blocks to the log file or only dumping the contents of blocks that were allocated since the last dump or reset.*

For more specific information on each of the API calls and configuration settings, see the sections below.

Invocations

All API functions do nothing (but return a zero return code) if you disable heap checking.

Settings-Related Functions

void btSetValidationFrequency(unsigned short freq)

Sets the frequency at which the list of allocated memory blocks in the heap is checked for corruption. When the frequency is set to 0, then automatic heap checking is disabled. When the frequency is non-0, then an internal counter is incremented every time a memory block is allocated or freed. When this counter is greater or equal to the frequency value, the btValidateHeap() function is called to check for heap corruption. This also sets the counter to 0. See the btValidateHeap() for additional details on heap checking. The default validationFrequency is 0.

unsigned short btGetValidationFrequency()

Returns the current value *validationFrequency*. The default validationFrequency is 0.

void btSetBlockDumpLimit(unsigned long newLimit)

Sets the maximum number of bytes per memory block that is output to the log file when dumping the contents of a heap block. Memory blocks in the heap are dumped to the log file whenever corruption is detected or during calls to *btDumpAllocated()*, *btDumpAllocatedDelta()*, and *btReportLeakage()*. The default *blockDumpLimit* is `0xFFFFFFFF`.

unsigned long btGetBlockDumpLimit()

Returns the current value of *blockDumpLimit*.

void btSetDeferredSettings(const btDeferredSettings & newSets)

Sets the deferred free queue settings. When deferred freeing is active, memory blocks are not freed immediately. Instead, the memory blocks are filled with a known byte pattern and placed in a deferred free queue. When the heap is validated, the contents of the deferred blocks are checked to ensure that user code has not changed them. (If so, an error is logged.) Enabling deferred freeing is useful when you think that some code is trying to change memory after it has already been freed.

The heap checking level must be `checkpoints` or higher in order for the deferred freeing settings to have any effect.

Values of the `btDeferredSettings` structure determine how the queue of deferred blocks is managed. Two of the settings control the minimum and maximum sizes of the blocks placed in the deferred queue. If a block is outside the range, it is not added to the queue and is freed from the heap immediately. The other two settings control the maximum number of deferred bytes and memory blocks that can be in the queue at any one time. If the queue exceeds these limits after a new block is added, the oldest blocks in the queue are removed (and freed) until the queue size is within limits.

The following fields are in the `btDeferredSettings` structure:

- *unsigned long maxBlocks* The maximum number of memory blocks in the deferred free queue at any one time. New blocks added that exceed the limit cause the oldest blocks in the queue to be removed until the queue size is within limits. The default value for `maxBlocks` is `0xFFFFFFFF` (which is no maximum).
- *unsigned long maxMem* The maximum number of bytes in the deferred free queue at any one time. New blocks added that exceed the limit cause the oldest blocks in the queue to be removed until the queue size is within limits. The default value for `maxMem` is `0`, which disables deferred freeing.
- *unsigned long minBlockSize* The minimum size block that is added to the deferred free queue. Blocks smaller than `minBlockSize` are freed immediately. *Only the user's portion of a memory block is considered in size calculations.* The default value for `minBlockSize` is `0`.
- *unsigned long maxBlockSize* The maximum size block that is added to the deferred free queue. Blocks larger than `maxBlockSize` are freed immediately. *Only the user's portion of a*

memory block is considered in size calculations. The default value for `maxBlockSize` is `16384` (16 Kb).

void btGetDeferredSettings(btDeferredSettings & output)

Returns the current values of the deferred free queue settings.

void btSetExitOnErrorRc(long exitRc)

Sets a value that controls the operation of the heap checker when an error is detected. If the exitRc is non-0, then the heap checker will exit the process when a heap error is detected; ending the process will be accomplished by calling *exit()* with the value of exitRc. If exitRc is 0, the heap checker will continue operation (returning the error to the caller if possible). Heap errors which cause the program to exit include: out of memory, corruption detected during validation, invalid memory block freed or reallocated. The default value of exitRc is 0.

int btGetExitOnErrorRc(void)

Returns the current value of *exitOnErrorRc*.

void btSetDumpOnTerm(int flag)

Controls whether the heap is dumped on termination. If the flag is `BT_DUMP_ON_TERM_STATS`, then *btDumpHeapStats()* is called when the process exits normally. If the flag is `BT_DUMP_ON_TERM_DELTA`, then *btDumpAllocatedDelta()* is called. If the flag is `BT_DUMP_ON_TERM_FULL`, then *btDumpAllocated()* is called. (For either delta or full dumps on termination, heap statistics are dumped as well.) If the flag is `BT_DUMP_ON_TERM_NONE`, no dump is done. See "Using the Heap Checker" on page 679 for a method of using this flag with SurePOS ACE.

The destructor of the heap checker object calls the dump function. The destructor runs after all other destructors of global objects in the system. The default value of `dumpOnTerm` is `BT_DUMP_ON_TERM_NONE`.

int btGetDumpOnTerm()

Returns the current value of *dumpOnTerm*.

void btSetBeepOnError(int flag)

Controls whether or not a beep will be sounded when errors (or leaks) are detected. If true, a beep is sounded. If false, a beep is not sounded. Beeping currently works only on targets in a Windows environment; beeping does not work on 4690 OS targets.

The default value of *beepOnError* is `false`.

int btGetBeepOnError()

Returns the current value of *beepOnError*.

void btSetLogAllHeapOps(int flag)

Controls the *verbosity* of the heap control code. When the flag is true, the heap control code logs *all* memory operations (like successful allocations and frees). This is useful in cases where you must know where a certain pointer is freed, for example. However, logging all memory operations can create *immense* log files. You can use the *btSetLogAllHeapOps()* function at runtime to set the verbosity selectively.

The default value of *logAllHeapOps* is `false`.

int btGetLogAllHeapOps()

Returns the current value of *logAllHeapOps*.

Heap Control Functions

void btSetCheckpoint(unsigned short newCp)

Sets the checkpoint value for the heap to the given value. All memory blocks allocated after this point will be given a checkpoint value of *newCp*. You can use *btReportLeakage* to log all allocated memory blocks with checkpoint values within a given range. The initial checkpoint value is 0.

unsigned short btGetCheckpoint()

Returns the current *checkpoint* value.

void btAdjustCheckpoint(short increment)

Adjusts the current checkpoint value by the given increment (which can be positive or negative). The checkpoint value won't wrap below 0 or beyond the limit of an unsigned short.

void btIncrementCheckpoint()

Increments the current checkpoint value by 1. The checkpoint value will not wrap if it is already at the limit of an unsigned short.

void btDecrementCheckpoint()

Decrements the current checkpoint value by 1. The checkpoint value will not wrap if it is already 0.

unsigned long btReportLeakage(unsigned short startCp, unsigned short endCp)

Dumps to the log file all allocated memory blocks that have a checkpoint value in the range *startCp* to *endCp* inclusively.

Returns the number of leaked or invalid blocks detected. It returns zero (0) if there are no leaks or invalid blocks.

int btValidateHeap()

Iterates through all allocated memory blocks in the heap and attempts to ensure that each block is valid. (It checks application code has not accidentally overwritten sentinels at the beginning and end of each block.)

This function is only really useful when corruption checking is enabled. In the case where corruption checking is *not* enabled, this function iterates through all the blocks of the heap but might not detect invalid blocks and could cause the program to trap. (As it would if the heap checking code were totally disabled.)

Returns a boolean value indicating if the heap is valid. If any invalid blocks are detected, this function returns `0`, which is false. Otherwise, it returns a non-zero value.

void btDumpAllocated()

Dumps the contents of all allocated memory blocks to the log file.

At the checkpoint and corruption checking levels, the memory blocks are dumped in order of their allocation. At the minimal level, the order will depend on the order in which they are stored in the underlying runtime heap.

void btDumpReset()

Causes the list of memory blocks tracked by *btDumpAllocatedDelta* to be reset. See *btDumpAllocatedDelta* for more information.

void btDumpAllocatedDelta()

Dumps the contents of any currently allocated memory blocks that were allocated since the last call to *btDumpAllocated*, *btDumpAllocatedDelta*, or *btDumpReset*.

The *btDumpXxx* functions provide a simple memory checkpointing scheme without resorting to checkpoints. For example, you can call *btDumpReset* before a lengthy operation and after it, call *btDumpAllocatedDelta* to report all memory allocated during the operation but not yet freed.

The checking level must be set to `checkpoints` or higher for this to work properly. In minimal mode, calling this function is equivalent to calling *btDumpAllocated()*.

void btGetHeapStats(btHeapStats &stats)

Fills in the parameter value with statistics about the heap. The following fields are in the `btHeapStats` structure:

- *unsigned long allocatedBlocks* The number of heap blocks currently allocated.
- *unsigned long maxBlocks* The maximum number of blocks allocated at any one time.
- *unsigned long totalBlocks* The total number of blocks allocated since the heap started operation.
- *unsigned long allocatedMem* The number of bytes of user memory currently allocated. (This does not include the overhead of the heap checker or runtime code).
- *unsigned long maxMem* The maximum number of bytes of memory allocated at any one time.
- *unsigned long totalMem* The total number of bytes of memory allocated since the heap started operation.
- *unsigned long deferredBlocks* The total number of blocks in the deferred free queue.
- *unsigned long deferredMem* The total number of bytes in the deferred free queue.
- *btBlockSizeHistogram totalsHistogram* A histogram object used to track the sizes of all member blocks ever allocated by user code. Refer to the comments for the `btBlockSizeHistogram` structure in btheap.hpp for more information about data contained in the histogram object.

void btDumpHeapStats()

Records the current heap statistics in the log file. It also records call stack information to help identify where statistics were logged.

void btLogUserMsg(const char *pMsg)

Logs a user-generated message in the log file. The `pMsg` argument is a pointer to a null-terminated string of characters to log. It also logs the call stack at the point of this call.

int btIsEnabled()

Returns true (`non-0`) if the heap checking code is enabled and false (`0`) if it is disabled.

Interactions

This section discusses the various tools that are related to the heap checker code

TYPE / TAIL

In some situations, it might be useful to monitor the contents of a text heap checker log file at runtime. An example of this is if you are attempting to recreate a heap corruption and wish to stop and examine the log file as soon as the corruption occurred.

One way to take a *snapshot* of the log file is to use the `TYPE` command to display the contents of the log file. If there is a large amount of data in the file, output from the `type` command can be redirected to a file as shown in this example:

```
type heaptest.h00 > out
```

You can also use the UNIX command `tail` to monitor the contents of log files in a more interactive manner. It shows the last few lines of a file. (You can control the number of lines with the `-n` flag.) Output from the `type` command can be piped to the `tail` command to see the last event in the file. In addition, some implementations of `tail` also support the `-f` flag that displays any data appended to the file as it grows. Thus, using a command such as `tail -f heaptest.h00` lets you use a separate window to interactively monitor any new events appended to a text log file.

The `-f` flag for `ACEPPHLG` that is described below behaves much like the `-f` flag for tail.

ACEPPHLG

The ACEPPHLG utility *post-processes* heap checker log files, primarily to convert binary log files to human readable form. However, you can also use it for text log files in cases where the heap checker cannot load the debug information at runtime. In these cases, caller addresses in the log file are prefixed with two tilde characters. These logs can be post-processed at a time when debug information is available.

In order to resolve function names and line number information in a log file, ACEPPHLG reads in compiler debug information. Under 4690 OS, this information is contained in the debug version of the .386 files that are generated during the SurePOS ACE build process. In a Windows environment, debug information is contained within the application's `.exe` file itself. In either case, ACEPPHLG must be able to locate the file containing the debug information. By default, ACEPPHLG looks for the file in the current directory. If the debug information file is in a different directory, use the `-d:` flag to specify the directory name. (The file name that ACEPPHLG looks for is stored in the log file.)

To post-process a binary log for which there is no file containing debug information (and you cannot regenerate it), use the `-i` flag of ACEPPHLG. In this case, all addresses are unresolved.

It is very important to ensure that debug information used to post-process a log file is identical to the application that created the log file. Otherwise, caller information is not correct. (There are incorrect line numbers, function names, and module names listed in the call stacks.) To help prevent mismatches, the SurePOS ACE build process embeds a version ID in both the application and debug information. This ID is also in the header of the file. If the version ID in the log does not match the version ID in the debug information file, ACEPPHLG reports the mismatch and does not attempt to process the log file. The `-i` flag makes ACEPPHLG ignore the error and attempt to use whatever information is available.

This tool also supports a form of interactive log file monitoring, similar to using the `tail -f` command to display updates to a text log file. When run with the `-f` flag, ACEPPHLG reads as much of the log file as it can and processes and displays all data that it finds. It then checks the file every second for additional data to process. This works with both binary and text log files. The only way to stop ACEPPHLG in interactive mode is to use `Ctrl-C` or `Ctrl-Break`. When using this flag, you typically want the output to be directed to `stdout`. (In that case, do not specify an output file on the command line.)

ACEGDGDB

The primary use of ACEDBGDB is as a debugging tool. It reads in a debug information file (a VisualAge® C/C++ EXE or .386 file) and builds an in-memory database of debug information. It then enters interactive mode and lets you type in code addresses. If a code address matches an entry in the database, ACEDBGDB prints out the module name, function name, and line number for that address. This is the same database used by the heap checker code and by ACEPPHLG to resolve call stack addresses.

You can also use ACEDBGDB to dump debugging information, either directly from the .386 file or after the database is built. You can use the resulting output to find the address of a particular function or line of code.

Configuration Specifications

Most of the heap checker code is configurable at runtime versus compile time. The configuration is done via a configuration file that is similar to the INI files used to configure SurePOS ACE. The name of the file is ACEHPCHK.CFG. Under 4690 OS and a supported Windows operating system, the file must be in the directory given by the logical name `<ACE:>`. Typically, you want this file to be on every controller in the store so that everyone sees the same settings. Therefore, when you install SurePOS ACE, mark the default configuration file as a compound file distributed on close so that everyone will see it.

A single configuration file is used for all programs and terminals, but the format of the file allows you to set varying options for different programs and terminals. The file consists of a set of section names (enclosed in square brackets) and a set of `keyword = value` pairs as in INI files. The section name specifies the application name and optional terminal ID that the set of options that follow apply to. The `keyword = value` pairs are the options values themselves.

The section name consists of the name of the application followed by an optional terminal or controller ID. An asterisk character can be used as a wildcard to match any application name or ID. If a terminal or controller ID is not given, the set of options is used for all terminals or controllers.

The CFG file is read a section at a time from top to bottom. As each section is encountered, the name of the current application is compared to the section name. If the names match (or the application name is an asterisk), then the following `keyword = value` pairs are used to update the heap checker options. If the value portion of a `keyword=value` pair is unrecognized or invalid, it is ignored and the option retains its current value. The current value might be the default value or the value set in a previous matching file section. In most cases, a blank value for an option is also considered to be invalid.

If the section name contains a colon, any text after the colon is considered to be a controller or terminal ID. For the set of options to be used, this ID must match the ID of the controller or terminal on which the application is being executed. For terminal applications, the value is compared to the ID of the terminal. For controller applications, the value is compared to the controller ID. (Applications running in a Windows environment are considered to be controller applications with an ID of "*". Thus, only sections with a missing ID or an ID of "*" will match in a Windows environment.)

CFG file processing continues even after a matching set of options has been read in. This allows global options to be given for all applications near the top of the file and then more specific options to be given later in the file. The global options can be used to set the most commonly used options for all applications and yet have the `checkingLevel` set to `disabled`. You can enable the heap checker individually for specific applications and/or terminals.

Lines beginning with a semicolon are considered comments and are ignored. Unrecognized option names and invalid option values are also ignored. Any lines that are not within a valid section are ignored.

Configuration Keywords

The following keywords are recognized in the heap checker configuration file.

checkingLevel = disabled | minimal | checkpoints | corruption

This setting controls the overall level of things done by the heap checker code. When the heap checker is enabled, it replaces the default new and delete operators with code that increases the amount of memory in each memory block and uses the extra space to hold additional data.

The following settings are available:

disabled

At this level, all checking code is disabled and only the default runtime heap calls are used to allocate memory. All of the heap checker API functions (btXxxXxx()) are also disabled and do nothing if called.

minimal

At this level, each user allocation is increased in size by 10 bytes. The extra data is used to hold information about the calling function that allocated the memory block. This option is intended to be used only in memory constrained environments. At this level checkpoints and a call stack are not supported. In addition, the memory dump may incorrectly identify blocks as being allocated by the memory checker when they really haven't been.

checkpoints

At this level, 20 extra bytes per block are allocated plus 4 bytes per entry in the call stack. Check point values are stored in each block and when the heap is dumped, blocks are dumped in the order in which they were allocated (which can make debugging easier).

corruption

At this level, 20 extra bytes per block are allocated plus 4 bytes per entry in the call stack and any additional space required for the sentinels. At this level, sentinels and check values are added to each memory block to attempt to catch heap memory corruption. Sentinels are designed to catch cases where a memory block is overwritten by a few bytes. This is the level at which the heap validation functions become useful.

The default `checkingLevel` is disabled.

callStackSize = 1...16

This is the number of entries to place in the call stack in the header of each memory block that is allocated. This information is displayed during memory dumps and lets you identify the caller of the memory. At the minimal level, only the ID of the caller is stored in a block. At all other checking levels, the maximum value for this field is 16.

The default value of callStackSize is 3.

logFileDir = directory_name

This is the directory name used to store the heap checker log files. All status and error output from the heap checking code is written to this file. The directory name can contain SurePOS ACE-style logical names. If the resolved name does not end in the directory separator, one will be appended to it. For applications running on a terminal, the prefix R:: is automatically added to the directory name.

By default this setting is <ADX_IDT4:>. If the directory name is blank, the log file will be placed in the current directory.

A new log file is created or opened every time an application using the heap checking code is started. For controller apps, the base file name will be the same as the name of the application. For terminal apps, the name will consist of the first 5 characters of the application name (padded with underscores if necessary) followed by the 3 character terminal ID.

The extension of the file consists of the letter H followed by a two digit number from 0 to 99, for example: H00, H01, ..., H99. The maximum number of log files created for an application is controlled by the maxLogFiles configuration parameter. To avoid overwriting existing log files, the heap checker code chooses an extension that is not already being used. If all extensions are already in use, the file with the oldest date is overwritten.

logFileType = binary | text | depends

This controls whether the log file is written in binary or text mode. A binary log file is smaller and written more quickly at runtime than a text log file. However, a binary file is not in a form that humans can read. You can run the post-processor tool ACEPPHLG at a later time to convert a binary log file to text mode. (See "ACEPPHLG" on page 670 for more information on this tool.)

When a text log file is created, the heap checker code attempts to load debug information to resolve function and module names. This increases memory requirements and processing time as events are logged. If there is a problem loading debug information, it still creates a text log file. Function addresses in the file are preceded by special characters (two tildes). ACEPPHLG can also post-process this file to resolve the addresses.

If the flag is set to binary, the heap checker code creates a binary log file. If set to text, it creates a text log file. If set to depends, it creates a binary file on 4690 OS and a text file in a Windows environment. The depends setting is a good default to use on both 4690 OS and Windows for several reasons. Because debug versions of the .386 files are not shipped to customer sites, it does not make sense to create a text log file by default on 4690 OS because addresses cannot be resolved to function and module names. However, in a Windows environment, the debug information is contained within the EXE itself and is always available.

Windows also uses virtual memory and is typically run on faster machines, so the additional memory and CPU requirements of a text log file are of much less concern.

Support for loading debug information (.386 files) is not compiled into the heap checker code in 4690 OS, primarily to save space. This means that text mode log files in 4690 OS never have their symbols resolved, which requires the use of ACEPPHLG to post-process the files.

The default value for logFileType is depends.

cacheAllDebugInfo = true | false

Every time a memory block is allocated or a heap function called, the heap checker code stores the function address and call stack of the code that called it. When event data is written to the log file (and the log file is not in binary mode), the binary addresses are resolved to function and module names. In order to correctly identify the function and module names, the heap checker code reads the debugger information for the application. For 4690 OS applications, the information is read directly from the .386 file (which must have been linked with the linker flag that includes the debug information). For Windows applications, the information is read directly from the EXE file (which must have been linked with the linker flag that includes the debug information).

If a text mode log file is being generated, this option controls whether the full set of debug information is read during initialization. If it is true, the entire contents of the debug information file is read when the heap checker is initializing and then the file is closed. If it is false, the file is kept open during execution and debug information is read from the file as needed. For example, when `btReportLeakage()` or `btDumpAllocated()` is called and the heap checker code needs to determine the module name and line number that allocated a memory block, it reads the symbol and line number information from the file.

The advantage to setting this flag to `false` is that only the parts of the file that are needed are read. The amount of time and memory taken up by the heap checker during program initialization is thereby reduced. The advantage to setting this flag to `true` is to ensure that the debug information database does not consume any time or memory resources at *random* places during the later execution of the application.

The default value of `cacheAllDebugInfo` is `false`.

maxLogFiles

This is the maximum number of log files to allow. A special value of `0` can be used during debugging to force the extension to HLG. This makes it easier to find the correct log file instead of hunting for it.

The default value for `maxLogFiles` is `16`.

firstSentinelsize

This indicates the number of sentinel bytes to place at the beginning of the user's heap block when corruption checking is enabled.

The default value for `firstSentinelSize` is `4`.

lastSentinelSize

This indicates the number of sentinel bytes to place at the end of the user's heap block when corruption checking is enabled.

The default value for `lastSentinelSize` is `4`.

memoryWiping = true | false

Controls whether memory blocks are *wiped* with special values. If this value is `true`, memory blocks are filled with non-0 values immediately after allocation and before being returned to the runtime heap. This prevents callers from accidentally being dependent on the prior (undefined) contents of newly allocated memory blocks. It also helps catch cases where the contents of deallocated memory blocks are used after being freed. If this value is `false`, memory wiping does not occur.

The default value of `memoryWiping` is `false`.

validationFrequency = unsigned short (0...65535)

The frequency at which the list of allocated memory blocks in the heap is checked for corruption. When you set this frequency to 0, automatic heap checking is disabled. When the frequency is non-0, an internal counter is incremented every time a memory block is allocated or freed. When this counter is greater or equal to the frequency value, the `btValidateHeap()` function is called to check for heap corruption. This also sets the counter to 0. See the `btValidateHeap()` for additional details on heap checking.

The default `validationFrequency` is `0`.

blockDumpLimit = unsigned long (0...4294967295)

Sets the maximum number of bytes per memory block that is output to the log file when the contents of a heap block is dumped. Memory blocks in the heap are dumped to the log file whenever corruption is detected or during calls to `btDumpAllocated()`, `btDumpAllocatedDelta()`, and `btReportLeakage()`.

The default `blockDumpLimit` is `0xFFFFFFFF`.

maxDeferredBlocks

This is the maximum number of memory blocks placed in the deferred free queue at any one time. New blocks that exceed the limit cause the oldest blocks in the queue to be removed until the queue size is within limits.

The default value for `maxDeferredBlocks` is `0xFFFFFFFF`, which mean there is no maximum.

maxDeferredMem

This is the maximum number of bytes placed in the deferred free queue at any one time. New blocks that exceed the limit cause the oldest blocks in the queue to be removed until the queue size is within limits.

The default value for `maxDeferredMem` is `0`, which disables deferred freeing.

minDeferredBlockSize

This is the minimum sized block added to the deferred free queue. Blocks smaller than this are freed immediately. Only the user's portion of a memory block is considered in size calculations.

The default value for `minDeferredBlockSize` is `0`.

maxDeferredBlockSize

This is the maximum sized block added to the deferred free queue. Blocks larger than this are freed immediately. Only the user's portion of a memory block is considered in size calculations.

The default value for `maxDeferredBlockSize` is `16384` (16 Kb).

exitOnErrorRc = long (-2147483648...2147483647)

Sets a value that controls the operation of the heap checker when an error is detected. If the `exitOnErrorRc` is non-zero, the heap checker exits the process when it detects a heap error. Ending the process is accomplished by calling *exit()* with the value of `exitOnErrorRc`. If `exitOnErrorRc` is `0`, the heap checker continues its operation (returning the error to the caller if possible). Heap errors that cause the program to exit include:

- Out of memory
- Corruption detected during validation
- Invalid memory block freed
- Invalid memory block reallocated

The default value of `exitOnErrorRc` is `0`.

dumpOnTerm = full | delta | stats | none

This controls whether the heap is dumped on termination, when the process exits normally. The call to the dump function is done by the destructor of the heap checker object itself. The destructor runs after all other destructors of global objects in the system.

The following settings are available:

- *full* `btDumpHeapStats()` and `btDumpAllocated()` are called at termination.
- *delta* `btDumpHeapStats()` and `btDumpAllocatedDelta()` are called at termination.
- *stats* `btDumpHeapStats()` is called at termination.
- *none* No dump is done at termination.

See “Using the Heap Checker” on page 679 for a method of using this flag with SurePOS ACE.

The default value of `dumpOnTerm` is `none`.

beepOnError = true | false

Controls whether or not a beep will be sounded when errors (or leaks) are detected. If it is true, a beep sounds. If false, a beep does not sound. Beeping currently works only on targets in a Windows environment; beeping does not work on 4690 OS targets.

The default value of `beepOnError` is `false`.

logAllHeapOps = true | false

This controls the *verbosity* of the heap control code. When true, the heap control code logs *all* memory operations, such as successful allocations and frees. This is useful in cases when you must know where a certain pointer is freed, for example. However, it can create *immense* log files.

You can use the `btSetLogAllHeapOps()` function at runtime to set this selectively as needed.

The default value of `logAllHeapOps` is `false`.

Example Configuration File

```
 The possible values for each setting are:
;
;    checkingLevel        = disabled | minimal | checkpoints | corruption
;    callStackSize        = 1...16
;    logFileDir           = directory_name
;    logFileType          = binary | text | depends
;    cacheAllDebugInfo    = true | false
;    maxLogFiles          = 0...99
;    firstSentinelSize    = 2...32767
;    lastSentinelSize     = 2...32767
;    memoryWiping         = true | false
;    validationFrequency  = unsigned short (0...65535)
;    blockDumpLimit       = unsigned long  (0...4294967295)
;    maxDeferredBlocks    = unsigned long  (0...4294967295)
;    maxDeferredMem       = unsigned long  (0...4294967295)
;    minDeferredBlockSize = unsigned long  (0...4294967295)
;    maxDeferredBlockSize = unsigned long  (0...4294967295)
;    exitOnErrorRc        = long           (-2147483648...2147483647)
;    dumpOnTerm           = full | delta | stats | none
;    beepOnError          = true | false
;    logAllHeapOps        = true | false
```

Figure 45. Possible Values For Each Setting

```
 Global options can be given here

[*]
checkingLevel        = disabled
callStackSize        = 3
logFileDir           = logFileType          = depends
cacheAllDebugInfo    = false
maxLogFiles          = 16
firstSentinelSize    = 4
lastSentinelSize     = 4
memoryWiping         = false
validationFrequency  = 0
blockDumpLimit       = 0xFFFFFFFF
maxDeferredBlocks    = 0xFFFFFFFF
maxDeferredMem       = 0
minDeferredBlockSize = 0
maxDeferredBlockSize = 16384
exitOnErrorRc        = 0
dumpOnTerm           = none
beepOnError          = false
logAllHeapOps        = false


; This can also be used to set global options. Note that the last valid
; setting for a given keyword in a matching section is the one that is used.
```

```
[*:*]
blockDumpLimit = 0x1F
blockDumpLimit = 255
```

Figure 46. Global Options

```
; Override settings for ACEACCNT on all controllers. These values will
; override any of the global options.

[ACEACCNT]
checkingLevel = checkpoints


; Override settings for ACEACCNT on controller DD only. Note that the
; settings in the previous section will also have been used for controller
; DD since it applies to all controllers.  Any settings in this section
; will override values in previous sections.

[ACEACCNT:DD]
checkingLevel = minimal


; Override settings for all applications running on controller CC
; (Store log files in a data directory on a bigger drive).

[*:CC]
logFileDir = D:\BUZZY\


; Override settings for ACETSFSL on terminal 2

[ACETSFSL:002]
...


; Override settings for all applications on terminal 2

[*:002]
...


; Override settings for ACETSFSL on all terminals

[ACETSFSL]
...


; An INVALID entry which is ignored ('*' can only be used by itself)

[ACETS*]
...


; Some invalid settings. The checkingLevel line is invalid because
; comments must appear on lines by themselves.

[TEST]
checkingLevel = checkpoints     ; Turn on checking ? (NOT)


; Hmm, global options at the end of the file? Note that values in this
; section will override any options prior to this section in the file.
; (None of the other settings for blockDumpLimit would have any effect).

[*]
blockDumpLimit = 0
```

Figure 47. Override Settings for Applications and Terminals

Using the Heap Checker

This section will discuss how to use the heap checker to debug SurePOS ACE programs. There are several possible scenarios depending on how many changes you want to add to the program being debugged. Typically, you will want to put as few changes as possible in the SurePOS ACE code, particularly in common code. Even though the heap checker API calls are very fast when the heap checker is disabled, scattering a lot of calls throughout common code to debug one given application can interfere with the ability to debug other applications (the applications will be stepping on each other's toes).

Debugging with No Code Changes

Obviously, it would be extremely nice to be able to debug programs without changing any code at all. Luckily, this is possible by using the keywords in configuration file. If you suspect that your program is corrupting memory by writing beyond the end of allocated memory blocks, then simply set the `checkingLevel` parameter to `corruption` in the configuration file. The `firstSentinelSize` and `lastSentinelSize` parameters can be adjusted if necessary to check larger regions before and after each heap block.

The `memoryWiping` option can be enabled in cases where you think code is allocating memory from the heap and then using it without initializing it. When memory wiping is enabled, all memory allocated by the user is filled with the value 0xAA (`A` for allocated) before being returned to the caller. Later, when the user frees the memory block, it is again filled ; this time with the value 0xDD (`D` for deallocated). This may help catch problems where the constructors of C++ objects do not initialize every class member variable. Since memory blocks are also wiped when they are freed, this setting may also detect cases where code attempts to access data in freed memory blocks.

In cases where you think someone may be modifying memory after it has been freed (for example, clearing out a pointer or incrementing an integer), it may be helpful to enable deferred memory freeing. There are 4 configuration parameters which control deferred freeing. Three of these have reasonable defaults, so that in most cases, all you need to do to enable deferred freeing is to set `maxDeferredMem` to a non-zero value. The `maxDeferredBlocks` parameter was set to its maximum value because in most cases, the number of blocks being deferred is not as important as the number of bytes deferred. This is especially true on 4690 OS, which does not page virtual storage to disk. Many of these settings above can be combined with setting `exitOnErrorRc` to a non-zero value. This causes the program to end as soon as it detects an error.

Some types of memory leaks can also be detected without making any code changes. If the `dumpOnTerm` flag is set to `full`, the entire contents of the heap is dumped when the program ends normally (for example, when it calls `exit()`, `abort()`, or returns from `main()`). This can cause the log file to be very large. However, it might be possible to identify leaks by looking for cases where there are many memory blocks allocated by the same caller in the program. If there should not normally be that many blocks (i.e. if they should normally be freed soon after use), then you may have found a leak. Note that `dumpOnTerm` does not work when a program ends because of a trap or is terminated from the task manager (or 4690 OS control window). This means that programs that run continuously (such as Terminal Sales or Checkout Support) cannot be debugged in this manner.

Dumping Delta Allocations

One method for checking for memory leaks with very few code changes is to use `btDumpReset()` and `btDumpAllocatedDelta()`. These two functions allow you to dump all memory blocks that were allocated during a particular period of time in your program. Typically, the way to use these functions is to identify the section of code in your program that you think is causing leaks (such as running a sales transaction). You would then insert a call to `btDumpReset()` at the beginning of this section (in the event that signals the start of the transaction) and a call to `btDumpAllocatedDelta()` at the end of the section (in the event that signals the end of the transaction). When `btDumpAllocatedDelta()` at runtime, it dumps out any still allocated memory blocks that were allocated since `btDumpReset()` was called (at the beginning of the transaction). These two functions essentially implement a simpler form of checkpoints, which are discussed below.

Many current SurePOS ACE files (such as `aceaccnt.cpp`) contain *SmartHeap* calls that simulate this form of leak detection (by using checkpoints). At the beginning of the `main()` function (after all factories have been created), the current checkpoint value is set to 2. Then, at the end of `main()`, the program dumps all (still allocated) memory that was allocated during checkpoint 2.

SmartHeap functions can be emulated by inserting a call to `btDumpReset()` at the beginning of `main()` after factories have been created and a call to `btDumpAllocatedDelta()` when `main()` is about to return. If your program ends normally in some way other than returning from `main()`, then your call to `btDumpAllocatedDelta()` is never executed. In this case, you can also set the `dumpOnTerm` configuration setting to `delta` and achieve a similar result. The resulting dump will contain all of the non-factory related memory blocks that were still allocated when the program ended. Not all of these memory blocks are necessarily leaks, but looking at the dump might help you narrow down where leaks are coming from.

Checkpointing

Checkpoints are by far the most powerful method the heap checker offers for leak detection, unfortunately it also requires the most work. Typically, most of the work involves determining how to set or increment the checkpoint value and where to report checkpoint leakage. As an example, assume that memory is being leaked for every transaction run in terminal sales.

One of the simplest methods of using checkpoints to track leakage is to call `btIncrementCheckpoint()` at the beginning of each transaction. This results in any memory allocated during a given transaction to have a unique checkpoint value, which lets you possibly determine what type of operation caused the leak. To *see* the memory blocks, you need some way of causing the heap checker to dump the memory blocks to the log file. If you wanted to see all memory blocks, you could just call `btDumpAllocated()`. However, this might provide too much information. The `btReportLeakage()` API is designed to dump only currently allocated memory blocks that were allocated within a given range of checkpoint values. The difficult part in using `btReportLeakage()` is determining when and where to call it. The most obvious approach is to report the leakage after every transaction, dumping all memory allocated during the current checkpoint:

```
btReportLeakage(btGetCheckpoint(), btGetCheckpoint());
```

While this works, it might also very well dump memory blocks that are not true leaks. The reason for this is that there may be cases in the code where memory is allocated during one transaction and then intentionally cached (not freed) until the next transaction. In this case, you might want to wait until (for example) the third transaction and report any memory leaked during the first transaction:

```
if (btGetCheckpoint() >= 3)
   btReportLeakage(btGetCheckpoint()-2, btGetCheckpoint()-2);
```

Even this method could dump memory blocks that are not really leaks. This is especially true of memory allocated during *important* events, such as the first transaction after signon or the first non-cash transaction. It is possible that the code might read additional information from the store controller for these types of transactions and then, instead of freeing the data when done with it, cache the information for later use. (Then, it does not have to spend time fetching it from the controller again.)

Finally, instead of reporting leakage at the end of each transaction, you could also log *leaked* memory only when the program is about to end (right before *main()* returns). This gives the program plenty of time to free any memory (and hopefully any caches as well) before it is logged. For applications such as Terminal Sales, which normally never end, a good place to do this is during an event that you can easily control during testing, such as sign off.

Performance

Linking the heap checking code into a SurePOS ACE executable causes a minor increase in sizes of code and data in the executable. The modules making up the heap checking code reside in the library files, acehchkc.lib (for 4690 OS controller and Windows applications) and acehchkt.lib (for 4690 OS terminal applications).

Runtime performance is impacted only slightly when the heap checking code is disabled. The extra code consists of an additional function call and a few `if` statements. As stated above, as long as a SurePOS ACE program does not use the function calls in `btheap.cpp`, the heap checker is not required to be linked in. If your SurePOS ACE code extensions always use these functions, you can create a set of replacement stub functions that you can link into one of the default SurePOS ACE code libraries.

When the heap checker is enabled, performance is impacted to a greater degree. The amount of additional time spent in heap code checking depends on whether or not corruption checking is enabled and how often a full heap check is done. In addition, the amount of memory required for each allocation is increased by the size of the heap checker headers, which can range from 10 to 40 bytes by default. This can increase memory requirements significantly. However, because you can enable heap checker code only when you require it, this is not a major concern.

Interpreting Output Records from a Heap Log

This section describes how to interpret output records from a sample heap log.

File Header

The first part of the log file is a 128-byte header containing information about the program that generated the log as well as other information. The second word of the header (`v4`) indicates the version of the heap checker code that generated the log. Any time the format of the log file changes, this number will be incremented to ensure that the post-processor tool doesn't blow up trying to read bad data out of a log file. You will receive an error message if you attempt to run `ACEPPHLG` on a log file that doesn't have a matching version identifier.

```
--Text-HLG-file-- v4 32-bit Win32 Controller bthtest.exe tm913240522 dvid134
```

Process Information and Configuration

Following the header is more information about the process that created the log file, including the process name, process ID, and terminal ID. Process information is followed by a listing of configuration settings at initialization (as read from the ACEHPCHK.CFG file).

```
** ACE Heap Checking Code initialized
   Process name   = bthtest
   Process id     = 152
   Terminal id    = *
   Block overhead = 10/10, 40/40, 48/52

Current configuration:
   checkingLevel        = corruption
   ...
```

Heap Block Information

One of the most common entries in a heap checker log file is the information for a given heap block (which could be logged as a result to `btDumpAllocated()` or other functions). An example of this output follows:

```
0xB744F0: 10 bytes allocated by bthtest.cpp on line 213 at CP(0)
    bthtest.cpp ( 214) - testDelta
    bthtest.cpp ( 108) - main
    dde4strt    (   0) - ?_exestart
    fn(0x77F1B304)
  0000: AA AA AA AA AA AA AA AA - AA AA -- -- -- -- -- --  | ..........
```

The first line of this entry contains information about the heap block. The first value is the user's address of the memory block (the pointer returned when calling `new` or `malloc()`). This is followed by the size and then the module and line number that allocated the memory. Finally, the checkpoint at which the memory was allocated is listed.

The next four lines of this entry indicate the call stack at the time the memory block was allocated. Each of these lines list the module name, line number, and function name for each entry in the call stack. The first of these entries is the function that actually allocated the memory. The call stack information is stored in the header of each memory block. The `callStackSize` configuration parameter can be used to configure the maximum amount of information stored. In the sample above, the `callStackSize` was set to `6`, but only 4 entries are displayed because the end of the stack was reached.

The last line of the call stack is an address value. This is displayed because debugging information from the EXE file did not contain information about the module or function name that existed at that address. Usually, this problem only affects a Windows environment and usually only the last address on the call stack. However, it can affect any platform if the debugging information is incomplete or if the call stack addresses are corrupted.

The last line of information for a heap block contains the contents of the block itself. The data is displayed in both hex and ASCII formats (with periods representing non-printable characters). The number at the start of each data line is the hexadecimal offset from the beginning of the block. Depending on the setting of the `blockDumpLimit` configuration setting, not all data for a block is dumped. However, the size of the block, as listed by the first line, is correct.

In cases where the debug information in an EXE or .386 file was nonexistent, the heap checker is not able to format any of the call stack addresses. Usually, this happens only with text-mode log files on 4690 OS where debug versions of the .386 files are not read in at runtime. In this case, the heap checker code prefixes addresses with a special string (two tildes), which allows `ACEPPHLG` to later post-process the file. The example below shows the information for a data block both before and after post-processing.

```
0517:0488: 30 bytes allocated by ~~caller(0157:0700) at CP(0)
    ~~fn(0157:0700)
    ~~fn(0157:039c)
    ~~fn(0067:001b)
    ~~fn(0067:00f8)
  0000: 80 55 21 09 00 01 01 01 - 01 01 01 01 01 01 7B 00  | .U!...........{.
  0010: 0B 00 11 80 68 21 09 00 - 01 01 01 01 01 01 -- --  | ....h!........

0517:0488: 30 bytes allocated by BTHTEST on line 218 at CP(0)
    BTHTEST     ( 218) - testdelta
    BTHTEST     ( 108) - _main
    C0HC        (   0) - main
    INIT        (   0) - _mwinit
```

```
   0000: 80 55 21 09 00 01 01 01 - 01 01 01 01 01 01 7B 00  | .U!...........{.
   0010: 0B 00 11 80 68 21 09 00 - 01 01 01 01 01 01 -- --  | ....h!........
```

User Messages

The `btLogUserMsg()` function allows client code to add messages to the log file (to indicate significant events or other traceable processes). An example of a user message record follows below. The information provided is similar to that for an allocated block, including the line number and call stack at the point that the message was logged. The main difference is that the call stack for user messages will be up to 16 entries, regardless of the setting of the `callStackSize` configuration setting. Each call to `btLogUserMsg()` generates one record. To log multiline messages, use a new-line character (`\n`) to separate each line.

```
** User message from bthtest.cpp on line 41
      bthtest.cpp (  41) - logMsg__FPCce
      bthtest.cpp ( 169) - leakageTest__Fv
      bthtest.cpp ( 108) - main
      dde4strt    (   0) - ?_exestart
      fn(0x77F1B304)
   Checkpoint should be 2, it is 2
```

Reports

Several heap checker API calls result in *reports* being written to the log file. In particular, `btDumpAllocated()`, `btDumpAllocatedDelta()`, and `btReportLeakage()` generate reports consisting of heap blocks that meet the *search criteria*. In addition, `btValidateHeap()` reports invalid blocks after it is called directly (or indirectly by setting the `validationFrequency` configuration setting or by calling `btSetValidationFrequency()`).

Each of the reports begins with a line describing the type of report and the caller that requested it (as illustrated below). Any heap blocks in the report are then listed. Finally, a line indicating the end of the report is displayed along with the number of blocks in the report.

```
** DUMP ALLOCATED REPORT - requested by bthtest.cpp on line 186 at CP(1)
...
** END OF REPORT, 12 blocks logged

** DUMP ALLOCATED DELTA REPORT - requested by bthtest.cpp on line 220 at CP(1)
...
** END OF REPORT, 14 blocks logged

** REPORT LEAKAGE FROM CP 1 TO 1 - requested by bthtest.cpp on line 184 at CP(1)
...
** END OF REPORT, 1 blocks logged

** Heap validation - requested by bthtest.cpp on line 199 at CP(1)
...
** Heap validation done, 1 invalid blocks detected
```

Invalid Blocks

The heap checking code logs invalid heap blocks a little differently than normal heap blocks. If corruption checking is enabled and the block's sentinels were corrupted, all normal information about the block is displayed along with the reason why the block was considered corrupt. Typically, this type of error is detected when the heap is dumped or validated, as shown below.

```
** Heap validation - requested by bthtest.cpp on line 412 at CP(1)

** INVALID BLOCK - 0xB747A0: 6 bytes allocated by bthtest.cpp on line 199 at CP(1)
   Reason: Corrupt sentinel at beginning of block (offset 3)
       bthtest.cpp ( 199) - testFreq__Fv
       bthtest.cpp ( 108) - main
       dde4strt    (   0) - ?_exestart
       fn(0x77F1B304)
     0000: AA AA AA AA AA AA -- -- - -- -- -- -- -- -- -- --  | ......

** Time: Wed Dec  9 16:55:42 1998
** Heap validation done, 1 invalid blocks detected
```

Another case where bad blocks can be detected are when bad pointers are passed to the free or delete functions. In this case, a full set of call stack information about the caller of the free function is logged along with information about the block that was freed. If the header of the heap block is intact, the block information will be logged as shown above. However, if a problem is detected in the header of a heap block, then the caller information and block contents are not logged. The reason for the failure is logged along with the *raw* contents of the block header as shown below.

The amount and type of information in the heap block header depends on the configuration values. Sometimes, if the heap block header was only slightly damaged, some of the information might be useful. However, if the caller attempts to free a invalid memory pointer, the data may be of very little use at all.

```
** Free failed at CP(1)
   Caller: bthtest.cpp on line 398
           bthtest.cpp ( 398) - __dt__8MyStructFv
           bthtest.cpp ( 405) - wrongDelOp__Fv
           bthtest.cpp ( 108) - main
           dde4strt    (   0) - ?_exestart
           fn(0x77F1B304)

** INVALID BLOCK - B744F8 - unknown size
   Reason: Invalid memory block - bad eyecatcher (0x44C0)
   Caller information not available (error in block header)
      eyeCatcher = 0x44C0 (expected = 0xBAAB)
      userSize   = 1153433783 (0x44C000B7)
      allocCp    = 183 (0x00B7)
      pPrev      = 0x72E1BC0
      pNext      = 0x42F30D
      checkValue = 0x00410863 (expected = 0xC5159331)
      callStack  = 0x410324, 0x43444F, 0x77F1B304, 0x0, 0xCCCCCCCC, 0x2
      sentinel   = (expected CC's) EFBE1010
   Raw header data:
     0000: C0 44 B7 00 C0 44 B7 00 - C0 1B 2E 07 0D F3 42 00  | .D...D........B.
     0010: 63 08 41 00 24 03 41 00 - 4F 44 43 00 04 B3 F1 77  | c.A.$.A.ODC....w
     0020: 00 00 00 00 CC CC CC CC - 02 00 00 00 EF BE 10 10  | ................
```

Modified Deferred Blocks

If deferred memory freeing is enabled, memory blocks are placed in a queue before being freed. The content of the memory block is filled with a specific hex value (`0xEE`). Later, when the list gets full and the memory block is freed, the contents are compared with this value. If the contents of the memory block have changed after the caller *freed* it an error is logged as indicated below. (The byte at offset 12 was incremented to `0xEF`.) The heap checking code also checks the list of deferred free blocks for modifications whenever the heap is validated by *btValidateHeap()*.

```
** DEFERRED BLOCK CHANGED - 0xB74540: 100 bytes allocated by bthtest.cpp on line 461 at CP(0)
   Changed at offset: 12
       bthtest.cpp ( 461) - deferredCorrupt__Fv
       bthtest.cpp ( 108) - main
       dde4strt    (   0) - ?_exestart
       fn(0x77F1B304)
     0000: EE EE EE EE EE EE EE EE - EE EE EE EE EF EE EE EE  | ................
     0010: EE EE EE EE EE EE EE EE - EE EE EE EE EE EE EE EE  | ................
     0020: EE EE EE EE EE EE EE EE - EE EE EE EE EE EE EE EE  | ................
     0030: EE EE EE EE EE EE EE EE - EE EE EE EE EE EE EE EE  | ................
     0040: EE EE EE EE EE EE EE EE - EE EE EE EE EE EE EE EE  | ................
     0050: EE EE EE EE EE EE EE EE - EE EE EE EE EE EE EE EE  | ................
     0060: EE EE EE EE -- -- -- -- - -- -- -- -- -- -- -- --  | ....
```

Timestamps

Whenever the heap checker detects *important* events (such as heap corruption), it logs the current time of day in the log file. An example timestamp event appears below.

The time value used to log the event is retrieved using the C function *time()* and formatted using *ctime()*. If a text mode log file is being created, the time value is formatted immediately. If a binary log file is used, the time value is formatted by ACEPPHLG. This distinction is important because the *ctime()* function uses the current time zone and locale settings to format the time value, which is returned from *time()* as a value relative to GMT/CUT. An incorrect time zone setting can cause the time value to be off by several hours (although the difference should remain constant within a log file).

```
** Time: Wed Dec  9 16:55:22 1998
```

Appendix G. Debugging SurePOS ACE

This section describes some tools and techniques that can aid in debugging SurePOS ACE applications.

Debugging SurePOS ACE is made easier because the application can be run in Windows environments. The SurePOS ACE Development Toolkit runs in a Windows environment and builds two sets of ACE executables and data files: one set for 4690 OS and one set for Windows. This means most of your ACE development, building and debugging, can be done on a Windows machine with intermediate and final verification on the 4690 OS machine.

The Windows version of the SurePOS ACE executables is made possible by a set of scaffolding Windows classes that cancel out or implement some of the lower-level 4690 OS to provide some of these functions. A large portion of these classes is devoted to simulating the 4690 OS terminal devices to allow SurePOS ACE Terminal Sales applications to run in the Windows environment. When you build a Windows environment application, the build process automatically includes these simulator classes.

The Windows environment applications are for development use only. They are not to be used in any customer production environment.

Running in the Windows Environment

To start debugging extensions, you must first build executables that are compiled and linked. You build executables using the SurePOS ACE Development Toolkit running in the Windows environment. When you install the toolkit, it personalizes two batch files, `acent.bat` for producing Windows targets and `aceflat.bat` for 4690 targets, with your answers to the installation process questions. Running these batch files sets the environment variables for the ACE Toolkit to build the selected set of targets.

This is the procedure for building a SurePOS ACE executable:

1. Open a command prompt window.
2. Run the toolkit batch file, either ACENT or ACEFLAT, for the type of target to build. See Table 136 for help selecting the batch file.
3. If you did not set the VisualAgeC++ paths in your default environment, run its provided batch file to add its paths to your current window by entering `drive::\ibmcxxr\bin \setenv`.
4. Change to your toolkit \ACE\MAKE directory. The default installed directory is drive:: \SP_ACE\ACE\MAKE.
5. Enter your make command as `Make -f makefilename targetname`, where *makefilename* is the name of the make file used to make your target and *targetname* is the name of the target. See Table 136 for help selecting the `make` command to use.
6. For a successful build, make your changes to address any compile or link problems and repeat Step 5.
7. By default, your new executables will go in your toolkit root directory: \nt\bin for Windows and \flat\bin for 4690.

Table 136. Make Commands for Target Executables

Target	Batch File to Use	Make Command
ACEDMMLL.EXT (Win)	ACENT	make -f acedmmll.mak acedmmll.exe
ACEDMMLL.386 (4690)	ACEFLAT	make -f acedmmll.mak acedmmll.386

Target	Batch File to Use	Make Command
ACETS10L.EXE (Win)	ACENT	make -f acets10l.mak acets10l.exe
ACETS10L.386 (4690)	ACEFLAT	make -f acets10l.mak acets101.386
All libs + executables for target environment		make -f makefile.ace all
ACEDMMLL.EXE depend	ACENT	make -f acedmmll.mak dependencies

After you have successfully built your extended applications, you can run them in the Windows environment to verify that they function properly. You do not need to set environment variables to run the SurePOS ACE executables, but you do need ACE Toolkit Services (ATS) running to handle tasks, such as logical name resolution and keyed file support. If you do not have ATS running when you start a SurePOS ACE Windows executable, an ASSERT dialog box will pop up with text similar to `File: f:\ace\mgv\commonc\cdds.c Line 58 Reason:`. If this occurs, you must start ATS.

If you have the ACE toolkit installed in the c:\sp_ace directory:

- Enter this command to start ATS: `start C:\sp_ace\ace\lfns\bin\fdsgo debug`
- To stop ATS, close the command window that was opened when ATS was started. The window has "c:\sp_ace\ace\lfns\bin\fdsgo.exe" in the title bar.

In addition, ATS must be using the correct set of SurePOS ACE logical name definitions that it receives from its logical name file, FDSLN.LN. The toolkit used your answers to the installer questions to modify the base ACE-provided logical name file, FDSLN.LN, and copied that modified file to the correct directory. When ATS starts, it reads that logical name file and uses it to resolve any SurePOS ACE logical names in the Windows environment. If your extensions require changes to the SurePOS ACE logical names, you must place those changes into the logical name file. Then, you must stop and restart ATS to pick up the changes.

You also need to define and set up the application adx_ directories on your Windows computer. Again, the SurePOS ACE Development Toolkit installer makes this set of directories, installs the necessary default files to them, and changes the logical names file to resolve those logical names to the proper directories. SurePOS ACE will run with the default set of files. However, to get your particular behavior, you may want to copy any necessary files from your 4690 machine to the corresponding toolkit directory tree on your Windows computer.

There are several ways to move files between your Windows development computer and your 4690 machine. You can use a diskette, or if you have network access, you can use FTP if your 4690 environment is set up for FTP.

You can run one or several SurePOS ACE applications by starting each in its own command window. For example, if you need Terminal Sales, Checkout Support, GIPC, and DM all running at the same time, start each one in its own command window.

Using the SurePOS ACE Terminal Simulator

When you use the SurePOS ACE Development Toolkit to build a Terminal Sales application for the Windows environment, it will automatically include some Windows scaffolding classes that simulate the 4690 terminal devices. This lets you debug most of your Terminal Sales code on your Windows machine, and then do only intermediate and final testing on a 4690 terminal.

The set of Windows scaffolding classes is referred to as the *SurePOS ACE terminal simulator* or *virtual cash register*. When you start a Windows Terminal Sales application in a command window, a setup window for the terminal simulator is displayed, as shown in Figure 48.

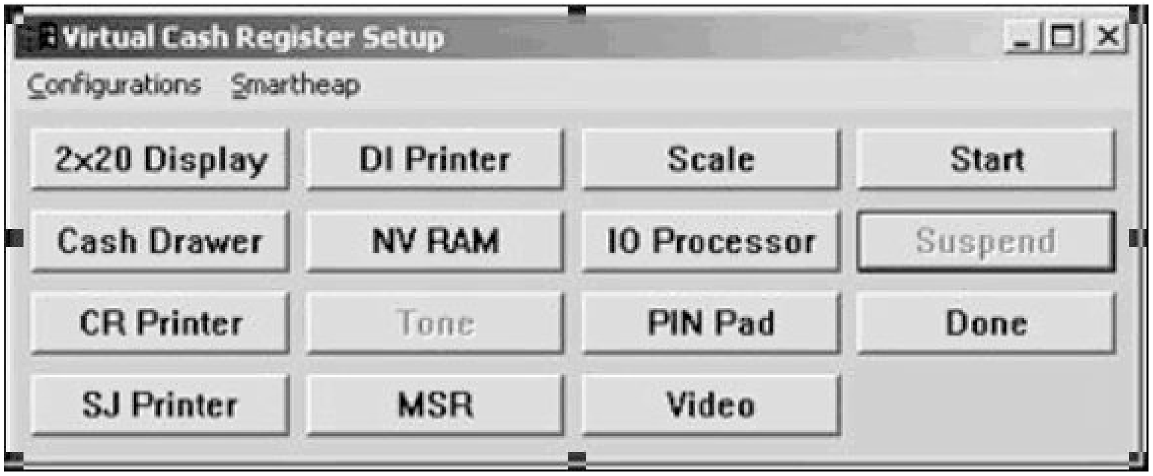


Figure 48. Virtual Cash Register Setup Window

Use the Setup window shown in Figure 48 to add the terminal devices for your application, and to start or stop the application. Before the application can run, you must configure the required devices. The Setup window has buttons for every device that the terminal simulator supports. To add a device to your configuration, click the button for that device. Another window to represent the device opens. For example, click IO Processor to open windows for a keyboard and scanner. You can resize or move the windows to any location on your desktop. Continue clicking and moving until you have added all the devices that you want.

The first time that you run the Terminal Simulator, you must specify your devices. After the first time, if you have a previously-saved configuration, you can load that configuration, or you can specify your devices at every run.

Figure 49 shows a completed configuration. The 2x20 Display window should be extended and the the System_Operator checkbox should be selected for viewing operator messages. The Drawer window has the Drawer Locked checkbox selected. If this is not selected, each time the cash drawer would open, the Drawer Open check box would be set by the simulator, and a prompt would be displayed. To continue, you would have to clear the check box and press the CLEAR key.

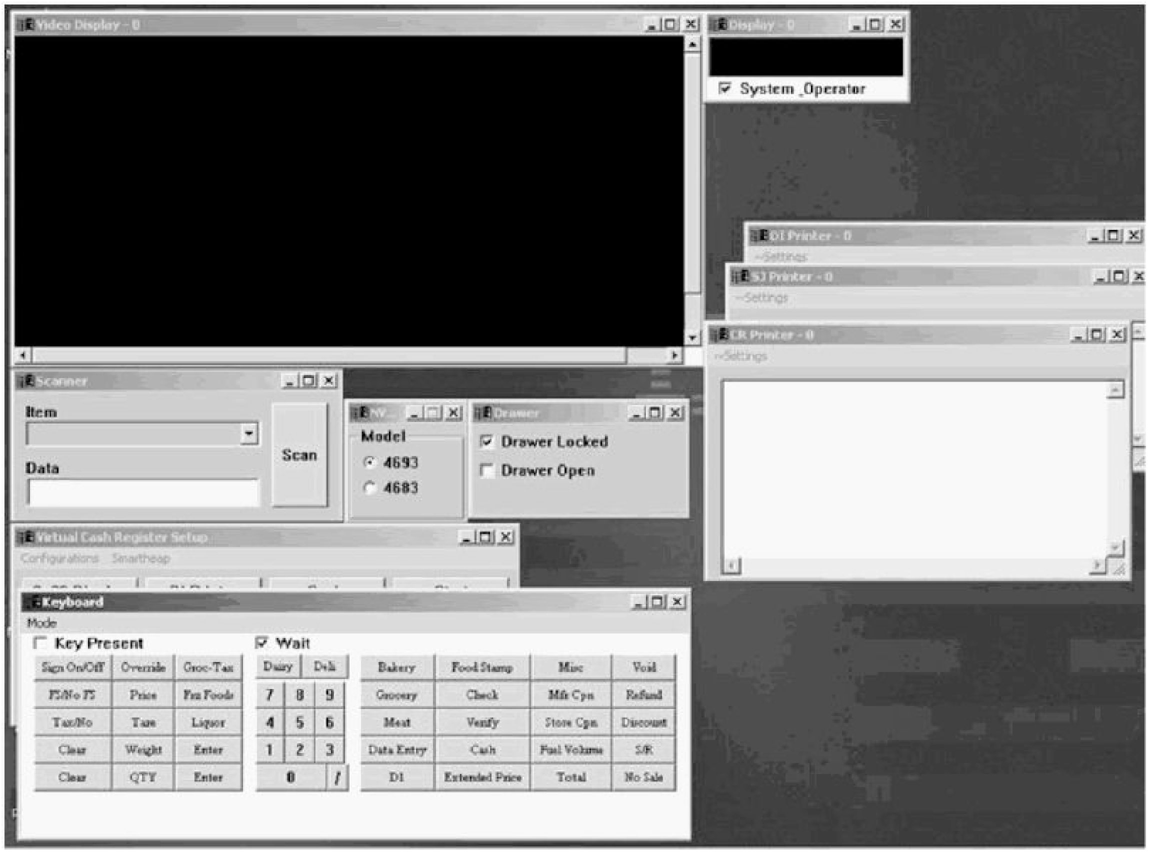


Figure 49. Completed Terminal Simulator Configuration

When you have the configuration that you want, go back to the Setup window and click Configurations --> Save to save the devices, their sizes, and their locations. The next time you start the terminal sales application, you can use the saved configuration by clicking Configurations --> Load.

Running the Terminal Sales Application

To run the Terminal Sales application, click Start in the Setup window. The 2x20 Display window will show the terminal loading options and then go to the `CLOSED` state. The full screen will also show the `CLOSED` state. The application starts in the state that it was in when you ended, either `TERMINAL SECURED` or `CLOSED`. If you were in the middle of a transaction, it will start where you left off.

To change the configuration, you must first delete the old configuration using the Registry Editor. Using extreme caution, start regedit and search for the `vrtlcshr.ini` key. When you find the key, delete it. Click Configurations --> Save to save the currrent configuration as the new default.

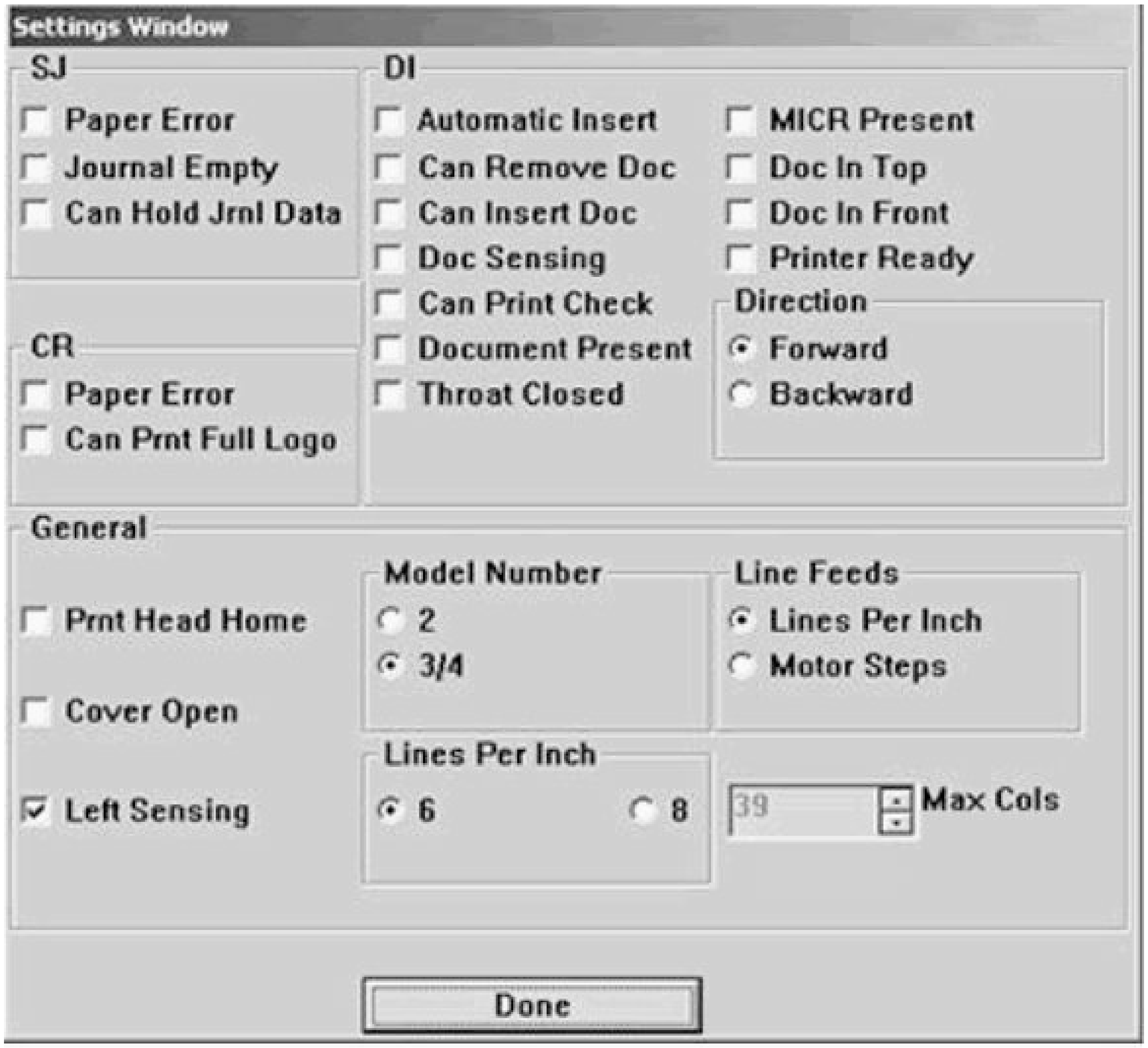
Using the Simulator Keyboard

To use the terminal simulator keyboard, go to the Keyboard window and click the keys that you want to enter. You can use your desktop keyboard for the numeric keys, the ENTER key and the Esc key (for the CLEAR key), but the Keyboard window must be active.

To sign on to the simulated terminal, select the Keyboard window and click the sequence `1, /, 1, Sign On/Off`. You can use your desktop keyboard to enter the two numeric one's. As you enter your keying sequence, you can watch the 2x20 Display window.

Configuration Settings for Printers

You can specify additional configuration settings or error conditions for your printers. To access the Settings window for a printer, click Settings on the Printer Stations window. The Settings window is displayed, as shown in Figure 50. Select the check boxes and radio buttons for the settings that you want. When you are finished, click Done to close the window.


Settings Window

SJ
- Paper Error
- Journal Empty
- Can Hold Jrnl Data

CR
- Paper Error
- Can Prnt Full Logo

DI
- Automatic Insert
- Can Remove Doc
- Can Insert Doc
- Doc Sensing
- Can Print Check
- Document Present
- Throat Closed
- MICR Present
- Doc In Top
- Doc In Front
- Printer Ready

Direction: [✓ Forward] Backward

General
- Prnt Head Home
- Cover Open
- [✓ Left Sensing]

Model Number: 2 [✓ 3/4]

Line Feeds: [✓ Lines Per Inch] Motor Steps

Lines Per Inch: [✓ 6] 8

39 Max Cols

Done

Figure 50. Settings Window

Stopping the Terminal Sales Application

When you have finished running your Terminal Sales application, go back to the Setup window and click Done.

Using the Windows Debugger

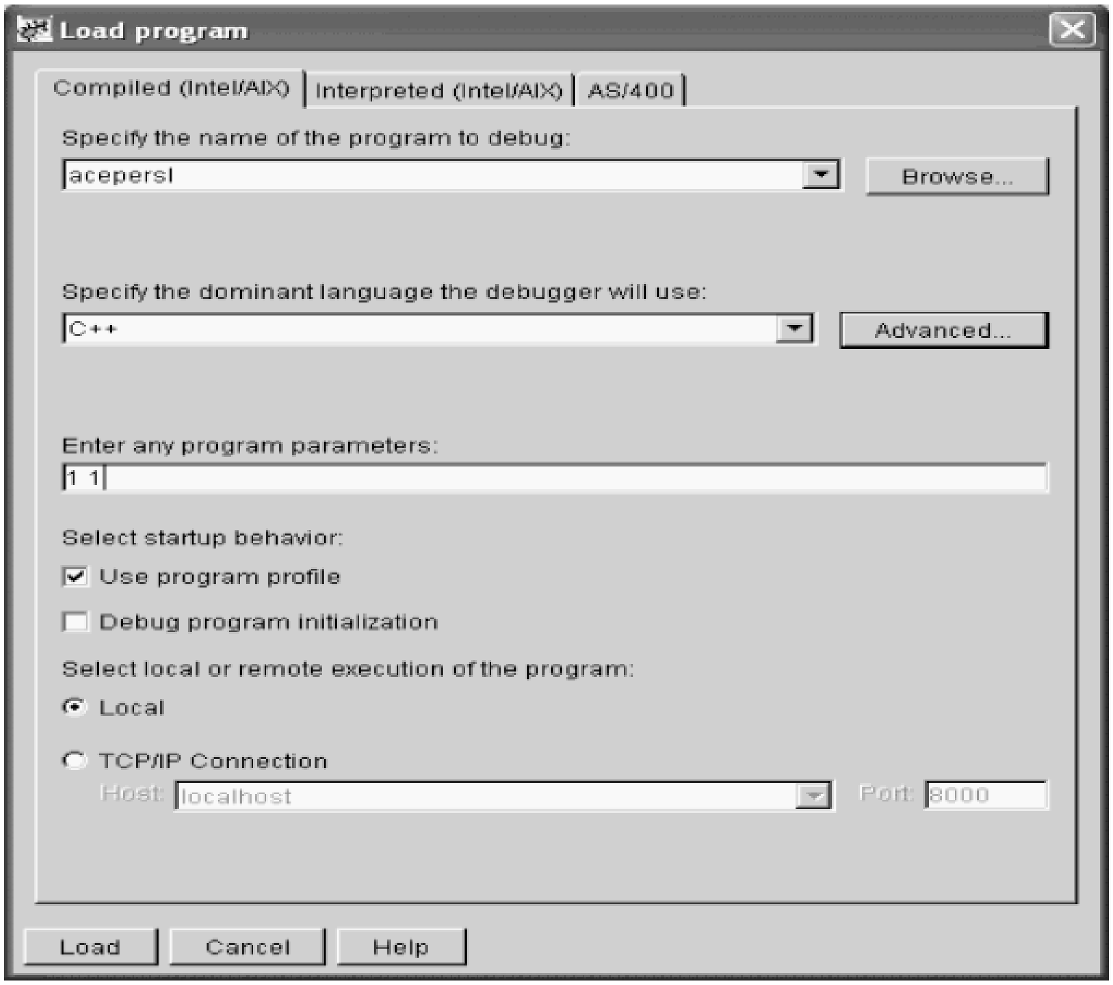
VisualAge C++ compiler (version 3.6.5) Fixpack 2 is used to create the Windows and 4690 executables. (See "Software Requirements" on page 554 for more information about this compiler and a debugger, which can also remotely debug SurePOS ACE applications on 4690 OS, that are included in the Toshiba Global Commerce Solutions Development Environment for OS4690.)

Starting the Debugger

The name of the debugger application is IDEBUG. It is located in the \IBMCXXR\BIN directory. There are two ways to start the debugger at a command prompt:

- Enter the debugger's name.

  The default screen (see Figure 51) is displayed. Enter the program name and parameters in the boxes, select the check boxes that you want, and click OK.


Load program

Compiled (Intel/AIX) | Interpreted (Intel/AIX) | AS/400

Specify the name of the program to debug: acepersl

Specify the dominant language the debugger will use: C++

Enter any program parameters: 1 1

Select startup behavior:

[✓] Use program profile

[ ] Debug program initialization

Select local or remote execution of the program:

(●) Local

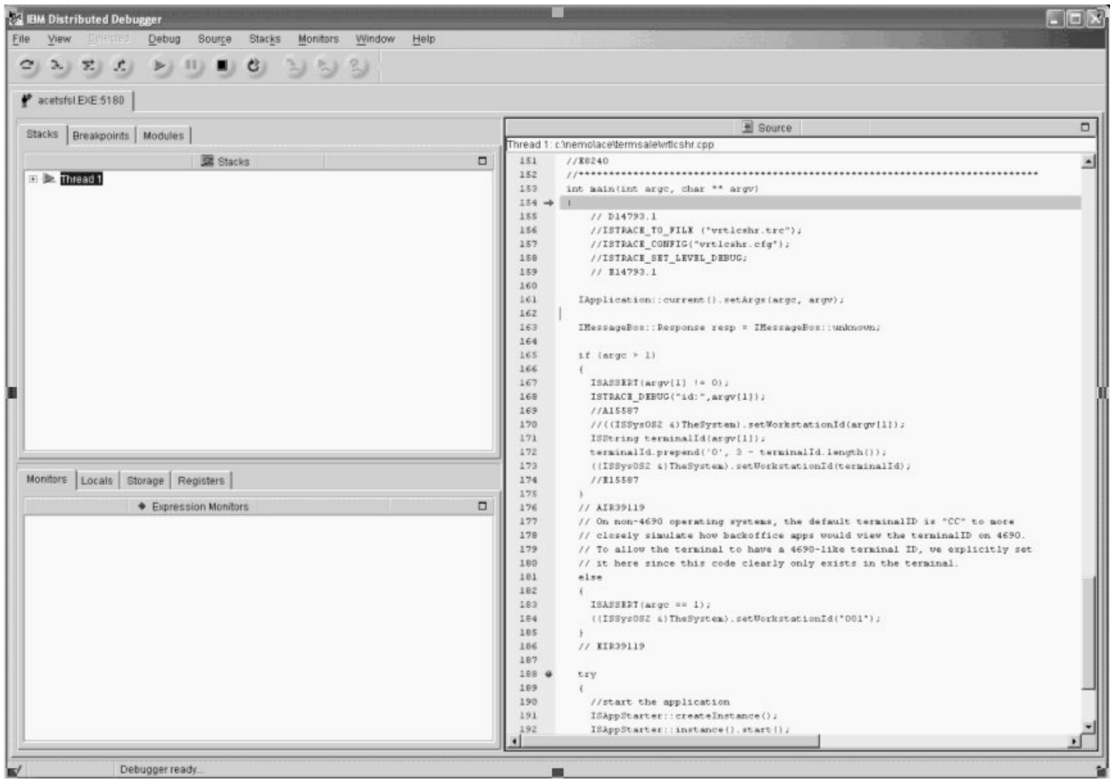
( ) TCP/IP Connection

Host: localhost Port: 8000

Figure 51. Load Program Window

- Enter the debugger's name, and the name of the program to debug and any parameters that it needs.

  If you specifiy a program when you start the debugger, you will skip the Load Program window and go directly to the Debugger pane (see Figure 52).


IBM Distributed Debugger
File View Selected Debug Source Stacks Monitors Window Help
acetsfsl.EXE:5180
Stacks Breakpoints Modules
Stacks
Thread 1
Monitors Locals Storage Registers
Expression Monitors
Source
Debugger ready...

Thread 1: c:\nemo\ace\termsale\vrtlcshr.cpp

```
//E8240
//****************************************************************************
int main(int argc, char ** argv)
{
    // D14793.1
    //ISTRACE_TO_FILE ("vrtlcshr.trc");
    //ISTRACE_CONFIG("vrtlcshr.cfg");
    //ISTRACE_SET_LEVEL_DEBUG;
    // E14793.1

  IApplication::current().setArgs(argc, argv);

  IMessageBox::Response resp = IMessageBox::unknown;

  if (argc > 1)
  {
    ISASSERT(argv[1] != 0);
    ISTRACE_DEBUG("id:",argv[1]);
    //A15587
    //((ISSysOS2 &)TheSystem).setWorkstationId(argv[1]);
    ISString terminalId(argv[1]);
    terminalId.prepend('0', 3 - terminalId.length());
    ((ISSysOS2 &)TheSystem).setWorkstationId(terminalId);
    //E15587
  }
  // AIR39119
  // On non-4690 operating systems, the default terminalID is "CC" to more
  // closely simulate how backoffice apps would view the terminalID on 4690.
  // To allow the terminal to have a 4690-like terminal ID, we explicitly set
  // it here since this code clearly only exists in the terminal.
  else
  {
    ISASSERT(argc == 1);
    ((ISSysOS2 &)TheSystem).setWorkstationId("001");
  }
  // EIR39119

  try
  {
    //start the application
    ISAppStarter::createInstance();
    ISAppStarter::instance().start();
```

Figure 52. Debugger Pane

Both methods should start the debugger positioned at the entry to your `main()` routine, just as if you had set a breakpoint, if you did not select Debug program initialization, which starts the debugger in a low-level initialization function. You can use this point to set breakpoints in any static classes that you need to debug.

If your source modules are not in the same directory as your executable files, the debugger will not be able to find your source code and will provide a prompt in a dialog box for you to enter the path. To enable the debugger to find your source code, without having to prompt you each time, there is an environment variable that you can set as a source path for the debugger. The environment variable name is `DER_DBG_PATH`, which can be set just like the PATH variable, to a semicolon-separated list of paths.The variable should be set automatically by either acent.bat or aceflat.bat. The following example sets the environment variable to three SurePOS ACE paths used in ACEPERSL.

```
set der_dbg_path=f:\sp_ace\ace\person;f:\sp_ace\ace\ui;f:\sp_ace\ace\tvision
```

Note: When debugging a program for the first time, click File --> Preference --> Debug --> program name --> Exception Filter Preferences Settings. Make sure all of the boxes are selected, then select Debugger Defaults.

When you are in a source code window, you can use either the menu selections or keyboard hot keys to operate the debugger. These are some of the hot keys:

Double-click line

Toggle breakpoint at current line

F5

Run

F10

Instruction step over

F11

Instruction step into

Displaying or Editing Variables

There are two ways to display or edit variables:

- Using the Program Monitor window - To add a variable to the Program Monitor, select a variable, right-click the variable, then select Add to Program Monitor. If the Program Monitor window is not already displayed, it will be started.
- Using local variable view - To display all of the local variables, select the Locals tab at the bottom left of the debugger window. The bottom-left window in the debugger will switch to the local variable view.

You can work with variables in either window by right-clicking the variable in the window and then clicking your selection. For example, `Next Representation` toggles between the hexadecimal and decimal display of the value.

Remote Debugging of 32-bit Controller Executables

To perform remote debugging, you must have the Toshiba Global Commerce Solutions Development Environment for OS4690 installed.

1. On your Windows machine, run aceflat.bat to setup your environment variables. Of particular interest is the `DER_DBG_DEBUGGEE_INTERFACE=tcpip 4000` environment variable, which indicates the debugger should listen on a socket for an external probe to drive the debugging.
2. On your 4690 machine, start the debugger probe by executing this command

   ```
   derdpcmd tcpipc your_machine_address 4000
   ```

   your_machine_address should be the IP address of your Windows computer from which you will run the front end of the debugger. If you have entered an alias for that address in the hosts file on your 4690 system, that name will also work. Upon execution, some information should get printed out to the screen with one of the last lines indicating that the probe is waiting for a connection.
3. On your Windows machine, change to the bin directory where your executable (.386) is located. Start the debugger by executing

   ```
   idebug full_4690_path
   ```

   where `full_4690_path` is the full path of your executable on your 4690 system. For example, if you are trying to debug acecsmll.386 and you put it in c:\adx_ipgm on your 4690 system, the debug command will be

   ```
   idebug c:\adx_ipgm\acecsmll.386
   ```

   The debugger front end will ignore the path portion and just look for acecsmll.386 in your current directory. If you are running a software firewall, you should configure it to allow external programs to initiate connections to port 4000.

Remote Debugging of 32-bit Terminal Executables

To perform remote debugging, you must have the Toshiba Global Commerce Solutions Development Environment for OS4690 installed.

1. Make sure TCP/IP is installed on your terminal and that your terminal is on a LAN with your Windows system. You should be able to ping your terminal from your Windows machine.
2. Create, if necessary, and edit adx_sdt1:adxenvt.dat to add the following line:

   ```
   DER_DBG_DERDPCMD_PARMS=tcpipc your_machine_address 4000
   ```

   your_machine_address should be the IP address of your Windows computer from which you will run the front end of the debugger.
3. On your 4690 system, run the derdpcmd.386 executable as a terminal application.
4. Start the debugger front end on Windows exactly as you would for a controller executable (as in Step 3 on page 695).

Troubleshooting Debugger Problems

If you are having problems starting the debugger, check the output on the 4690 screen. If, after starting the probe on 4690 and the debugger on Windows, you do not see `Connection Established`, then the two programs are not establishing communications correctly. Check your firewall settings to make sure that the communication can be established. You should also check the 4690 optional diskettes for the derdtcp2.dll file. The file should be copied to the bin directory where you installed the Toshiba Global Commerce Solutions Development Environment for 4690.

If you see `Connection Established`, then you should double-check the 386 executable that you are trying to debug on your 4690 system against the one you are using for debug information on your Windows system. The two executables must be produced from the same build.

Debugging Tools Available with 4690 OS

These debugging tools are provided with 4690 OS:

- ADXCSJ0L is a hexadecimal editor shipped with 4690 OS. Use it to display or edit file data.
- XE is a text editor shipped with 4690 OS. Use it to display, edit, or create text files, such as usX.
- FTP and Telnet (limited keys), when set up, are supported by 4690 OS.

Debugging Tools Available with 4690 OS

These debugging tools are provided with 4690 OS:

- ACETVREL SQL Query Report, which is intended for development use, can be activated by renaming `acereini.usx` to `acereini.us3`. This will add a new selection to the Reports menu. This report can be helpful in debugging SQL commands and SAPATH definitions. The report can also be a good problem determination tool because it provides a powerful SQL interface for viewing the ACE database.

- ACE Keyed File Rebuild Utility (ACEKFRBL) can be used to build keyed files from direct files for testing purposes. For example, to test the SurePOS ACE Sample report you need a keyed file with Zip Code data in it. You can get this file by making a fixed record-length direct file with your test data and then using this utility to make a keyed file from the direct file.
- Serviceability INI File (`svc.ini`) activates miscellaneous serviceability options. In addition to serviceability aids, when tracing or trap code is needed to collect documentation for a problem, control for the problem determination code is added to the INI file. You activate the code via the use of an overlay for the `svc.ini` file. This allows you to maintain one set of programs across the enterprise, yet selectively enable the trace/trap code.

Tools Provided As-Is

Other tools have also been included in the product to enable Toshiba Global Commerce Solutions Support to better assist retailers in supporting their stores. While these programs are shipped with the product, formal support is not available. When working with Toshiba Global Commerce Solutions Support personnel, you may be instructed to use these, or other tools in order to troubleshoot or collect documentation for a problem. In order to use these tools in the toolkit batch file, either ACENT or ACEFLAT must be run at the command prompt, and then the program must be run in that command window. These are some currently-shipping support tools:

ACEDIFF

A file comparison tool used to compare files in line-by-line mode. This program is useful for comparing the output of options dumped to file using personalization, and can be used to compare store to store, or terminal specific to terminal specific options.

ACEKYVER

A keyed-file validation program and can be used to report statistics about keyed files, generate a list of keys within the files, report the records in a keyed file, and detect corruption within the keyed file.

BNLOOKUP

Lookup data from the BIN File (`acebninp`).

HEXDUMP

Dumps the contents of a file to the screen in hex mode; useful for a quick view of a file in hex.

HOSTSIM

Host Simulator for a default (DEF) host.

LOGFMT

Formats the input EPS log file in a more readable manner.

SNAPSHOT

A documentation collection tool which can be run using various profiles for collecting needed files for Toshiba Global Commerce Solutions Support. The snapshot.txt file contains a description of the profile templates. You should use INI overlay files to specify additional files needed for your own unique operation.

Snaphot can run other utilities to prepare data for collection. Examples of this include running `ACEPERSL /S` to save options to INI files and then archiving the INI files.

Note: Be careful when running Snapshot. Do not specify an editor when running via RCP or the IPL Command Processor.

Multiple archives of collected documentation are managed via the INI file to limit the disk space and number of files kept. The default INI files place the documentation files in the root directory of C: However, Toshiba Global Commerce Solutions Support recommends that you make a new directory for Snapshot archives and direct all archives there through the use of an INI file overlay.

Methods of Tracing Code

SurePOS ACE provides two methods of tracing that you can use when debugging code:

- Configurable Tracing Utility
- Macros placed within the code

Configurable Tracing Utility

SurePOS ACE includes a configurable tracing facility (CTF) to aid third-party developers in understanding the code paths in base SurePOS ACE code so that they can debug their own code. CTF is available only for 4690 OS target executables built from the SurePOS ACE Toolkit, not for Windows targets.

To activate the CTF, customers must create configuration files. Each ACE executable searches for a different file. See Table 137 for a list of which executables have been enabled for tracing and the configuration files for which they look. If only a file name is given, then the application will look for that file in the directory from which the application is run.

Typically, tracing information is output into files in the adx_idt4 directory.

Table 137. Executables that are Enabled for Tracing

Application	Configuration File	Output File
Any Terminal Sales application	adx_ipgm:trmtrcfg.*XXX*, where *XXX* is the terminal number	adx_idt4:trmtrace.*XXX*, where *XXX* is the terminal number
acecsmll	adx_ipgm:cstrace.cfg	adx_idt4:cstrace.trc
aceaccn	aceaccnt.cfg	aceaccnt.trc
aceapssr	\iotest.cfg	\iotest.trc
aceborun	aceborun.cfg	aceborun.trc
aceubatp	aceubatp.cfg	aceubatp.trc
acegipc	\gipctest.cfg	\gipctest.trc
aceejrpt	aceejrpt.cfg	aceejrpt.trc
acetvrel	acetvrel.cfg	acetvrel.out
aceslirl	aceslirl.cfg	aceslirl.out

All executables write tracing statements as they are executed, with the exception of the Checkout Support (acecsmll) and Terminal Sales applications. Terminal Sales applications buffer their output until the operator executes a special signoff. Checkout Support buffers output until an interval crossing is reached. These applications will buffer at most 200,000 bytes before wrapping output in their buffer.

Configuration file contain lines of the following form:

```
ClassName::FunctionName traceLevel entryExitLevel localFlag
```

with the parameters defined as follows:

ClassName
: The name of a class. If no class name is specified, the line refers to a free function.

FunctionName
: The name of a function (which should be the name of a member function of ClassName if one is given, although no error or warning will be given if it is not.) To specify a free function, the double-colon must be included, otherwise the trace facility treats the name as a class name.

traceLevel
: Specifies the tracing level to be used on entry to the function (or any member of the class, if no function name is given). This is an optional flag.

  The valid levels are `none`, `error`, `alert`, `normal`, `debug` and `diag` (the values are not case-sensitive).

entryExitLevel
: Causes the entry/exit tracing flag to be turned on or off on entry to the specified function. This is an optional flag.

  The valid levels are `entryExitOn` and `entryExitOff` (the values are not case-sensitive).

localFlag
: Specifies that the trace level and entry/exit trace flag are to be changed just in the function specified and not by all functions called by it. This is an optional flag.

  The valid value is `local` (the value is not case-sensitive).

Table 138 contains examples of possible trace configuration file entries.

Table 138. Examples of Trace Configuration File Entries

CTF Entry	Purpose
ISFile::open debug	Causes the `ISFile::open` member, and all functions called by it, to trace at debug level.
ISFile none	Turns off all tracing from the `ISFile` class.
::globalFunction error	Causes the (global) function `globalFunction` to trace at the error level. (The leading :: is required to indicate that this is a function name rather than a class name.)
ISFile::open entryExitOn	Turns entry/exit tracing on for the `ISFile::open` member and all functions called by it.
ISFile::open debug local	Turns on debug level tracing for the `ISFile::open` member, but does not propagate the trace level change to functions called by `ISFile::open`.
$Timestamp	Turns on timestamping.
$ElapsedTime	Prints the amount of time spent in each traced function for which entry/exit tracing is enabled.

CTF Entry	Purpose
$Performancettt	This limits trace output to only entry/exit traces that ran longer than the specified time. Replace "ttt" in the keyword with a time (1-720000 centiseconds on 4690 or 1-7200000 milliseconds on Windows) without commas or periods.
$NoWrap	Prevents the memory traces from wrapping the trace buffer, which otherwise can cause loss of trace data not yet written to disk, typically caused by verbose trace-level settings. All trace levels are written to disk, flushing when the memory buffer becomes full, even during transaction processing. As such, a minimal amount of tracing should be enabled when $NoWrap is enabled, due to the loss of performance.

Using Tracing Macros

Developers extending ACE also have the option of adding tracing macros to their code. While developers can choose macros that trace at different levels, ACE 4690 executables are always compiled with the most inclusive levels. However, the configuration file can be used to skip over tracing statements at lower tracing levels.

Selecting a Trace Destination

Developers creating a new executable must specify a tracing destination and configuration file to use. The tracing destination can be either a file or a memory buffer.

Tracing to a File

Use the ISTRACE_TO_FILE(filename) macro to trace to a file. Each tracing statement that produces output (as dictated by the configuration file) will write directly to this file.

You can use the macro ISTRACE_SIZE to limit the maximum size of the trace file. By default, the size is unlimited.

Tracing to a Memory Buffer

Use the ISTRACE_TO_MEMORY(buffer size,file name) macro to trace to a memory buffer. The parameters specify the size of the buffer in memory, and the file that will ultimately contain the trace data.

Because the memory buffer size is fixed, it will continuously wrap output as data is added. The macro ISTRACE_FLUSH must be called to write the memory buffer to the file.

Example: ISTRACE_TO_MEMORY(200000,"adx_idt4:userprog.trc");

Setting the Trace Level

Use the ISTRACE_SET_LEVEL_xx macros to set the trace level. These are the available macros:

- ISTRACE_SET_LEVEL_NONE;
- ISTRACE_SET_LEVEL_ERROR;
- ISTRACE_SET_LEVEL_ALERT;
- ISTRACE_SET_LEVEL_NORMAL;
- ISTRACE_SET_LEVEL_DEBUG;
- ISTRACE_SET_LEVEL_DIAG;

Toshiba Global Commerce Solutions recommends that you use the ISTRACE_SET_LEVEL_DEBUG macro for all programs. Issue this macro after setting the trace destination and configuration file.

Function Entry

You should include the ISTRACE_IN(functionName) macro as the first line of each function that is to be traced. The macro enables the trace facility to track the function calls that the application makes. You do not need to enclose the function name in quotes and you do not need to include the class name; the trace code will do that for you. This is an example:

```
ReturnCode ISSalesTransaction::getNextItem(ISItemEntry& item)
{
  ISTRACE_IN(getNextItem);
  ... function body ...
}
```

You can have the trace facility generate trace lines on function entry and exit by specifying one of the following macros:

- ISTRACE_ENTRY_EXIT_ON;
- ISTRACE_ENTRY_EXIT_OFF;

You can have the trace facility start or stop generating a timestamp (HH:MM:SS) at the start of each trace line by specifying one of the following macros:

- ISTRACE_TIMESTAMP_ON;
- ISTRACE_TIMESTAMP_OFF;

You can have the tracing facility turn on and off wrapping control of the memory buffer by specifying one of the following macros. Turning off wrapping causes all trace data to be written to disk when the memory buffer becomes full and can have a negative impact on performance.

- ISTRACE_WRAP_ALLOWED_ON;
- ISTRACE_WRAP_ALLOWED_OFF;

Tracing the Flow of Control

The ISTRACE_xxx macros allow a string to be written to the trace output. The string can be used for tracing code flow and other informational messages which have no associated data.

These are the available macros:

- ISTRACE_CHAR_DIAG("text message");

  Note: Class header information is suppressed.
- ISTRACE_POS("text message");

- ISTRACE_POS_ALERT("text message");
- ISTRACE_POS_DIAG("text message");
- ISTRACE_POS_DEBUG("text message");
- ISTRACE_POS_ERROR("text message");

Tracing Objects

There is a set of macros that allows the tracing of primitive types (int, char*, and others) and of class type objects. In order for these macros to compile correctly for a class object, the class must have a trace member. The trace member is assumed to trace out important details of the object state. See "Class Tracing" on page 702 for more information.

These are the available macros:

- ISTRACE("object description", <object>);
- ISTRACE_ALERT("object description", <object>);
- ISTRACE_DEBUG("object description", <object>);
- ISTRACE_DIAG("object description", <object>);
- ISTRACE_ERROR("object description", <object>);
- ISTRACE_LITE_DEBUG("object description", <object>);

  Note: Class header information is suppressed.
- ISTRACE_LITE_DIAG("object description", <object>);

Note: Class header information is suppressed for LITE macros.

Tracing Data Buffers

There is a set of macros that allows buffers of data to be traced in one call. The data is traced in both hexadecimal and ASCII formats. The length parameter specifies the number of bytes of data to be traced. The data parameter must be of type char* or unsigned char*.

These are the available macros:

- ISTRACE_BUFFER("buffer description", length, data);
- ISTRACE_BUFFER_ALERT("buffer description", length, data);
- ISTRACE_BUFFER_DEBUG("buffer description", length, data);
- ISTRACE_BUFFER_DIAG("buffer description", length, data);
- ISTRACE_BUFFER_ERROR("buffer description", length, data);

Class Tracing

This section describes how a class should be written to enable tracing. Enabling tracing for a class allows the trace facility to track member function entry and exit for that class, if you have specified ISTRACE_IN at the start of the function. Enabling tracing for a class also allows clients of the class to trace details of an object of the class using the ISTRACE_... macros.

To enable class tracing, the ISTRACE_DECLARE(className) macro should appear in the protected section of the class declaration.

Note: There should be no semicolon terminating this line. If you put a semicolon and attempt to compile with tracing turned off, the ISTRACE_DECLARE macro will be null. This will leave a single semicolon, which is not valid C++ code in a class declaration.

Also the ISTRACE_IMPLEMENT(className); macro (including the semicolon) should appear in the implementation file for the class. If more than one class is implemented in the file, an ISTRACE_IMPLEMENT macro should appear for each class.

Object Tracing

To allow clients of a class to write trace statements with an object as the value to be traced, you must provide a tracing function in the class to be traced. This function has the following form:

```
#ifdef ISTRACEON
void ISClassName::trace(void) const
{
  // Tracing code to go here
  ........
}
#endif
```

This function is declared for you by the ISTRACE_DECLARE macro; you just need to provide an implementation. The #ifdef...#endif pair removes the function when tracing is turned off, thereby preventing compiler errors. Inside the trace member you can use the macros described above to trace out the object state. It may be useful to use a range of macros at different tracing levels so that, for example, more detailed information is traced at the debug level than at the alert level. If the trace member function is being implemented for a derived class, you can call the base class trace member to write details known by the base. You should add tracing only for the data members added by the derived class.

Tracing the Call Stack

You can use the ISTRACE_DUMP_TRACED_CALL_STACK; macro to write the current call stack to the trace output. This macro can only record entries and exits of functions that are traced (that is, functions that have ISTRACE_IN specified at the start of the function). Calls to functions that are not traced (or have been compiled with tracing off) will not be shown.

Common Debugging Problems

Debugging problems occur in these categories:

- Problems with file names in a terminal application, but not in the controller or controller/terminal
- Problems that occur running in 4690 OS, but not in Windows

Problems with File Names

File names, including the path, have a maximum length of 26 bytes in the terminal but can be up to 128 bytes in the controller or controller/terminal. If you are having a problem with your terminal application running in a controller/terminal but not in a terminal, check any file or path names that you have added.

Logical names cannot be resolved in a terminal, but can be resolved in a controller or controller/terminal. If you try to resolve a logical name in the terminal, you will get back the name that you passed. The terminal generally passes the logical name to the controller in its request, and the controller resolves the name.

OS-specific Problems

Class variables that are not properly initialized might work in Windows but fail intermittently in 4690 OS. A Windows `new( )` gives back zeroed data space but 4690 OS does not.

Accessing FREED objects in 4690 OS can cause unpredictable results. When an instantiated object is FREED on the 4690 heap, its first two bytes are overwritten. For most objects, this will be the offset portion of their virtual function table pointer. If you later try to call one of its virtual functions that are not inline, you will get unpredictable results, which could include appearing to work or even working.

Attempting to access data outside the intended object's actual space can have unpredictable results in 4690 OS. It may cause the application to abend, appear to work, or, in some cases, actually work. The SurePOS ACE Heap Manager in 4690 OS gets 4K blocks from the OS and then segments that out to satisfy individual memory requests. Thus the application has access to the entire 4K, not just its allocated segment. If the code tries to access outside its allocated segment, it will be able to access data, but it will be the wrong data, as long as it does not go past the end of the actual 4K block. This can result in an intermittent problem.

Appendix H. Online Help

SurePOS ACE provides online help for menu bar options, pull-down menu items, and dialogs (notebook pages), which is accessed by pressing F1. Currently, help for fields is only available through links from the help for the panel on which the field appears.

Each help source file is used with one of the programs that provide the SurePOS ACE user interface. Table 139 lists each help file and the program that uses it.

Table 139. Help Source Files for SurePOS ACE

Help File	SurePOS ACE Program	Functions Included
aceachlp.txt	ACEACCNT	Accounting
acebohlp.txt	ACEBORUN	Main Menu, Manager Procedures menu and functions accessed from it, User Procedures menu and functions accessed from it
aceddhlp.txt	ACEDDMML	Batch (Delayed) Data Maintenance
acedmhlp.txt	ACEDMMLL	Data Maintenance, including maintenance of Item Data, Operator Authorization, Tender Verification, Customer Accounts, and Targeted Coupon Messages
aceevhlp.txt	ACEEVLMT	Electronic Voucher Authorization
acepehlp.txt	ACEPERSL	Personalization, including optional SurePOS ACE EPS options
acerehlp.txt	ACETVREL	Reports
acewkhlp.txt	ACEWRKEY	Working Key Management

Modifying SurePOS ACE Help

Some examples of why you might modify existing panels are:

- To add instructions for your store managers on one of the report help panels
- To redo the link text part of some index entries
- To add file name guidelines to the help for filing reports

When you use the SurePOS ACE Toolkit to extend the SurePOS ACE code, sometimes you will also want to add new help panels. Always add the new panel to the help source file for the program you have modified. For example, if you add user procedures to be accessed through the User Procedures button on the SurePOS ACE Main Menu, add the help panel to the acebohlp.txt file.

As long as you follow the formats and rules in "Formats, Rules and Guidelines for Help Panels" on page 706, you can modify existing help panel text or add new help panels to the source files. However, if you modify the source files, you increase the difficulty of integrating your changes into maintenance releases of SurePOS ACE. The preferred method of making changes to the help files uses files similar to the tiered overlay files that are used to make customer changes to the INI files. The syntax that is used in the help file overlays is described in "Using Tiered Overlays for Help Files" on page 710.

Formats, Rules and Guidelines for Help Panels

Help File Formats

Each file contains help panels for the programs associated with the file. In addition to help that describes the data that can be entered through the user interface, there is also an index to the help panels in the file, and a panel that describes keys used for navigating through the program. In some files there are also linked panels that describe concepts (for example, pricing methods in acedmhlp.txt).

Note: Because it contains help for multiple parts of the user interface, the acebohlp.txt file has a more complex structure than the other help files. Therefore, the following discussion has these parts:

- Standard information
- acebohlp.txt information

Standard Help File Format

With the exception of acebohlp.txt, the help files consist of the following:

- Main help panel for the function
- Panel with index to help panels
- Help panels for each screen or message box displayed in the user interface
- Panel that lists the common keys used by the program

acebohlp.txt Format

acebohlp.txt contains help for the SurePOS ACE Main Menu, manager procedures, and user procedures.

Each section of the help file contains the following:

- Help panel for the main panel in that section
- Panel with index to help panels
- Help panels for each screen or message box displayed in the user interface
- Panel that lists the common keys used by that part of SurePOS ACE

Help Panel Format Rules

Some characteristics of the source for all SurePOS ACE help panels are:

- Each help panel starts with a line that has *.topic* starting in column 1, followed by at least one space, the name of the help panel, an equal sign, and a topic ID. The following example is the first line for a help panel named *HelpPanel5* that has been assigned an ID of 2400.

  ```
  .topic     HelpPanel5=2400
  ```
- Each topic ID and help panel name must be unique within the help file in which it is used.
- Line comments can be used and are indicated by placing a # in both column 1 and column 2. You cannot use comments that start anyplace else within the line.

- Link statements have the format *Link Text:PanelName* enclosed in braces. For example, if the following character string is included in the text of a help panel, *next link* will display as part of the help and provide a link to the panel named *HelpPanel*.

```
{next link:HelpPanel}
```

- The maximum line length is 76 characters. If a line includes a link statement, the braces {} and the characters in *PanelName* are excluded when counting the number of characters.
- All parts of a link statement must be on one line. A link statement cannot begin on one line and end on the next line.

In all help files except acebohlp.txt, the following topic IDs in each help file are reserved for the specified use:

15000-15499
Help topics added by Toshiba Global Commerce Solutions Business Partners

15500-15999
Help topics added by users

The topic IDs for the sections of the acebohlp.txt help file are:

0-1999
Main Menu help

2000-2999
Help for the manager procedures, some of which are Selective Item Report and Terminal Monitor

3000-8999
Help for any user procedures that are accessed from the User Procedures push button on the SurePOS ACE Main Menu

9000-12999
Additional help for the manager procedures

13000-14999
Reserved

15000-15499
Help topics added by Toshiba Global Commerce Solutions Business Partners

15500-15999
Help topics added by users

Help Panel Format Guidelines

The guidelines used with the base SurePOS ACE help panels are:

- The second line of each panel contains a link to the Index panel for the file. In some help files, there are also links to higher-level help panels.
- The third and sixth lines of each help panel contain only blank characters.
- The fourth line of each help panel contains the title of the help panel, beginning in column 3. The fifth line contains characters that underscore the help panel title.
- The text of the help panel begins in column 1 of the seventh line.

- Each panel ends with a link to the main help panel for the function, a link to a General Help panel, and a link to a Common Keys panel.

Examples of Help Panel Formats

The main help panel for each program includes an outline of the menu options in the program. The menu options are given as link statements so that you can access the corresponding help panels. On the main help panel, as shown in Figure 53, sometimes the link to the Common Keys panel is included in text instead of being a standalone link at the end of the panel.

Help Panel

Figure 53 is an example of a main help panel. It shows representative parts of the main help panel for the Data Maintenance program.

```
.topic Nocontext=0, Main
{Index:Index}

  Data Maintenance
  ■■■■■■■■■■■■■■■■

Use Data Maintenance to view (display/print), add, erase, and change
specific data records.

For information on data maintenance tasks, press Enter to get the Index, or
tab to any of these highlighted phrases and press Enter for a description
of menu bar and pull-down menu options:

  *  {Item Data Maintenance:ItemData}
      --  {Pricing Data:PricingData}
      --  {Coupon:Coupon}
      --  {Item Type:ItemType}
      --  {Item Controls:ItemControls}
      --  {Misc/Coupon:MiscCoupon}
      --  {Taxes:Taxes}
      --  {User Data:UserData}
      --  {Optional:Optional}

  *  {Exit:ExitDM}

Tab to this highlighted phrase, {Common Keys:ComKeys} and press Enter
to view keys help.

  {General Help:GenHelp}
```

Figure 53. Example of Main Help Panel

Figure 54 is an example of a help panel other than the main help panel for a program. It shows representative parts of a help panel for the Data Maintenance program.

```
.topic UserData=1160
{Index:Index}  * {Item Data Maintenance:ItemData}

  User Data
  ■■■■■■■■■


Maintains user data and options applicable to item.

  1.  Select Data Maintenance from Main Menu and press Enter.
  2.  Select Item Data Maintenance from the pull-down menu and press Enter.
  3.  Page up or down to select User Data.
  4.  Enter the {item code:ItemCode} and retrieve or begin creating the record.
  5.  Select following information:

      User Data 1       Specifies user data value.
```

```
                        blank     No user data (default);
                        0-32767   Value of user data (32767 is $327.67);


  {Data Maintenance:Nocontext}
  {General Help:GenHelp}
  {Common Keys:ComKeys}
```

Figure 54. Example of Data Help Panel

Index Panel

Figure 55 is an example of the format of an Index panel. It shows representative parts of the Index help panel for the Data Maintenance program.

You can include links to any kind of panel that exists in the file. This example includes links to the AccountStatus panel, which is help for a panel in Tender Verification Maintenance. It also includes a link to the PM5 panel, which is conceptual information about pricing method 5.

```
.topic Index=1

  Index
  ■■■■■
  {Account Status:AccountStatus}
  {Account Limits:AccountLimits}
  {Address:CustomerAddress}
  {Adjustments:Adjustments}
  {Alias Item:PricingData}
  {Alias Pricing (Pricing Method 5):PM5}
```

Figure 55. Example of Index Help Panel

Common Keys Panel

Figure 56 is an example of the format of a Common Keys help panel. It shows representative parts of the Common Keys help panel for the Data Maintenance program.

```
.topic ComKeys=300
 {Index:Index}  * {General Help:GenHelp}

  Common Keys
  ■■■■■■■■■■■


  Alt + E         Edits targeted coupon message;
                  Erases saved record in all other DM functions.
  Alt + H         Shows host pricing in Item Data Maintenance.
  Alt + L         Loads (looks up) data from saved records.
  Alt + P         Prints record in Item DM;
                  Prints record in Operator Authorization Records.
  Alt + R         Restores host pricing in Item Data Maintenance.
  Alt + S         Saves edited changes.
  Alt + X         Returns to the application's Main Menu.

  Backspace       Deletes the character to the left in an input field.

  Ctrl + End      Moves to last notebook page.
  Ctrl + Home     Moves to first notebook page.

  Delete          Deletes the character to the right in an input field.
  Down arrow      Moves cursor down in a control;
                  Expands choices in a combo box.
```

```
Enter           Links to another help topic;
                Expands a combo box;
                Closes an expanded combo box;
                Performs function of highlighted push button.

F1              Enters the help system from any panel.
F2              Returns to the previous help panel.

Insert          Toggles between insert/overwrite in an input field.

{Data Maintenance:Nocontext}
```

Figure 56. Example of Common Keys Help Panel

Using Tiered Overlays for Help Files

The tiered overlay method of processing changes to help files accommodates conditional changes to help files (at build time) as well as customer changes or changes resulting from the distribution of maintenance. Overlays occur in files that have the same file name as the base help file. Up to ten tiered overlay files can exist for each help file. Tiered overlay file name extensions are:

.us0 through .us3
Toshiba Global Commerce Solutions-issued maintenance

.us4 through .us6
Toshiba Global Commerce Solutions Business Partner modifications

.us7 through .us9
User modifications

When a request is made to access a help file during a build, the base help file is accessed first. SurePOS ACE next attempts to open a series of files with the same name but with sequentially named extensions, from `us0` to `us9`. For example, after processing values in the base help file aceachlp.txt, SurePOS ACE attempts to process related files `aceachlp.us0`, `aceachlp.us1`, and so on to `aceachlp.us9`.

All Toshiba Global Commerce Solutions, Business Partner, and user-defined modifications are made in the sequential tiered overlay files that contain additional, replacement, and displacement entries for the base help file. SurePOS ACE applies these files in sequence, beginning with any Toshiba Global Commerce Solutions-issued maintenance, including Business Partner modifications, and finishing with your modifications.

Each applied tiered overlay file *overlays* the combination of information from preceding files. Although these overlay files are overlaid in sequence (from `us0` to `us9`), it is not necessary for all preceding files to exist to use a higher order file. For example, `aceachlp.us9` can exist without any preceding tiered overlay files and overlays the base as expected.

The tiered overlay method uses the same format as base help files with certain syntax extensions.

Syntax extensions overlay (change) or delete previously defined entries, or add new ones.

Note: The lines are added to the help file exactly as they are formatted in the overlay file. Be sure that your overlay file follows the format, rules, and guidelines given in "Formats, Rules and Guidelines for Help Panels" on page 706.

If you attempt to add a panel that already exists in the composite help file, there is no effect. However, an attempt to add a panel that already exists overlays the panel with the new panel.

If you attempt to delete or overlay a panel that does not have a match in the base or composite help file, SurePOS ACE does not perform any action for the entry and does not provide any notification that the attempt failed. Such a failed operation might occur if you specify a topic ID that another tiered overlay file has already deleted, or if you misspell a panel name.

Adding or Deleting Entire Panels

Delete entire panel

To delete an existing help panel, identify the panel name and topic ID and use the /D: parameter with the panel name. The syntax is:

```
 /D:.topic panelName=topicID
```

You will probably also need to remove references to the topic from the index section to avoid an 'unresolved forward reference' error.

Add entire panel

To add a panel to the base help file, use the /A: parameter with the new panel name and topic ID. The syntax is:

```
/A:.topic panelName=topicID
a line of text
another line of text
.
.
last line of text
```

This will add the topic to the help file only if the topic does not already exist. It will not replace an existing topic.

Modifying Existing Panels

To modify existing panels, you must have a way to identify what part of the panel will have information added, changed, or deleted. RogueWave regular expressions (*a regex expression* in the following syntaxes) and strings are the identification methods that SurePOS ACE uses.

Note: If you need to be precise in identifying where a change will be made, you can replace *"[a regex expression]"* in the following syntax statements with *"a regex expression as a line of text"*.

Topic modifications can use either text strings, or Rogue Wave regular expressions (RE). The line in a topic will be modified provided the text string is contained in the line (matching is case-sensitive), or if the RE yields a match.

A text string is what we think of as a line of text (for example, "F1 displays help for this window"). For detailed information about regular expressions, see “Using Rogue Wave Regular Expressions” on page 712.

Modifications in the middle of a help panel are done on a line by line basis, not by paragraph. Therefore, the regular expression for each line must include some text from the preceding line. For example, the syntax to add two lines after a specific line would be:

```
.topic panelName=topicID
/AA"[a regex expression]":first new line of text
/AA"[a regex expression using part of first new line of text]":a line of text
```

Adding lines at the end of a panel

The syntax is:

```
.topic panelName=topicID
/A:a line of text
/A:the next line of text
```

Adding lines after a specified point in a panel

The syntax is:

```
.topic aTitle=xxxx
/AA"[a bracketed regex rxpression]":a line of text
 Or
/AA"a multiple-character regex expression":a line of text
Or
/AAS"a line of text as a string":a line of text
```

Adding lines before a specified point in a panel

The syntax is:

```
.topic aTitle=xxxx
/AB"[a bracketed regex expression]":a line of text
Or
/AB"a multiple-character regex expression":a line of text
Or
/ABS"a line of text as a string":a line of text
```

Deleting a line at a specified point in a panel

The syntax is:

```
.topic aTitle=xxxx
/D"[a bracketed regex expression]":
Or
/D"a multiple-character regex expression":
Or
/DS"a line of text as a string":
```

Replacing a specified line

The syntax is:

```
.topic aTitle=xxxx
/R"[a bracketed regex expression]":a replacement line of text
Or
/R"a multiple-character regex expression":a replacement line of text
Or
/RS"a line of text as a string":a replacement line of text
```

Use this if you want to replace the links in the line after the .topic line for a help panel.

Modify a line within a topic with a text string, or a Regular Expression (RE)

These operation commands will have an "S" when you need to match based on strings. If the "S" is not present, then a regular expression will be used (single character, multiple character, or whatever is in the double quotes).

Users need to keep in mind this will cause the entire line to be replaced.

Using Rogue Wave Regular Expressions

Single-Character RE

A set of characters enclosed in brackets ([]) is a one-character RE that matches any of the characters in the set. For example, *[akm]* matches ether "a", "k", or "m".

A range of characters can be indicated with a dash, so [a-z] matches any lower case letter.

The caret (^) as the first character of a set will cause the RE to match any character except those in the set. For example, *[^akm]* matches any character except "a", "k", or "m". The caret loses this special meaning when it is not the first character of the set.

The period (.) matches any single character except new line. So ".umpty" would match either "Humpty" or "Dumpty".

Multiple-Character RE

A one-character RE followed by an asterisk (*) matches zero, or more occurrences of the RE. So, *[a-z]** matches zero, or more lowercase characters.

A one-character RE followed by a plus (+) matches one, or more occurrences of the RE. So, *[a-z]+* matches one, or more lowercase characters.

The question mark (?) specifies the RE can occur zero, or once in the string, but not multiple times. So, *xy?z* matches *xyz* or *xz*, but not *xyyz*.

Single-character REs can be concatenated. So, [A-Z][a-z] would match any capitalized word.

RE Special Characters

Regular expressions can be a single character, or multiple characters. The RE special characters are:

```
+  *  ?  .  [  ]  ^  $
```

If you need to match a special character in a line using an RE, the special character must be preceded by a backslash (\).

REs can be anchored to match only the beginning, or end of a line. If a caret (^) is at the beginning of the RE, then the matched string must be at the beginning of the line. If a dollar sign ($) is at the end of the RE, then the matched string must be at the end of the line.

The following escape codes can be used to match control characters:

/b
: Backspace

/e
: ESC (escape)

/f
: Formfeed

/n
: Newline

/r
: Carriage return

/t
: Tab

/xddd
: The literal hex number 0xddd

\^C

Control code (for example, \^D is "control-D)

How do you really use regular expressions?

A multiple-character RE *"Push"* would yield a match on any line with the word "Push" anywhere in it. You could add a caret to this RE *"^Push"*, and now a match occurs only when the line starts with the word *"Push"*. If you add a dollar sign to the end of this RE, *"Push$"*, now a match occurs only when the line ends with the word *"Push"*. You can concatenate multiple single-character REs.*[P][u][s][h]* yields a match on the line that contains*"P"*, *"u"*, *"s"*, *"h"* within a line in this order, but not necessarily together. You can have characters outside the brackets, so *"p[a-z]*"* would match the words *"park"*, *"par"*, *"p"*. If you are trying to match a string with a quote in it, you'll need to escape the character with a back slash. So, if you want to match the text *""move""* your RE would look like this: *"\"move\""*.

Some generic examples on what an RE might look like:

"^Push"

Any line that starts with Push

"I.*rt"

A word that starts with "I", and has a character after it, and ends with "rt". This would match a word like "Insert".

"\"F1\""

Would match a quoted F1 "F1".

"F10"

Would match the text F10.

Using TVHC to Compile SurePOS ACE Help

To apply your changes to the SurePOS ACE help panels, you must first compile the help source file. The TurboVision command is:

```
tvhc filename.txt filename.hlp filename.hpp
```

Copy the .hlp file into the adx_ipgm directory on your SurePOS ACE system.

Appendix I. SAPATH Migration

SurePOS ACE uses length-specific `sapath.rvt` and `sapath.ddf` files. The specific lengths are those item record lengths that have been supported at any time on SurePOS ACE. The item record lengths and their corresponding files are:

46
: `sap046.rvt` and `sap046.ddf`

101
: `sap101.rvt` and `sap101.ddf`

127
: `sap127.rvt` and `sap127.ddf`

169
: `sap169.rvt` and `sap169.ddf`

Messages

Various messages are issued during migration of the user's `sapath.ddf` file, depending on whether the user or Toshiba Global Commerce Solutions has made changes to the `sapath.ddf` file or the length of the Item Record File. The messages are logged in the migration log with reference to the migration state.

Migration State 0

This state occurs when there are no changes to the sapath file for the installed item record size and there is no new item record size from Toshiba.

Status Message Text: Sapath Migration: The installed sapath.ddf and the sapath.ddf from Toshiba Global Commerce Solutions are the same. The installed sapath.ddf was migrated.

Status Message Explanation: Migration completed without changes to the `sapath.ddf` file or the item record size.

Migration State 1

This state occurs when there are changes to the `sapath.ddf` file from Toshiba Global Commerce Solutions for the installed item record size, there are no customer changes to the `sapath.ddf` file, and there is no difference between the Toshiba Global Commerce Solutions and customer item record lengths.

Status Message Text: Sapath Migration: The latest sapath.ddf definitions from Toshiba Global Commerce Solutions have been installed. See Programmer Reference, Sapath Migration State 1.

Status Message Explanation: Toshiba Global Commerce Solutions has added or updated field definitions in the `sapath.ddf` file that apply to your configuration. The changes will be applied to your installed `sapath.ddf` file.

Migration State 2a

This state occurs when there are no changes to the `sapath` file for the installed item record size, but there is a difference between the installed item record size and the item record size from Toshiba Global Commerce Solutions.

Status Message Text: Sapath Migration: Toshiba has defined a new item record size, but there are no other sapath changes. Your currently installed sapath has been migrated. See Programmer Reference, Sapath Migration State 2a.

Status Message Explanation: The Toshiba-supplied item record length has changed. To preserve your environment, your previous `sapath.ddf` file was migrated. If you want to use the new item record length and any new fields in it, do the following after you have applied maintenance:

1. Copy the length-specific `sapath.ddf` and `sapath.rvt` files in the adx_ipgm directory to `adx_ipgm\sapath.ddf` and `adx_ipgm\sapath.rvt`.
2. Use the Keyed File Rebuild utility to resize your eamitemr.dat file to the new size.
3. Use Options -> Database personalization to enable the new fields that you want to use.
4. Set default values for the new fields in the item record. See "Setting Default Values for New Item Record Fields" on page 718.

Migration State 2b

This state occurs when there are new Toshiba definitions in the `sapath` file, there were no customer changes to the `sapath` file, and there is a difference between the installed item record size and the item record size from Toshiba.

Status Message Text: Sapath Migration: Toshiba has defined a new item record size, but the latest `sapath.ddf` matching your current item record size has been installed. See Programmer Reference, Sapath Migration State 2b.

Status Message Explanation: The Toshiba-supplied item record length has changed. To preserve your environment, your previous item record length was migrated and any new Toshiba record definitions within that length were updated. If you want to use the new item record length and any new fields in it, do the following after applying maintenance:

1. Copy the length-specific `apath.ddf` and `sapath.rvt` files in the adx_ipgm directory to `adx_ipgm\sapath.ddf` and `adx_ipgm\sapath.rvt` before applying the maintenance.
2. Use the Keyed File Rebuild utility to resize your eamitemr.dat file to the new size.
3. Use Options -> Database personalization to enable the new fields that you want to use.
4. Set default values for the new fields in the item record. See "Setting Default Values for New Item Record Fields" on page 718.

Migration State 3

This state occurs when there are new Toshiba definitions in the `sapath` file and there were customer overrides to the `sapath` file, but there is no difference between the installed item record size and the item record size from Toshiba.

Status Message

Message text: Sapath Migration: New sapath.ddf definitions from Toshiba have been defined. They have not been installed because your installation uses a customized sapath. See Programmer Reference, Sapath Migration State 3.

Message Explanation: The Toshiba-supplied item record definitions have changed. Your installation is using a version of the `sapath.ddf` file that has been modified from a previous Toshiba version. Your version as defined in the `acesql.ini` file has been migrated. It is important that you or your third-party vendor have done the following before installing SurePOS ACE to ensure that your modifications are preserved and that you integrate any new necessary Toshiba definitions or new optional Toshiba definitions that you want to use.

1. Save a text copy of your current sapath definitions to be used later to check that your existing modifications were preserved. (For example, you can use the DataMiner tool that is supplied with the SurePOS ACE Toolkit to print a copy of the item record definitions. On Windows, add a printer that uses the port FILE and device Generic/Text Only, then open your modified sapath definitions in DataMiner, select the file, and select File/Print.)
2. Rebuild your sapath definitions using the DataMiner tool `GCONF32.EXE` that is supplied with the SurePOS ACE Toolkit.

   a. Copy the new `sapath.rvt` file from Toshiba to the directory where you run the DataMiner tool and where you are editing your delta version of the sapath file. (This replaces your original Toshiba base file with the new version of the sapath file from Toshiba.)
   b. Save your delta RVT file and generate your new delta DDF file.
3. Repeat Step 1 on page 717, saving the output to a different file name.
4. Compare the two output files using any comparison tool.
5. Verify that any differences are Toshiba changes only and that your modifications have not been altered or reset to Toshiba values.
6. If the comparison results are not what you need, repeat Step 2 on page 717, reapplying your modifications. This might be necessary - there are known instances where customer modifications are reset to Toshiba values by the DataMiner tool.
7. After applying maintenance, if there are new Toshiba field definitions that you want to use, enable them in Options -> Database personalization.
8. Set default values for the new fields in the item record. See "Setting Default Values for New Item Record Fields" on page 718.

Error Messages

If either of the following messages occurs, it indicates a major error that you must correct for your installation to complete successfully.

Error Message 1 Text: Sapath Migration: Your installation uses a customized sapath file, but there is no customized DDF file in the installation directory, adx_imnt. See the Programmer Reference.

Error Message 1 Explanation: Your installation has an overlay file in the adx_ipgm directory. The name of the overlay file is of the form `acesql.us?`, where ? is a number from 0 to 9. The presence of this file in the adx_ipgm directory means that your installation is overriding some of the definitions in the Toshiba-supplied `sapath.ddf` file. The `acesql.us?` file contains the name of the delta DDF file that will be used to overlay Toshiba's definitions. This customized DDF file cannot be named `sapath.ddf` or begin with the letters "SAP", because these names are used to identify files from Toshiba. When you receive this error, it means that your customized DDF file was not found in the installation directory, adx_imnt, and therefore will not be installed when maintenance is applied.

Error Message 1 Action: Update your product control file to include your modified DDF file, so that it will be included in your installation package.

Error Message 2 Text: Sapath Migration: Your installation uses a customized sapath file, but the customized DDF file in the installation directory, adx_imnt, has not been integrated with new Toshiba definitions. See the Programmer Reference.

Error Message 2 Explanation: Your installation has an overlay file in the adx_ipgm directory. The name of the overlay file is of the form `acesql.us?`, where ? is a number from 0 to 9. The presence of this file in the adx_ipgm directory means that your installation is overriding some of the definitions in the Toshiba-supplied `sapath.ddf` file. The `acesql.us?` file contains the name of the delta DDF file that will be used to overlay Toshiba's definitions. This customized DDF file cannot be named `sapath.ddf` or begin with the letters "SAP", because these names are used to identify files from Toshiba. When you receive this error it means that your customized DDF file found in the installation directory, adx_imnt, had a date earlier than the new Toshiba-supplied DDF definitions. This indicates that you have not integrated your definitions with the Toshiba-supplied definition using the DataMiner tool.

Error Message 2 Action: Using the DataMiner tool, `GCONF32.EXE`, that is supplied in the SurePOS ACE Toolkit, apply your customized definitions to the new Toshiba base `sapath.ddf` file.

Migration State 4

This state occurs when there are new Toshiba definitions in the `sapath` file, there were customer changes to the `sapath` file, and there are differences between the installed item record size and the item record size from Toshiba.

Status Message Text: Sapath Migration: Toshiba has defined a new item record size. It has not been installed because your installation uses a customized sapath. See Programmer Reference, Sapath Migration State 4.

Status Message Explanation: This message occurs under conditions similar to the Migration State 3 message, except that there are file size differences. If this message is received, it is important that you or your third-party vendor have done the following before installing SurePOS ACE:

1. Perform the steps in the explanation for Migration State 3.
2. If you choose to use the new record size from Toshiba, use the Keyed File Rebuild utility to resize your eamitemr.dat to the new record size.

Error Messages: See “Error Messages” on page 717 for information about the two supplemental messages that indicate errors if they occur.

Setting Default Values for New Item Record Fields

If new fields are added to the item record from new Toshiba definitions, a new record size, or new fields added by you, Toshiba suggests that you initialize the new fields with appropriate data values.

Appendix J. Personalization Management via DIF API

This appendix provides detailed information on the messages used to manage personalization using DIF. The setup of this solution is documented in "Using DIF to Maintain Personalization Options" on page 586.

Communications overview

Communication in this API follows two messaging models: requests and events. The retailer can make requests to perform personalization actions and ACEPERSL can send events to the retailer to provide information on when options change. The solution used by the retailer to interact with DIF is referred to as the "retailer's enterprise" in this appendix.

To make requests, the retailer's enterprise communicates with DIF using this API. DIF will construct a response message that conforms to this API to send to the retailer's enterprise.

Component responsibilities

The responsibility of each component includes:

ACEPERSL

- Provide functionality to update options, notify terminals to load options, and query current options settings.
- Via DIF, notifies the retailer's enterprise whenever an option is changed, regardless of the method used to make the change.
- When running in UI mode, ACEPERSL provides a way for DIF to cause the program to end without making any option changes. This will be used to automatically end a UI instance of ACEPERSL when an options update request is issued. Note that UI users will be warned of this occurrence via a message indicating that ACEPERSL must be restarted.

DIF

- Interprets API messages from the retailer's enterprise and invokes the appropriate ACEPERSL command.
- Passes the update options request to ACEPERSL to be processed.
- Passes API messages from ACEPERSL to the retailer's enterprise. Also provides functionality to update options, notify terminals to load options, and query current option settings.

Retailer's enterprise

- Makes a request to perform personalization actions: update option, notify terminals to load options, and query current options settings.
- In the UpdateOptionsRequest, only options that can be updated using the ACEPERSL command line parameter can be updated. Areas such as tax tables and miscellaneous reports cannot be updated since they cannot be updated using the /u parameter.
- Handles unsolicited messages describing options changes.

Implementation notes

To successfully conform to the DIF API, keep in mind the following guidelines:

- Update options requests are atomic operations. The changes are only made if all of the requested changes can be made; no partial updates will occur.

- In the ACE personalization GUI and .ini files, a value of +++ can be used to indicate "not applicable". However, ACEPERSL actually stores those values as the largest possible number that data type can hold, as "+++" is not a valid number.

  Note: Because the schema will expect proper numerical values, this API will use the ultimate largest possible numbers rather than "+++". For integers, this number is 32768. For long date types, this number is 2147483648.
- Only options stored in ACEPERSL, APSOPTNS, NLSOPTNS, or ACECATE5 are considered. Areas such as tax tables and miscellaneous reports, which reside outside of those four files, are not processed in this solution.
- The XML option can contain the following special characters:

  ```
  <   &lt;
  >   &gt;
  &   &amp;
  "   &quot;
  '   &apos;
  ```

  For example, use `&quot;` for the `"` character.

  If the host read pipe name is changed to <HOSTPIPE>, it would appear as follows in the OptionsChangedEvent:

  ```
  <HostReadPipe>&lt;HOSTPIPE&gt;</HostReadPipe>
  ```

  Option XML is formed using the section and option names found in the options layout files, not the names found in the GUI. These names are similar, but not identical. The following is an example of how to take an option name from the GUI and find its name in the options layout file:

  Example: The four options layout files are optnparm.ini (contains options from EAMOPTNS), apsparms.ini (APSOPTNS), nlsparms.ini (NLSOPTNS), and acecpntr.ini (ACECATE5). To locate the option Options->Security->Authorization->Password Expiration (days), open optnparm.ini and search on PasswordExpiration to locate the following line:

  ```
  PasswordExpirationDays           = 21, UInt,           0
  ```

  Therefore, the correct option name to use is PasswordExpirationDays. Scrolling up will locate the section name in brackets. For example:

  ```
   [Security]
  ```

  Therefore, to update this option's value to 10, the following XML would be used:

  ```
  <Security>
            <PasswordExpirationDays>10</PasswordExpirationDays>
  </Security>
  ```

Message details

The following section includes the details of each message:

Load options request

- Request ID - A universally unique identifier (UUID) assigned to the request by the retailer.
- Date - In YYMMDD format
- Time - In HH:MM (24 hour) format
- Terminal numbers (optional) - Represents the terminal numbers where load options are included. Use a range of the format x-y; you can specify multiple terminals or

ranges that should be delimited by commas. A value of 0 indicates all terminals. If this field is included, the terminal group numbers field will be ignored if present.
- Terminal group numbers (optional) - Represents the terminal group numbers where load options are included. The same formatting as the terminal number fields applies. A value of 0 indicates all terminal groups. This field will be ignored if the terminal number field is included.

Load options result

- Request ID - A universally unique identifier (UUID) assigned to the request by the retailer.
- Return code - An integer with one of the following possible values:
  - 0 - Success, no errors encountered
  - 1 - Invalid request XML
  - 2 - One or more of the terminals specified does not exist. The terminals that existed are still notified to load options.
  - 3 - Represents one of the following:
    - One or more of the terminal groups specified does not exist. The groups that do exist are still notified to load options.
    - The terminal or terminal groups are specified using invalid syntax (that is, `<TerminalNumbers>one</TerminalNumbers>`; therefore, terminals will not be directed to load options.
- Error text – A string containing a plain text description of the error. Present only if the return code is non-zero.

Update options request

- Request ID - A universally unique identifier (UUID) assigned to the request by the retailer.
- Date - In YYMMDD format.
- Time - In HH:MM (24 hour) format.
- File to update – Indicates the file that will be updated. The four valid files are EAMOPTNS, APSOPTNS, NLSOPTNS, and ACECATE5. Additionally, EAMOPTNS can be qualified with a terminal or terminalGroup attribute to indicate terminal or terminal group specific options.
- The structure of this message is best suited to examples. Essentially, the field names are all taken from the option paths, so there are hundreds of different fields that can be present.

  Note: Only one options file can be changed per request.

Update options result

- Request ID - A universally unique identifier (UUID) assigned to the request by the retailer.
- Return code - An integer with one of the following possible values:
  - 0 - Success, no errors encountered
  - 1 - Invalid request XML
  - 2 - A nonexistent terminal or terminal group number was specified
  - 3 - The terminal or terminal group was specified using invalid syntax, for example:

    ```
    <TerminalNumbers>one</TerminalNumbers>
    ```
  - 4 - Invalid section name
  - 5 - Invalid option name

    - 6 - Invalid option value (for example, letters instead of numbers)
    - 7 - Unable to obtain lock on ACEPERSL options files, which can be the result of a previous UpdateOptionsRequest still running when another is received
    - 8 - Error accessing files used to communicate between ACE and DIF
    - 9 - No valid file found in UpdateOptionsRequest
    - 10 - Error reading options file (for example, *eamoptns.dat* does not exist)
- Error text - A string containing a plain text description of the error (included only if the return code is non-zero)

  Note: If the return code is non-zero, options have not been updated (that is, the UpdateOptionsRequest is an atomic, "all or nothing" operation).

Options changed event

- Date - In YYMMDD format
- Time - In HH:MM (24 hour) format
- Operator ID – The ID of the operator who made the changes. If the changes were made via an options update request, the operator ID will be taken from Rpt_Backgrnd in aceglrsr.ini, which is used in the same capacity by reports. The default is Backgrnd.
- Terminal number
- Terminal group number
- The structure of this message is very similar to the options update request. It will simply contain all of the options that were changed.
- The structure of this message is best suited to examples. Essentially, the field names are all taken from the option paths, so there are hundreds of different fields that can be present.

  Note: Since only one options file can be changed per request, a separate options changed event message is sent for each file, terminal, or terminal group changed. This event is sent whenever option changes are saved to disk, regardless of whether the changes occurred via the GUI interface or as a result of an options update request.

Options query request

- Request ID - A universally unique identifier (UUID) assigned to the request by the retailer.
- Date - In YYMMDD format.
- Time - In HH:MM (24 hour) format.

Options query result

- Request ID - A universally unique identifier (UUID) assigned to the request by the retailer.
- Options, using the same syntax as the options changed event.

ExceptionResult

Sent when an unrecognized root element is received. Includes a message with the unrecognized root element. This is generally the result of improper DIF configuration (that is, an UpdateOptionsRequest being routed to the LoadOptionsRequest actor).

API Examples

The following are examples of DIF APIs:

Example 1: Load options for all terminals

In this example, all terminals are specified to load options.

Message 1

```
<persl:LoadOptionsRequest
       xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <RequestID>eb92b140-b0c9-11e1-b51b-0002a5d5c51b</RequestID>
            <Date>120701</Date>
            <Time>20:00</Time>
            <TerminalNumbers>0</TerminalNumbers>
</persl:LoadOptionsRequest>
```

Message 2

The response indicates that all terminals were successfully told to load.

Note: This does not mean they have actually reloaded options yet; just that the EAMTERMS bit indicating to do so has been set.

```
<persl:LoadOptionsResult
       xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <RequestID>eb92b140-b0c9-11e1-b51b-0002a5d5c51b</RequestID>
            <ReturnCode>0</ReturnCode>
</persl:LoadOptionsResult>
```

Example 2: Load options for a specific subset of terminals

In this example, terminals 1 through 5, 8, and 12 are specified to load options:

Message 1

```
<persl:LoadOptionsRequest
       xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <RequestID>04cf4380-b0ca-11e1-8dae-0002a5d5c51b</RequestID>
            <Date>120701</Date>
            <Time>20:00</Time>
            <TerminalNumbers>1-5,8,12</TerminalNumbers>
</persl:LoadOptionsRequest>
```

Message 2

```
<persl:LoadOptionsResult
       xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <RequestID>04cf4380-b0ca-11e1-8dae-0002a5d5c51b</RequestID>
            <ReturnCode>0</ReturnCode>
</persl:LoadOptionsResult>
```

Example 3: Load options for a terminal group

In this example, all terminals in terminal group 3 are specified to load options:

Message 1

```
<persl:LoadOptionsRequest
       xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <RequestID>1034be80-b0ca-11e1-aedf-0002a5d5c51b</RequestID>
            <Date>120701</Date>
            <Time>20:00</Time>
            <TerminalGroupNumbers>3</TerminalGroupNumbers>
</persl:LoadOptionsRequest>
```

Message 2

```
<persl:LoadOptionsResult
       xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <RequestID>101034be80-b0ca-11e1-aedf-0002a5d5c51b</RequestID>
            <ReturnCode>0</ReturnCode>
</persl:LoadOptionsResult>
```

Example 4: Update global options

In this example, several different options with a format of boolean, integer, array, or string are updated. The options are not terminal or terminal-group specific.

Message 1

The message indicating which options to change is sent. The names for the options, as well as the path to the options, is taken from the structure laid out in optnparm.ini. Each actual option has an attribute indicating its type, and an attribute "array" if it is an array. Again, this mirrors the format of the .ini files used to update options.

Note: It is invalid for an XML schema to have an element name that starts with a number, so any ACE section or option name that starts with a number will have the single, lower case character "n" prepended to it.

```
<persl:UpdateOptionsRequest
       xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
         <RequestID>19240960-b0ca-11e1-9203-0002a5d5c51b</RequestID>
         <Date>120701</Date>
         <Time>20:00</Time>
         <EAMOPTNS>
              <Store>
                <DiskSpaceWarningC>40</DiskSpaceWarningC>
                   <PromptForEatIn>TRUE</PromptForEatIn>
                      <Phone1>(123) 555-6789</Phone1>
              </Store>
              <Coupons>
                   <AllowAsFirstItem>TRUE</AllowAsFirstItem>
                      <StoreCouponsTaxable>TRUE</StoreCouponsTaxable>
              </Coupons>
              <Security>
                      <PriceOverrideLimit>
                      <Element>200</Element>
                      <Element>1000</Element>
                      <Element>10000</Element>
                      </PriceOverrideLimit>
              </Security>
              <n4610Specific>
                     <LogoFile>logo.lgo</LogoFile>
              </n4610Specific>
      </EAMOPTNS>
</UpdateOptionsRequest></persl:UpdateOptionsRequest>
```

Message 2

This is the event that is sent whenever an options change takes place. The PromptForEatIn option was already set to TRUE and is not included in this event because it did not change.

```
<persl:OptionsChangedEvent
       xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <Date>120701</Date>
            <Time>20:00</Time>
            <OperatorID>Backgrnd </OperatorID>
            <EAMOPTNS>
                        <Store>
```

```
                                    <DiskSpaceWarningC>40</DiskSpaceWarningC>
                                    <Phone1>(123) 555-6789</Phone1>
                        </Store>
                        <Coupons>
                                    <AllowAsFirstItem>TRUE</AllowAsFirstItem>
                                    <StoreCouponTaxable>TRUE</
StoreCouponsTaxable>
                        </Coupons>
                        <Security>
                                    <PriceOverrideLimit>
                                    <Element>200</Element>
                                    <Element>1000</Element>
                                    <Element>10000</Element>
                                    </PriceOverrideLimit>
                        </Security>
            </EAMOPTNS>
</persl:OptionsChangedEvent>
```

Message 3

A result of success is sent because all options are now set as requested, even though PromptForEatIn did not actually change.

```
<persl:UpdateOptionsResult
   xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
     <RequestID>19240960-b0ca-11e1-9203-0002a5d5c51b</RequestID>
     <ReturnCode>0</ReturnCode>
</persl:UpdateOptionsResult>
```

Example 5: Update terminal specific and terminal group specific options

This example demonstrates how multiple sets of options can be updated using only one message. Terminal specific and terminal group specific options are updated. Two options update messages must be sent, as only one file can be updated per request.

Message 1

The message indicating which options to change is sent. The options file has an attribute indicating which terminal or group it pertains to.

```
<persl:UpdateOptionsRequest
     xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
      <RequestID>23015dc0-b0ca-11e1-8efb-0002a5d5c51b</RequestID>
      <Date>120701</Date>
      <Time>20:00</Time>
      <EAMOPTNS terminal="325">
         <Printing>
                 <StoreHeader1>****************</StoreHeader1>
                    <StoreHeader2>Customer Service</StoreHeader2>
              <StoreHeader3>****************</StoreHeader3>
         </Printing>
      </EAMOPTNS>
</persl:UpdateOptionsRequest>
```

Message 2

```
<persl:OptionsChangedEvent
     xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
      <Date>120701</Date>
      <Time>20:00</Time>
      <OperatorID>Backgrnd </OperatorID>
      <EAMOPTNS terminal="325">
         <Printing>
```

```
               <StoreHeader1>****************</StoreHeader1>
                     <StoreHeader2>Customer Service</StoreHeader2>
                     <StoreHeader3>****************</StoreHeader3>
        </Printing>
     </EAMOPTNS>
</persl:OptionsChangedEvent>
```

Message 3

```
<persl:UpdateOptionsResult
       xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <RequestID>23015dc0-b0ca-11e1-8efb-0002a5d5c51b</RequestID>
            <ReturnCode>0</ReturnCode>
</persl:UpdateOptionsResult>
```

Message 4

```
<persl:UpdateOptionsRequest
       xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <RequestID>2c693400-b0ca-11e1-80da-0002a5d5c51b</RequestID>
            <Date>120701</Date>
            <Time>20:00</Time>
            <EAMOPTNS terminalGroup="4">
                    <Printing>
                            <StoreHeader1>===============</StoreHeader1>
                            <StoreHeader2>Self Checkout</StoreHeader2>
                            <StoreHeader3>===============</StoreHeader3>
                    </Printing>
            </EAMOPTNS>
</persl:UpdateOptionsRequest>
```

Message 5

```
<persl:OptionsChangedEvent
       xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <Date>120701</Date>
            <Time>20:00</Time>
            <OperatorID>Backgrnd </OperatorID>
            <EAMOPTNS terminalGroup="4">
                    <Printing>
                            <StoreHeader1>===============</StoreHeader1>
                            <StoreHeader2>Self Checkout</StoreHeader2>
                            <StoreHeader3>===============</StoreHeader3>
                    </Printing>
            </EAMOPTNS>
</persl:OptionsChangedEvent>
```

Message 6

```
<persl:UpdateOptionsResult
     xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
      <RequestID>2c693400-b0ca-11e1-80da-0002a5d5c51b</RequestID>
      <ReturnCode>0</ReturnCode>
</persl:UpdateOptionsResult>
```

Example 6: Updating an option list

This example demonstrates how to update an option that resides inside a list.

Message 1

In the previous examples, all options were constructed using the section and options names from optnparm.ini. When updating a list, the procedure is somewhat different; in optnparm.ini, a separate section name is present for each member of a list.

These section names end with a unique numerical identifier that is not necessarily the same as the ID attribute of the section. For example, in default ACE options the Visa section has an ID attribute of 41, but the section name has a numerical identifier of 9.

When constructing an options update request, the ID attribute should be used to identify the section of the list that should be updated, not the section name. In this example, two different tenders have their options updated and a third tender is added. Also, a list can have a similar name to another, non-list set of options. For example, there are both global tender options and list tender options.

```
<persl:UpdateOptionsRequest
    xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
      <RequestID>37379f20-b0ca-11e1-99d6-0002a5d5c51b</RequestID>
      <Date>120701</Date>
      <Time>20:00</Time>
      <EAMOPTNS>
         <Tender>
                    <ForeignCurrency>TRUE</ForeignCurrency>
                </Tender>
                <TenderList>
                            <TenderListSection id="41">
                                <OpenDrawer>FALSE</OpenDrawer>
                            </TenderListSection>
                            <TenderListSection id="11">
                                <OpenDrawer>TRUE</OpenDrawer>
                            </TenderListSection>
                            <TenderListSection id="12">
                                <Type>1</Type>
                                <Variety>2</Variety>
                                <Description>New tender</Description>
                                <ShortDescription>NTND</ShortDescription>
                            </TenderListSection>
            </TenderList>
     </EAMOPTNS>
</persl:UpdateOptionsRequest>
```

When adding a section to a list, if not all fields are specified, default values will be used for the missing fields. This will be reflected in the options changed event.

Message 2

The values for the tender limits are set at 2147483648. This is what ACEPERSL stores +++, as in the options files for options of type long.

```
<persl:OptionsChangedEvent
   xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
     <Date>120701</Date>
     <Time>20:00</Time>
     <OperatorID>Backgrnd </OperatorID>
     <EAMOPTNS>
        <TenderList>
                <TenderListSection id="41">
                    <OpenDrawer>FALSE</OpenDrawer>
                </TenderListSection>
                <TenderListSection id="11">
                    <OpenDrawer>TRUE</OpenDrawer>
                </TenderListSection>
                <TenderListSection id="12">
                    <ID>12</ID>
                         <Type>1</Type>
                         <Variety>2</Variety>
                         <ShortDescription>NTND</ShortDescription>
                         <ChangeLimit>
                             <Element>2147483648</Element>
```

```
                              <Element>2147483648</Element>
                              <Element>2147483648</Element>
                         </ChangeLimit>
                         <MaxLimit>
                              <Element>2147483648</Element>
                              <Element>2147483648</Element>
                            <Element>2147483648</Element>
                         </MaxLimit>
                         <MinLimit>
                             <Element>2147483648</Element>
                             <Element>2147483648</Element>
                             <Element>2147483648</Element>
                         </MinLimit>
                         <MaxQuantityLimit>999</MaxQuantityLimit>
                         ... (rest of the default values)
                 </TenderListSection>
       </TenderList>
     </EAMOPTNS>
</persl:OptionsChangedEvent>
```

Message 3

```
<persl:UpdateOptionsResult
    xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
     <RequestID>37379f20-b0ca-11e1-99d6-0002a5d5c51b</RequestID>
            <ReturnCode>0</ReturnCode>
</persl:UpdateOptionsResult>
```

Message 4

An option list can be deleted as follows providing the DeleteFlag is present in the list when you perform an options query. See "Example 8: Query all options" on page 729.

```
<persl:UpdateOptionsRequest
   xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
     <RequestID>19240960-b0ca-11e1-9203-0002a5d5c51b</RequestID>
     <Date>121030</Date>121030>
   <Time>10:28</Time>10:28>
   <EAMOPTNS>
     <RestrictedSalesList>
        <RestrictedSalesListSection id="2">
          <DeleteFlag>true</DeleteFlag>
        </RestrictedSalesListSection>
     </RestrictedSalesList>
   </EAMOPTNS>
 </persl:UpdateOptionsRequest>
```

Message 5

```
<persl:OptionsChangedEvent
    xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
      <Date>121220</Date>
   <Time>14:14</Time>
   <OperatorID>1</OperatorID>
   <EAMOPTNS>
     <RestrictedSalesList>
        <RestrictedSalesListSection id="2">
          <DeleteFlag>true</DeleteFlag>
        </RestrictedSalesListSection>
     </RestrictedSalesList>
   </EAMOPTNS>
 </persl:OptionsChangedEvent>
```

Message 6

```
<persl:UpdateOptionsResult
   xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
     <RequestID>19240960-b0ca-11e1-9203-0002a5d5c51b</RequestID>
```

```
            <ReturnCode>0</ReturnCode>
</persl:UpdateOptionsResult>
```

Example 7: Options updated via the UI

As noted in the message details, an options changed event is sent whenever an options file is updated, whether it is updated by an XML request or the UI. This example simply shows an unsolicited options change event sent when a user updated options is using the ACEPERSL UI.

Message 1

```
<persl:OptionsChangedEvent
   xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
     <Date>120701</Date>
            <Time>20:00</Time>
            <OperatorID>1</OperatorID>
            <EAMOPTNS>
                    <Store>
                            <DiskSpaceWarningC>40</DiskSpaceWarningC>
                            <Phone1>(123) 555-6789</Phone1>
                    </Store>
                    <Coupons>
                            <AllowAsFirstItem>FALSE</AllowAsFirstItem>
                            <StoreCouponTaxable>FALSE</StoreCouponsTaxable>
                    </Coupons>
            </EAMOPTNS>
</persl:OptionsChangedEvent>
```

When adding a section to a list, if not all fields are specified, default values are used for the missing fields. This is reflected in the options changed event.

Example 8: Query all options

In this example, the settings for all options are queried.

Message 1

```
<persl:OptionsQueryRequest
       xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <RequestID>412ee420-b0ca-11e1-80e5-0002a5d5c51b</RequestID>
            <Date>120701</Date>
            <Time>20:00</Time>
</persl:OptionsQueryRequest>
```

Message 2

The results will probably be over one megabyte in size, as it contains every option. In the following example, only a few options are shown to establish the pattern.

```
<persl:OptionsQueryResult
       xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <RequestID>412ee420-b0ca-11e1-80e5-0002a5d5c51b</RequestID>
            <EAMOPTNS>
                    <Store>
                                <Name>Store Name</Name>
                                <Division>SUREPOS ACE</Division>
                                <Number>1</Number>
                                … (Rest of store options)
                    </Store>
                    <Security>
                                <EncryptPasswords>FALSE</EncryptPasswords>
                                <DrawerOpenTimeLimit>30</DrawerOpenTimeLimit>
                                … (Rest of security options)
                        </Security>
                        … (Rest of global EAMOPTNS)
                </EAMOPTNS>
                … (APSOPTNS, NLSOPTNS)
                <EAMOPTNS terminal="3">
                        <ItemLimits>
                                <ItemSaleAmount>
                                        <Element>1000</Element>
                                        <Element>2000</Element>
                                        <Element>5000</Element>
                                </ItemSaleAmount>
                        </ItemLimits>
                        … (Rest of EAMOPTNS specific to terminal 3)
                </EAMOPTNS>
                … (Rest of terminal specific and/or terminal group specific
options)
</persl:OptionsQueryResult>
```

Example 9: Error loading terminal options

In this example, an invalid terminal number is specified when trying to load terminal options.

Message 1

```
<persl:LoadOptionsRequest
        xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <RequestID>04cf4380-b0ca-11e1-8dae-0002a5d5c51b</RequestID>
            <Date>120701</Date>
            <Time>20:00</Time>
            <TerminalNumbers>789</TerminalNumbers>
</persl:LoadOptionsRequest>
```

Message 2

```
<persl:LoadOptionsResult
        xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <RequestID>04cf4380-b0ca-11e1-8dae-0002a5d5c51b</RequestID>
            <ReturnCode>2</ReturnCode>
            <ErrorText>Record not found for terminal 789.</ErrorText>
</persl:LoadOptionsResult>
```

Example 10: Error updating options

In this example, an invalid option is specified; therefore, options are not updated.

Message 1

```
<persl:UpdateOptionsRequest
   xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
    <RequestID>19240960-b0ca-11e1-9203-0002a5d5c51b</RequestID>
    <Date>120701</Date>
    <Time>20:00</Time>
    <EAMOPTNS>
            <Store>
                    <DiskSpaceWarningC>40</DiskSpaceWarningC>
                    <PromptForEatIn>TRUE</PromptForEatIn>
                        <Phone1>(123) 555-6789</Phone1>
            </Store>
            <Coupons>
```

```
                     <AllowAsFirstItem>TRUE</AllowAsFirstItem>
                     <StoreCouponsTaxable>TRUE</StoreCouponsTaxable>
                     <NonexistentOption>3</NonexistentOption>
             </Coupons>
     </EAMOPTNS>
</persl:UpdateOptionsRequest>
```

Message 2

In this example, the return code indicates an error. The OptionsChangedEvent was not sent, even though several valid options were contained in the update request.

```
<persl:UpdateOptionsResult
       xmlns:persl="http://bc.ace.retail.ibm.com/PersonalizationSchema">
            <RequestID>19240960-b0ca-11e1-9203-0002a5d5c51b</RequestID>
            <ReturnCode>5</ReturnCode>
            <ErrorText>Option “NonexistentOption” does not exist in section
       “Coupons”.</ErrorText>
</persl:UpdateOptionsResult>
```

Example 11: Invalid request error

In this example, the retailer sends an invalid message to DIF.

Message 1

```
<InvalidRequest>
            <Anything>here</Anything>
</InvalidRequest>
```

Message 2

```
<ExceptionResult>Unrecognized root element
"InvalidRequest".</ExceptionResult>
```

Notices

Trademarks

Glossary

This glossary includes terms and definitions from the *IBM Dictionary of Computing* (New York; McGraw-Hill, Inc., 1994).

address
: A location in the storage of a computer where particular data is stored. Also, the digits that identify such a location.

alphanumeric
: Pertaining to a character set containing letters, digits, and other characters, such as punctuation marks.

amount tendered
: Partial or complete payment given to a salesperson by a customer for merchandise.

application
: (1) A program written for or by a user that applies to his own work. (2) A program, written for or by a user that applies to a particular application. (3) A program written for or by a user that is specific to the user's own application.

at close mode
: A file mode in which a file is distributed when the file is closed. Contrast with *per update mode*.

backup
: Pertaining to a system, device, file, or facility that can be used in the event of a malfunction or the loss of data.

bar code
: A code representing characters by sets of parallel bars of varying thickness and separation that are read optically by transverse scanning.

BASIC
: Beginner's All-purpose Symbolic Instruction Code. A programming language that uses common English words.

byte
: The space allotted in a computer processing unit for storing data that represents an alphanumeric character in the character set.

call
: The action of bringing a function or subprogram into effect, usually by specifying the entry conditions and jumping to an entry point.

cash drawer
: A drawer at a point of sale terminal that can be programmed to open automatically. See *till*.

cash receipt
: An itemized list of merchandise purchased and paid for by the customer.

Catalina marketing
: A method of providing coupons to the customer for use with future purchases. Using a computer and software provided by the Catalina Marketing Company, coupons based on the items purchased may be printed at the end of a transaction.

chaining
: A method of storing records in which each record belongs to a list or group of records and has a linking field for tracing the chain.

charge
: A sales transaction in which a customer has the partial or total value of purchased merchandise added to an account for later payment.

checkout system
: The hardware and software products which perform functions at the front end of a store (checkout area).

checkpoint

A point at which information about the status of a job and the system can be recorded so that the job step can be restarted later.

clear

To delete data from a screen or from storage.

command

A request made from a terminal to perform an operation or to run a program.

configuration

The group of devices and programs that make up a data processing system or network.

control character

A character whose occurrence in a particular context initiates, modifies, or stops a control operation. A control character may be recorded for use in a subsequent action, and it may have a graphic representation in some circumstances.

cursor

A movable point of light (or a short line) that indicates where the next character is to be entered on the display screen.

customer display

The terminal display that shows only sales transaction information. Compare with *system display*.

customize

To tailor a program or store system through option selection.

data

Facts, concepts, or instructions suitable for communication, interpretation, or processing.

database

A collection of data fundamental to a system. A collection of interrelated or independent data items stored together without redundancy, to serve one or more applications.

data file

A collection of related data records organized in a specific manner; for example, a payroll file (one record for each employee, showing such information as rate of pay and deductions) or an inventory file (one record for each inventory item, showing such information as cost, selling price, and number in stock.) See also *data set, file.*

data processing system

A network, including computer systems and associated personnel, that accepts information, processes it according to a plan, and produces the desired results.

data set

Logically related records treated as a single unit. See also *file.*

default value

The value the system supplies when the user does not specify a value.

department motor key

An item lookup key with a price entry required.

disk

A round, flat plate coated with a magnetic substance on which data for a computer is stored. See also *integrated disk.*

diskette

A thin, flexible magnetic disk permanently enclosed in a protective jacket. A diskette is used to store information until it is required for processing.

display

(1) A visual presentation of data. (2) A device that presents visual information to the point of sale terminal operator and to the customer, or to the display station operator.

distributed

Physically separate but connected by cables.

EAN

See *European article number*.

electronic funds transfer (EFT)

Electronic funds transfer is a method of payment where the customer uses a bank debit card and a personal identification number in a magnetic stripe reader/PIN pad to debit their account for their purchases.

element

In a set, an object, entity, or concept having the properties that define a set.

error message

A message that is issued because an error has been detected.

European article number (EAN)

A number that is assigned to and encoded on an article of merchandise for scanning in certain IBM World Trade countries.

evaluation

Reduction of an expression to a single value.

feature

A specific design addition to an IBM product, quoted by the sales manual and can be ordered by the user.

field

In a record, a specified area used for a particular category of data.

file

The term for a collection of logically related records treated as a single unit. For example, an invoice may form a record and the complete set of such records may form a file. See also *data set*.

file access

Methods of entering a file to retrieve the information stored in the file.

file mode

Attribute that tells when a file is distributed.

file name

(1) A name assigned to a file. (2) A term that identifies a file in the control unit.

file type

Extension to a file name. A filetype is optional and can be up to three characters long. All characters permitted in a file name are permitted in a file type. The file type must be separated from the file name by a period.

fixed disk

A disk of rigid material with a magnetic coating, used for mass storage and retrieval of data.

flag

A character that signals the occurrence of some condition, such as the end of a word.

franking

Printing an indication on a document that the document has been processed. This franking may be a store header line, a "total" line, or a transaction number that is printed when a check, a discount coupon, or a gift certificate is inserted in the document insert station of the point of sale terminal during certain types of transactions.

front end

(1) Operators located at checkout registers. (2) One of the two main components of the compiler. The front end consists of the lexical analyzer, the parser, and the symbol table generator.

FSI coupon

A free-standing insert coupon.

function

(1) A specific purpose or characteristic action. (2) In data communications, a machine action such as a carriage return or line feed. (3) A subroutine that returns the value of a single variable and usually has a single exit.

function key

A key on a terminal, such as an Enter key, that causes the transmission of a signal not associated with a character that can be printed or displayed. Detection of the signal usually causes the system to perform some predefined action for the operator.

gross minus

The accumulation, during a sales period, of all negative amounts (such as refunds, allowances, or discounts) entered or calculated in sales transactions at the point of sale terminal. Contrast with *gross plus*.

gross plus

The accumulation, during a sales period, of all positive amounts (such as merchandise prices, taxes, or fees) entered or calculated in sales transactions at the point of sale terminal. Contrast with *gross minus*.

group

Category of identification defined for file access protection.

hashing

In an indexed data set, using an algorithm to convert the key of a record to an address for that record, for storing and retrieving data. Synonymous with *randomizing*.

header message

A message, printed at the top of a cash receipt or sales check, that may contain the salesperson number, transaction type, transaction number, terminal number, and date.

initial program load (IPL)

The initialization procedure that causes an operating system to begin operation.

INI file

A file that contains initialization parameters for a program.

integrated disk

An integral part of the processor that is used for magnetically storing files, application programs, and diagnostics. Synonymous with *disk*.

interface

Hardware or software that allows two independent systems to communicate with each other.

I/O

Input/output.

IPL

See *initial program load*.

IRI Shopper's Network

Information Resources Inc. Shopper's Network. A network to which stores can subscribe that enables them to collect data about their customer's shopping patterns.

item

(1) One member of a group. (2) In a store, one unit of a commodity, such as one box, one bag, or one can. Usually an item is the smallest unit of a commodity to be sold.

item code

(1) The number assigned to an item that can indicate the manufacturer as well as the specific item in that manufacturer's line of goods. (2) The number assigned to an item for indexing into the user's files.

item lookup key

A key set up to allow item entry with a single key stroke.

K

When referring to storage capacity, a symbol that represents two to the tenth power, or 1024.

keyboard

A group of numeric keys, alphabetic keys, special character keys, or function keys used for entering information into the terminal and into the system.

keyed file

Type of file composed of keyed records. Each keyed record has two parts: a key and data. A key is used to identify and access each record in the file.

label

Constant, either numeric or literal, that references a statement or function.

label format record

A variable length record that explicitly defines the format of a shelf label.

LAN

See *local area network*.

line

On a terminal, one or more characters entered before a return to the first printing or display position.

link

A transmission medium (such as telephone lines), the associated communication devices (such as modems), and the protocols (rules) for sending data through a network.

linetype

A category of message information that is displayed or printed. For example, a "sales prompt" linetype gives instructions to the operator about what sales information to enter.

listing

A printout, usually prepared by a language translator, that lists. Usually, chained lists are used so that the logical order of items can be changed without altering their physical locations.

load

In computer programming, to enter data into storage or working registers.

local area network (LAN)

A communication system for moving information between devices.

logging

The chronological recording of events occurring in a system or a subsystem for accounting or data collection purposes.

lookup key

See *item lookup key*.

loop

A set of instructions that may be executed repeatedly while a certain condition prevails.

magnetic stripe

The magnetic material (similar to recording tape) on merchandise tickets, credit cards, and employee badges. Information is recorded on the stripe for later "reading" by the magnetic stripe reader (MSR) attached to the point of sale terminal.

magnetic stripe reader (MSR)

A device that reads coded information from a magnetic stripe on a card, such as a credit card, as it passes through a slot in the reader.

memory

Program-addressable storage from which instructions and other data can be loaded directly into registers for subsequent execution or processing.

message

(1) An arbitrary amount of information whose beginning and end are defined or implied. (2) A group of characters and control bit sequences transferred as an entity. (3) In telecommunication, a combination of characters and symbols transmitted from one point to another. (4) See *error message, operator message*.

mix and match grouping

See *multiprice grouping*.

modulo check

A function designed to detect most common input errors by performing a calculation on values entered into a system by an operator or scanning device.

MSR

See *magnetic stripe reader*.

multiprice grouping

Multiprice grouping lets a customer purchase items within the same group while mixing the items but matching the price.

national language support (NLS)

The modification or conversion of a United States English product to conform to the requirements of another language or country. This can include the enabling or retrofitting of a product and the translation of nomenclature or documentation of a product.

network
A system of computers that are interconnected so that they can communicate with each other.

node
A point of physical or logical connection to a network.

no sale
A function at the point of sale terminal that enables an operator to open the cash drawer for other than a sales transaction.

NLS
See *National Language Support*.

online
Operation of a functional unit that is under the continual control of a computer or control unit. The term also describes a user's access to a computer using a terminal.

operating system
A collection of programs that is loaded when the processor is turned on and works as a supervisor for the computer.

operator
(1) A symbol that represents the action be performed in a mathematical operation. (2) A person who operates a machine.

operator message
A message from the operating system or a problem program telling the operator to perform a specific function or informing the operator of a specific condition within the system, such as an error condition.

pad character
A character introduced to use up time or space while a function is being accomplished (for example, a carriage return or form eject).

path
Reference that specifies the location of a particular file within the various directories and subdirectories of a hierarchical file system.

personal identification number (PIN)
A numeric identification code assigned to a customer to protect funds and data from unauthorized users.

per update mode
A file mode in which a file is distributed every time a record is written or deleted. Synonymous with *record mode*. Contrast with *at close mode*.

PIN
See *personal identification number*.

pipe
A sequential file in a memory buffer that is used to pass messages from one program to another.

point of sale terminal
A terminal that provides point of sale transaction data collection, credit authorization, price look-up, and other inquiry and data entry functions.

polling characters (address)
A set of characters peculiar to a terminal and the polling operation; response to these characters indicates to the computer whether the terminal has a message to enter.

primary application
A program that controls the normal operating environment of your store (for example, programs that provide sales support).

procedure
A sequenced set of statements that may be used at one or more points in one or more computer programs, and that usually has one or more input parameters and yields one or more output parameters.

Programming Request for Price Quotation (PRPQ)
A customer request for a price quotation on alterations or additions to the functional capabilities of system control programming or licensed programs. The PRPQ may be used in conjunction with computing system RPQs to solve unique data processing problems.

prompt
Character displayed by the operating system to indicate that it is ready to accept input.

PRPQ
See *Programming Request for Price Quotation*.

randomizing
See *hashing*.

read
To acquire or to interpret data from a storage device, from a data medium, or from another source.

reconciliation
A control procedure that identifies and accounts for any difference between the values of a given balance and its associated control total.

record
A collection of related items of data, treated as a unit; for example, in stock control, each invoice could constitute one record. A complete set of such records may form a file.

record key
One or more characters within a file record (or data set record) that is used to identify the record or control its use.

record mode
See *per update mode*.

record number
Position of a specific record in a fixed-length file, relative to record number 1.

report descriptor
An application message that appears on store controller reports.

response
The information the network control program sends to the access method, usually in answer to a request received from the access method. (Some responses, however, result from conditions occurring within the network control program, such as accumulation of error statistics.)

root directory
The main directory on the hard disk. It contains the system files used during the startup of the controller. It also contains all of the subdirectories for the files on the hard disk.

sales descriptor
An application message that displays or prints to help the terminal operator in performing customer checkout.

sales transaction
See *transaction*.

scan
To pass an item over or through the scanner so that the encoded information is read.

scanner
A device that examines the bar code on merchandise tickets, credit cards, and employee badges and generates analog or digital signals corresponding to the bar code.

source
The origin of any data involved in a data transfer.

state
A condition of a terminal for which there is a set of allowed inputs.

station
(1) One of the input or output points of a communications system. (2) A point of sale terminal that consists of a processing unit, a keyboard, and a display. It can also have input/output devices, such as a printer, a magnetic stripe reader and cash drawers.

store controller
A programmable unit in a network running IBM 4690 Operating System used to collect data, to direct inquiries, and to control communication within a system.

stub
A blank module, or section of code, that contains only an entry point and an exit point. Stub modules are useful in structured programming, as they hold a place in the

overall code for a module that will be written or added later.

subdirectory

Any level of file directory lower than the root (or main) directory within a hierarchical file system.

summary journal

A record of the terminal operational activity that is printed at the terminal.

system

See *data processing system*.

system display

The terminal display that shows information about sales transactions and system messages and errors. Compare with *customer display*.

task

A basic unit of work.

tender

Money, checks, or coupons used as payment for merchandise or service.

terminal

A device, usually equipped with a keyboard and a display, capable of sending and receiving information over a communication channel.

terminal number

A number assigned to a terminal to identify it for addressing purposes.

terminal offline mode

The mode of a terminal after an attempt to write a transaction to the Transaction Summary Log on the controller failed. The terminal is storing transactions locally on either the hard drive or the RAM disk for despooling later to the controller.

till

A tray in the cash drawer of the point of sale terminal, used to keep the different denominations of bills and coins separated and easily accessible.

TOF mode

See *terminal offline mode*.

TOF-recovered transaction

A transaction that was stored locally by the terminal and later despooled to the Transaction Summary Log on the controller.

transaction

The process of recording item sales, processing refunds, recording coupons, handling voids, verifying checks before tendering, and arriving at the amount to be paid by or to a customer. The receiving of payment for merchandise or service is also included in a transaction.

transaction log

A record of transactions performed at the point of sale terminal. This log is magnetically recorded and stored on the disk or diskette.

unit-pricing

Used for pricing individual items. Multiplying the item price by the quantity of items gives the extended price.

universal product code (UPC)

An encoded number that can be assigned to and printed on or attached to an article of merchandise for scanning.

UPC

See *universal product code*.

user

Category of identification defined for file access protection.

user exit

A point in an IBM-supplied program at which a user-written program is given control.

user exit routine

A routine written by a user to take control of a program supplied by IBM at a user exit.

variable

A named entity that is used to refer to data and to which values can be assigned. Its attributes remain constant, but it can refer to different values at different times.

variety

A subdivision of tender. Each tender type can have up to nine varieties associated with it.

velocity code

A user-defined item code.

void item

A function at the point of sale terminal that enables an operator to delete a single line entry previously entered during the current transaction. The line entry is electronically deleted from the terminal but remains on the printout at the terminal. See also *void transaction*.

void transaction

A function at the point of sale terminal that enables the operator to delete the entire transaction in progress. See also *void item*.

work file

A file that is both created and deleted in the same job.

Index

TC62-0046-02