USER GUIDE

NCR SelfServ™ Cash Tender POS Integration SDK (MS Windows)

Release 1.2

B005-0000-2151
Issue E

The product described in this book is a licensed product of NCR Corporation.

NCR is a registered trademark of NCR Corporation. NCR SelfServ is a trademark of NCR Corporation in the United States and/or other countries. Other product names mentioned in this publication may be trademarks or registered trademarks of their respective companies and are hereby acknowledged.

Where creation of derivative works, modifications or copies of this NCR copyrighted documentation is permitted under the terms and conditions of an agreement you have with NCR, NCR's copyright notice must be included.

It is the policy of NCR Corporation (NCR) to improve products as new technology, components, software, and firmware become available. NCR, therefore, reserves the right to change specifications without prior notice.

All features, functions, and operations described herein may not be marketed by NCR in all parts of the world. In some instances, photographs are of equipment prototypes. Therefore, before using this document, consult with your NCR representative or NCR office for information that is applicable and current.

To maintain the quality of our publications, we need your comments on the accuracy, clarity, organization, and value of this book. Please use the link below to send your comments.

EMail: FD230036@ncr.com

Preface

Audience

This book is written for software developers, NCR Customer IT (CS), NCR Support, NCR Customer Engineers, NCR Professional Services Engineers, and third–party partners.

References

- *NCR SelfServ™ Cash Tender Module (7352) Site Preparation and Hardware Installation Guide* (B005-0000-2147)
- *NCR SelfServ™ Cash Tender Module (7352) Hardware Service Guide* (B005-0000-2148)
- *NCR SelfServ™ Cash Tender Module (7352) Hardware User Guide* (B005–0000–2149)
- *NCR SelfServ™ Cash Tender Module (7352) Parts Identification Manual* (B005–0000–2150)
- *NCR Automatic Device Detection (ADD) Release 3.X Installation and Configuration Guide* (B005–0000–2330)
- *NCR SelfServ™ Checkout Release 1.0 Profile Manager Lite User Guide for ADD 3.X* (B005–0000–2279)
- Complete list of return codes: *http://msdn.microsoft.com/en-us/library/windows/desktop/aa368542%28v=vs.85%29.aspx*

Table of Contents

Chapter 4: Cash Tender Module Client Java API

Revision Record

Issue	Date	Remarks
A	Dec 2012	First Issue
B	Mar 2014	Release 1.1 Added the Windows 7 and Windows POSReady 7 operating systems in the list of supported operating systems. Added information on the Ending a Cash Management Transaction function. Removed any information on the Resetting Dispensable Note Counts API for C and Java because the bills in the BNR cannot be emptied. Removed any information on the Setting the Dispensable Counts API for C and Java because the function is already available through the Setting the Dispensable Note Counts and Resetting the Dispensable Coin Counts APIs. Added instructions for uninstalling the Cash Tender POS Integration SDK.
C	July 2014	Release 1.1 Added information on the non–device dispense error. Added diagrams on customer transactions with extra deposit events. Added back the information on setting dispensable counts for both the C and Java APIs. Added new appendix on return codes list.
C	Sept 2014	Release 1.1 Added minor changes for the maintenance release.
D	Apr 2015	Release 1.1 Added minor changes for the maintenance release.
E	May 2015	Release 1.2 Added a new section on the CTM-SSCO interoperability feature.

Chapter 1: Cash Tender Module

Overview

The NCR SelfServ™ Cash Tender Module (7352) is a new hardware product based on the NCR SelfServ™ Checkout 7350 core body but without the monitor, receipt printer, scanner, and E–Box.

The Cash Tender Module (CTM) is designed for installation in the store's Point–of–Sales (POS) assisted lanes. The store's POS system communicates with the CTM hardware to perform Cash Tender functionalities. Cashiers use the CTM to process cash payments. Store customers can also use the CTM to process cash payments and receive change without the assistance of any cashier. A head cashier or any cash–handling personnel can perform advanced cash management operations on the Cash Tender unit depending on the POS integration. In all cases, the user requesting the operation and the details of the operation are logged for subsequent analysis if needed.

In a customer transaction, notes and coins are inserted into the recycler devices, and change is distributed from the recycler units. The CTM supports two devices, the Bulk Coin Recycler (BCR) and the Bulk Note Recycler (BNR).

The CTM does not provide graphics user interface (GUI) operations but it must communicate specific messages on the maintenance of the devices in the context of POS operations. Thus, the CTM provides the POS with both robust error notifications and services to provide the POS with localized content describing the error conditions and corrective actions.

Integrating the Cash Tender Module to the store's existing POS system is possible through the Cash Tender POS Integration Software Developer's Kit (SDK).

Benefits of Cash Tender Module

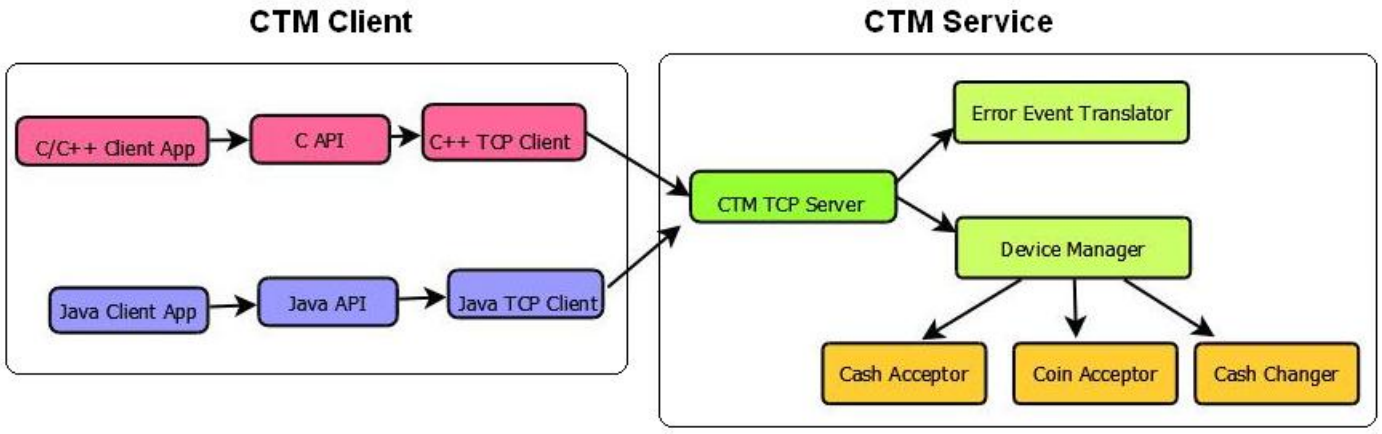
Cash Tender Module (CTM) enables retailers to automate cash handling in relation to their existing POS systems. The solution provides shoppers the ability to interact directly with the POS system in making payments and receiving change, therefore removing the need for cashiers to handle cash and eliminating the risk of providing an incorrect change. The following are the additional benefits of the CTM solution:

- Provides greater security for the cash–handling personnel.
- Provides greater cash accuracy at the POS assisted lane.
- Improves counterfeit detection.
- Removes possibility for cash shrinking caused by employee theft at the POS assisted lane.
- Provides faster cash tender process through automation.
- Provides clear cash accountability.
- Improves operational efficiency.

Components of the Cash Tender Module System

The Cash Tender Module (CTM) solution includes two main components:

- A service that directly controls the cash devices.
- A client API that communicates to that service through a TCP connection.



Cash Tender Module Service

The CTM service is a Java application that manages the CTM devices, such as the Bulk Coin Recycler (BCR) and the Bank Note Recycler (BNR), and processes the commands that are sent by the CTM client.

The CTM service is responsible for many of the low–level details that are typically related to interacting with cash devices. The CTM client, on the other hand, provides cash accepting, cash dispensing, and cash management functions.

Cash Tender Module Client

There is no user interface for the CTM system. The POS application uses the CTM client to connect to the CTM service and perform Cash Tender operations. At present, client applications, such as the store's existing POS application, can use either Java or C to interact with the CTM system.

Cash Tender Module Devices

The NCR SelfServ™ Cash Tender Module (7352) hardware includes two recycler devices: the Bulk Coin Recycler (BCR) and the Bulk Note Recycler (BNR). These devices provide the functionality for automated cash tender at assisted lanes and are equipped with currency recycling features, which enable retailers to reduce cash handling and replenishment requirements because currency accepted from the shoppers is used to provide change in future transactions.

Bulk Coin Recycler (BCR)

The Bulk Coin Recycler (BCR) accepts coins from shoppers as tender and then recycles these coins to provide change to future shoppers. The BCR recycles up to six different coin denominations. Specific denominations are held in the coin bins for use as change.

For more information about the BCR device, refer to the *NCR SelfServ™ Cash Tender Module (7352) Hardware User Guide* (B005–0000–2149).

Bulk Note Recycler (BNR)

The Bank Note Recycler (BNR) accepts notes (bills) from shoppers as tender and then recycles these notes to provide change to future shoppers. The BNR combines the functions of a note acceptor and note dispenser.

The BNR recycles up to four different note denominations. Specific denominations are held in the Recycler cassettes for use as change.

For more information about the BNR device, refer to the *NCR SelfServ™ Cash Tender Module (7352) Hardware User Guide* (B005–0000–2149).

Cash Tender POS Integration SDK

The Cash Tender Module (CTM) is integrated into a POS system through the Cash Tender POS Integration Software Developer Kit (SDK). The Cash Tender SDK includes programming information intended for software developers and others who want to install the CTM into a store's existing POS system. It is packaged with high–level APIs in C and Java for successful POS integration.

The Cash Tender SDK supports the following functionalities:

- Accept and dispense operations
- Cash management operations
- Advanced Cash Office (ACO) interoperation
- Error–handling operations
- Platform configuration management

Accept and Dispense Operations

The Cash Tender SDK provides functions that POS systems can call to perform the following:

- Accepting notes and coins.
- Dispensing notes and coins by total value.
- Reporting the results of accept and dispense operations.

Cash Management Operations

The CTM provides cash management functionalities that include balance inquiries, loans, and pickups. The Cash Tender SDK provides functions for the following cash management operations:

- Setting the number of notes in the loader cassette.
- Dispensing cash by denomination.
- Transferring notes from BNR loader cassette to cashbox.
- Transferring notes from recyclers to cashbox.
- Refilling notes and coins.
- Resetting dispensable and non–dispensable coin counts.
- Resetting non–dispensable note counts.
- Setting the dispensable counts.
- Querying the dispensable and non–dispensable capacities of coins and notes.

Advanced Cash Office (ACO) Interoperation

The CTM is connected to a listening back–office service, the Advanced Cash Office (ACO), which receives and keeps information of all customer transactions and cash management operations that occur in the Cash Tender device. These transactions include loans, pickups, and accept and dispense operations. The Cash Tender device transmits the transaction ID and the cash balance data of each occurrence of these operations to the ACO.

Error–handling Operations

The Cash Tender SDK provides functions that POS systems use to handle unsolicited errors. These errors include device errors and dispense errors.

When an error occurs in the Cash Tender device and the POS system learns about it, the POS system sends the error information to the CTM. The CTM then sends back textual instructions to the POS system, which displays the instructions for the cashier or cash–handling personnel to read and perform.

The cashier or cash–handling personnel follows the instructions to resolve the error and performs a step that indicates the resolution steps are complete. The POS system then performs a system health check to make sure that the error is resolved and that the system is ready to get back in service.

Platform Configuration

The Cash Tender SDK provides the functionality to configure the language and currency settings of the CTM system during the platform software installation. During platform software installation, the CADDConfigure user interface displays the available language and currency, in addition to the option to enable accept and dispense operations for both coins and notes.

Chapter 2: Installing the Cash Tender POS Integration SDK

Overview

This chapter discusses the different system requirements of the Cash Tender Module (CTM) system. It also discusses the prerequisites and installations steps of the Cash Tender POS Integration SDK.

Cash Tender Module Requirements

This section discusses the different system, software, and hardware requirements of the Cash Tender Module (CTM) system.

System Requirements

The CTM system is expected to use 80 Megabytes (MB) of memory while running. It requires two available USB ports in the POS hardware.

Software Requirements

This section describes the software requirements of the CTM system.

Operating System

The CTM system is compatible with the following Windows operating systems:

- Windows® XP Embedded (XPe) SP3
- Windows® Embedded POSReady 2009
- Windows® Embedded POSReady 7
- Windows® 7

Platform Software

The CTM system requires the installation of the NCR Retail and CTM Platform components, which are all included in the installation package. The NCR Retail Platform components include all the OPOS controls and the Retail Platform Software for Windows (RPSW). After installing the platform software, installing the Automatic Device Detection (ADD) software with its relevant hotfixes and the NCR SS Retail API (SSRAPI) must follow. For more information, refer to *Installing the Retail Platform and ADD Software* on page 11.

Third–party Application

Installing the Java™ Runtime Environment (JRE) version 1.6 or later is required before installing the Cash Tender POS Integration SDK. For more information, refer to *Installing Java™ Runtime Environment (JRE)* on page 2.

Scot Directory

Another prerequisite of installing the Cash Tender POS Integration SDK is the `C:\scot` directory. Ensure that the `scot` directory exists in the drive C. This directory contains the logs created from installing and uninstalling the Cash Tender POS Integration SDK.

Hardware Requirements

The NCR SelfServ™ Cash Tender Module (7352) hardware is required for the CTM system. For more information about the CTM hardware, refer to the *NCR SelfServ™ Cash Tender Module (7352) Hardware User Guide* (B005–0000–2149).

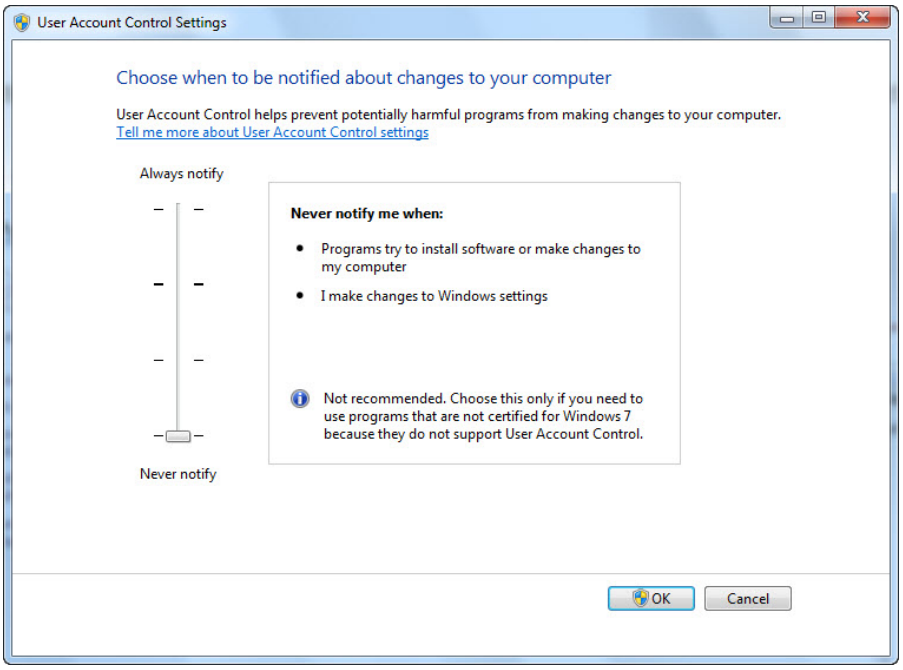
Prerequisites

Before installing the Cash Tender POS Integration SDK, ensure that the platform and ADD software and the third–party applications are already installed.

Setting the User Account Control

To ensure that the installation completes successfully, set the User Account Control (UAC) settings to the lowest level. To set the UAC, follow these steps:

1. Select **Start → Control Panel → User Accounts → User Accounts→ Change User Account Control Settings**. The system displays the following window.



2. Set the bar to the lowest level at **Never notify**.
3. Select **OK** to save the changes and to close the window.

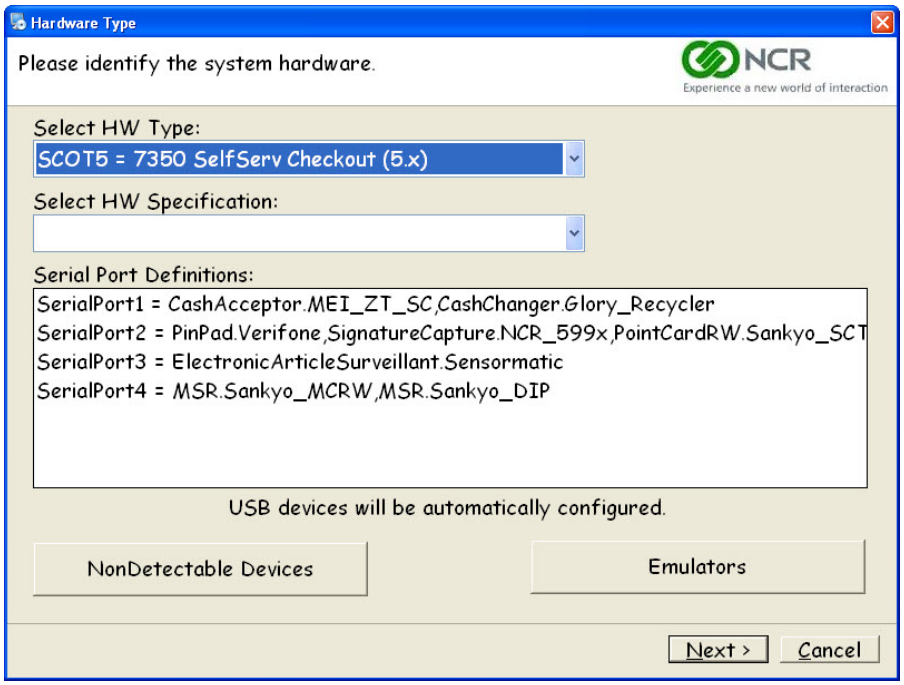
Installing the Retail Platform and ADD Software

To install the platform software and the Automatic Device Detection (ADD) software, follow these steps:

1. Browse for the installation file of RPSW 4.0 from the installation CD and select **`HF1309_NCRSSCO_RPSW_4031_Installer_v1.3`**, or a later version if available, to install the platform software.
2. Reboot the system after installing the platform software.
3. Browse for the installation file of RPSW 4.0 from the installation CD and select **`HF1405_NCRSSC_RPSW_4034_Patch_v1.4`**.
4. Reboot the system after installing the platform patch software.
5. Browse for the installation file of the ADD software from the installation CD and select **`ADDPackage_V3.3.0.194`** to extract the package to the `C:\temp\master\` directory.
6. Open a Command Prompt window and then go to the `C:\temp\master\ADDPackage` directory.
7. Run the following line:

   `InstallADDPackage.exe SKIPOS SKIPPCI`

   Wait for the system to display the Hardware Type window of the CADDConfigure user interface.

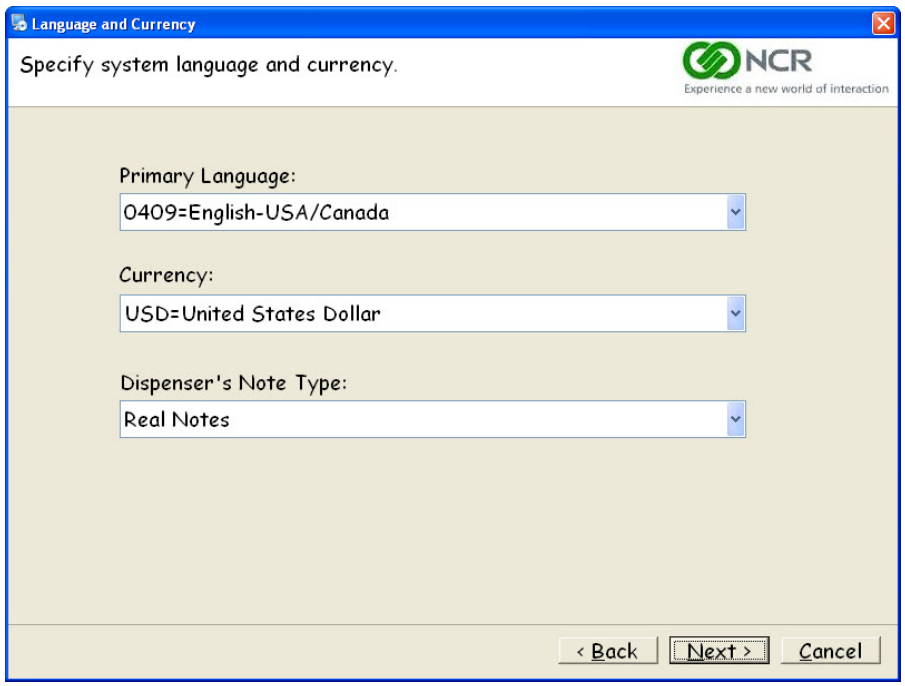


8. Select **SCOT5 = 7350 SelfServ Checkout (5.x)** for the HW type.
9. Select **CTM = SelfServ Cash Tender Module** for the HW Specification.

10. If you are working with emulators, select **Emulators**, and then select the **Note Acceptor**, **Note Dispenser**, **Coin Acceptor**, and **Coin Dispenser** devices.

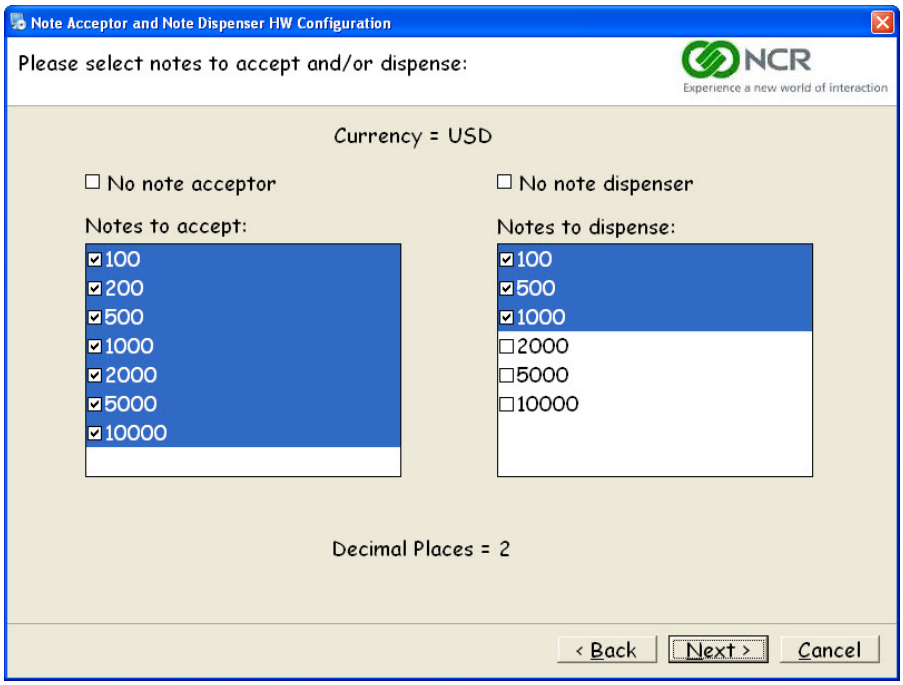
    For more information, refer to *Working with Emulators* on page 22.

11. Select **Next**. The system displays the Language and Currency window.

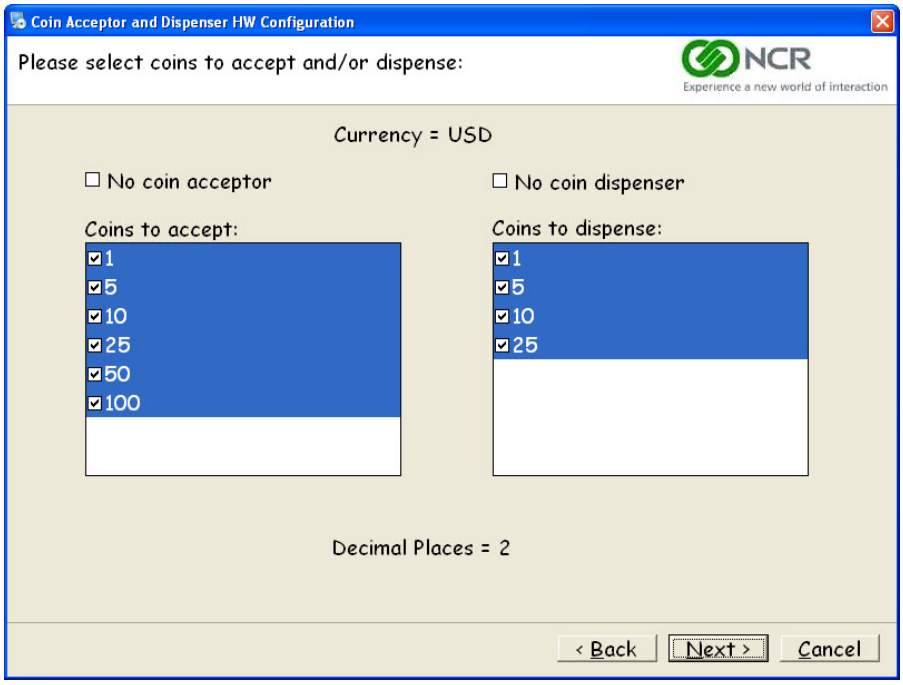


12. Select your preferred **Primary Language**, **Currency**, and **Dispenser's Note Type**.

13. Select **Next**. The system displays the Note Acceptor and Note Dispenser HW Configuration window.

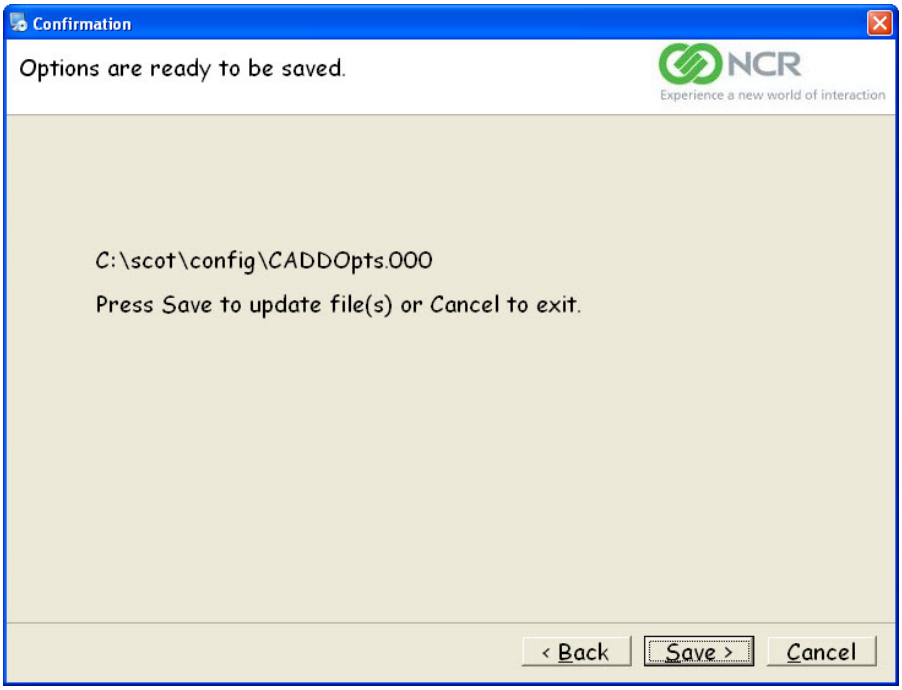


14. Select whether to accept and dispense notes, and the notes to accept and dispense.
15. Select **Next**. The system displays the Coin Acceptor and Dispenser HW Configuration window.



16. Select whether to accept and dispense coins, and the coins to accept and dispense.

17. Select **Next**. The system displays a Confirmation window.



18. Select **Save** to save the settings and wait for the window to close.
19. Reboot the system.

Note: After installing the ADD software, you may need to install the latest mandatory ADD hotfixes for specific hardware devices. Contact your NCR PS representative or customer manager for the current list of the required platform hotfixes. The hotfixes may include the following for the NCR SelfServ™ Checkout 5.x hardware for all supported operating systems:

- HF1405_Global_Note_v16.31
- HF1405_Global_Note_16_Firmware_Update_v6.0
- HF1406_Global_Coin_v14.17
- HF1405_Global_Coin_14_Firmware_Update_v1.9

Note: All hotfixes must be installed in the correct chronological order.

20. On the Command Prompt window, run the following line:

    `C:\SCOT\bin\ADD.bat`

    The platform software then detects your connected devices or emulators and configures them according to the language and currency settings you specified using the CADDConfigure user interface.

21. Reboot the system after installing the ADD software and required hotfixes.

Note: The system may automatically reboot after installing the ADD software and required hotfixes. For more information, refer to *Verifying the ADD Installation* below.

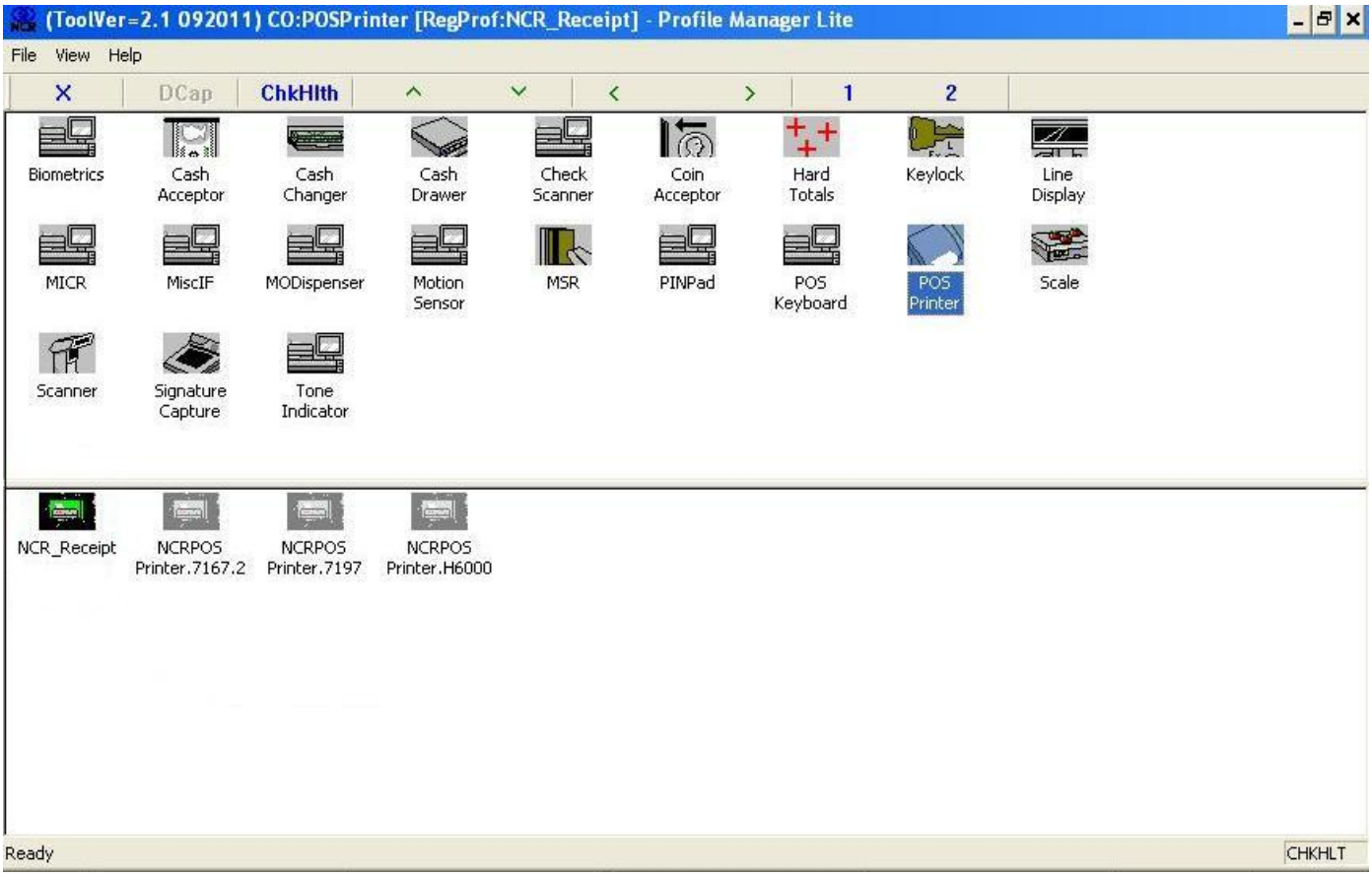
Verifying the ADD Installation

Before installing the Cash Tender POS Integration SDK, you must verify the installation of the ADD software and check whether the cash devices are working properly through the Profile Manager Lite (PML) application. To verify the ADD installation, follow these steps:

1. Select **Start→ Run** to display the Run window.
2. Run the following line:

   ```
   profilemanagerlite.exe /nodongle /nocalibration
   ```

   The PML window appears.



3. Select the appropriate BCR and BNR devices to test.

Installing the Cash Tender POS Integration SDK

To install the Cash Tender POS Integration SDK, follow these steps:

1. Insert the Cash Tender POS Integration SDK installation CD into the CD–ROM drive of the POS hardware.
2. Copy the following installation packages from the Cash Tender POS Integration SDK installation CD to any directory in the POS system:
   - CTMCClient (optional; used for C API only)
   - CTMJavaClient (optional; used for Java API only)
   - CTMDevelopmentKit
   - CTMMedia
   - CTMService
3. Double–click the **CTMCClient** folder and select **`Install.bat`**. The system installs the CTM C Client.

Note: Perform this step only if using the C API.

4. Double–click the **CTMJavaClient** folder and select **`Install.bat`**. The system installs the CTM Java Client.

Note: Perform this step only if using the Java API.

5. Double–click the **CTMDevelopmentKit** folder and select **`Install.bat`**. The system installs the CTM Development Kit.
6. Double–click the **CTMMedia** folder and select **`Install.bat`**. The system installs the CTM Media.
7. Double–click the **CTMService** folder and select **`Install.bat`**. The system installs the CTM Service.

After these packages are installed, the modules and binaries are installed at `C:\Program Files\NCR\CashTenderModule\`.

The following install logs are created at `C:\scot\`install:

- `CTMC-ClientInstall.log` (if using the C API)
- `CTMJavaClientInstall.log` (if using the Java API)
- `CTMDevelopmentKitInstall.log`
- `CTMMediaInstall.log`
- `CTMServiceInstallLog.log`

To verify the installation of the CTM package, refer to *Verifying the CTM Package Installation* on the facing page. To verify the installation of the Java Client, refer to *Verifying the Java Client Installation* on page 19. To verify the installation of the C Client, refer to *Verifying the C Client Installation* on page 21.

Verifying the CTM Package Installation

There are two steps to verify whether the CTM package is installed successfully:

1. Check the install logs.
2. Check the connection to the CTM devices.

Checking the Install Logs

The install logs indicate whether the CTM package installation is successful. To check the install logs, follow these steps:

1. Go to `C:\scot\` install and open the following CTM install logs:
   - `CTMC-ClientInstall.log` (if using the C API)
   - `CTMJavaClientInstall.log` (is using the Java API)
   - `CTMDevelopmentKitInstall.log`
   - `CTMMediaInstall.log`
   - `CTMServiceInstallLog.log`
2. If the CTM package installation is successful, the following line appears at the end of the files:

   `MainEngineThread is returning 0`

3. If the CTM package installation is not successful, the **MainEngineThread** line would indicate a non–zero return.

   Example: `MainEngineThread is returning 1798`

Note: For more information, refer to *Installer and Uninstaller Return Codes List* on page 9.

Checking the Device Connections

To verify whether the CTM service can successfully communicate with the Bulk Note Recycler (BNR) and Bulk Coin Recycler (BCR) devices, follow these steps:

1. Go to **`C:\Program Files\NCR\CashTenderModule\Service\`**.
2. Run the **`CTMService.bat`** to connect to the BNR and BCR devices.
3. Wait until you see the message *"Bootstrapping CTM Device manager complete!"* in the output of the service.
4. Check the output of the other devices:
   - *"CashAcceptorServiceEx claim status 0"*
   - *"CashChangerServiceEx claim status 0"*
   - *"CoinAcceptorServiceEx claim status 0"*

If you see these messages, it means that the CTM service successfully communicated with both the BNR and BCR devices or with their associated emulators. The CTM service begins accepting TCP connections from the client at this point.

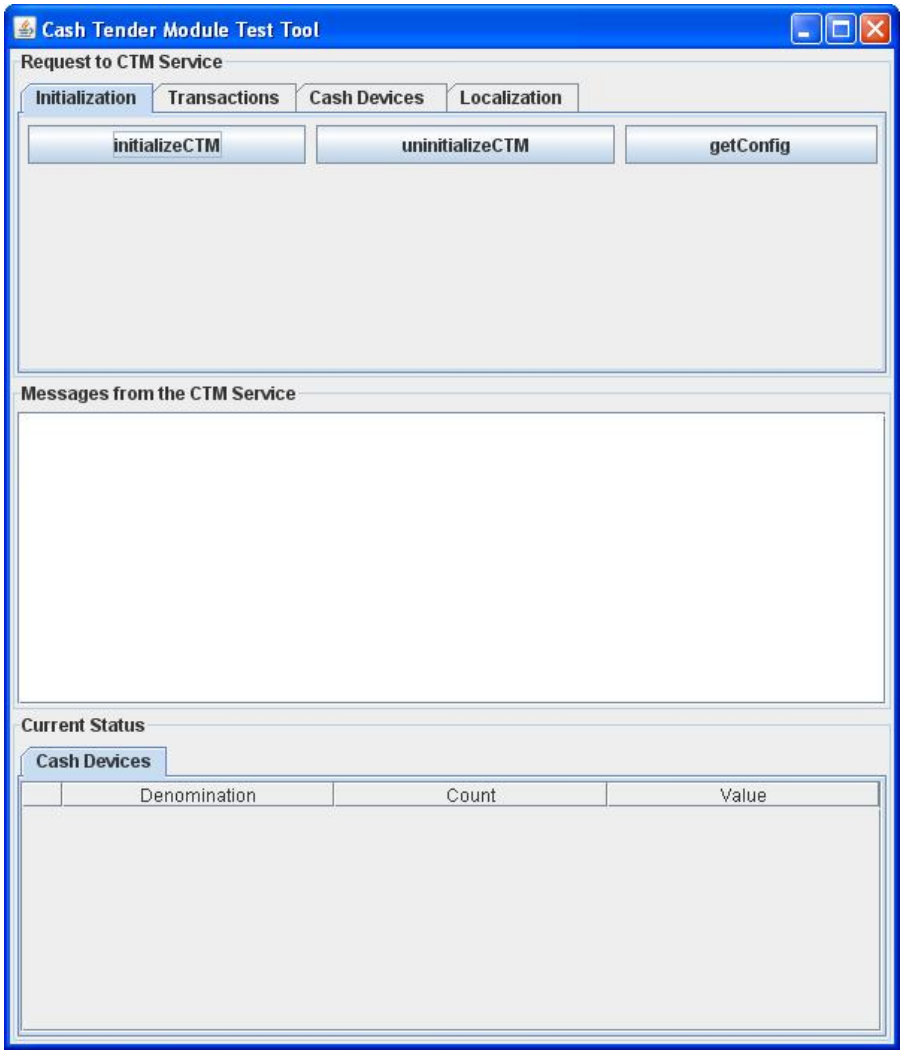
Note: After verifying the CTM service installation, you must verify the Java or C client installation. For more information, refer to the next sections.

Verifying the Java Client Installation

Note: Before verifying the Java client installation, verify the CTM package installation first. For more information, refer to *Verifying the CTM Package Installation* on page 17.

The Cash Tender POS Integration SDK provides a Java test program, which is a graphical user interface (GUI) application that interacts with the Java client APIs. Use this Java test program to verify the Java API installation. To run the test program, follow these steps:

1. Go to **`C:\Program Files\NCR\CashTenderModule\Development-Kit\Examples\Java\`**.
2. Run the **`CTMTestProgramme.bat`** batch file. The Java test program GUI appears.



3. Select the **initializeCTM** button and then select Enter to connect to the CTM service. If you see the ***[initializeCTM] return value: true*** message in the *Messages from the CTM Service* section of the GUI, it means that the Java client API successfully communicated with the CTM service.

You can view the logs of the CTM service and the CTM test program in the following log files:

- `C:\scot\logs\ctm.log`
- `C:\scot\logs\ctmSampleApp.log`

For more information, refer to *Cash Tender Module Client Java API* on page 73.

Verifying the C Client Installation

Note: Before verifying the C client installation, verify the CTM package installation first. For more information, refer to *Verifying the CTM Package Installation* on page 17.

To verify whether the C client is installed successfully, follow these steps:

1. Copy all the DLL files from `C:\Program Files\NCR\CashTenderModule\C-Client\bin` to this new location:
   `C:\Program Files\NCR\CashTenderModule\Development-Kit\Examples\C\bin`
2. From this new location, select a `ctm_client_example_##` application to run.
3. For information on how the selected application works, refer to the corresponding file with the same file name in the `C:\Program Files\NCR\CashTenderModule\Development-Kit\Examples\C\src\` directory.

You can view the logs of the CTM service and the CTM test application in the following log files:

- `C:\scot\logs\ctm.log`
- `C:\scot\logs\ctm_c_api`

For more information, refer to *Cash Tender Module Client C API* on page 107.

Working with Emulators

Response (RSP) files are required in systems that work with emulators. For systems working with emulators, copy the following response files from the `CTM_x.x.x.x\SDK\RSP\` directory of the release package to the `C:\` drive.

- `CashAcceptor.rsp`
- `CoinAcceptor.rsp`
- `CashChanger.rsp`

Note: Do not copy RSP files to systems that work with real devices.

Using the RSP files

For example, if you are sending a DirectIO command "1004" to the cash changer, the `CashChanger.rsp` file should contain the following response:

```
[1004] (1004 is the DirectIoCommand)
Response = 0;1;"123456789" (response expected from the command)
```

CTM-SSCO Interoperability

It is possible to use the CTM software on SelfServ™ Checkout (SSCO) hardware that is configured with the BNR and the BCR devices. In this configuration, it is possible to switch back and forth between the SelfServ Checkout software and a separate Point-of-Sale (POS) solution, if the POS solution is integrated with the CTM software.

Note: This feature has only been certified on ADK 5.0 software. If you require certification of this feature on another version of the ADK software, see your customer manager or other NCR representatives.

Requirements of CTM-SSCO Interoperability

The CTM-SSCO interoperability feature requires the installation of hotfixes on some ADK versions. The following table displays these ADK versions and their corresponding hotfixes.

ADK version	Required hotfix
ADK G2	HF10015_Retrofit2.1.1_Update1_v12
ADK 4.5.1	HF12057_4.5.1.28_Xmode_Reporting_v3

Note: ADK version 5.0 and versions after 4.5.1 do not require any hotfix installation.

After installing the ADK and the required hotfix (if any), open the `scotopts.000` file and then add the following line under the "[Locale]" section:

"UseFileCashCounts = Y"

When this option is set to "Y" (enabled), cash counts are read from and written to two new files:

- `c:\scot\data\AcceptorCount.dat`
- `c:\scot\data\DispenserCount.dat`

Registry for cash counts are removed. By default, this option is set to "N" (disabled) so cash counts are read from and written to registry.

Note: When switching currencies, delete the `AcceptorCount.dat` and `DispenserCount.dat` cash counts files before starting either SSCO or CTM. Although SSCO supports non-recycler devices, the CTM-SSCO interoperability feature should be used only for recycler devices that CTM supports.

Uninstalling the Cash Tender POS Integration SDK

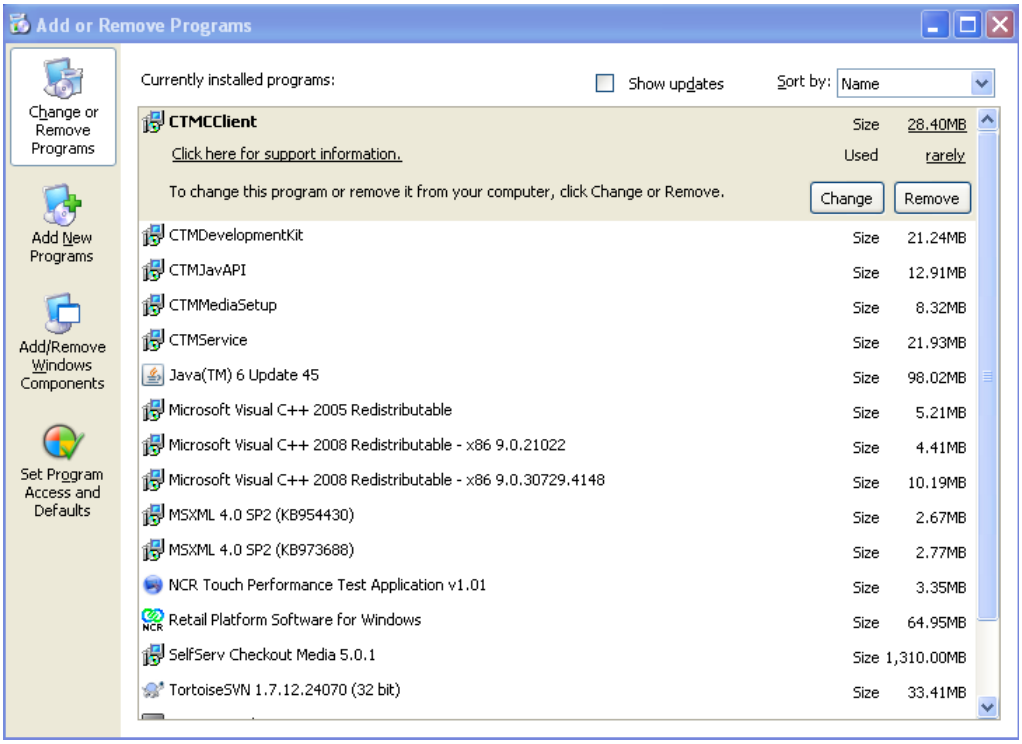
There are two ways to uninstall the Cash Tender POS Integration SDK from the system:

- through the Control Panel
- through the `Uninstall.bat` files

Uninstalling the SDK through the Control Panel

To uninstall the Cash Tender POS Integration SDK using the Control Panel, follow these steps:

1. Select **Start → Control Panel → Add or Remove Programs**. The system displays the list of programs currently installed in the system, including the CTM packages.



2. Select each of the following CTM packages, and then select **Remove** to uninstall the package:
   - CTMCClient (if installed)
   - CTMJavaAPI (if installed)
   - CTMDevelopmentKit
   - CTMMediaSetup
   - CTMService

Uninstalling the SDK through the Uninstall Batch Files

The CTM installer package contains Uninstall batch files that can be used to uninstall the Cash Tender POS Integration SDK. To uninstall the Cash Tender POS Integration SDK using the Uninstall batch files, follow these steps:

1. Go to each of the following directories in the CTM installation CD:
   - CTMCClient
   - CTMDevelopmentKit
   - CTMJavaClient
   - CTMMedia
   - CTMService
2. Double–click the `Uninstall.bat` file found in each of these directories to uninstall the corresponding package.

To verify the uninstallation of the CTM packages, refer to *Verifying the Uninstallation of CTM Packages* on the next page.

Verifying the Uninstallation of CTM Packages

To verify whether the CTM packages are successfully uninstalled from the system, follow these steps:

1. Go to `C:\scot\`install and ensure that the following log files are created after uninstalling the CTM packages.
   - `CTMC-ClientUNInstall.log`
   - `CTMDevelopmentKitUNInstall.log`
   - `CTMJavaClientUnInstall.log`
   - `CTMMediaUNInstall.log`
   - `CTMServiceUNInstall.log`
2. Check each of these uninstall log files and verify that the following line appears at the end of the file:

   ```
   MainEngineThread is returning 0
   ```

Note: This verification process applies only when uninstalling the Cash Tender POS Integration SDK using the Uninstall batch files. Uninstall log files are not created when uninstalling the SDK using the Control Panel.

Note: For more information, refer to *Installer and Uninstaller Return Codes List* on page 9.

Chapter 3: Understanding the Cash Tender Module Operations

Overview

The Cash Tender Module (CTM) is designed to work together with the store's existing POS application. The POS application communicates with the CTM hardware to perform Cash Tender operations. Cashiers use the CTM to process cash payments. Store customers can also use the CTM to process cash payments and receive change without the assistance of any cashier.

The following are the different Cash Tender operations that customers, cashiers, and any cash–handling personnel can perform with the CTM system:

- Registering and unregistering callbacks
- Initializing and uninitializing the CTM client
- Querying configuration information
- Reading dispensable and non-dispensable counts
- Handling customer transaction operations
- Handling cash management transactions
- Handling error conditions
- Connecting to an Advanced Cash Office (ACO) feed

Registering and Unregistering Callbacks

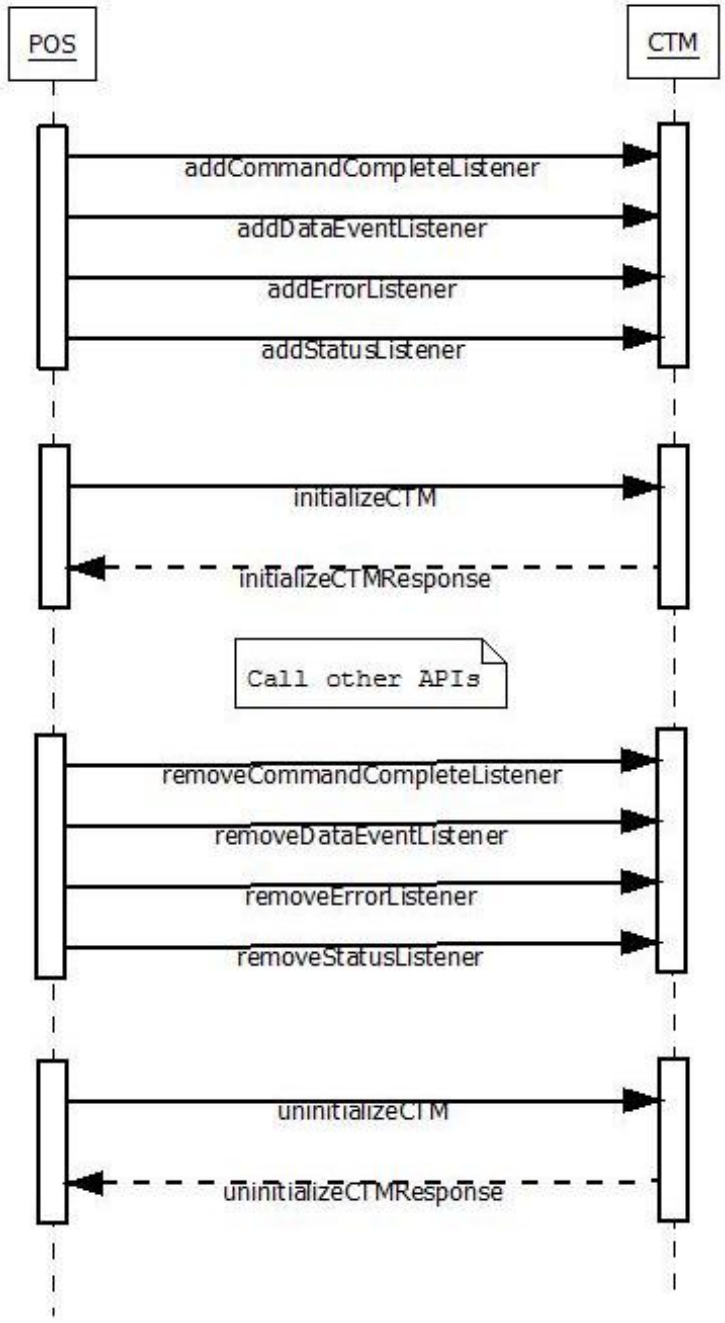
Before a POS application can connect to the CTM service, it must first register callbacks or listeners with the CTM client to receive feedback from the CTM service. The following are the four types of callbacks:

- Data—refers to the callback that signals when a data event has occurred, such as when money is deposited.
- Command results—refers to the callback that signals when a command has completed.
- Error—refers to the callback that signals when an error has occurred.
- Status—refers to the callback that indicates informational events about the CTM devices, such as when cash levels are low.

When the POS application is shutting down or it no longer needs to receive events from the CTM service, the POS application unregisters the registered callbacks.

The POS application calls the C or Java API to register and unregister callbacks in the CTM client.

Refer to the following diagram for more information.



Initializing and Uninitializing the CTM Client

After registering the necessary callbacks in the CTM client, the POS application must initialize the CTM client to connect to the CTM service. Initializing and uninitializing the CTM client follows this process:

1. The POS application initializes the CTM client by calling the C or Java initialization function.
2. The function returns a value that indicates whether the initialization was successful or that it encountered an error.
3. If the initialization process is successful, the POS application obtains configuration information from the CTM service by calling a query function.
4. The CTM service sends back a response containing static configuration values for accepted denominations, dispensed denominations, currency code, language code, and other configuration information.

Note: Before you initialize the CTM client, ensure that you have created the necessary callbacks. Also, make sure you start the CTM service first.

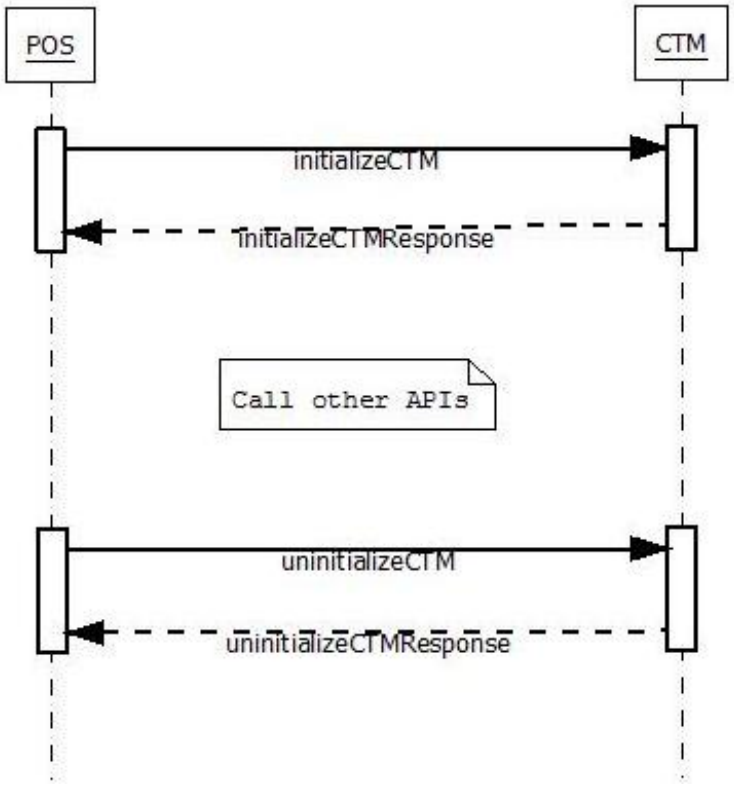
5. If the connection between the CTM client and CTM service is unnecessary, the POS application can uninitialize the CTM client to stop its connection to the CTM service.

Note: Before you uninitialize the CTM client, ensure that you have unregistered the existing callbacks first.

6. When the CTM client fails to connect to the CTM service, it sends a response back to the POS about the connection failure.
7. If the response indicates that the CTM system is offline, the POS application displays an offline message and the cashier follows the instructions that the POS vendor provides for resolving the error.

Note: After a successful initialization, it is recommended to run a diagnostic test using **selfTestAll** to ensure that the cash devices are healthy and working properly. For more information, refer to *selfTestAll* on page 80.

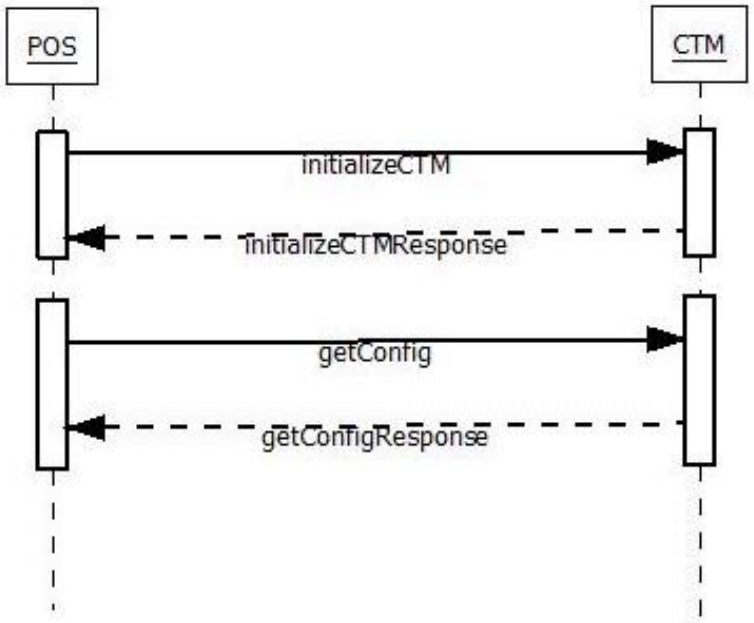
Refer to the following diagram for more information.



Querying Configuration Information

The POS application can request for CTM configuration information from the CTM service through a C or Java query function. If the request is successful, the CTM service sends back the configuration information, such as the names of the all the configured devices, the CTM client software version, the CTM service software version, and so forth.

Refer to the following diagram for more information.



Reading Dispensable and Non–dispensable Counts

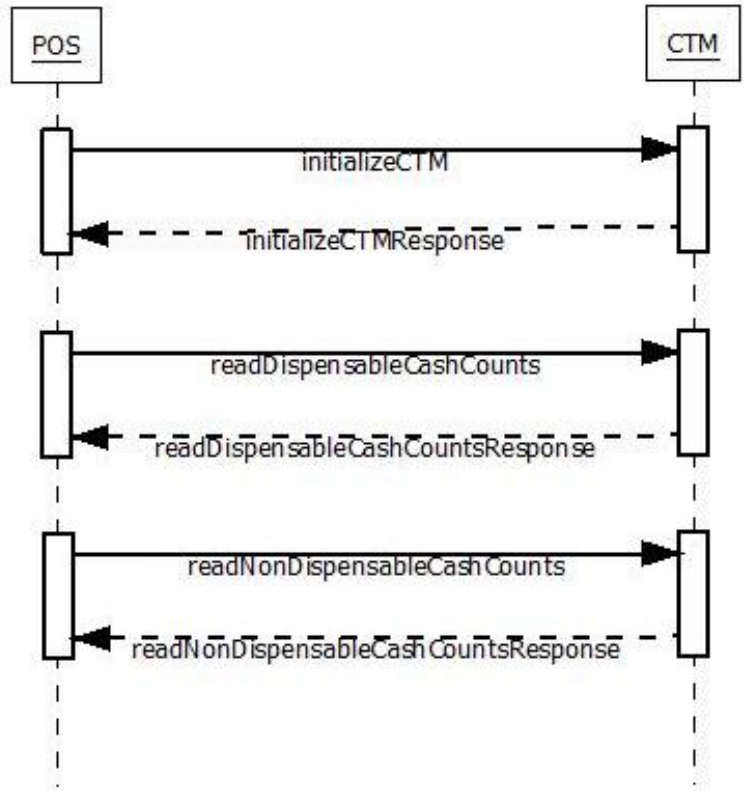
Before a customer transaction starts, cashiers must determine the quantity of each note and coin denomination that the CTM can and cannot dispense as change to shoppers. This process is necessary to perform the following:

- Determine whether the dispensable count level is low, which may require refilling the cash changer device.
- Determine whether the coin overflow or cash box containers are almost full.
- Update the display of a component in the POS application with the count information.

Reading dispensable and non–dispensable counts follows this process:

1. When the POS application sends a function to read dispensable counts to the CTM service, the CTM service queries the cash changer interface for the dispensable counts.
2. The CTM service sends the response back to the POS application and an indication whether the function succeeded or failed.
3. If the function is successful, the POS application receives the updated counts of all denominations that the CTM can dispense to a customer as change.
4. The same process occurs when the POS application requests for the non–dispensable counts except that it calls the function to read non–dispensable counts and obtains the updated counts of all denominations that cannot be dispensed to a customer as change from the CTM service.
5. If the function for reading the dispensable and non–dispensable counts failed, the POS system displays an error message and provides the steps that the cashier must follow to correct the error.

Refer to the following diagram for more information.



Handling Customer Transaction Operations

This section lists the operations involved during a customer transaction:

- Starting a customer transaction
- Accepting money in a customer transaction
- Ending a deposit transaction
- Dispensing change in a customer transaction
- Ending a customer transaction

To view sample diagrams of customer transactions, refer to *Sample Customer Transaction Diagrams* on page 38

Starting a Customer Transaction

Starting a customer transaction follows this process:

1. When a shopper presents store items to be purchased, the POS application sends a begin transaction function to the CTM service to begin the customer transaction.
2. The CTM service sends back a response to the POS application indicating whether the begin transaction function succeeded or failed.
3. If the function is successful, the cashier can now scan the shopper's items and the CTM is ready to accept cash and dispense change.

Note: A customer transaction can successfully start if the CTM client is initialized and the CTM system has not started a cash management transaction.

Accepting Money in a Customer Transaction

Accepting money in a customer transaction follows this process:

1. When shopper provides money for payment during a customer transaction, the POS application signals the CTM service to accept cash or coin through a begin deposit function and specifies the expected amount that the CTM must accept.
2. The CTM service enables the cash or coin acceptor devices and sends an indication to the POS application on whether the begin deposit function succeeded or failed.
3. If the function succeeded, the shopper can now deposit coins or notes.
4. The acceptor devices generate a data input event that signals the CTM service of the deposit.
5. The CTM service then signals the POS application of the amount that has been deposited.
6. The POS then updates the display of the current amount deposited.

7. If the amount deposited so far is less than the expected amount of payment, the CTM continues to wait and accept payment.
8. If the amount deposited so far is equal or greater than the expected amount of payment, the CTM service disables the cash or coin acceptance.
9. The CTM service sends a data input event to the POS application, which in return queries the total amount deposited so far through a get deposited amount function.
10. The CTM service sends the response back to the POS application, which then updates its display.
11. The CTM service ends the deposit transaction automatically without user interaction and again sends the total deposited amount to the POS application.
12. For deposited amounts that are greater than the expected amount of payment, the POS application issues a dispense request for dispensing change.
13. Sometimes, extra deposit events are made after the expected amount of payment. POS should dispense back the extra deposit events.

Note: If the begin deposit function failed, the POS system displays an error message and provides the steps that the cashier must follow to correct the error.

Ending a Deposit Transaction

Cashiers end a deposit transaction through an end deposit function. When the POS application sends the signal to the CTM service to end the deposit transaction, the CTM service disables the cash and coin acceptor devices, and sends back the total deposited amount to the POS application.

Dispensing Change in a Customer Transaction

Dispensing change in a customer transaction follows this process:

1. In a customer transaction, when the deposited amount is greater than the expected amount of payment, the POS application issues a dispense request to the CTM service by calling a dispense change function.
2. When the CTM service receives the request, it issues a dispense request to the cash changer service, which then dispenses the requested amount of change.
3. The CTM service sends back an indication to the POS application whether the dispense change function succeeded or failed.
4. If the function is successful, the shopper receives the change and the customer transaction is complete.

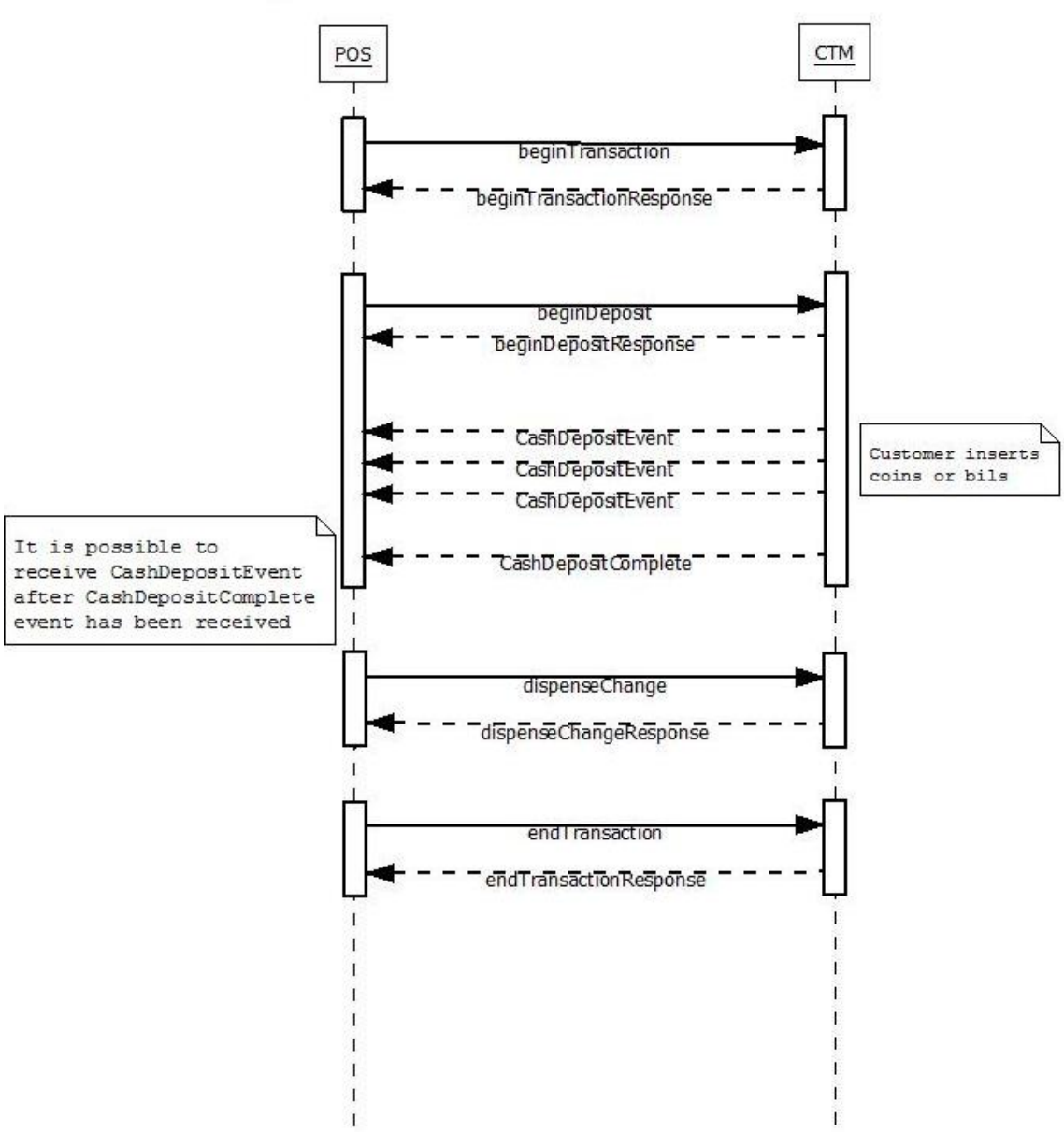
Ending a Customer Transaction

Ending a customer transaction follows this process:

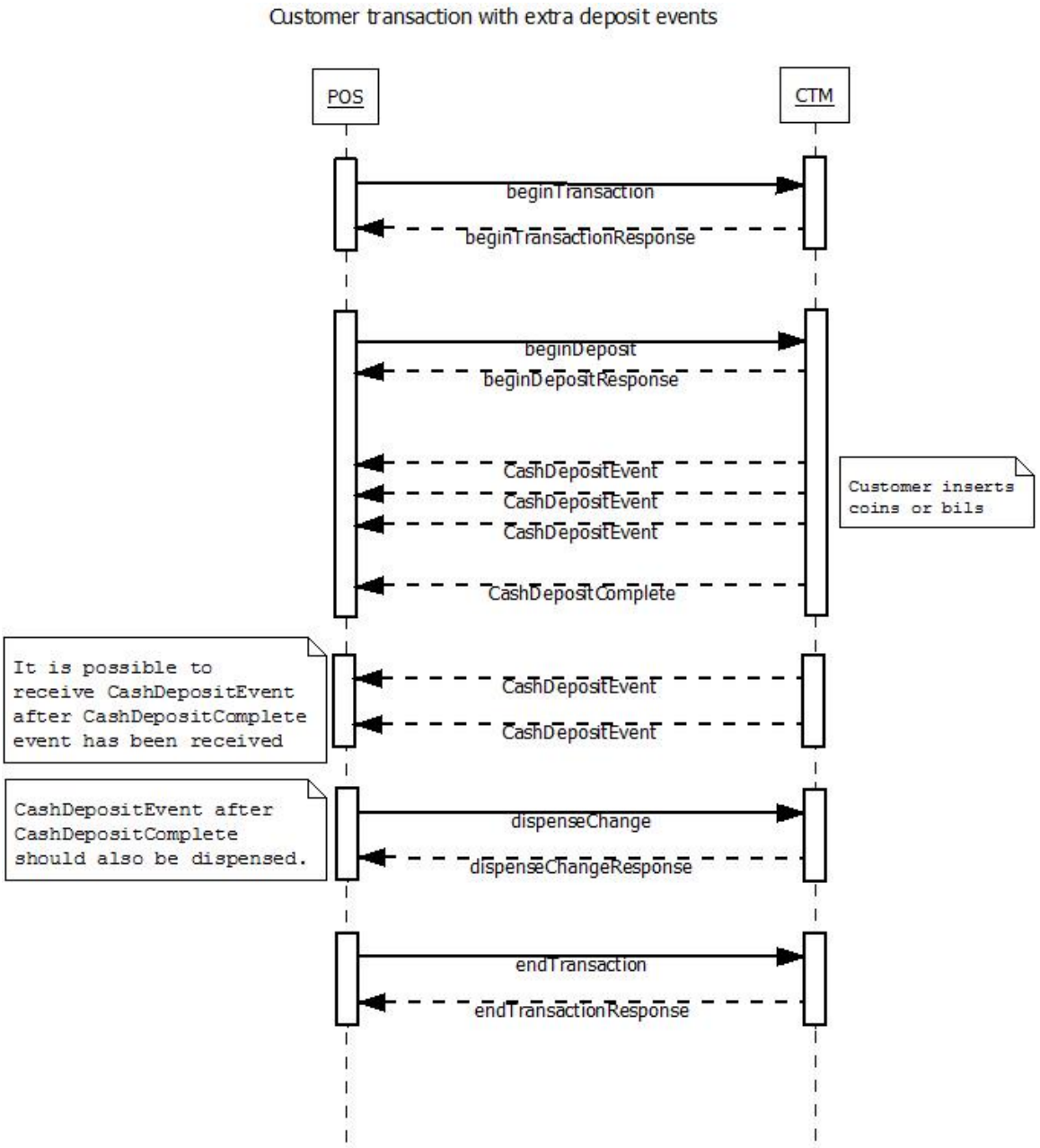
1. When a customer transaction is finished, the POS application signals the CTM service to end the customer transaction through an end transaction function.
2. When the CTM service receives the request, it determines whether cash has been deposited or dispensed during the customer transaction.
3. If cash has been deposited or dispensed, the CTM service generates and saves the cash activity information in an `.xml` file and sends it to the Advanced Cash Office (ACO) server.
4. The CTM service sends back an indication to the POS application whether the end transaction function succeeded or failed.
5. If the function is successful, the CTM is no longer in a transaction mode.

Sample Customer Transaction Diagrams

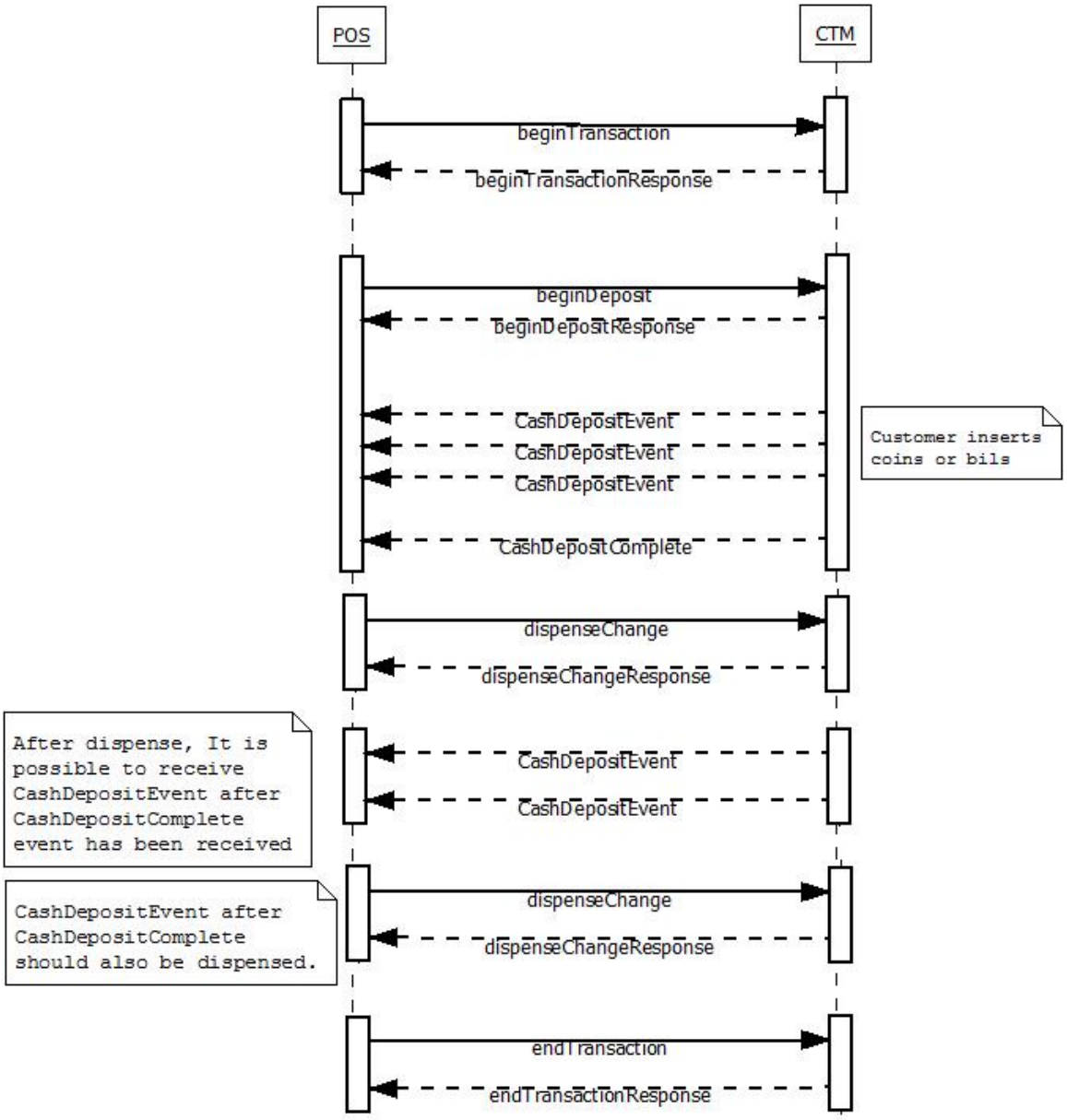
The following is a sample diagram for a customer transaction:



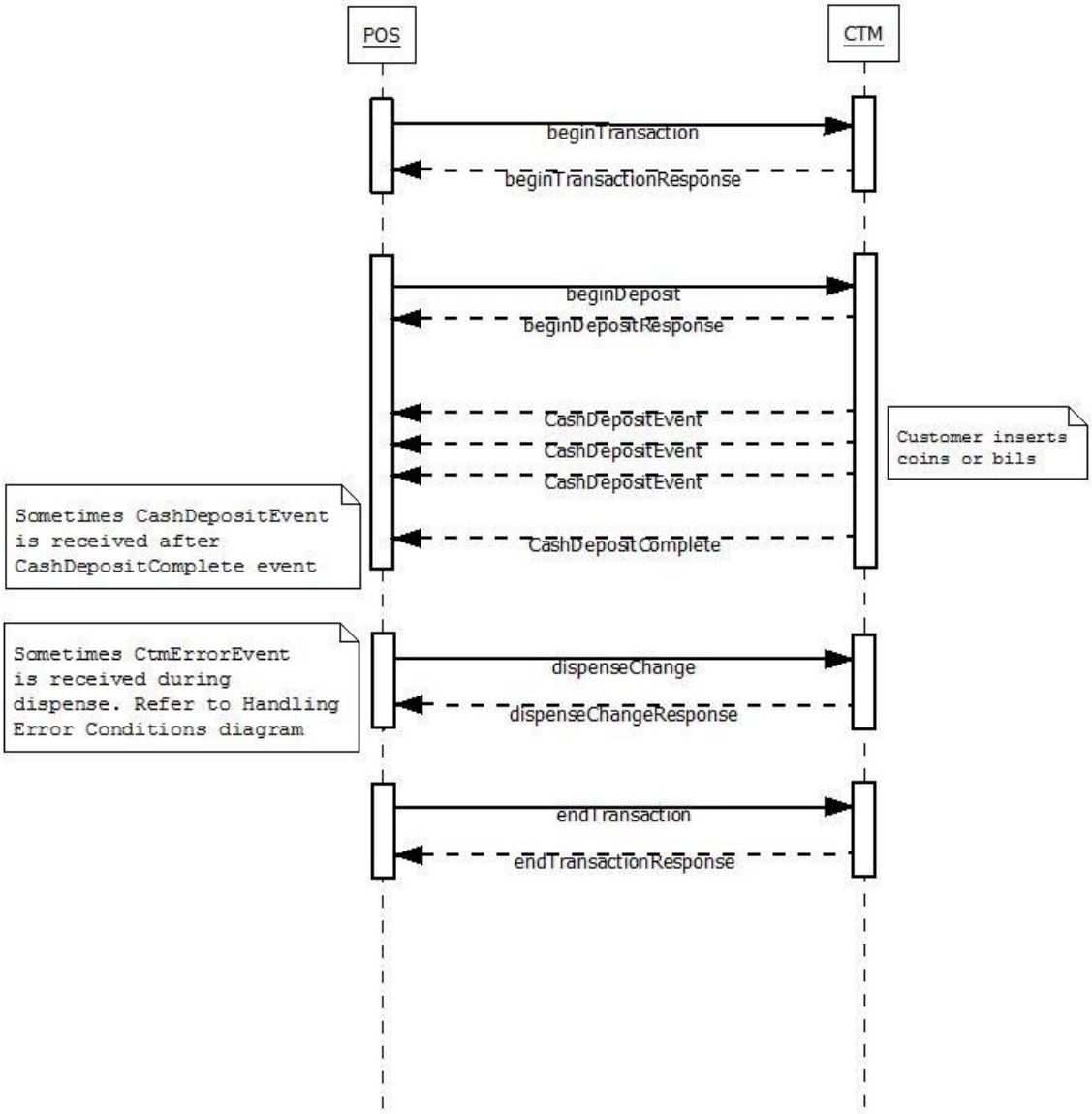
The following is a sample diagram for a customer transaction with an extra deposit event:



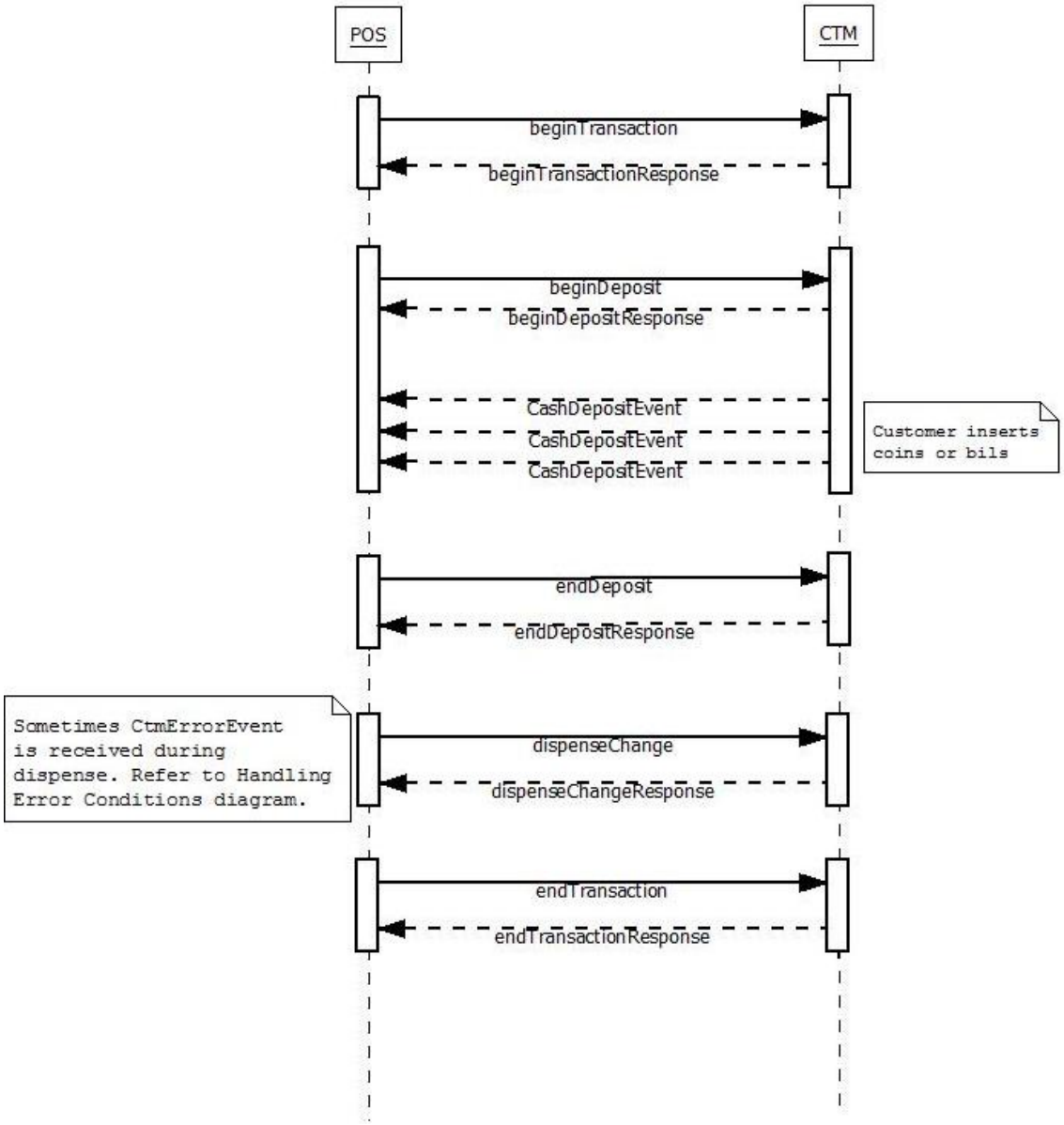
The following is a sample diagram for a customer transaction with an extra deposit event after a dispense event:



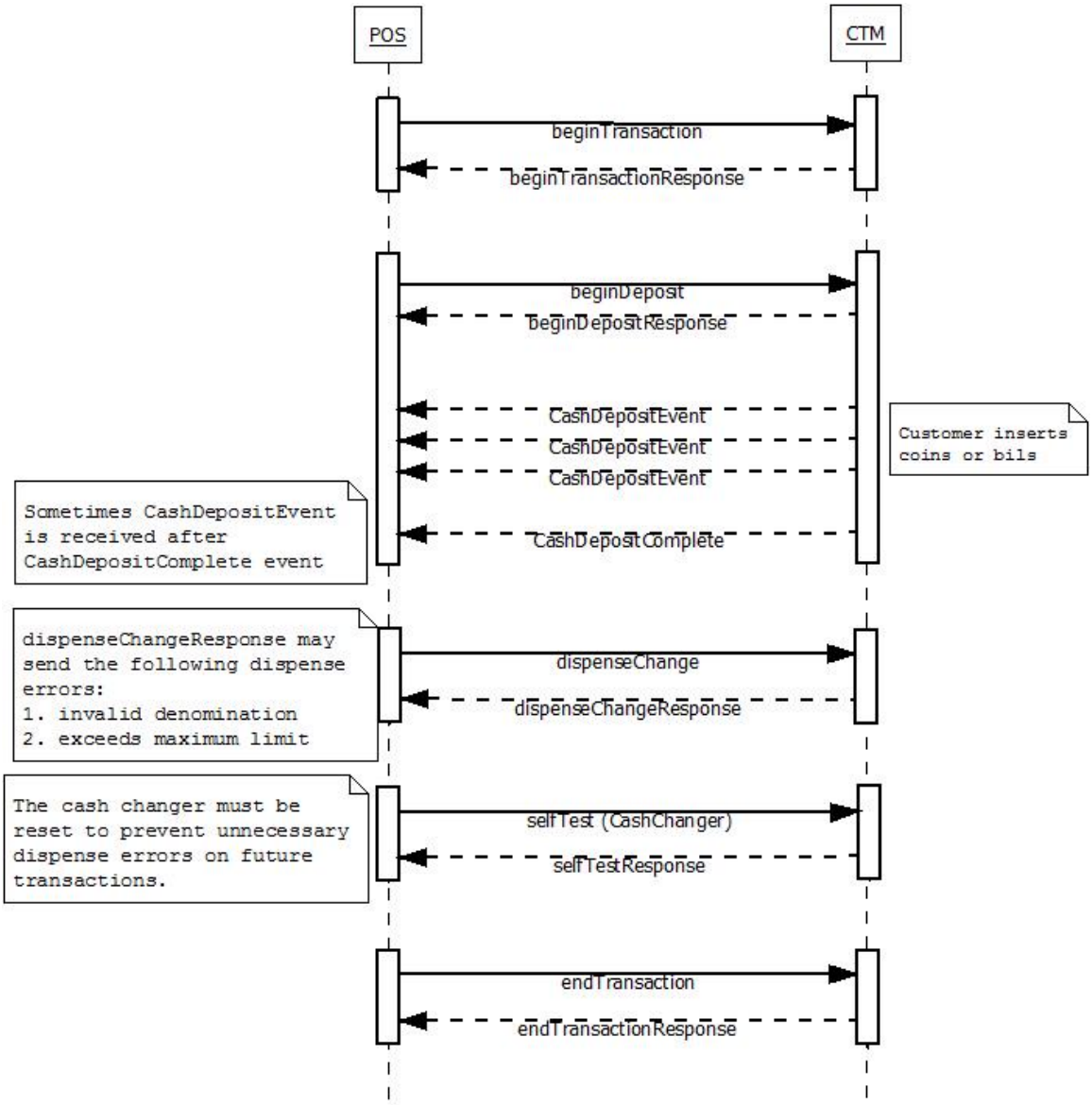
The following is a sample diagram for a customer transaction with a device error:



The following is a sample diagram of a customer transaction without a target amount:



The following is a sample diagram of a customer transaction with a dispense non–device error:



Note: Resetting the cash changer is required only if the dispense error is caused by any of the following events:

- The denomination specified is an invalid denomination.
- The amount specified exceeds the maximum limit.

Handling Cash Management Transactions

This section describes the cash management transactions performed in a CTM system:

- Starting a cash management transaction
- Ending a cash management transaction
- Setting the loader cassette counts
- Setting the loader cassette counts - load money first
- Dispensing cash by denomination
- Transferring notes from the loader cassette to the cash box
- Transferring notes from the BNR recycler to the cash box
- Refilling the recycler devices
- Resetting the dispensable coin counts
- Resetting the non-dispensable coin counts
- Resetting the non-dispensable note counts
- Setting the dispensable counts
- Querying and clearing the purge status
- Querying the dispensable capacities
- Querying non-dispensable capacities

Starting a Cash Management Transaction

The Advanced Cash Office (ACO) requires the CTM service to send cash management loan and pickup data after a loan or pickup operation. To determine when a loan or pickup operation started, the POS application must first signal the CTM system to start a cash management transaction. Before starting a cash management transaction, you must first meet the following preconditions:

- Ensure that the CTM client is initialized.
- Ensure that the CTM system has not started a customer transaction.
- Ensure that the store personnel requesting the start of a cash management transaction is authorized to perform cash management operations and is authenticated at the POS application.

Starting a cash management transaction follows this process:

1. To start a cash management transaction, the POS application sends a begin cash management transaction function to the CTM service. It also sends the user IDs of the current POS operator and of the head cashier who is in–charge of cash handling operations.
2. The CTM service resets all internal counters and other data related to cash management transaction, and records the current balance at that point to determine the difference at the end of the transaction.
3. The CTM service sends back an indication to the POS application whether the transaction was started successfully.
4. If the cash management transaction is started successfully, the CTM system is ready for cash management operations.

Ending a Cash Management Transaction

Ending a cash management transaction follows this process:

1. When a cash management transaction is finished, the POS application signals the CTM service to end the cash management transaction through an end transaction function.
2. When the CTM service receives the request, it determines whether cash has been loaned or picked up during the cash management transaction.
3. If cash has been loaned or picked up, the CTM service generates and saves the cash activity information in an `.xml` file and sends it to the Advanced Cash Office (ACO) server.
4. The CTM service sends back an indication to the POS application whether the end transaction function succeeded or failed.
5. If the function is successful, the CTM is no longer in a transaction mode.

Setting the Loader Cassette Counts

The Bulk Note Recycler (BNR) device includes a loader cassette and a recycler. Bills are added to the recycler through the loader cassette. The loader cassette holds a buffer of a single note denomination to maintain the recycler level for a specific denomination at a configurable level. The POS application must inform the BNR device of the number of notes to be placed in the loader cassette to track the total number of notes in the device. This cash management operation is restricted to the cash–handling personnel only.

Setting the loader cassette counts follows this process:

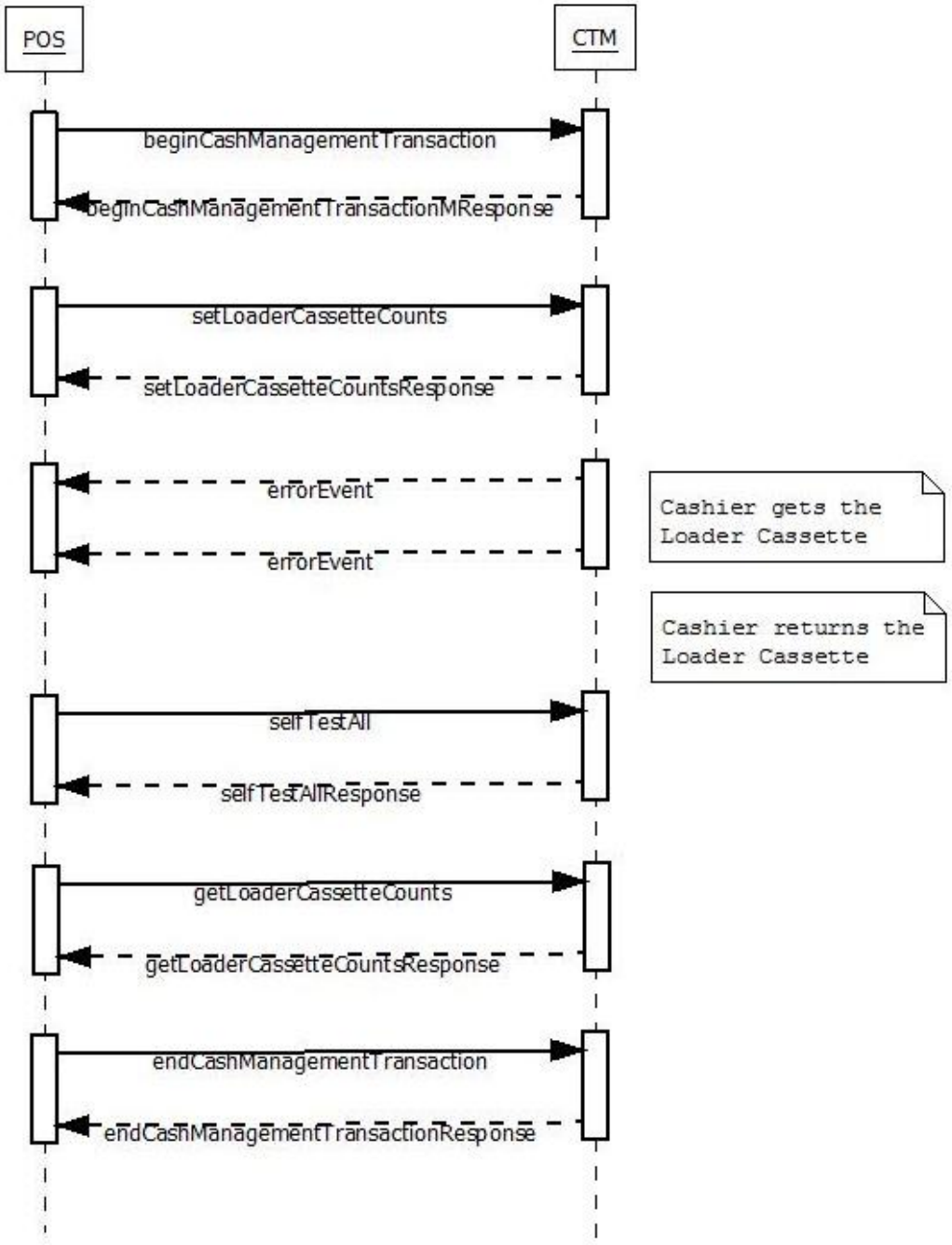
1. The cash–handling staff must first start a cash management transaction.
2. The staff must then enter the count of each note denomination to be placed into the loader cassette through the POS application.
3. The POS application issues a set loader count function to the CTM service, which relays the function to the cash changer interface.
4. The CTM service then sends an indication whether the function succeeded or failed.
5. If the request to set the loader count is successful, the cash–handling staff opens the CTM device, takes out the loader cassette device, and brings it to the cash office to be filled with bills.

Note: CTM does not have a way to query the actual purge counts but instead only queries the purge status. A purge bill would cause the loader counts to be inconsistent. Place valid bills inside the loader cassette as much as possible.

6. After filling the loader cassette device with bills, the cash–handling staff replaces the loader cassette into the BNR device and clears the error generated when the CTM device was opened earlier.
7. The BNR device transfers one or more notes from the loader cassette to the recyclers.
8. The POS application updates the display of the loader count through a get loader count function.

Note: The BNR device does not maintain the loader cassette counts but it is maintained by the CTM software. Upon initialization, external files `AcceptorCount.dat` and `DispenserCount.dat` are created, which update the total counts that is used to get the current loader counts when requested. The files are located by default at `C:\scot\Data\`. When getting the loader counts and incorrect counts are reported, empty all bills inside the BNR and delete the `AcceptorCount.dat` and `DispenserCount.dat` files before resuming operations.

Refer to the following diagram for more information.



Setting the Loader Cassette Counts—Load Money First

This cash management operation is similar to the cash management operation discussed in the previous section except for the following:

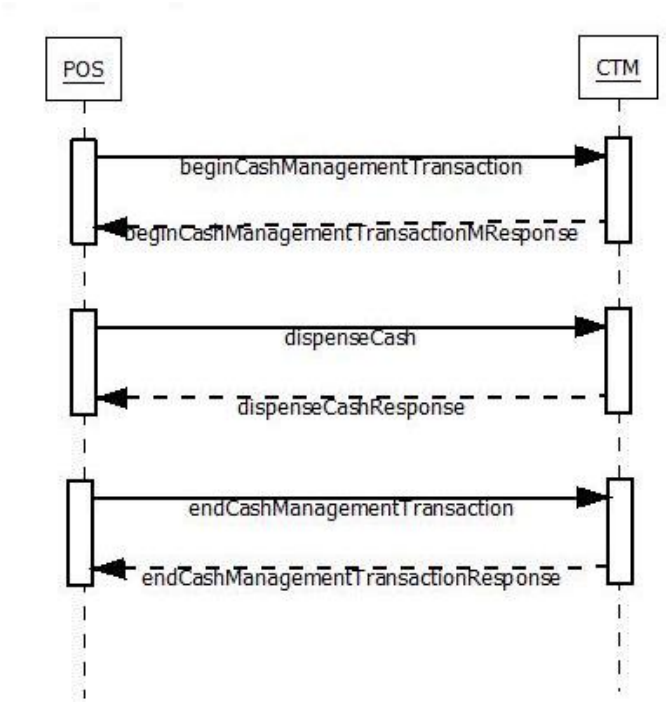
- Taking out the loader cassette from the BNR device, filling it with bills, and transferring of bills from the loader cassette to the recycler occur before setting the count of each note denomination to be placed into the loader cassette device.
- The number of notes in the loader cassette is less than the number that the cash–handling staff originally entered.

Dispensing Cash by Denomination

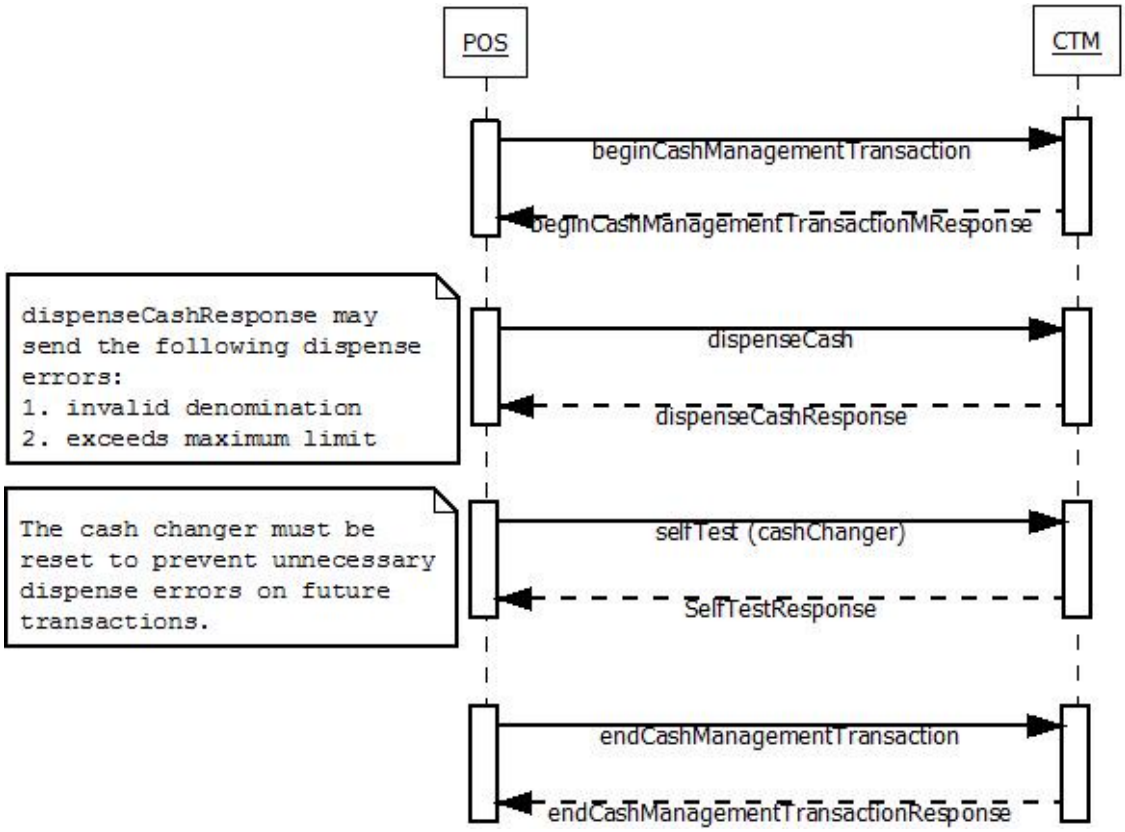
The cash–handling personnel can dispense cash by denomination from recycler devices to remove excess cash at the end of the day. Dispensing cash by denomination follows this process:

1. The cash–handling staff must first start a cash management transaction.
2. The cash–handling staff specifies the quantity of each denomination that is dispensed and checks whether the cash exit is not blocked, for example, when the Bulk Coin Recycler (BCR) is empty.
3. If the cash exit is not blocked, the POS application issues a dispense request for cash by denomination.
4. The CTM services receives the request and issues a dispense request to the cash changer service.
5. The cash changer service dispenses the requested amount.
6. The CTM service queries the cash changer services for the amount that was dispensed and sends back an indication to the POS application of whether the dispense operation succeeded or failed.
7. If the dispense operation is successful, the cash–handling staff obtains the requested quantity of each denomination.

Refer to the following diagram for more information.



The following is a sample diagram of a non–device error encountered while dispensing cash by denomination:



Note: Resetting the cash changer is required only if the dispense error is caused by any of the following events:

- The denomination specified is an invalid denomination.
- The amount specified exceeds the maximum limit.

Transferring Notes from the Loader Cassette to the Cash Box

At the end of the day, the cash–handling staff removes cash from the BNR device. Notes from the BNR loader cassette are transferred to the cash box, which is then transported to the cash office for counting. Transferring cash to the cash box avoids the presence of loose cash on the store floor, which provides greater cash security.

Note: It takes approximately one to two seconds to transfer each note from the loader cassette to the cash box. Therefore, if the loader cassette is filled to its capacity of 200 notes, completing the transfer can take up to seven minutes.

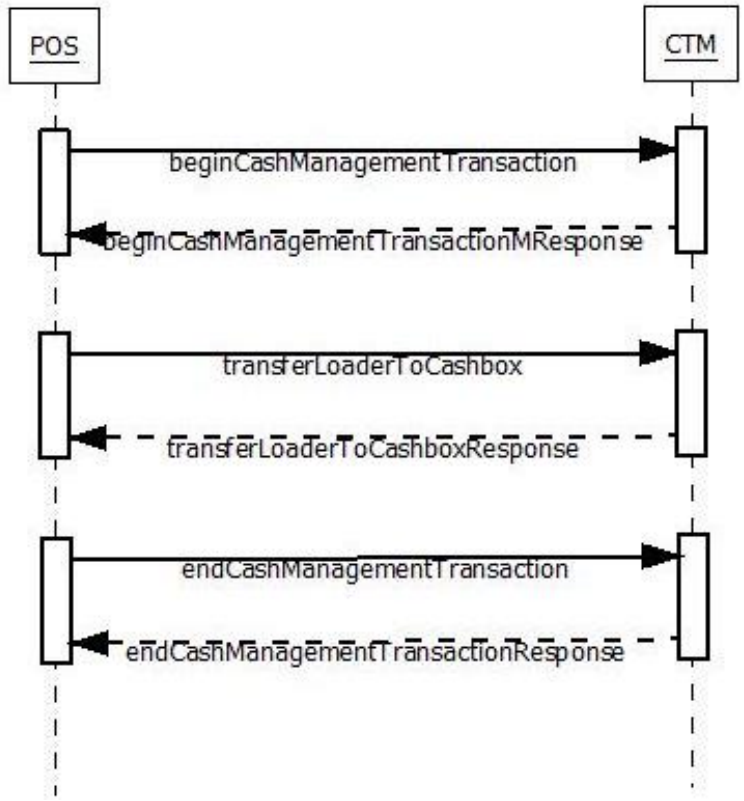
Aside from transferring all notes from the loader cassette to the cash box, the cash–handling personnel must also transfer all notes from the recycler device to the cash box. For more information, refer to *Transferring Notes from the BNR Recycler to the Cash Box* on page 54.

Transferring notes from the loader cassette to the cash box follows this process:

1. The cash–handling personnel must first start a cash management transaction.
2. The cash–handling personnel issue a request to transfer notes and the POS application issues the corresponding function.
3. The CTM service receives the request and verifies if the cash box has enough space to accommodate all the notes from the loader cassette.
4. The CTM service then issues the transfer request to the cash changer device.
5. The BNR device starts to transfer all the notes from the loader cassette to the cash box.
6. The CTM service sends an indication to the POS application that the transfer request succeeded.

After the transfer, the net balance of the BNR device is unchanged. The loader cassette is empty and the cash box count is the previous count plus the number of notes that was transferred from the loader cassette.

Refer to the following diagram for more information.



Transferring Notes from the BNR Recycler to the Cash Box

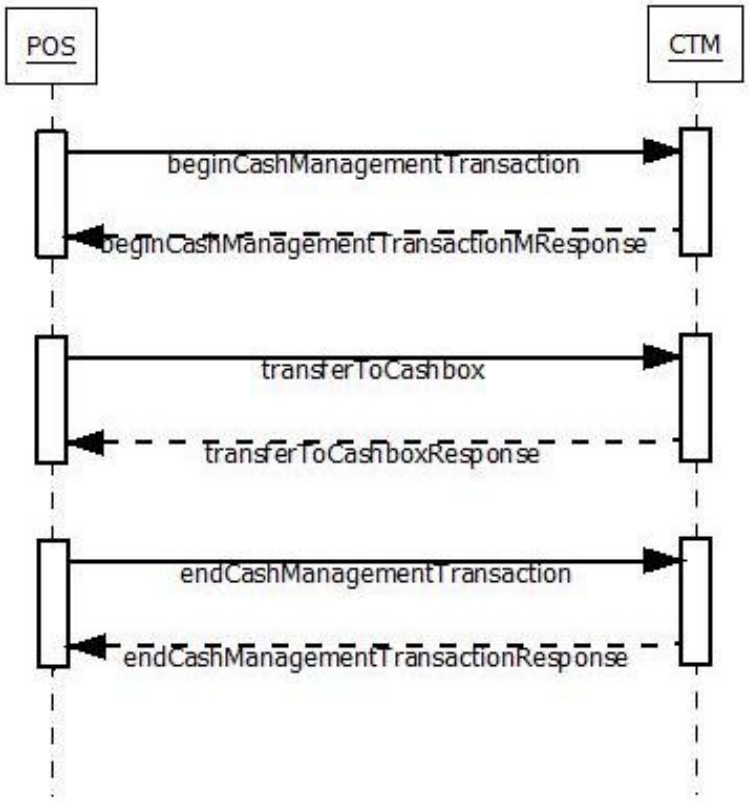
The cash–handling personnel can transfer notes from the BNR recycler device to the cash box as part of the process of emptying the BNR device at the end of the day. This cash management operation provides greater cash security because loose cash is avoided on the store floor.

Transferring notes from the BNR recycler to the cash box follows this process:

1. The cash–handling personnel must first start a cash management transaction and then issue a request to transfer one or more denominations from the note recyclers to the cash box.
2. The POS application then calls a transfer function and passes the denomination count data to the CTM service.
3. The CTM service verifies if there is enough space in the cash box to accommodate the recycler notes.
4. If there is enough space, the CTM service sends a request to the cash changer device to start the transfer.
5. The BNR device transfers all the notes and corresponding counts from the recyclers to the cash box.
6. The CTM service sends back an indication to the POS application that the transfer request succeeded.

After the transfer, the net note balance of the BNR device is unchanged. The recycler counts are decreased by the number of notes that was transferred to the cash box, and the cash box count is increased by the number of notes that was transferred from the recyclers.

Refer to the following diagram for more information.



Refilling the Recycler Devices

Replenishment operations are required to refill the recycler devices when coin and note levels are low. Before starting a replenishment operation, the cash–handling personnel must first start a cash management transaction and ensure that deposit operations for customer transactions are disabled.

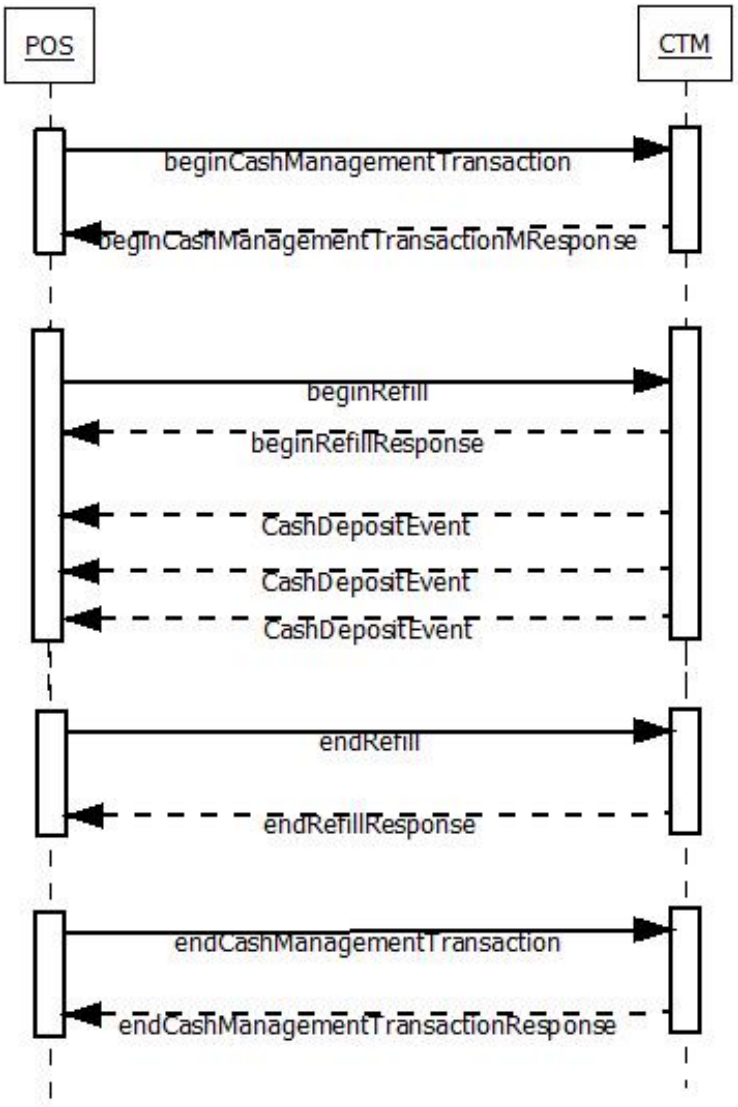
Refilling the recycler devices follows this process:

1. The POS application calls a function to begin a refill.
2. The CTM service receives the request and then enables coin and note acceptance and sets the Bulk Coin Recycler (BCR) device to *relaxed refill* mode.
3. The cash–handling personnel start depositing one or more coins or notes to the recycler devices.
4. The CTM service calls an end refill function, after which the cash–handling personnel cannot deposit anymore.
5. The CTM service sends back an indication to the POS application that the deposit has ended.

The cash–handling personnel may obtain the latest balance by calling a function to read the dispensable and non–dispensable counts. For more information, refer to *Reading Dispensable and Non–dispensable Counts* on page 33.

Note: When performing a refill and accepting a large number of coins, the high–level hopper sensor is covered by default. Because hoppers sometimes fill unevenly, the BCR will dispense three coins to try to settle the coins in the hopper so that it can accept more before the hopper high–level sensor redirects the coins to the overflow bin. Because cash deposit events were already sent to the POS for the three dispensed coins, using the sum of cash deposit events to determine the total deposited cash is unreliable. Use the function to read the dispensable counts instead.

Refer to the following diagram for more information.



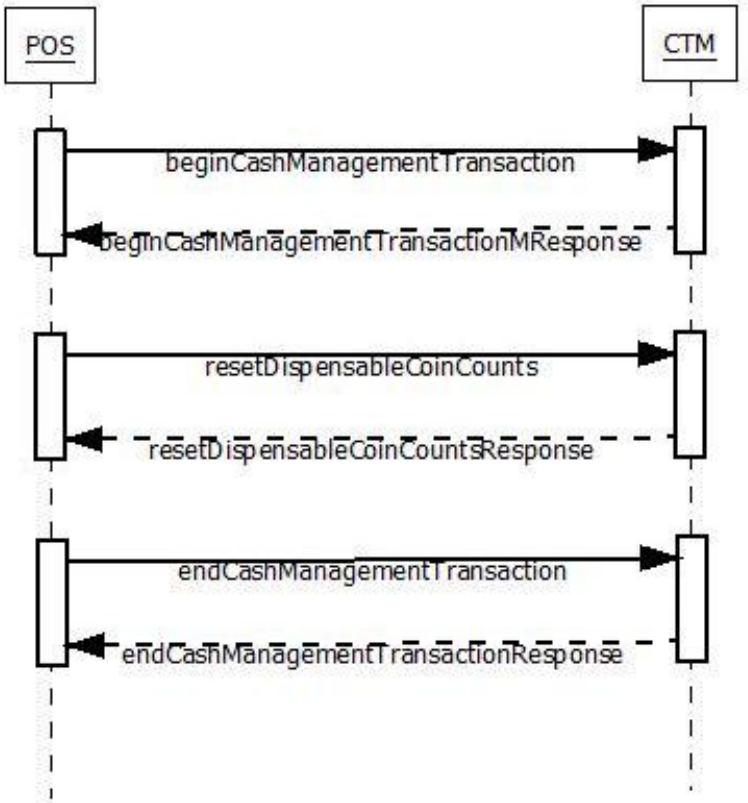
Resetting the Dispensable Coin Counts

The cash–handling personnel can remove money from the Bulk Coin Recycler (BCR) without assistance from the device. When the cash–handling personnel perform this operation, the dispensable coin counts must be reset to zero. Before setting the dispensable coin counts to zero, the cash–handling personnel must first start a cash management transaction.

Resetting the dispensable coin counts follows this process:

1. When the cash–handling personnel send a signal that the coin dispenser or recycler will be logically emptied, the POS application calls a reset function.
2. The CTM receives the request from the POS application and issues a request to reset the dispensable coin counts in the cash changer device.
3. The CTM service sends back an indication to the POS application whether the request succeeded or failed.
4. If the request is successful, the cash–handling personnel can physically remove all the coins from the recycler device.
5. The coin recycler or dispenser counts are set to zero.

Refer to the following diagram for more information.



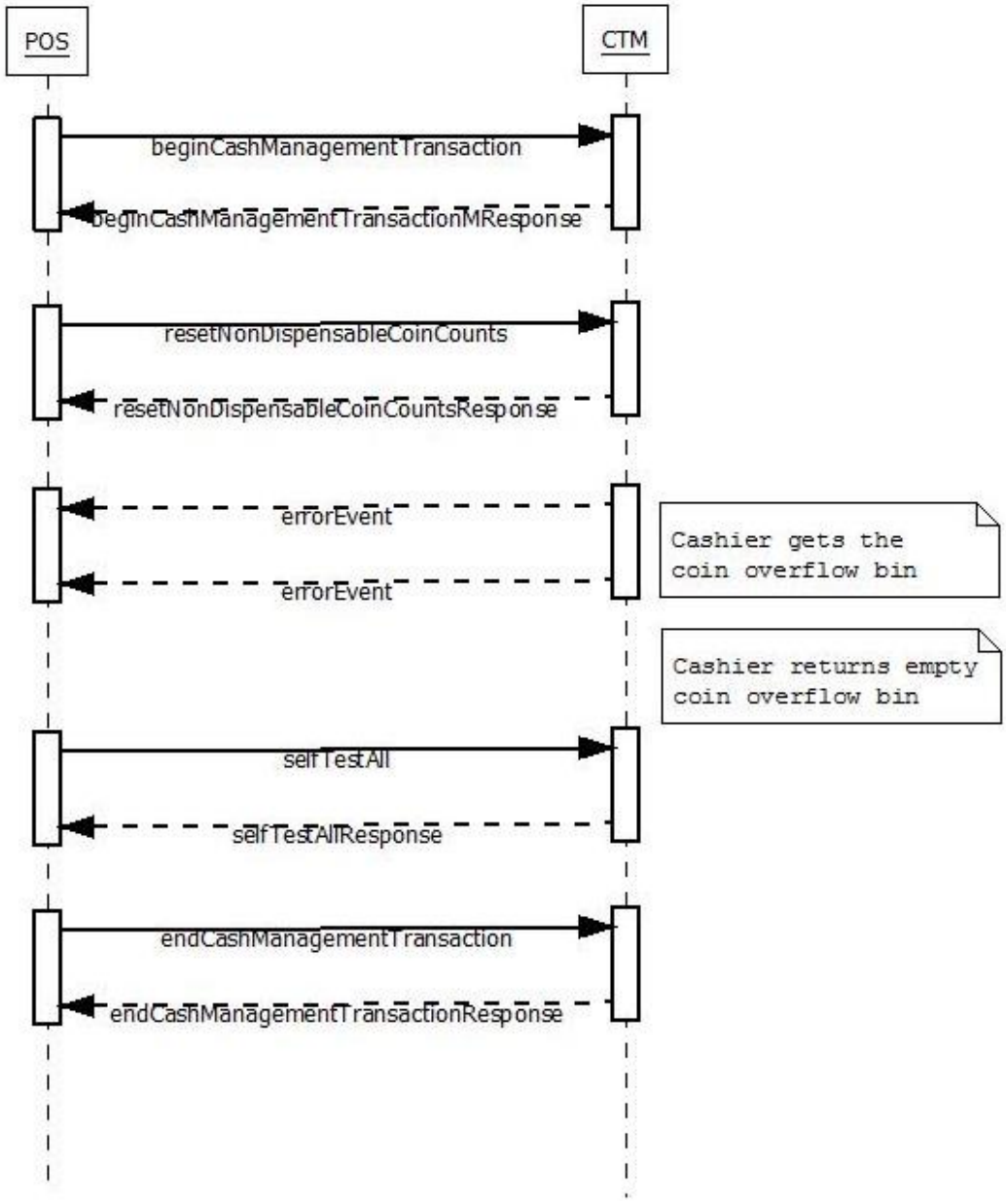
Resetting the Non–dispensable Coin Counts

The cash–handling personnel can remove all the non–dispensable coins or the coins from the coin overflow container or coin container of the BCR device. When the cash–handling personnel perform this operation, the non–dispensable coin counts must be reset to zero. The cash–handling personnel must first start a cash management transaction before resetting the non–dispensable coin counts to zero.

Resetting the non–dispensable coin counts follows this process:

1. When the cash–handling personnel send a signal that the coin overflow or coin acceptor will be logically emptied, the POS application calls a reset function.
2. The CTM service receives the request from the POS application and issues a request to reset the non–dispensable coin counts in the coin acceptor device.
3. The CTM service sends back an indication to the POS application whether the request succeeded or failed.
4. If the request is successful, the cash–handling personnel remove the coin overflow or coin acceptor container and the coins from it.
5. The cash–handling personnel must replace the empty coin overflow or coin acceptor container to clear any error from taking out the container.

Refer to the following diagram for more information.



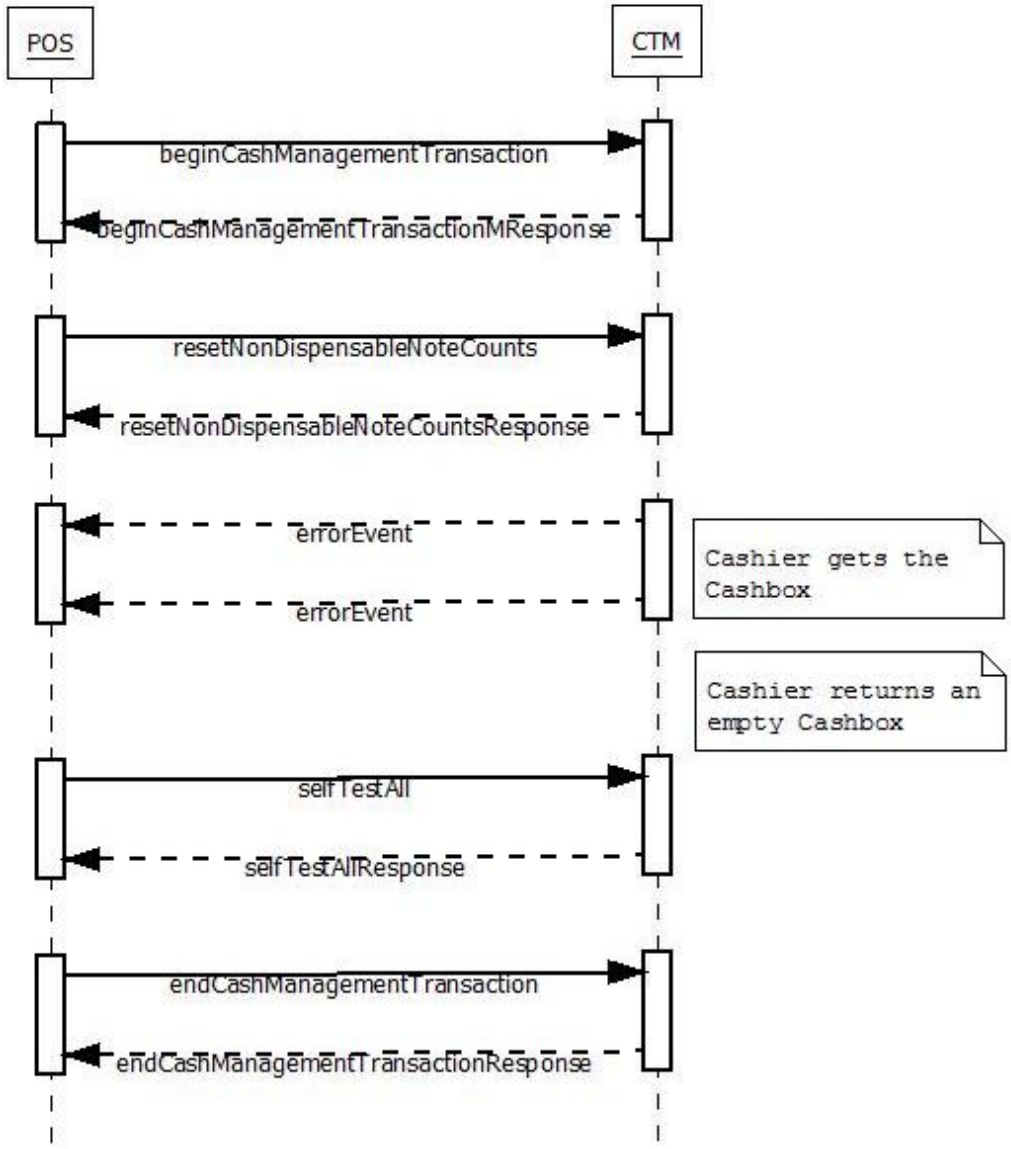
Resetting the Non–dispensable Note Counts

The cash–handling personnel can remove non–dispensable notes or the notes from the note cashbox of the BNR device. When the cash–handling personnel perform this operation, the non–dispensable note counts must be set to zero. Before resetting the non–dispensable note counts to zero, the cash–handling personnel must first start a cash management transaction.

Resetting the non–dispensable note counts follows this process:

1. When the cash–handling personnel send a signal that the cash box will be logically emptied, the POS application calls a reset function.
2. The CTM service receives the request from the POS application and issues a request to reset the non–dispensable note counts in the cash box.
3. The CTM service sends back an indication to the POS application whether the request succeeded or failed.
4. If the request is successful, the cash–handling personnel remove the cash box and the notes from it.
5. The cash–handling personnel must replace the empty cash box to clear any error from taking out the container.

Refer to the following diagram for more information.



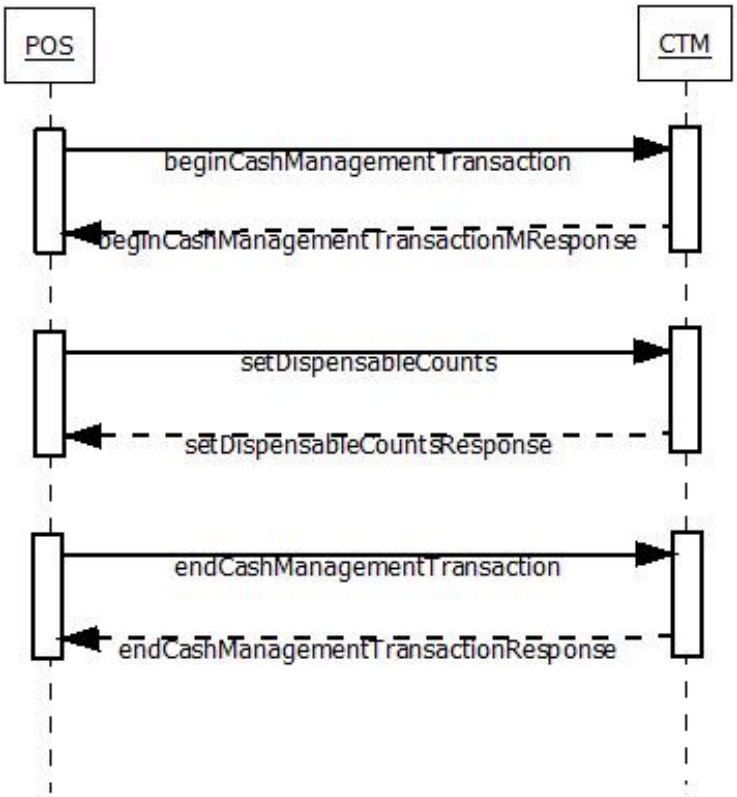
Setting the Dispensable Counts

Cashiers can set the logical cash counts of recyclers or dispensers in the BCR device. This operation is not applicable to the BNR device. Before setting the dispensable counts in the BCR device, the cashier must first start a cash management transaction.

In setting the dispensable counts of the BCR device, the cashier enters the number of each denomination that will be added to the recycler or dispenser devices. The POS application calls the set dispensable counts function and sends the entered denomination counts to the CTM service. The CTM service issues a request to set the dispensable counts in the cash changer device and sends back an indication to the POS application whether the request succeeded.

If the request is successful, the cashier places the corresponding counts of each coin into the dispenser cassettes.

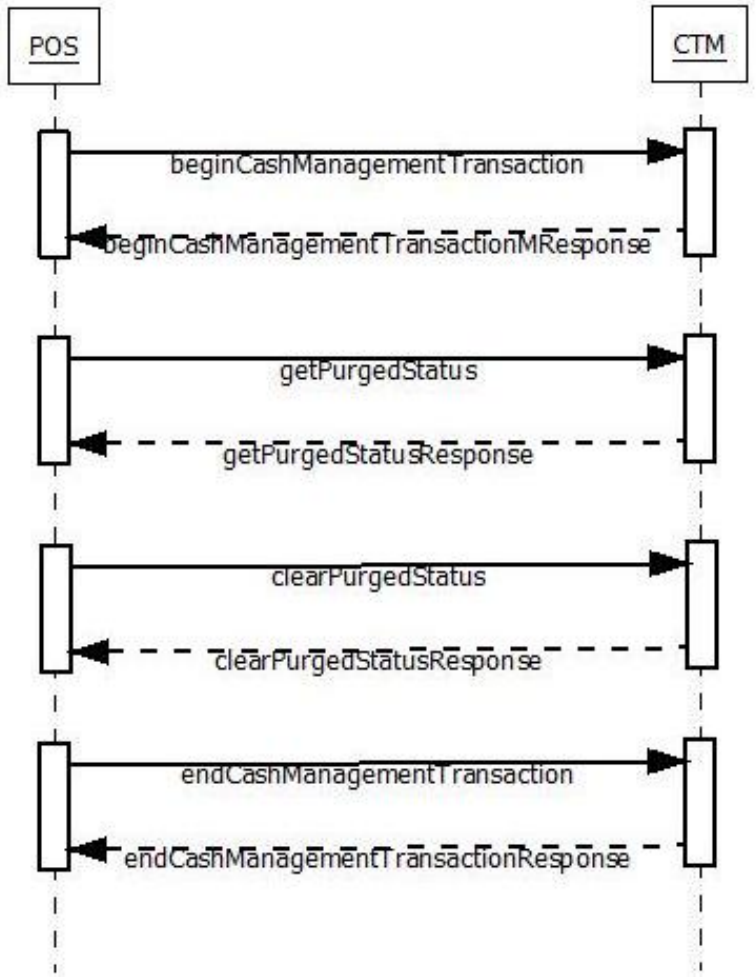
Refer to the following diagram for more information.



Querying and Clearing the Purge Status

The cash–handling personnel can determine whether the CTM device has sent any defective dispensable notes to the purge bin. The POS application also has the ability to determine whether a defective note is placed in the BNR loader cassette that the validator cannot read. The BNR device can report the purge status but it cannot indicate the denomination. After the purge notes are removed, the purge status can be reset to false by clearing the purge status.

Refer to the following diagram for more information.



Querying the Dispensable Capacities

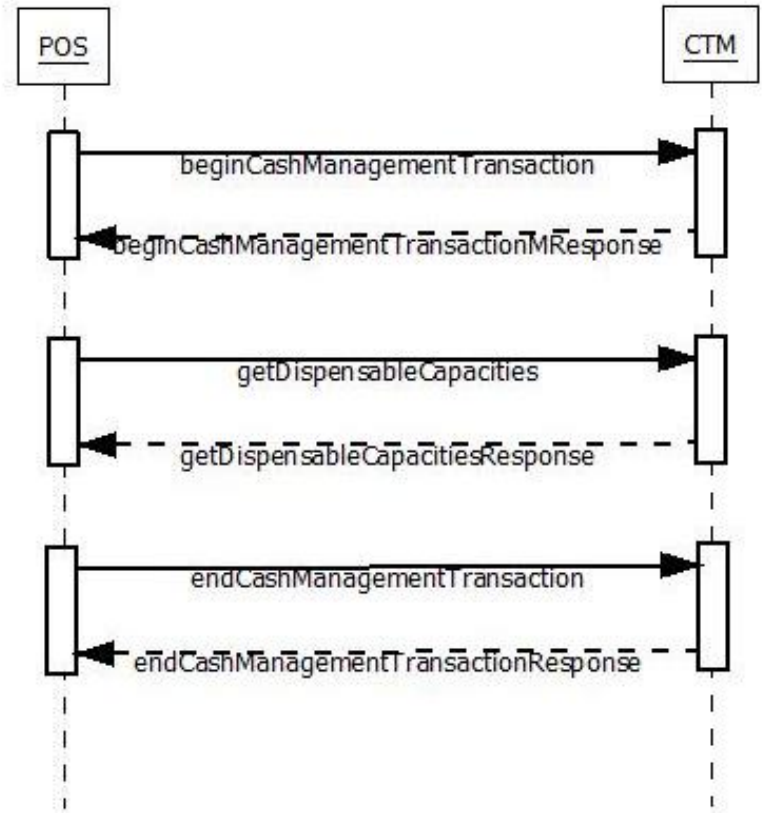
To determine the full percentage status of the dispensable denominations of the CTM device, the cash–handling personnel must determine the capacity of the storage locations for the dispensable denominations. To determine the total dispensable capacities, the cash–handling personnel must first start a cash management transaction.

Querying the dispensable capacities follows this process:

1. The cash–handling personnel issue a request to get the dispensable capacities through the POS application.
2. The CTM service receives the request and queries the cash changer interface for the capacities of all dispensable denominations.
3. The CTM service then sends an indication back to the POS application whether the request succeeded or failed.
4. If the request is successful, the POS application receives information on the capacities of all denominations.

Note: In this operation, the returned capacities do not include the capacity of the loader cassette of the BNR device.

Refer to the following diagram for more information.



Querying Non–dispensable Capacities

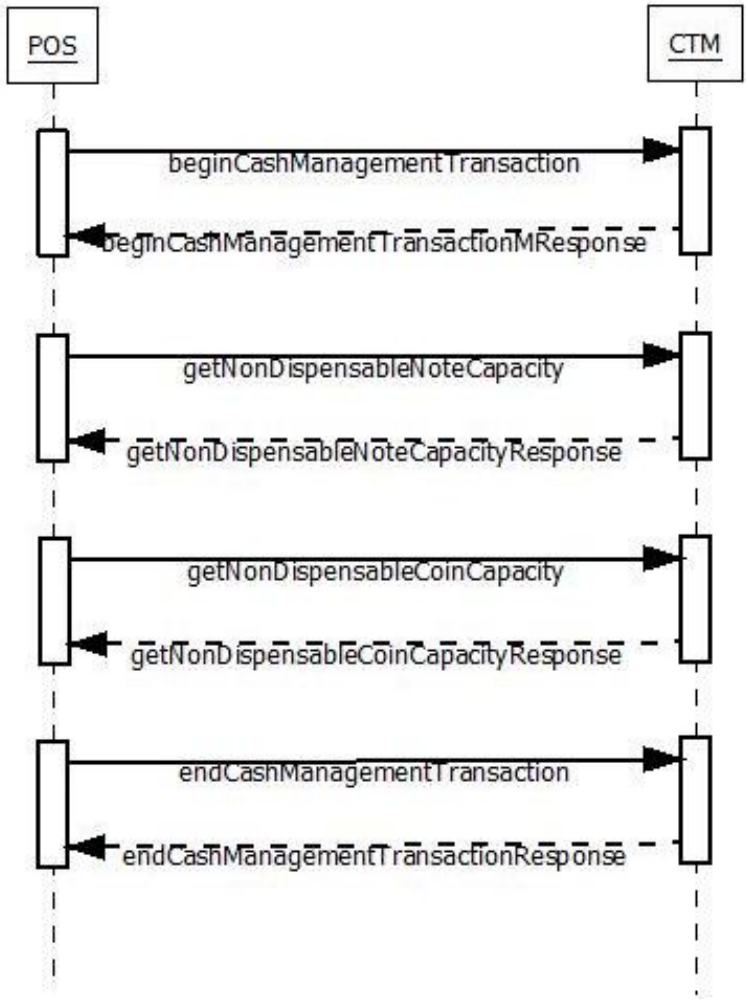
To determine the full percentage status of the non–dispensable denominations of the CTM device, the cash–handling personnel must determine the capacity of the storage locations for the non–dispensable denominations. These storage locations include the coin overflow, coin acceptor, cash box, cash acceptor, and so forth. To determine the total non–dispensable capacities, the cash–handling personnel must first start a cash management transaction.

Querying the non–dispensable capacities follows this process:

1. The cash–handling personnel issue a request to get the non–dispensable capacities through the POS application.
2. The CTM service receives the request and queries the coin acceptor and cash acceptor devices for the capacities of all non–dispensable denominations.
3. The CTM service then sends an indication back to the POS application whether the request succeeded or failed.
4. If the request is successful, the POS application receives information on the capacities of all denominations.

Note: The BCR device cannot provide the coin overflow capacity. A configurable `CTMConfig.properties` file, however, has the estimated value, which the user can change as needed. It is set to 1000 on the CoinOverFlowCapacity by default. The file path is `C:\ctm\products\cash-tender-module\trunk\cash-tender-module-shared\src\main\resources\CTMConfig.properties`.

Refer to the following diagram for more information.



Handling Error Conditions

This section describes the error conditions that cashiers or any cash–handling personnel may encounter during a customer transaction or a cash management transaction.

- Device error
- Dispense error

Handling error conditions follows this process:

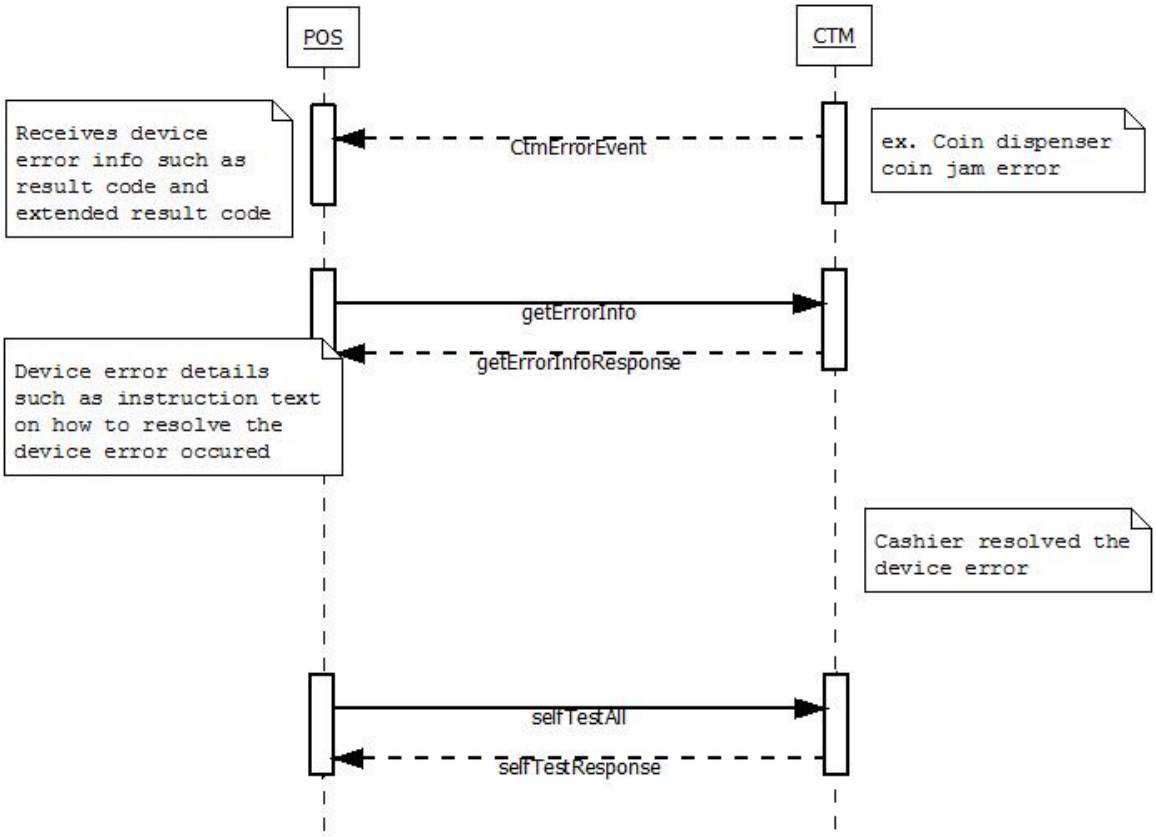
1. When error events occur, the POS application receives and displays textual instructions from the CTM system, which the cashier or cash–handling personnel must follow to correct the error.
2. After resolving the error, the cashier or cash–handling personnel send an indication that the resolution steps are complete.
3. The POS application then requests the CTM system to perform a system health check.
4. If the health check result is successful, the CTM system is back in service.

Device Error

A device error occurs when an event happens to one of the CTM devices that causes an error condition. Examples of these events include the following:

- The cashier or any cash–handling personnel removes the coin overflow bin of the BCR device.
- A bill jam occurs during money deposit.

The following is a sample diagram on an error event caused by a coin jam device error.

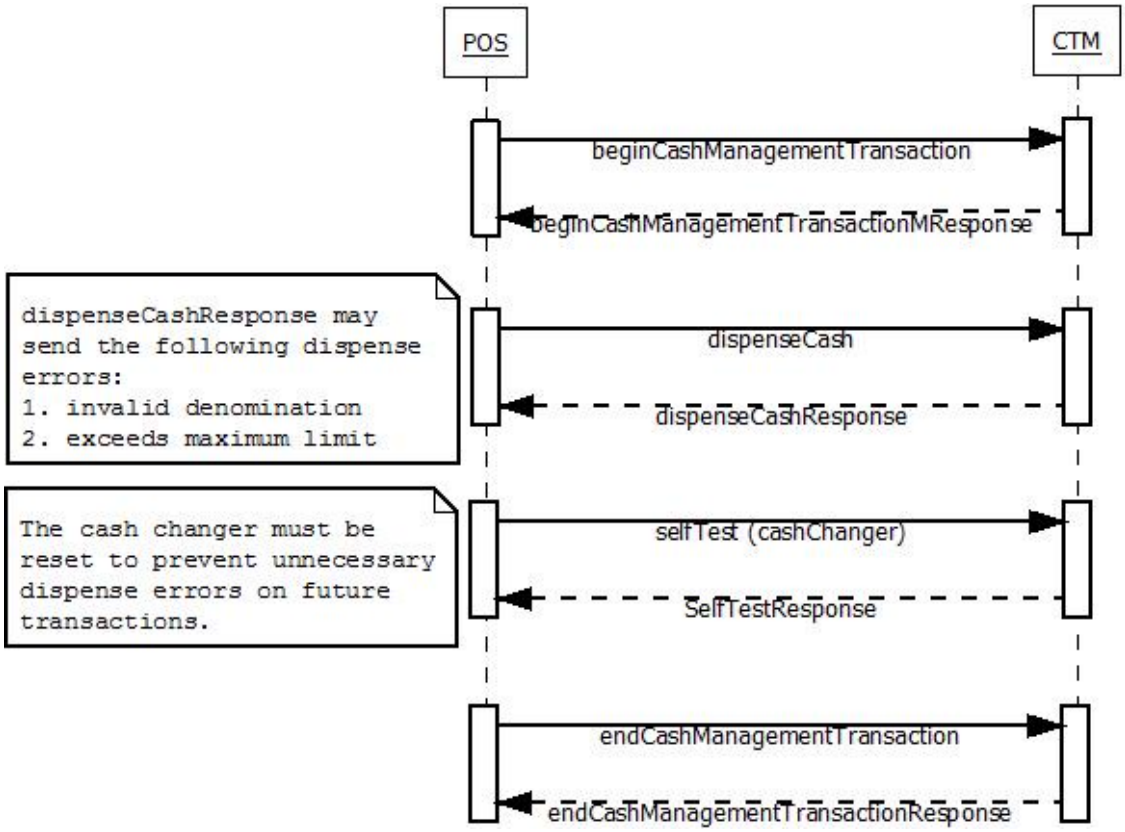


Dispense Error

A dispense error occurs when one of the following events happens during a dispense transaction:

- There is no transaction ID for the transaction.
- The amount specified is a negative number.
- The denomination specified is an invalid denomination
- The amount specified exceeds the maximum limit.
- The CTM client is not connected to the CTM service.

The following is a sample diagram of a non–device error encountered while dispensing cash by denomination:



Note: Resetting the cash changer is required only if the dispense error is caused by any of the following events:

- The denomination specified is an invalid denomination.
- The amount specified exceeds the maximum limit.

An error event may also occur when a command that the POS application sends to the CTM system fails.

Connecting to an Advanced Cash Office (ACO) Feed

After each customer transaction or cash management transaction, the CTM service must transmit information about any money that was deposited or dispensed during the transaction to a listening Advanced Cash Office (ACO) back–office service. An ACO client is installed in each CTM station. The ACO client transmits cash balance data back to the Advanced Cash Office server.

The cash–handling personnel must perform the following to ensure the connection to the ACO service is successful:

- Configure the connection information of the Advanced Cash Office service.
- Ensure that the CTM service is initialized and connected to the ACO server.

The ACO feed follows this process:

1. After each customer transaction and cash management transaction, the ACO client detects whether new XML data is available to be transmitted to the back–office ACO server.
2. If new data exists, the ACO client reads the oldest file and transmits it to the ACO server.
3. If the transmission is successful, the ACO client receives acknowledgement about it and then deletes the file that it transmitted.
4. The ACO server then keeps the updated cash balance information for that particular CTM station.

Chapter 4: Cash Tender Module Client Java API

Overview

This chapter describes the Java API used by the Cash Tender Module (CTM) service to perform the different Cash Tender operations. The different classes and interfaces of the Java API are classified in the following packages:

- com.ncr.ssc.ctm
- com.ncr.ssc.ctm.events
- com.ncr.ssc.ctm.result

Each of these packages contains different classes and interfaces that the CTM client uses to connect to the CTM service.

com.ncr.ssc.ctm

The com.ncr.ssc.ctm package contains the `ICashTenderModuleClient` interface. The `ICashTenderModuleClient` is the interface specification for the client interaction with the CTM service.

ICashTenderModuleClient Interface

The ICashTenderModuleClient interface contains asynchronous methods with return types that are signified by the corresponding specified class of the com.ncr.ssc.ctm.result package. Asynchronous results are returned in the `ICommandCompleteListener` listener method.

Syntax

```
public interface ICashTenderModuleClient
```

ICashTenderModuleClient Methods

The following are the different methods classified in the `ICashTenderModuleClient` interface:

- `initializeCTM`
- `uninitializeCTM`
- `getConfig`
- `addDataEventListener`
- `removeDataEventListener`
- `addCommandCompleteListener`
- `removeCommandCompleteListener`
- `readDispensableCashCounts`
- `readNonDispensableCashCounts`
- `beginTransaction`
- `endTransaction`
- `beginDeposit`
- `endDeposit`
- `getErrorInfo`
- `selfTest`
- `selfTestAll`
- `dispenseChange`
- `beginCashManagementTransaction`
- `endCashManagementTransaction`

- `setLoaderCassetteCounts`
- `getLoaderCassetteCounts`
- `dispenseCash`
- `transferLoaderToCashbox`
- `transferToCashbox`
- `beginRefill`
- `endRefill`
- `resetDispensableCoinCounts`
- `resetNonDispensableCoinCounts`
- `resetNonDispensableNoteCounts`
- `setDispensableCounts`
- `getPurgedStatus`
- `clearPurgedStatus`
- `getDispensableCapacities`
- `getNonDispensableNoteCapacity`
- `getNonDispensableCoinCapacity`
- `getLoaderCassetteCapacity`

initializeCTM

This method connects the current client instance to the CTM service.

Syntax

```
boolean initializeCTM(long timeout)
```

Parameters

- `timeOut`—maximum time to wait for the connection to succeed in milliseconds.

Returns

- true—if initialization succeeded.
- false—if initialization failed.

uninitializeCTM

This method disconnects the current client instance from the CTM service.

Syntax

```
boolean uninitialize CTM()
```

Returns

- true—if disconnection succeeded.
- false—if disconnection failed.

getConfig

This method obtains CTM configuration information.

Syntax

```
CtmConfigResultInfo getConfig()
```

Returns

```
com.ncr.ssc.ctm.result.CtmConfigResultInfo
```

addDataEventListener

This method registers a listener to monitor incoming data events from the CTM service. The user of this class would typically implement the `IDataListener` interface and then register.

Syntax

```
void addDataEventListener(IDataListener listener)
```

Parameters

- `listener`—instance of `IDataListener` that receives incoming data events.

For more information, refer to *IDataListener Interface* on page 88.

removeDataEventListener

This method unregisters a data event listener.

Syntax

```
void removeDataEventListener(IDataListener listener)
```

Parameters

- `listener`—registered listener that you want to remove.

addCommandCompleteListener

This method is used to register a listener to monitor command completion events from the CTM service. When any of the asynchronous methods defined in this interface are called, all registered `ICommandCompleteListeners` are notified when execution of the methods has finished. The user of this class would typically implement the `ICommandCompleteListener` interface and then register.

Syntax

```
void addCommandCompleteListener(ICommandCompleteListener listener)
```

Parameters

- `listener`—instance of `ICommandCompleteListener` that receives incoming data events.

For more information, refer to *ICommandCompleteListener Interface* on page 88.

removeCommandCompleteListener

This method unregisters a command event listener.

Syntax

```
void removeCommandCompleteListener(ICommandCompleteListener listener)
```

Parameters

- `listener`—registered listener that you want to remove.

readDispensableCashCounts

This method returns the number of each denomination that can be dispensed as a change. This count does not include any notes that are in the loader cassette.

Syntax

```
java.util.Set<CashUnit> readDispensableCashCounts()
```

Returns

This method returns a set of `CashUnit` objects sorted from the lowest coin to the highest note denomination.

Throws

- `java.lang.IllegalStateException`—if called when the current client has not been initialized.

readNonDispensableCashCounts

This method returns the number of each denomination that cannot be dispensed as change.

Syntax

```
java.util.Set<CashUnit> readNonDispensableCashCounts()
```

Returns

This method returns a set of `CashUnit` objects sorted from lowest coin to highest note denomination.

Throws

- `java.lang.IllegalStateException`—if called when the current client has not been initialized.

beginTransaction

This method signals the start of a new customer transaction to the CTM service. The POS application must call this method at the beginning of each customer transaction to reset any data that is accumulated in a prior transaction.

Syntax

```
int beginTransaction(java.long.String transactionID)
```

Parameters

- `transactionID`—transaction identifier that the POS application supplies. This identifier is included in the Advanced Cash Office (ACO) feed and can be used to synchronize CTM activity with POS transaction activity. If the `transactionID` is null, no identifier is used.

Returns

This method returns a unique sequence number, which is distinct from the supplied `transactionID` parameter.

Throws

- `java.long.IllegalStateException`—if this method is called while a customer transaction is already in progress or if this client instance is not connected to the CTM service.

endTransaction

This method signals the end of a customer transaction to the CTM service. The POS application must call this method at the end of each customer transaction.

Syntax

```
int endTransaction()
```

Return

This method returns a sequence number that was returned by the call to `beginTransaction`.

beginDeposit

This asynchronous method enables coin and note acceptance. After this command is completed, currency may be deposited in either the coin or note acceptor, and the event handlers of all registered `IDataListeners` are called for each deposit.

Syntax

```
void beginDeposit(long targetAmount, java.long.Object context)
```

Parameters

- `targetAmount`—the amount that must be tendered to satisfy the current transaction. After this amount is deposited, the CTM service automatically disables the coin and note acceptance. If the amount is not deposited or is set to zero, acceptance is enabled until `endDeposit` is called.
- `context`—optional user–supplied context object.

Throws

- `java.lang.IllegalException`—if called outside of a customer transaction.

For more information, refer to *IDataListener Interface* on page 88.

endDeposit

This asynchronous method disables coin or note acceptance. Deposit events may continue to arrive after this method is executed if the acceptors are handling deposited currency when the `endDeposit` request is issued.

Syntax

```
void endDeposit(java.lang.Object context)
```

Parameters

- `context`—optional user–supplied context object.

getErrorInfo

This method retrieves detailed textual information for a device error event.

Syntax

```
DeviceErrorDetails getErrorInfo(DeviceError errorInfo,
java.lang.String languageCode)
```

Parameters

- `errorInfo`—the error information to retrieve.
- `languageCode`—language code to which error details need to be translated. If this parameter is not provided, the default language code is used. The default language code is provided in the configuration.

Note: When a CashChanger device error event is received, a "BNR & BCR" device model is provided. Please set "Any" as the device model on the `errorInfo` parameter.

Returns

This method returns `DeviceErrorDetails` object that contains textual information that can be displayed to an end–user.

Throws

- `java.lang.IllegalStateException`—if called while the current client instance is not connected to the CTM service.

selfTest

This asynchronous method runs a diagnostic on specific devices.

Syntax

```
void selfTest(java.util.Set<DeviceInfo> devices, java.lang.Object
context)
```

Parameters

- `devices`—set of devices to be diagnosed.
- `context`—optional user–supplied context object.

Throws

- `java.lang.IllegalStateException`—if called while the current client instance is not connected to the CTM service.

Remarks

If a device error still exists after `selfTest` has completed, the device error information is sent to the application through the CTMErrorEvent object.

selfTestAll

This asynchronous method runs a diagnostic on all CTM devices.

Syntax

```
void selfTestAll(java.lang.Object context)
```

Parameters

- `context`—optional user–supplied context object.

Throws

- `java.lang.IllegalStateException`—if called while the current client instance is not connected to the CTM service.

Remarks

If a device error still exists after `selfTestAll` has completed, the device error information is sent to the application through the CTMErrorEvent object.

dispenseChange

This asynchronous method dispenses a specified amount of change.

Syntax

```
void dispenseChange(int amount, java.lang.Object context)
```

Parameters

- `amount`—the amount to be dispensed. The actual denominations that are dispensed are determined by the device–level components.
- `context`—optional user–supplied context object.

Throws

- `java.lang.IllegalArgumentException`—if an invalid amount is specified, such as a negative number.
- `java.lang.IllegalStateException`—if called outside of a customer transaction.

beginCashManagementTransaction

This method signals the start of a new cash management transaction to the CTM service. The POS application must call this method at the beginning of each cash management loan or pickup operation.

Syntax

```
int beginCashManagementTransaction(java.lang.String userId,
java.lang.String cashierId)
```

Parameters

- `userId`—login ID of the store operator for this terminal.
- `cashierId`—login ID of the store employee who is to carry out this cash management transaction (may be the same as `userId`).

Returns

This method returns a unique sequence number, which is distinct from the supplied `transactionID` parameter.

Throws

- `java.lang.IllegalStateException`—if this method is called while a customer or cash management transaction is in progress or if this client instance is not connected to the CTM service.

endCashManagementTransaction

This method signals the end of a cash management transaction to the CTM service. The POS application must call this method at the end of a cash management loan or pickup operation that results in a cash balance change.

Syntax

```
int endCashManagementTransaction()
```

Returns

This method returns a sequence number that was returned by the call to `beginTransaction`.

setLoaderCassetteCounts

This method sets the logical note counts of the note loader cassette. This method does not apply to cash devices that do not have a loader cassette.

Syntax

```
boolean setLoaderCassetteCounts(java.util.Set<CashUnit> cashCounts)
```

Parameters

- `cashCounts`—set of CashUnits that contains the counts of each denomination to be dispensed.

Returns

- true—if transaction succeeded.
- false—if transaction failed.

Throws

- `java.lang.IllegalStateException`—if called outside of a cash management transaction.
- `java.lang.IllegalArgumentException`—if an invalid amount is specified, such as negative or too large counts for the denomination.

getLoaderCassetteCounts

This method reads the number of each note denomination in the loader cassette.

Syntax

```
java.util.Set<CashUnit> getLoaderCassetteCounts()
```

Returns

This method returns a set of `CashUnit` objects sorted from the lowest coin to the highest note denomination.

Throws

- `java.lang.IllegalStateException`—if called when the current client has not been initialized.

dispenseCash

This asynchronous method dispenses cash by denomination.

Syntax

```
void dispenseCash(java.util.Set<CashUnit> cashCounts,
java.lang.Object context)
```

Parameters

- `cashCounts`—set of `CashUnits` that contains the counts of each denomination to be dispensed.
- `context`—optional user–supplied context object.

Throws

- `java.lang.IllegalArgumentException`—if an invalid amount is specified, such as a negative number.
- `java.lang.IllegalStateException`—if called outside of a cash management transaction.

transferLoaderToCashbox

This asynchronous method transfers notes from loader cassette to cash box if supported by the device configuration. This method does nothing if the note recycler does not utilize a loader cassette.

Syntax

```
void transferLoaderToCashbox(java.lang.Object context)
```

Parameters

- `context`—optional user–supplied context object.

Throws

- `java.lang.IllegalStateException`—if called when the current client is not initialized.

transferToCashbox

This asynchronous method transfers a specified number of notes from the recyclers to the cash box if supported by the device configuration. This method does nothing if a note recycler is not present.

Syntax

```
void transferToCashbox(java.util.Set<CashUnit> cashCounts,
java.lang.Object context)
```

Parameters

- `cashCounts`—set of `CashUnits` that contains the counts of each denomination to be dispensed.
- `context`—optional user–supplied context object.

Throws

- `java.lang.IllegalArgumentException`—if an invalid amount is specified, such as a negative number.
- `java.lang.IllegalStateException`—if called outside of a cash management transaction.

beginRefill

This asynchronous method enables coin or note replenishment through the acceptors. This method does nothing for devices that do not support this capability. Replenishment differs from the normal `beginDeposit` method in that this functionality is utilized for refilling the recycler devices. As with the `beginDeposit` method, `CtmDataEvents` are generated while coins or notes are deposited.

However, there will not be an event for each deposited denomination due to the potentially large number of `CtmDataEvents` that could be generated in the refilling cycle. Ensure that if this method is called during a customer transaction, care must be taken to not credit funds deposited in replenishment mode to the current customer transaction.

Syntax

```
void beginRefill(java.long.Object context)
```

Parameters

- `context`—optional user–supplied context object.

Throws

- `java.lang.IllegalStateException`—if called outside of a cash management transaction.

endRefill

This asynchronous method ends cash replenishment operation. Due to timing, it is possible for additional `CtmDataEvents` to be received after this method is executed. The command result indicates that no additional deposit events should arrive.

Syntax

```
void endRefill(java.lang.Object context)
```

Parameters

- `context`—optional user–supplied context object.

resetDispensableCoinCounts

This method logically empties the coin dispenser or recycler containers. This method does nothing for cash devices that do not permit resetting of dispensable counts.

Syntax

```
boolean resetDispensableCoinCounts()
```

Returns

- true—if transaction succeeded.
- false—if transaction failed.

Throws

- `java.lang.IllegalStateException`—if called outside of a cash management transaction.

resetNonDispensableCoinCounts

This method logically empties the non–dispensable containers of the Bulk Coin Recycler (BCR) device, such as acceptors, overflow, cash box, and so forth.

Syntax

```
boolean resetNonDispensableCoinCounts()
```

Returns

- true—if transaction succeeded.
- false—if transaction failed.

Throws

- `java.lang.IllegalStateException`—if called outside of a cash management transaction.

resetNonDispensableNoteCounts

This method logically empties the non–dispensable containers of the Bulk Note Recycler (BNR) device, such as acceptors, overflow, cash box, and so forth.

Syntax

```
boolean resetNonDispensableNoteCounts()
```

Returns

- true—if transaction succeeded.
- false—if transaction failed.

Throws

- `java.lang.IllegalStateException`—if called outside of a cash management transaction.

setDispensableCounts

This method sets the logical counts of recyclers or dispensers in the BCR device. Setting dispensable counts is not applicable to the BNR device or to cash devices that do not allow resetting of dispensable counts.

Syntax

```
boolean setDispensableCounts(java.util.Set<CashUnit> cashCounts)
```

Parameters

- `cashCounts`—set of `CashUnits` that contains the counts of each denomination to set.

Throws

- `java.lang.IllegalStateException`—if called outside of a cash management transaction.
- `java.lang.IllegalArgumentException`—if an invalid amount is specified, such as negative or too large counts for the denomination.

getPurgedStatus

This method returns an indication whether one or more notes have been diverted to the purge bin location.

Syntax

```
boolean getPurgedStatus()
```

Returns

- true—if notes are present in the purge container.
- false—if no notes are present in the purge container.

clearPurgedStatus

This method resets the purge status to indicate that all notes have been removed from the purge container.

Syntax

```
boolean clearPurgedStatus()
```

Returns

- true—if notes are present in the purge container.
- false—if no notes are present in the purge container.

getDispensableCapacities

This method obtains the device storage capacity of each dispensable denomination.

Syntax

```
java.util.Set<CashUnit> getDispensableCapacities()
```

Returns

This method returns a set of `CashUnit` objects sorted from the lowest coin to the highest note denomination where the counts indicate the capacity of each denomination.

Throws

- `java.lang.IllegalStateException`—if called when the current client has not been initialized.

getNonDispensableNoteCapacity

This method obtains the non–dispensable storage capacity of the BNR device.

Syntax

```
int getNonDispensableNoteCapacity()
```

Returns

This method returns the maximum note storage count.

Throws

- `java.lang.IllegalStateException`—if called when the current client has not been initialized.

getNonDispensableCoinCapacity

This method obtains the non–dispensable storage capacity of the BCR device.

Syntax

```
int getNonDispensableCoinCapacity()
```

Returns

This method returns the maximum coin storage count.

Throws

- `java.lang.IllegalStateException`—if called when the current client has not been initialized.

getLoaderCassetteCapacity

This method obtains the estimated maximum number of notes that the loader cassette can contain. Zero is returned if the device does not have a loader cassette.

Syntax

```
int getLoaderCassetteCapacity()
```

Returns

This method returns the loader cassette capacity.

Throws

- `java.lang.IllegalStateException`—if called when the current client has not been initialized.

com.ncr.ssc.ctm.events

The com.ncr.ssc.ctm.events package includes the following interfaces and classes:

- Interface
  - `ICommandCompleteListener`
  - `IDataListener`
- Class
  - `CtmCommandCompleteEvent`
  - `CtmDataEvent`
  - `CtmEvent`

ICommandCompleteListener Interface

The `ICommandCompleteListener` interface is the event listener for when an asynchronous command has completed.

Syntax

```
public interface ICommandCompleteListener
extends java.util.EventListener
```

ICommandCompleteListener Methods

The `ICommandCompleteListener` interface contains the `commandCompleted` method.

commandCompleted

This method is called when an asynchronous command has completed.

Syntax

```
void commandCompleted(CtmCommandCompleteEvent event)
```

Parameters

- `event`—completion event object that contains result information for the executed command.

For more information, refer to *CtmCommandCompleteEvent Class* on the facing page.

IDataListener Interface

The `IDataListener` interface listens to incoming deposit events.

Syntax

```
public interface IDataListener
extends java.util.EventListener
```

IDataListener Methods

The `IDataListener` interface contains the `dataOccurred` method.

dataOccurred

This method is called whenever a coin or note deposit has occurred.

Syntax

```
void dataOccurred(CtmDataEvent event)
```

Parameters

- `event`—descriptor for this event.

For more information, refer to *CtmDataEvent Class* on page 91.

CtmCommandCompleteEvent Class

The `CtmCommandCompleteEvent` class listens to a command complete event.

Syntax

```
public abstract class CtmCommandCompleteEvent
extends CtmEvent
```

CtmCommandCompleteEvent Constructor

The `CtmCommandCompleteEvent` class contains the `CtmCommandCompleteEvent` constructor.

Syntax

```
public CtmCommandCompleteEvent()
```

CtmCommandCompleteEvent Methods

The following are the different methods classified in the `CtmCommandCompleteEvent` class:

- `getDeviceInfo`
- `setErrorInfo`
- `getCommandResults`
- `setCommandResults`

getDeviceInfo

This method obtains the device information.

Syntax

```
public DeviceInfo getDeviceInfo()
```

Returns

This method returns null if no error condition is associated with the command. If an error occurred, this method returns the `CtmErrorEvent` object that describes the command error condition.

setErrorInfo

This method sets the error information.

Syntax

```
public void setErrorInfo(DeviceInfo deviceInfo)
```

Parameters

- `deviceInfo`—the new error information.

getCommandResults

This method obtains the command results.

Syntax

```
public CtmCommandResultInfo getCommandResults()
```

Returns

This method returns the result data for this command. The result may be null if no result data is associated with the command or if an error occurred when executing the command. This would typically be cast by the caller to whatever type is expected as the command result.

setCommandResults

This method sets the command results.

Syntax

```
public void setCommandResults(CtmCommandResultInfo commandResults)
```

Parameters

- `commandResults`— the new command results.

Inherited Methods

The following are the methods that the `CtmCommandCompleteEvent` class inherited from the `com.ncr.ssc.ctm.events.CtmEvent` class:

- `getSequenceNumber`
- `getWhen`
- `setSequenceNumber`
- `setWhen`

The following are the methods that the `CtmCommandCompleteEvent` class inherited from the `java.lang.Object` class:

- `clone`
- `equals`

- `finalize`
- `getClass`
- `hashCode`
- `notify`
- `notifyAll`
- `toString`
- `wait`

CtmDataEvent Class

The `CtmDataEvent` class is the CTM data event object.

Syntax

```
public final class CtmDataEvent
extends CtmEvent
```

CtmDataEvent Constructor

The `CtmDataEvent` class contains the `CtmDataEvent` constructor.

Syntax

```
public CtmDataEvent()
```

CtmDataEvent Methods

The following methods are classified in the CtmDataEvent class.

- `getStatus`
- `getDeposited`
- `setStatus`
- `setDeposited`

getStatus

This method returns the value of the Status identifier.

Syntax

```
public int getStatus()
```

Returns

This method returns the status identifier, if there is any.

getDeposited

This method obtains the amount that was deposited for the current event.

Syntax

```
public CashUnit getDeposited()
```

Returns

This method returns the deposited amount for the current event.

setStatus

This method sets the value of the Status identifier.

Syntax

```
public void setStatus(int status)
```

Parameters

- `status`—the new status.

setDeposited

This method sets the deposited amount and casts it to a set of `CashUnit` objects.

Syntax

```
public void setDeposited(CashUnit deposited)
```

Parameters

- `deposited`—the new deposited amount.

Inherited Methods

The following are the methods that the `CtmDataEvent` class inherited from the `com.ncr.ssc.ctm.events.CtmEvent` class:

- `getSequenceNumber`
- `getWhen`
- `setSequenceNumber`
- `setWhen`

The following are the methods that the `CtmDataEvent` class inherited from the `java.lang.Object` class:

- `clone`
- `equals`
- `finalize`
- `getClass`
- `hashCode`
- `notify`
- `notifyAll`
- `toString`
- `wait`

CtmEvent Class

The `CtmEvent` class is the base event class for all cash tender module events. The direct known subclasses of the `CtmEvent` class are the `CtmCommandCompleteEvent` and `CtmDataEvent` classes.

Syntax

```
public abstract class CtmEvent
extends java.lang.Object
```

CtmEvent Constructor

The `CtmEvent` class contains the `CtmEvent` constructor.

Syntax

```
public CtmEvent()
```

CtmEvent Methods

The following methods are classified in the `CtmEvent` class:

- `getSequenceNumber`
- `setSequenceNumber`
- `getWhen`
- `setWhen`

getSequenceNumber

This method obtains the sequence number of the current event.

Syntax

```
public long getSequenceNumber()
```

Returns

This method returns the sequential identifier of the current event.

setSequenceNumber

This method sets the sequence number of the current event.

Syntax

```
public void setSequenceNumber(long sequenceNumber)
```

Parameters

- `sequenceNumber`—the new sequence number of the current event.

getWhen

This method obtains the current system time.

Syntax

```
public long getWhen()
```

Returns

This method returns the current system time in milliseconds.

setWhen

This method sets the current time to display when the deposit event occurred.

Syntax

```
public void setWhen(long when)
```

Parameters

- `when`—the new time.

Inherited Methods

The following are the methods that the `CtmEvent` class inherited from the `java.lang.Object` class:

- `clone`
- `equals`
- `finalize`
- `getClass`
- `hashCode`
- `notify`
- `notifyAll`
- `toString`
- `wait`

com.ncr.ssc.ctm.result

The com.ncr.ssc.ctm.result package includes the following classes and enumerations.

- Class
  - `CtmCashCountResultInfo`
  - `CtmCommandResultInfo`
  - `CtmConfigResultInfo`
  - `CtmLongResultInfo`
- Enumeration
  - `CtmCommandTypes`

CtmCashCountResultInfo Class

The `CtmCashCountResultInfo` class obtains the asynchronous result of a dispense or transfer command.

Syntax

```
public class CtmCashCountResultInfo
extends CtmCommandResultInfo
```

CtmCashCountResultInfo Constructor

The `CtmCashCountResultInfo` class contains the `CtmCashCountResultInfo` constructor.

Syntax

```
public CtmCashCountResultInfo(CtmCommandTypes commandType)
```

Parameters

- `commandType`—specifies the type for this command.

CtmCashCountResultInfo Methods

The following methods are classified in the `CtmCashCountResultInfo` class:

- `getCashCounts`
- `setCashCounts`

getCashCounts

This method returns the cash counts that are set by a cashier or cash–handling personnel.

Syntax

```
public java.util.Set<CashUnit> getCashCounts()
```

Returns

This method returns a set of cash counts.

setCashCounts

This method sets the cash counts by denomination.

Syntax

```
public void setCashCounts(java.util.Set<CashUnit> cashCounts)
```

Parameters

- `cashCounts`—the new cash count.

Inherited Methods

The following are the methods that the `CtmCashCountResultInfo` class inherited from the `com.ncr.ssc.ctm.result.CtmCommandResultInfo` class:

- `getCommandType`
- `getContext`
- `getCtmTransactionId`
- `getPosTransactionId`
- `getSuccess`
- `setCommandType`
- `setContext`
- `setCtmTransactionId`
- `setPosTransactionId`
- `setSuccess`

The following are the methods that the `CtmCashCountResultInfo` class inherited from the `java.lang.Object` class:

- `clone`
- `equals`
- `finalize`
- `getClass`
- `hashCode`
- `notify`
- `notifyAll`
- `toString`
- `wait`

CtmCommandResultInfo Class

The `CtmCommandResultInfo` class encapsulates the command result of an asynchronous method. The direct known subclasses of this class are the `CtmCashCountResultInfo` and `CtmLongResultInfo` classes.

Syntax

```
public class CtmCommandResultInfo
extends java.lang.Object
```

CtmCommandResultInfo Constructor

The `CtmCommandResultInfo` class contains the `CtmCommandResultInfo` constructor.

Syntax

```
public CtmCommandResultInfo(CtmCommandTypes commandType)
```

Parameters

- `commandType`—specifies the type for this command.

CtmCommandResultInfo Methods

The following methods are classified in the `CtmCommandResultInfo` class:

- `getCommandType`
- `setCommandType`
- `getContext`
- `setContext`
- `getSuccess`
- `setSuccess`
- `setPosTransactionId`
- `setCtmTransactionId`
- `getPosTransactionId`
- `getCtmTransactionId`

getCommandType

This method returns the type of command if a command is set to perform.

Syntax

```
public CtmCommandTypes getCommandType()
```

Returns

This method returns the type identifier for this command. The type identifier is used to distinguish different return types to caller.

setCommandType

This method sets the command type of the command to perform.

Syntax

```
public void setCommandType(CtmCommandTypes commandType)
```

Parameters

- `commandType`—the new command type.

getContext

This method returns the supplied context object that is associated with the command that provided this result.

Syntax

```
public java.lang.Object getContext()
```

Returns

This method returns a user–supplied context object associated with the command that is associated with this result.

setContext

This method sets the supplied context object associated with the command that provided this result.

Syntax

```
public void setContext(java.lang.Object context)
```

Parameters

- `context`—the new context.

getSuccess

This method returns the status of the command completion, whether it is a success or a failure.

Syntax

```
public java.lang.Boolean getSuccess()
```

Returns

- true—if the command succeeded.
- false—if the command failed.

setSuccess

This method sets the status of the command completion, whether it is a success or a failure. It uses the true and false values to indicate the success status.

Syntax

```
public void setSuccess(java.lang.Boolean success)
```

Parameters

- `success`—the new value whether the command is a success or a failure.

setPosTransactionId

This method sets the POS transaction identifier that the POS provides. This identifier must be set to perform CTM events.

Syntax

```
public void setPosTransactionId(java.lang.String posTransactionId)
```

Parameters

- `posTransactionId`—the transaction identifier that the POS provides.

setCtmTransactionId

This method sets the CTM transaction identifier. This CTM transaction ID is required to perform CTM events.

Syntax

```
public void setCtmTransactionId(int ctmTransactionId)
```

Parameters

- `ctmTransactionId`—the transaction ID to use for performing cash management transactions.

getPosTransactionId

This method returns the POS transaction ID.

Syntax

```
public java.lang.String getPosTransactionId()
```

Returns

This method returns the POS transaction identifier that was supplied as an argument to `beginTransaction`.

getCtmTransactionId

This method returns the CTM transaction ID.

Syntax

```
public int getCtmTransactionId()
```

Returns

This method returns the sequence number that was returned by the last call to `beginTransaction` or `beginCashManagementTransaction`.

Inherited Methods

The following are the methods that the `CtmCommandResultInfo` class inherited from the `java.lang.Object` class:

- `clone`
- `equals`
- `finalize`

- getClass
- hashCode
- notify
- notifyAll
- toString
- wait

CtmConfigResultInfo

The CtmConfigResultInfo class obtains the results of the getConfig command.

Syntax

```
public final class CtmConfigResultInfo
extends java.lang.Object
```

CtmConfigResultInfo Constructor

The CtmConfigResultInfo class contains the CtmConfigResultInfo constructor.

Syntax

```
public CtmConfigResultInfo()
```

CtmConfigResultInfo Methods

The following methods are classified in the CtmConfigResultInfo class:

- getConfigData
- setConfigData

getConfigData

This method returns a series of key or value configuration strings for static configuration values.

Syntax

```
public java.util.Map<java.lang.String,java.lang.String> getConfigData
()
```

Returns

This method returns configuration data as string key or value pairs.

setConfigData

This method sets the configuration data.

Syntax

```
public void setConfigData
(java.util.Map<java.lang.String,java.lang.String> configData)
```

Parameters

- configData—the new configuration data.

Inherited Methods

The following are the methods that the `CtmConfigResultInfo` class inherited from the `java.lang.Object` class:

- `clone`
- `equals`
- `finalize`
- `getClass`
- `hashCode`
- `notify`
- `notifyAll`
- `toString`
- `wait`

CtmLongResultInfo

The `CtmLongResultInfo` class obtains the asynchronous result of a command that returns a long value result.

Syntax

```
public class CtmLongResultInfo
extends CtmCommandResultInfo
```

CtmLongResultInfo Constructor

The `CtmLongResultInfo` class contains the `CtmLongResultInfo` constructor.

Syntax

```
public CtmLongResultInfo(CtmCommandTypes commandType)
```

Parameters

- `commandType`—specifies the type for this command.

CtmLongResultInfo Methods

The following methods are classified in the `CtmLongResultInfo` class:

- `getLongResultInfo`
- `setLongResultInfo`

getLongResultInfo

This method returns the result of the command.

Syntax

```
public long getLongResult()
```

Returns

This method returns the long result of the current command.

setLongResultInfo

This method sets the result of the command in long data type.

Syntax

```
public void setLongResult(long longResult)
```

Parameters

- `longResult`—the new long result.

Inherited Methods

The following are the methods that the `CtmLongResultInfo` class inherited from the `com.ncr.ssc.ctm.result.CtmCommandResultInfo` class:

- `getCommandType`
- `getContext`
- `getCtmTransactionId`
- `getPosTransactionId`
- `getSuccess`
- `setCommandType`
- `setContext`
- `setCtmTransactionId`
- `setPosTransactionId`
- `setSuccess`

The following are the methods that the `CtmLongResultInfo` class inherited from the `java.lang.Object` class:

- `clone`
- `equals`
- `finalize`
- `getClass`
- `hashCode`
- `notify`
- `notifyAll`
- `toString`
- `wait`

CtmCommandTypes Enum

The `CtmCommandTypes` enumeration provides the enumeration values for the CTM command types. The following is the syntax for using this enumeration:

```
public enum CtmCommandTypes
extends java.lang.Enum<CtmCommandTypes>
```

CtmCommandTypes Constants

The following constants are classified in the `CtmCommandTypes` enumeration:

- `CMD_DISPENSE_CHANGE`
- `CMD_DISPENSE_CASH`
- `CMD_TRANSFER_LOADER_TO_CASHBOX`
- `CMD_TRANSFER_TO_CASHBOX`
- `CMD_BEGIN_DEPOSIT`
- `CMD_END_DEPOSIT`
- `CMD_BEGIN_REFILL`
- `CMD_END_REFILL`

CMD_DISPENSE_CHANGE

This constant is the CTM event used to dispense change.

Syntax

```
public static final CtmCommandTypes CMD_DISPENSE_CHANGE
```

CMD_DISPENSE_CASH

This constant is the CTM event used to dispense cash by denomination.

Syntax

```
public static final CtmCommandTypes CMD_DISPENSE_CASH
```

CMD_TRANSFER_LOADER_TO_CASHBOX

This constant is the CTM event used to transfer money from the loader device to the cashbox.

Syntax

```
public static final CtmCommandTypes CMD_TRANSFER_LOADER_TO_CASHBOX
```

CMD_TRANSFER_TO_CASHBOX

This constant is the CTM event used to transfer money to the cashbox.

Syntax

```
public static final CtmCommandTypes CMD_TRANSFER_TO_CASHBOX
```

CMD_BEGIN_DEPOSIT

This constant is the CTM event used to enable the cash or coin acceptor devices to begin deposit.

Syntax

```
public static final CtmCommandTypes CMD_BEGIN_DEPOSIT
```

CMD_END_DEPOSIT

This constant is the CTM event used to disable the cash or coin acceptors to end the deposit.

Syntax

```
public static final CtmCommandTypes CMD_END_DEPOSIT
```

CMD_BEGIN_REFILL

This constant is the CTM event used to enable cash or coin acceptors to refill money in the recyclers.

Syntax

```
public static final CtmCommandTypes CMD_BEGIN_REFILL
```

CMD_END_REFILL

This constant is the CTM event used to disable cash or coin acceptors to stop the refill process.

Syntax

```
public static final CtmCommandTypes CMD_END_REFILL
```

CtmCommandTypes Methods

The following methods are classified in the `CtmCommandTypes` enumeration:

- `values`
- `valueOf`

values

This method returns an array containing the constants of the current enumeration type. This method may be used to iterate over the constants as follows:

```
for (CtmCommandTypes c : CtmCommandTypes.values())System.out.println(c);
```

Syntax

```
public static CtmCommandTypes[] values()
```

Returns

This method returns the array that contains the constants of the current enumeration type.

valueOf

This method returns the enumeration constant of this type with the specified name. The string must match exactly an identifier used to declare an enumeration constant in this type. Extraneous white space characters are not permitted.

Syntax

```
public static CtmCommandTypes valueOf(java.lang.String name)
```

Parameters

- `name`—the name of the enumeration constant to be returned.

Returns

This method returns the enumeration constant with the specified name.

Throws

- `java.lang.IllegalArgumentException`—if this enumeration type has no constant with the specified name.
- `java.lang.NullPointerException`—if the argument is null.

Chapter 5: Cash Tender Module Client C API

Overview

This chapter describes the C API used by the Cash Tender Module (CTM) service to perform the different Cash Tender operations. The different functions and data structures of the C API are classified in the following modules:

- Client Initialization
- Device Errors and System Messages
- Device Events
- Transactions
- Cash Dispensing
- Cash Managements
- Cash Accepting

Each of these modules discusses the different functions, data structures, enumerations, and type definitions (typedefs) that the CTM client uses to connect to the CTM service.

Client Initialization

The Client Initialization module lists the functions and return codes necessary to initialize the CTM API and connect it to the CTM service. It contains the following:

- Enumerations
- Functions

Enumerations

The Client Initialization module includes the `CTMInitializationResult` enumeration.

CTMInitializationResult

The `CTMInitializationResult` enumeration lists the possible return values for the `ctm_initialize()` function. The following table describes the different enumerators of the `CTMInitializationResult` enumeration:

Enumerator	Description
`CTM_INIT_SUCCESS`	Indicates that the CTM client has successfully connected to the CTM service.
`CTM_INIT_ERROR_RUNTIME`	Indicates that a standard C library call failed and populated the error number of the `errno.h` file.
`CTM_INIT_ERROR_INVALID_SERVICE_URL`	Indicates that the CTM client could not parse the service URL parameter.
`CTM_INIT_ERROR_COULD_NOT_CONNECT`	Indicates that the CTM client could not connect to the CTM service.
`CTM_INIT_ERROR_ALREADY_INITIALIZED`	Indicates that the CTM client is already initialized.
`CTM_INIT_ERROR_REJECTED_BY_SERVICE`	Indicates that the CTM service rejected the CTM client connection request.
`CTM_INIT_ERROR_AT_LEAST_ONE_DEVICE_NOT_READY`	Indicates that at least one device cannot be claimed.
`CTM_INIT_ERROR_MISSING_SYSTEM_MESSAGE_EVENT_HANDLER`	Indicates that you must register a callback to the `ctm_add_system_message_event_handler()` function.
`CTM_INIT_ERROR_MISSING_DEVICE_ERROR_EVENT_HANDLER`	Indicates that you must register a callback to the `ctm_add_device_error_event_handler()` function.

Functions

The Client Initialization module includes the following functions:

- `ctm_get_config()`
- `ctm_initialize()`
- `ctm_uninitialize()`

ctm_get_config()

The `ctm_get_config()` function is used to obtain CTM configuration information.

Syntax

```
struct CTMGetConfigResult ctm_get_config()
```

Returns

This function returns a key value pair containing information on the CTM configuration.

ctm_initialize()

The `ctm_initialize()` function is used to initialize the CTM client. When you use this function, a blocking call attempts to connect to the CTM service at the specified URL. If the CTM client is successfully initialized and the connection to the CTM service is successful, the CTM client starts calling existing callbacks while the corresponding events occur.

Syntax

```
enum CTMInitializationResult ctm_initialize(const char *
szServiceURL)
```

Parameters

- `szServiceURL`—the URL of the CTM service.

  Example: `ctm://localhost:3636`

Returns

This function returns one of the values from the `CTMInitializationResult` enumeration.

ctm_uninitialize()

The `ctm_uninitialize()` function is used to uninitialize the CTM client. This function stops the connection between the CTM client and the CTM service.

Syntax

```
void ctm_uninitialize()
```

Device Errors and System Messages

This module lists the asynchronous error events that may occur and the functions that provide localized instructions to help an attendant or cashier recover from these error events. It contains the following:

- Data structures
- Typedef
- Enumerations
- Functions

Data Structures

The module for device errors and system messages include the following data structures:

- `CTMDeviceError`
- `CTMDeviceErrorSet`
- `CTMDeviceTestResult`
- `CTMDeviceErrorDetails`
- `CTMDeviceInfo`

CTMDeviceError

The `CTMDeviceError` data structure lists the information on the cash device error.

The following table displays the data fields included in the `CTMDeviceError` data structure:

Data Field	Data Type	Description
`sDeviceInfo`	struct CTMDeviceInfo	Refers to the device information.
`iResultCode`	int32_t	Indicates whether a certain device is functioning well.
`iExtendedResultCode`	int32_t	Additional indicator to determine whether a device is functioning.
`piDenomination`	int32_t *	Refers to the denomination or bin associated with a cash device error.
`piChangeDue`	int32_t *	Refers to the value of the change due.

CTMDeviceErrorSet

The `CTMDeviceErrorSet` data structure lists the information of a set of device errors.

The following table displays the data fields included in the `CTMDeviceErrorSet` data structure:

Data Field	Data Type	Description
`iCount`	size_t	Refers to the number of device errors in a set.
`pstDeviceErrors`	struct `CTMDeviceError`	Refers to the information of the device errors in the set.

CTMDeviceTestResult

The `CTMDeviceTestResult` data structure lists all the fields for the possible return values of the following functions:

- `ctm_test_device()`
- `ctm_test_all_devices()`

The following table displays the data fields included in the `CTMDeviceTestResult` data structure:

Data Field	Data Type	Description
`stDeviceErrorSet`	struct `CTMDeviceErrorSet`	Contains the result of the device test, which determines whether the device or devices are functioning properly.
`error`	enum `CTMDeviceTestError`	Indicates whether the device test is successful.

CTMDeviceErrorDetails

The `CTMDeviceErrorDetails` data structure lists all the fields that are used to resolve a device error.

The following table displays the data fields included in the `CTMDeviceErrorDetails` data structure:

Data Field	Data Type	Description
`szTitle`	char *	Refers to the title description of the device error.
`szSubtitle`	char *	Refers to the subtitle description of the device error.

Data Field	Data Type	Description
szInstructionalText	char *	Refers to the set of textual instructions on how to resolve a device error.
szImageFilename	char *	Refers to the location of an image file that adds information on how to resolve a device error.
szVideoFilename	char *	Refers to the location of a video file that shows how to resolve a device error.
szErrorCode	char *	Refers to the error code of the device error.

CTMDeviceInfo

The CTMDeviceInfo data structure lists the fields used for the device information.

The following table displays the data fields included in the CTMDeviceInfo data structure:

Data Field	Data Type	Description
eDeviceType	enum CTMDeviceType	Refers to the type of the device.
szDeviceModel	char *	Refers to the model of the device type.
szDeviceSubModel	char *	Refers to the sub–model of the device type.
piDeviceID	int32_t *	Distinguishes between multiple devices of the same type.

Typedef

The module for device errors and system messages includes the CTMDeviceErrorCallback typedef.

CTMDeviceErrorCallback

The CTMDeviceErrorCallback typedef is the callback used for device error events.

Syntax

```
typedef void(* CTMDeviceErrorCallback)(struct CTMEventInfo, struct
CTMDeviceError)
```

Enumerations

The module for device errors and system messages includes the following enumerations:

- CTMDeviceTestError
- CTMDeviceType
- CTMErrorDetailsResult

CTMDeviceTestError

The `CTMDeviceTestError` enumeration lists the possible return values of the following functions:

- `ctm_test_all_devices()`
- `ctm_test_device()`

The following table displays the enumerators of the `CTMDeviceTestError` enumeration:

Enumerator	Description
`CTM_DEVICE_TEST_SUCCESS`	Indicates that the CTM service successfully tested the devices.
`CTM_DEVICE_TEST_ERROR_UNKNOWN_DEVICE_TYPE`	Indicates that an invalid identifier was provided for a device test request.
`CTM_DEVICE_TEST_ERROR_NOT_CONNECTED`	Indicates that the CTM client is not connected to the CTM service.

CTMDeviceType

The `CTMDeviceType` enumeration lists the possible types of a CTM device.

The following table displays the enumerators of the `CTMDeviceType` enumeration:

Enumerator	Description
`CTM_DEVICETYPE_CASHCHANGER`	Indicates that the device is a cash changer.
`CTM_DEVICETYPE_CASHACCEPTOR`	Indicates that the device is a cash acceptor.
`CTM_DEVICETYPE_COINACCEPTOR`	Indicates that the device is a coin acceptor.
`CTM_DEVICETYPE_OTHER`	Indicates that the device is another type of device that is not a cash changer, a cash acceptor, or a coin acceptor.

CTMErrorDetailsResult

The `CTMErrorDetailsResult` enumeration provides a list of possible return values for the `ctm_get_error_details()` function.

The following table displays the enumerators of the `CTMErrorDetailsResult` enumeration:

Enumerator	Description
`CTM_ERROR_DETAILS_SUCCESS`	Indicates that the CTM service successfully retrieved the error details.
`CTM_ERROR_DETAILS_UNKNOWN_DEVICE_TYPE`	Indicates that an invalid identifier was provided for a device error details request.
`CTM_ERROR_DETAILS_NOT_INITIALIZED`	Indicates that the CTM client is not initialized.
`CTM_ERROR_DETAILS_ERROR_DETAILS_STRUCT_IS_NULL`	Indicates that the CTM service returned a null pointer.
`CTM_ERROR_DETAILS_ERROR_DETAILS_STRUCT_CONTAINS_DATA`	Indicates that there is an error in the data contained the data structure.
`CTM_ERROR_DETAILS_ERROR_STRUCT_IS_INVALID`	Indicates that the data structure is invalid.
`CTM_ERROR_DETAILS_NO_DATA_RECEIVED`	Indicates that the CTM client did not receive any data.
`CTM_ERROR_DETAILS_RUNTIME_ERROR`	Indicates that a standard C library call failed and populated the error number of the `errno.h` file.

Functions

The module for device errors and system messages includes the following functions:

- `ctm_add_device_error_event_handler()`
- `ctm_free_error_details()`
- `ctm_get_error_details()`
- `ctm_test_all_devices()`
- `ctm_test_device()`

ctm_add_device_error_event_handler()

The `ctm_add_device_error_event_handler()` function is used to set an event handler callback for the device error event.

Syntax

```
void ctm_add_device_error_event_handler(CTMDeviceErrorCallback)
```

ctm_free_error_details()

The `ctm_free_error_details()` function is used to free the resources that the `ctm_get_error_details()` function used.

Syntax

```
void ctm_free_error_details(struct CTMDeviceErrorDetails *
psErrDetails)
```

ctm_get_error_details()

The `ctm_get_error_details()` function is used to retrieve detailed textual information for a device error event.

Syntax

```
enum CTMErrorDetailsResult ctm_get_error_details(const struct CTM
DeviceError sDeviceError, const char* szLocaleID, struct
CTMDeviceErrorDetails* psErrDetails)
```

Parameters

- `sDeviceError`—the device error that occurred.
- `szLocaleID`—the language used in retrieving the error.
- `psErrDetails`—the values of the retrieved error details.

Note: When a CashChanger device error event is received, a "BNR & BCR" device model is provided. Please set "Any" as the device model on the `psErrDetails` parameter.

Returns

This function returns the detailed textual information for the specified device error.

ctm_test_all_devices()

The `ctm_test_all_devices()` function is used to run a diagnostic on all CTM devices.

Syntax

```
struct CTMDeviceTestResult ctm_test_all_devices()
```

ctm_test_device()

The `ctm_test_device()` function is used to run a diagnostic on a specific device.

Syntax

```
struct CTMDeviceTestResult ctm_test_device(struct CTMDeviceInfo
sDeviceInfo)
```

Parameters

- `sDeviceInfo`—refers to the information of the device to test.

Device Events

The Device Events module lists the data that the CTM API passes to the CTM callback functions during asynchronous device events.

Data Structures

The Device Events module includes the following data structures:

- `CTMEventInfo`
- `CTMAcceptEvent`

CTMEventInfo

The `CTMEventInfo` data structure includes the field used to store the exact time when a particular event occurred.

The following table displays the data fields included in the `CTMEventInfo` data structure:

Data Field	Data Type	Description
`lTimestamp`	time_t	Refers to the time when the event occurred.

CTMAcceptEvent

The `CTMAcceptEvent` data structure lists all the fields used for cash accept events.

The following table displays the data fields included in the `CTMAcceptEvent` data structure:

Data Field	Data Type	Description
`u32Amount`	uint32_t	Refers to the amount accepted.
`stCashUnit`	struct `CTMCashUnit`	Refers to the CashUnit accepted, including the denomination, count, and currency.

Transactions

The Transactions module lists the functions used to define transactions for reporting purposes.

Any request that accepts, dispenses, or transfers cash must have a transaction associated with it for reporting purposes. To do associate a transaction for cash requests, you must create or assign a transaction ID through the `ctm_begin_transaction()` or `ctm_begin_cash_management_transaction()` functions before performing the request. When a customer or cash management transaction is finished, you must close the transaction by calling the `ctm_end_transaction()` function.

This module contains the following:

- Data structures
- Enumerations
- Functions

Data Structures

The Transactions module includes the `CTMBeginTransactionResult` data structure.

CTMBeginTransactionResult

The `CTMBeginTransactionResult` data structure lists all the fields for the results of a begin transaction event.

The following table displays the data fields included in the `CTMBeginTransactionResult` data structure:

Data Field	Data Type	Description
`szTransactionID`	char *	Refers to the transaction ID of the begin transaction event.
`error`	enum `CTMBeginTransactionError`	Indicates whether there is an error during the begin transaction event.

Enumerations

The Transactions module includes the following enumerations:

- `CTMBeginTransactionError`
- `CTMEndTransactionResult`

CTMBeginTransactionError

The `CTMBeginTransactionError` enumeration lists the possible return values of the following functions:

- `ctm_begin_customer_transaction()`
- `ctm_begin_cash_management_transaction()`

The following table displays the enumerators of the `CTMBeginTransactionError` enumeration:

Enumerator	Description
`CTM_BEGIN_TRX_SUCCESS`	Indicates that the transaction has started.
`CTM_BEGIN_TRX_ERROR_ALREADY_IN_PROGRESS`	Indicates that you must end the previous transaction first before starting a new transaction.
`CTM_BEGIN_TRX_ERROR_NOT_CONNECTED`	Indicates that the CTM client is not connected to the CTM service.

CTMEndTransactionResult

The `CTMEndTransactionResult` enumeration lists the possible return values of the `ctm_end_transaction()` function.

The following table displays the enumerators of the `CTMEndTransactionResult` enumeration:

Enumerator	Description
`CTM_END_TRX_SUCCESS`	Indicates that the transaction has ended.
`CTM_END_TRX_ERROR_NO_TRANSACTION_IN_PROGRESS`	Indicates that there is no transaction in progress.
`CTM_END_TRX_ERROR_MUST_SUPPLY_TRANSACTION_ID`	Indicates that you must provide a non–empty transaction ID string.
`CTM_END_TRX_ERROR_NOT_CONNECTED`	Indicates that the CTM client is not connected to the CTM service.

Functions

The Transactions module includes the following functions:

- `ctm_begin_customer_transaction()`
- `ctm_begin_cash_management_transaction()`
- `ctm_end_transaction()`

ctm_begin_customer_transaction()

The `ctm_begin_customer_transaction()` function is used to start a new customer transaction. The POS application must call this method at the beginning of each customer transaction to reset any data that were accumulated during a prior transaction.

Syntax

```
struct CTMBeginTransactionResult ctm_begin_customer_transaction(const
char* szTransactionID)
```

Parameters

- `szTransactionID`—a unique identifier for the customer transaction.

ctm_begin_cash_management_transaction()

The `ctm_begin_cash_management_transaction()` function is used to start a new cash management transaction. The POS application must call this method at the beginning of each cash management loan or pickup operation.

Syntax

```
struct CTMBeginTransactionResult ctm_begin_cash_management_
transaction(char* szUserID, char* szCashierID, char* szTransactionID)
```

Parameters

- `szUserID`—a unique identifier for the attendant operating the application.
- `szCashierID`—a unique identifier for the cashier or cash–handling personnel attempting to manage the cash devices. The value of this parameter may or may not be the same as the value for the `szUserID` parameter.
- `szTransactionID`—a unique identifier for the customer transaction.

ctm_end_transaction()

The `ctm_end_transaction()` function is used to end a customer or cash management transaction.

Syntax

```
enum CTMEndTransactionResult ctm_end_transaction(char*
szTransactionID)
```

Parameters

- `szTransactionID`—a unique identifier for the completed customer or cash management transaction. The CTM API frees the string and assigns `NULL` to `szTransactionID` if the CTM API had previously allocated the string through the `ctm_begin_cash_management_transaction()` or `ctm_begin_cash_management_transaction()` functions.

Cash Dispensing

The Cash Dispensing module lists the functions used to dispense cash from the cash devices and the data that the CTM service passes to the callback functions during the handling of the cash dispensed events. This module contains the following:

- Enumerations
- Functions

Enumerations

The Cash Dispensing module includes the `CTMDispenseCashError` enumeration.

CTMDispenseCashError

The `CTMDispenseCashError` enumeration lists the possible return values of the following functions:

- `ctm_dispense_cash()`
- `ctm_dispense_cash_by_denomination()`

The following table displays the enumerators of the `CTMDispenseCashError` enumeration:

Enumerator	Description
`CTM_DISPENSE_CASH_SUCCESS`	Indicates that the CTM service successfully dispensed cash.
`CTM_DISPENSE_CASH_ERROR_NEEDS_OPEN_TRANSACTION_ID`	Indicates that there is no transaction ID for the transaction.
`CTM_DISPENSE_CASH_ERROR_CANNOT_BE_NEGATIVE`	Indicates that the amount specified is a negative number.
`CTM_DISPENSE_CASH_ERROR_INVALID_DENOMINATION`	Indicates that the denomination specified is an invalid denomination
`CTM_DISPENSE_CASH_ERROR_EXCEEDS_CONFIGURABLE_MAXIMUM`	Indicates that the amount specified exceeds the maximum limit.
`CTM_DISPENSE_CASH_ERROR_NOT_CONNECTED`	Indicates that the CTM client is not connected to the CTM service.

Functions

The Cash Dispensing module includes the following functions:

- `ctm_dispense_cash()`
- `ctm_dispense_cash_by_denomination()`

ctm_dispense_cash()

The `ctm_dispense_cash()` function is used to dispense cash from the cash devices.

Syntax

```
struct CTMDispenseCashResult ctm_dispense_cash(uint32_t
u32AmountToDispense)
```

Parameters

- `u32AmountToDispose`—refers to the amount to dispense.

ctm_dispense_cash_by_denomination()

The `ctm_dispense_cash_by_denomination()` function is used to dispense cash by denomination.

Syntax

```
struct CTMDispenseCashResult ctm_dispense_cash_by_denomination(struct
CTMCashUnitSet stCashToDispense)
```

Parameters

- `stCashToDispense`—refers to the set of `CashUnits` that contains the counts of each denomination to be dispensed.

Cash Management

The Cash Management module lists the functions used to manage the notes and coins in the cash devices. It also lists the data that the CTM service passes to the callback functions during the handling of the cash transferred events. This module contains the following:

- Enumerations
- Functions

Enumerations

The Cash Management module includes the following enumerations:

- `CTMCashTransferLocation`
- `CTMGetCashCountsError`
- `SetCountsResult`
- `ResetCountsResult`
- `ResetCountsLocation`
- `CTMGetLoaderCassetteCountsError`
- `CTMGetPurgedStatusResult`
- `CTMClearPurgedStatusResult`
- `CTMTransferCashError`

CTMCashTransferLocation

The `CTMCashTransferLocation` enumeration lists the locations used to report the result of a cash transfer request.

The following table displays the enumerators of the `CTMCashTransferLocation` enumeration:

Enumerator	Description
`CTM_CASH_TRANSFER_LOCATION_LOADER`	Indicates that the cash transfer occurs in the loader cassette. The loader cassette is the location where cash is stored before being transferred to individual dispenser bins.

Enumerator	Description
CTM_CASH_TRANSFER_LOCATION_CASHBOX	Indicates that the cash transfer occurs in the cash box. The cash box is the location where bills that cannot be transferred to an individual dispenser bin go. The cash–handling personnel can remove bills from the dispenser bins by transferring the bills from the dispenser bins to the cash box and removing the cash box cassette from the Bank Note Recycler (BNR) device.
CTM_CASH_TRANSFER_LOCATION_INDIVIDUAL_BIN	Indicates that the cash transfer occurs at an individual dispenser bin. Cash devices can dispense cash that are stored in the dispenser bins

CTMGetCashCountsError

The `CTMGetCashCountsError` enumeration lists the possible return values of the following functions:

- `ctm_get_dispensable_cash_counts()`
- `ctm_get_non_dispensable_cash_counts()`

The following table displays the enumerators of the `CTMGetCashCountsError` enumeration:

Enumerator	Description
CTM_GET_CASH_COUNTS_SUCCESS	Indicates that the CTM service successfully got the cash counts.
CTM_GET_CASH_COUNTS_DEVICE_IN_ERROR	Indicates that the device is not present.
CTM_GET_CASH_COUNTS_NOT_CONNECTED	Indicates that the CTM client is not connected to the CTM service.

SetCountsResult

The `SetCountsResult` enumeration lists the possible return values of the following function:

- `ctm_set_loader_cassette_counts()`
- `ctm_set_dispensable_counts()`

The following table displays the enumerators of the `SetCountsResult` enumeration:

Enumerator	Description
`CTM_SET_COUNTS_SUCCESS`	Indicates that the CTM service successfully set the loader cassette and dispensable counts.
`CTM_SET_COUNTS_NEEDS_OPEN_CASH_MANAGEMENT_TRX_ID`	Indicates that there is no transaction ID for the begin cash management event.
`CTM_SET_COUNTS_AMOUNT_CANNOT_BE_NEGATIVE`	Indicates that the amount specified is a negative number.
`CTM_SET_COUNTS_AMOUNT_INVALID_DENOMINATION`	Indicates that the denomination specified is an invalid denomination
`CTM_SET_COUNTS_AMOUNT_EXCEEDS_CONFIGURABLE_MAX`	Indicates that the amount specified exceeds the maximum limit.
`CTM_SET_COUNTS_DEVICE_NOT_PRESENT`	Indicates that the device is not present.
`CTM_SET_COUNTS_NOT_CONNECTED`	Indicates that the CTM client is not connected to the CTM service.

ResetCountsResult

The `ResetCountsResult` enumeration lists the possible return values for the following functions:

- `ctm_reset_counts_dispensable_coins()`
- `ctm_reset_counts_non_dispensable_coins()`
- `ctm_reset_counts_non_dispensable_notes()`

The following table displays the enumerators of the `ResetCountsResult` enumeration:

Enumerator	Description
`CTM_RESET_COUNTS_SUCCESS`	Indicates that the CTM service successfully reset the device individual bin, cash box, or the coin overflow bin.

Enumerator	Description
CTM_RESET_COUNTS_NEEDS_OPEN_CASH_MANAGEMENT_TRX_ID	Indicates that there is no transaction ID for the begin cash management event.
CTM_RESET_COUNTS_DEVICE_NOT_PRESENT	Indicates that the device is not present.
CTM_RESET_COUNTS_NOT_CONNECTED	Indicates that the CTM client is not connected to the CTM service.

ResetCountsLocation

The `ResetCountsLocation` enumeration lists the possible reset counts locations.

The following table displays the enumerators of the `ResetCountsLocation` enumeration:

Enumerator	Description
CTM_RESET_COUNTS_LOADER_CASSETTE	Indicates that the reset of the counts occurs in the loader cassette.
CTM_RESET_COUNTS_DISPENSABLE_COINS	Indicates that the dispensable coin counts must be reset.
CTM_RESET_COUNTS_NON_DISPENSABLE_COINS	Indicates that the non–dispensable coin counts must be reset.
CTM_RESET_COUNTS_NON_DISPENSABLE_NOTES	Indicates that the non–dispensable note counts must be reset.

CTMGetLoaderCassetteCountsError

The `CTMGetLoaderCassetteCountsError` enumeration lists the possible return values of the `ctm_get_loader_cassette_counts()` function.

The following table displays the enumerators of the `CTMGetLoaderCassetteCountsError` enumeration:

Enumerator	Description
CTM_GETLOADERCOUNTS_SUCCESS	Indicates that the CTM service successfully got the loader cassette counts.
CTM_GETLOADERCOUNTS_NOT_CONNECTED	Indicates that the CTM client is not connected to the CTM service.
CTM_GETLOADERCOUNTS_NEEDS_OPEN_CASH_MANAGEMENT_TRX_ID	Indicates that there is no transaction ID for the begin cash management event.
CTM_GETLOADERCOUNTS_DEVICE_NOT_PRESENT	Indicates that the device is not present.

CTMGetPurgedStatusResult

The `CTMGetPurgedStatusResult` enumeration lists the possible return values of the `ctm_get_purged_status()` function.

The following table displays the enumerators of the `CTMGetPurgedStatusResult` enumeration:

Enumerator	Description
`CTM_GET_PURGED_STATUS_PURGE_BIN_CONTAINS_NOTES`	Indicates that the CTM service detected purged bills.
`CTM_GET_PURGED_STATUS_PURGE_BIN_DOESNT_CONTAINS_NOTES`	Indicates that the CTM service did not detect any purged bills.
`CTM_GET_PURGED_STATUS_NOT_CONNECTED`	Indicates that the CTM client is not connected to the CTM service.

CTMClearPurgedStatusResult

The `CTMClearPurgedStatusResult` enumeration lists the possible return values of the `ctm_clear_purged_status()` function.

The following table displays the enumerators of the `CTMClearPurgedStatusResult` enumeration:

Enumerator	Description
`CTM_CLEAR_PURGED_STATUS_SUCCESS`	Indicates that the CTM service cleared purged bills.
`CTM_CLEAR_PURGED_STATUS_ERROR_RUNTIME`	Indicates that the CTM service was not able to clear purged bills.
`CTM_CLEAR_PURGED_STATUS_NOT_CONNECTED`	Indicates that the CTM client is not connected to the CTM service.

CTMTransferCashError

The `CTMTransferCashError` enumeration lists the possible return values of the following functions:

- `ctm_transfer_from_bin_to_cashbox()`
- `ctm_transfer_all_from_loader_to_cashbox()`

The following table displays the enumerators of the `CTMTransferCashError` enumeration:

Enumerator	Description
`CTM_TRANSFER_SUCCESS`	Indicates that the CTM service successfully transferred cash.
`CTM_TRANSFER_NEEDS_OPEN_CASH_MANAGEMENT_TRX_ID`	Indicates that there is no transaction ID for the begin cash management event.
`CTM_TRANSFER_AMOUNT_CANNOT_BE_NEGATIVE`	Indicates that the amount specified is a negative number.
`CTM_TRANSFER_AMOUNT_INVALID_DENOMINATION`	Indicates that the denomination specified is an invalid denomination.
`CTM_TRANSFER_AMOUNT_EXCEEDS_CONFIGURABLE_MAX`	Indicates that the amount specified exceeds the maximum limit.
`CTM_TRANSFER_NOT_CONNECTED`	Indicates that the CTM client is not connected to the CTM service.

Functions

The Cash Management module includes the following functions:

- `ctm_get_purged_status()`
- `ctm_clear_purged_status()`
- `ctm_get_loader_cassette_capacity()`
- `ctm_get_non_dispensable_coin_capacity()`
- `ctm_get_non_dispensable_note_capacity()`
- `ctm_get_dispensable_capacities()`
- `ctm_get_non_dispensable_cash_counts()`
- `ctm_get_dispensable_cash_counts()`
- `ctm_transfer_from_bin_to_cashbox()`
- `ctm_transfer_all_from_loader_to_cashbox()`
- `ctm_set_loader_cassette_counts()`

- `ctm_get_loader_cassette_counts()`
- `ctm_set_dispensable_counts()`
- `ctm_reset_counts_dispensable_coins()`
- `ctm_reset_counts_non_dispensable_coins()`
- `ctm_reset_counts_non_dispensable_notes()`

ctm_get_purged_status()

The `ctm_get_purged_status()` function determines whether any dispensable notes have been sent to the purged bin. You may also call this function to determine whether a defective note is placed in a loader cassette and the validator cannot read it.

Syntax

```
enum CTMGetPurgedStatusResult ctm_get_purged_status()
```

Returns

This function returns one of the values from the `CTMGetPurgedStatusResult` enumeration.

ctm_clear_purged_status()

The `ctm_clear_purged_status()` function clears the purge status.

Syntax

```
enum CTMClearPurgedStatusResult ctm_clear_purged_status()
```

Returns

This function returns one of the values from the `CTMClearPurgedStatusResult` enumeration.

ctm_get_loader_cassette_capacity()

The `ctm_get_loader_cassette_capacity()` function obtains the estimated maximum number of notes that the loader cassette can contain.

Syntax

```
struct CTMGetCapacitiesResult ctm_get_loader_cassette_capacity()
```

Returns

This function returns the maximum loader cassette capacity or zero (0) if the device does not have a loader cassette.

ctm_get_non_dispensable_coin_capacity()

The `ctm_get_non_dispensable_coin_capacity()` function obtains the non–dispensable storage capacity for the coin device.

Syntax

```
struct CTMGetCapacitiesResult ctm_get_non_dispensable_coin_capacity()
```

Returns

This function returns the maximum coin storage count.

ctm_get_non_dispensable_note_capacity()

The `ctm_get_non_dispensable_note_capacity()` function obtains the non–dispensable storage capacity for the note device.

Syntax

```
struct CTMGetCapacitiesResult ctm_get_non_dispensable_note_capacity()
```

Returns

This function returns the maximum note storage count.

ctm_get_dispensable_capacities()

The `ctm_get_dispensable_capacities()` function obtains the storage capacity of each dispensable denomination.

Syntax

```
struct CTMGetCapacitiesResult ctm_get_dispensable_capacities()
```

Returns

This function returns a set of `CashUnit` where the counts indicate the capacity of each denomination.

ctm_get_dispensable_cash_counts()

The `ctm_get_dispensable_cash_counts()` function returns the number of each denomination that the CTM can dispense as a change. This count does not include the notes that are in the loader cassette.

Syntax

```
struct CTMGetCashCountsResult ctm_get_dispensable_cash_counts()
```

Returns

This function returns a set of `CashUnit` objects sorted from the lowest coin to the highest note denomination.

ctm_get_non_dispensable_cash_counts()

The `ctm_get_non_dispensable_cash_counts()` function returns the number of each denomination that the CTM cannot dispense as a change.

Syntax

```
struct CTMGetCashCountsResult ctm_get_non_dispensable_cash_counts()
```

Returns

This method returns a set of `CashUnit` objects sorted from the lowest coin to the highest note denomination.

ctm_transfer_from_bin_to_cashbox()

The `ctm_transfer_from_bin_to_cashbox()` function transfers a specified number of notes from the recyclers to the cash box if supported by the device configuration. If a note recycler is not present in the device, nothing happens when you call this function.

Syntax

```
struct CTMTransferFromBinToCashboxResult ctm_transfer_from_bin_to_
cashbox(struct CTMCashUnitSet stCashToTransfer)
```

Parameters

- `stCashToTransfer`—set of `CashUnits` that contains the counts of each denomination to be transferred.

Returns

This function returns one of the values from the `CTMTransferCashError` enumeration.

ctm_transfer_all_from_loader_to_cashbox()

The `ctm_transfer_all_from_loader_to_cashbox()` function transfers all cash from the loader cassette directly to the cashbox.

Syntax

```
struct CTMTransferAllFromLoaderToCashboxResult ctm_transfer_all_from_
loader_to_cashbox()
```

Returns

This function returns one of the values from the `CTMTransferCashError` enumeration.

ctm_set_loader_cassette_counts()

The `ctm_set_loader_cassette_counts()` function sets the logical note counts of the note loader cassette. If a loader cassette is not present in the device, nothing happens when you call this function.

Syntax

```
enum SetCountsResult ctm_set_loader_cassette_counts(struct
CTMCashUnitSet stCashToSet)
```

Parameters

- `stCashToSet`—set of `CashUnits` that contains the counts of each denomination to set.

Returns

This function returns `true` if the loader cassette count is successfully set, and `false` on failure.

ctm_get_loader_cassette_counts()

The `ctm_get_loader_cassette_counts()` returns the number of each note denomination in the loader cassette.

Syntax

```
struct CTMGetLoaderCassetteCountsResult ctm_get_loader_cassette_
counts()
```

Returns

This method returns a set of `CashUnit` objects sorted from the lowest coin to the highest note denomination.

ctm_set_dispensable_counts()

The `ctm_set_dispensable_counts()` function sets the logical cash counts of recyclers or dispensers in the BCR device. Setting dispensable counts is not applicable to the BNR device or to cash devices that do not allow resetting of dispensable counts.

Syntax

```
enum SetCountsResult ctm_set_dispensable_counts(struct CTMCashUnitSet
stCashToSet)
```

Parameters

- `stCashToSet`— set of `CashUnits` that contains the counts of each denomination to set.

Returns

This function returns one of the values from the `SetCountsResult` enumeration.

ctm_reset_counts_dispensable_coins()

The `ctm_reset_counts_dispensable_coins()` function is used to empty the coin dispenser or recycler containers. This method does nothing for cash devices that do not permit resetting of dispensable counts.

Syntax

```
enum ResetCountsResult ctm_reset_counts_dispensable_coins()
```

Returns

This function returns `true` if the dispensable coin count was successfully reset, and `false` on failure.

ctm_reset_counts_non_dispensable_coins()

The `ctm_reset_counts_non_dispensable_coins()` function is used to empty the non–dispensable containers (coin overflow bin) of the Bulk Coin Recycler (BCR) device.

Syntax

```
enum ResetCountsResult ctm_reset_counts_non_dispensable_coins()
```

Returns

This function returns `true` if the non–dispensable coin count was successfully reset, and `false` on failure.

ctm_reset_counts_non_dispensable_notes()

The `ctm_reset_counts_non_dispensable_notes()` function is used to empty the non–dispensable containers (cash box) of the BNR device.

Syntax

```
enum ResetCountsResult ctm_reset_counts_non_dispensable_notes()
```

Returns

This function returns `true` if the non–dispensable note count was successfully reset, and `false` on failure.

Cash Accepting

The Cash Accepting module lists the functions that are used to accept cash from the cash devices and the data that the CTM service passes to the callback functions during the handling of cash accept events.

This module contains the following:

- Typedefs
- Enumerations
- Functions

Typedefs

The Cash Accepting module includes the following typedefs:

- `CTMCashAcceptCallback`
- `CTMCashAcceptCompleteCallback`

CTMCashAcceptCallback

The `CTMCashAcceptCallback` typedef is the callback for cash deposit events.

Syntax

```
typedef void(* CTMCashAcceptCallback)(const struct CTMEventInfo,
const struct CTMAcceptEvent)
```

CTMCashAcceptCompleteCallback

The `CTMCashAcceptCompleteCallback` typedef is the callback used on events where the deposited amount is enough or more than the expected amount.

Syntax

```
typedef void(* CTMCashAcceptCompleteCallback)(const struct
CTMEventInfo)
```

Enumerations

The Cash Accepting module includes the following enumerations:

- `CTMAcceptCashRequestResult`
- `CTMStopAcceptingCashResult`

CTMAcceptCashRequestResult

The `CTMAcceptCashRequestResult` enumeration lists the possible return values of the following functions:

- `ctm_begin_refill()`
- `ctm_accept_cash()`

The following table displays the enumerators of the `CTMAcceptCashRequestResult` enumeration:

Enumerator	Description
`CTM_ACCEPT_CASH_SUCCESS`	Indicates that the CTM service has successfully performed the cash accept event.
`CTM_ACCEPT_CASH_ERROR_NEEDS_OPEN_TRANSACTION_ID`	Indicates that you must provide an open transaction ID to complete the current request.
`CTM_ACCEPT_CASH_ERROR_ALREADY_IN_PROGRESS`	Indicates that you must end the previous cash accept event first before starting a new one.
`CTM_ACCEPT_CASH_ERROR_NOT_CONNECTED`	Indicates that the CTM client is not connected to the CTM service.

CTMStopAcceptingCashResult

The `CTMStopAcceptingCashResult` enumeration lists the possible return values of the `ctm_stop_accepting_cash()` function.

The following table displays the enumerators of the `CTMStopAcceptingCashResult` enumeration:

Enumerator	Description
`CTM_STOP_ACCEPTING_CASH_SUCCESS`	Indicates that the CTM service successfully stopped the cash accepting event.
`CTM_STOP_ACCEPTING_CASH_ERROR_NO_DEPOSIT_IN_PROGRESS`	Indicates that there is no deposit in progress.
`CTM_STOP_ACCEPTING_CASH_ERROR_NOT_CONNECTED`	Indicates that the CTM client is not connected to the CTM service.

Functions

The Cash Accepting module includes the following functions:

- `ctm_add_cash_accept_event_handler()`
- `ctm_add_cash_accept_complete_event_handler()`
- `ctm_accept_cash()`

- `ctm_begin_refill()`
- `ctm_stop_accepting_cash()`

ctm_add_cash_accept_event_handler()

The `ctm_add_cash_accept_event_handler()` function is used to provide an event handler callback to CTM when the cash accept event is ongoing.

Syntax

```
void ctm_add_cash_accept_event_handler(CTMCashAcceptCallback)
```

ctm_add_cash_accept_complete_event_handler()

The `ctm_add_cash_accept_complete_event_handler()` function is used to provide an event handler callback to CTM when the cash accept event is complete.

Syntax

```
void ctm_add_cash_accept_complete_event_handler
(CTMCashAcceptCompleteCallback)
```

ctm_accept_cash()

The `ctm_accept_cash()` function is used to accept cash from cash accept devices during a transaction.

Syntax

```
enum CTMAcceptCashRequestResult ctm_accept_cash(uint32_t
u32AmountToAccept)
```

Parameters

- `u32AmountToAccept`—refers to the amount of cash to accept in a transaction. After this amount is deposited, the CTM service will automatically disable coin and note acceptance. If the amount is not deposited or is set to zero, acceptance is enabled until `ctm_stop_accepting_cash()` is called.

Returns

This function returns one of the values from the `CTMAcceptCashRequestResult` enumeration.

ctm_begin_refill()

The `ctm_begin_refill()` function is used to replenish coins and notes through the acceptors. This method does nothing for devices that do not support this capability.

Syntax

```
enum CTMAcceptCashRequestResult ctm_begin_refill()
```

Returns

This function returns one of the values from the `CTMAcceptCashRequestResult` enumeration.

ctm_stop_accepting_cash()

The `ctm_stop_accepting_cash()` function is used to disable coin and note acceptance.

Note: Deposit events may continue to arrive after the `ctm_stop_accepting_cash()` function is called if the cash accept devices are handling deposited currency when the `endDeposit` request is issued.

Syntax

```
enum CTMStopAcceptingCashResult ctm_stop_accepting_cash()
```

Returns

This function returns one of the values from the `CTMStopAcceptingCashResult` enumeration.

Appendix A: Installing Third–party Software

Overview

This appendix discusses the guide to installing the following third–party applications used in the functionality of the Cash Tender Module (CTM) system:

- Java™ Runtime Environment (JRE)

Installing Java™ Runtime Environment (JRE)

To download and install Java, follow these steps:

1. Open a compatible browser of your choice.
2. Go to *http://www.java.com/en/download* and download JRE version 1.6 or later.
3. Select **Windows Offline** to download the installer file.
4. Select **Save** to save the file to a desired installation path.
5. Double–click the file to install the application and follow the installation wizard instructions.

Appendix B: Open Source Software

Open Source Software List

This section discusses the different open source software used for the development and functionality of the CTM system.

Note: You must honor the license terms for each of the software listed in this appendix.

The following table displays the open source software used in the CTM system:

Name	Version
MinGW Port of the GNU Compiler Collection (GCC) Runtime Libraries	4.6.1–2
POSIX Threads for Win32	2.9
Google Protocol Buffers	2.4.1
Apache log4cxx	0.10.0
Apache Portable Runtime (APR)	
APR–util	1.4.6
1.4.1	
AOP alliance	1.0
Apache Ant Core	1.8.2
Apache Ant Launcher	1.8.2
Apache Velocity	1.5
args4j	2.0.8
ASM Analysis	3.2
ASM Commons	3.3.1
ASM Core	3.3.1
ASM Tree	3.3.1
ASM Util	3.2
Build Helper Maven Plugin	1.7

Name	Version
Classworlds	1.1
CLI	1.0
Code Generation Library	2.2.2
com4j runtime	20110320
com4j type library importer	20110320
Commons Collections	3.2.1
Commons IO	1.3.2
Commons Lang	2.6
Commons Logging	1.1.1
commons–beanutils	1.7.0
commons–digester	1.6
Default Plexus Container	1.0–alpha–9–stable–1
Doxia :: APT Module	1.0
Doxia :: Core	1.0
Doxia :: FML Module	1.0
Doxia :: Sink API	1.0
Doxia :: XDoc Module	1.0
Doxia :: XHTML Module	1.0
Doxia Sitetools :: Decoration Model	1.0
Doxia Sitetools :: Site Renderer Component	1.0
HttpClient	2.0.2
jdependency	0.7
jdom	1.0
JDOM	1.1
JSch	0.1.27
JTidy	4aug2000r7–dev
JUnit	3.8.1
Licensing Maven Mojo	1.7
Maven AntRun Plugin	1.7

Name	Version
Maven Archiver	2.5
Maven Artifact	2.0.6
Maven Artifact	2.0.9
Maven Artifact Manager	2.0.9
Maven Assembly Plugin	2.3
Maven Common Artifact Filters	1.2
Maven Core	2.0.9
Maven Dependency Analyzer	1.2
Maven Dependency Plugin	2.4
Maven Dependency Tree	1.1
Maven Dependency Tree	1.2
Maven Doxia Integration Tools	1.0.2
Maven Error Diagnostics	2.0.9
Maven File Management API	1.2.1
Maven Filtering	1.0
Maven Invoker	2.0.11
Maven Local Settings Model	2.0.6
Maven Local Settings Model	2.0.9
Maven Model	2.0.6
Maven Model	2.0.9
Maven Monitor	2.0.9
Maven Plugin API	2.0.6
Maven Plugin API	2.0.9
Maven Plugin Descriptor Model	2.0.9
Maven Plugin Parameter Documenter API	2.0.9
Maven Plugin Registry Model	2.0.6
Maven Plugin Registry Model	2.0.9
Maven Plugin Testing Mechanism	1.1
Maven Profile Model	2.0.6

Name	Version
Maven Profile Model	2.0.9
Maven Project Builder	2.0.6
Maven Project Builder	2.0.9
Maven Reporting API	3.0
Maven Reporting Implementation	2.0.5
Maven Repository Builder	1.0–alpha–2
Maven Repository Metadata Model	2.0.9
Maven Shade Plugin	1.6
Maven Shared I/O API	1.1
Maven Wagon API	1.0–beta–2
Maven Wagon File Provider	1.0–beta–2
Maven Wagon HTTP Shared Library	1.0–beta–2
Maven Wagon Lightweight HTTP Provider	1.0–beta–2
Maven Wagon SSH Common Library	1.0–beta–2
Maven Wagon SSH External Provider	1.0–beta–2
Maven Wagon SSH Provider	1.0–beta–2
Maven Wagon WebDav Provider	1.0–beta–2
MXP1: Xml Pull Parser 3rd Edition (XPP3)	1.1.4c
oro	2.0.8
Plexus :: Component Annotations	1.5.5
Plexus Archiver Component	2.0
Plexus Common Utilities	1.5.5
Plexus Common Utilities	3.0
Plexus Default Interactivity Handler	1.0–alpha–4
Plexus I18N Component	1.0–beta–7
Plexus Interpolation API	1.15
Plexus IO Components	2.0.1
Plexus Resource Component	1.0–alpha–7
Plexus Velocity Component	1.1.7

Name	Version
plexus–build–api	0.0.4
protobuf–java–format	1.2
Protocol Buffer Java API	2.4.1
SLF4J API Module	1.5.6
SLF4J Simple Binding	1.5.6
slide–webdavlib	2.1
Spring Integration Core	2.1.0
spring–aop	3.0.7
spring–aop	3.1.1
spring–asm	3.1.1
spring–beans	3.1.1
spring–context	3.1.1
spring–core	3.1.1
spring–expression	3.1.1
Validator	1.2.0
XML Commons External Components XML APIs	1.0.b2
XML Im–/Exporter	1.1
XML Pull Parsing API	1.1.3.1
XStream Core	1.4.2

Appendix C: Installer and Uninstaller Return Codes List

Verified Return Codes

The following table displays the verified return codes for this release of the Cash Tender Module SDK.

Error Code	Return Value	Description	Note
Success	0	The action completed successfully.	Applicable for both the installation and uninstallation process.
Install_ UserExit	1602	The user canceled the installation or the installation terminated unexpectedly.	Applicable for both the installation and uninstallation process.
Install_ Failure	1603	A fatal error occurred during installation.	Non–admin user (Applicable for both the installation and uninstallation process.)
Unknown Product	1605	This action is valid only for products that are currently installed.	Applicable for uninstalling already uninstalled products or products that are not yet installed.
Product Version	1638	Another version of this product is already installed. Installation of this version cannot continue.	Applicable for installing another version of a product that is already installed. For example, installing version 13 without uninstalling version 12.

Supported Return Codes

The following table displays the complete list of supported return codes.

Error Code	Value	Description
ERROR_SUCCESS	0	The action completed successfully.
ERROR_INVALID_DATA	13	The data is invalid.
ERROR_INVALID_ PARAMETER	87	One of the parameters was invalid.

Error Code	Value	Description
ERROR_CALL_NOT_ IMPLEMENTED	120	This value is returned when a custom action attempts to call a function that cannot be called from custom actions. The function returns the value ERROR_CALL_NOT_IMPLEMENTED. This is available since Windows Installer version 3.0.
ERROR_APPHELP_BLOCK	1259	If Windows Installer determines a product may be incompatible with the current operating system, it displays a dialog box informing the user and asking whether to try to install anyway. This error code is returned if the user chooses not to try the installation.
ERROR_INSTALL_SERVICE_ FAILURE	1601	The Windows Installer service could not be accessed. Contact your support personnel to verify that the Windows Installer service is properly registered.
ERROR_INSTALL_USEREXIT	1602	The user cancels the installation.
ERROR_INSTALL_FAILURE	1603	A fatal error occurred during installation.
ERROR_INSTALL_SUSPEND	1604	Installation suspended or incomplete.
ERROR_UNKNOWN_ PRODUCT	1605	This action is only valid for products that are currently installed.
ERROR_UNKNOWN_ FEATURE	1606	The feature identifier is not registered.
ERROR_UNKNOWN_ COMPONENT	1607	The component identifier is not registered.
ERROR_UNKNOWN_ PROPERTY	1608	This is an unknown property.
ERROR_INVALID_HANDLE_ STATE	1609	The handle is in an invalid state.
ERROR_BAD_ CONFIGURATION	1610	The configuration data for this product is corrupt. Contact your support personnel.
ERROR_INDEX_ABSENT	1611	The component qualifier not present.
ERROR_INSTALL_SOURCE_ ABSENT	1612	The installation source for this product is not available. Verify that the source exists and that you can access it.
ERROR_INSTALL_PACKAGE_ VERSION	1613	This installation package cannot be installed by the Windows Installer service. You must install a Windows service pack that contains a newer version of the Windows Installer service.

Error Code	Value	Description
ERROR_PRODUCT_ UNINSTALLED	1614	The product is uninstalled.
ERROR_BAD_QUERY_ SYNTAX	1615	The SQL query syntax is invalid or unsupported.
ERROR_INVALID_FIELD	1616	The record field does not exist.
ERROR_INSTALL_ALREADY_ RUNNING	1618	Another installation is already in progress. Complete that installation before proceeding with this install.
ERROR_INSTALL_PACKAGE_ OPEN_FAILED	1619	This installation package could not be opened. Verify that the package exists and is accessible, or contact the application vendor to verify that this is a valid Windows Installer package.
ERROR_INSTALL_PACKAGE_ INVALID	1620	This installation package could not be opened. Contact the application vendor to verify that this is a valid Windows Installer package.
ERROR_INSTALL_UI_FAILURE	1621	There was an error starting the Windows Installer service user interface. Contact your support personnel.
ERROR_INSTALL_LOG_ FAILURE	1622	There was an error opening installation log file. Verify that the specified log file location exists and is writable.
ERROR_INSTALL_ LANGUAGE_UNSUPPORTED	1623	This language of this installation package is not supported by your system.
ERROR_INSTALL_ TRANSFORM_FAILURE	1624	There was an error applying transforms. Verify that the specified transform paths are valid.
ERROR_INSTALL_PACKAGE_ REJECTED	1625	This installation is forbidden by system policy. Contact your system administrator.
ERROR_FUNCTION_NOT_ CALLED	1626	The function could not be executed.
ERROR_FUNCTION_FAILED	1627	The function failed during execution.
ERROR_INVALID_TABLE	1628	An invalid or unknown table was specified.
ERROR_DATATYPE_ MISMATCH	1629	The data supplied is the wrong type.
ERROR_UNSUPPORTED_ TYPE	1630	Data of this type is not supported.
ERROR_CREATE_FAILED	1631	The Windows Installer service failed to start. Contact your support personnel.

Error Code	Value	Description
ERROR_INSTALL_TEMP_ UNWRITABLE	1632	The Temp folder is either full or inaccessible. Verify that the Temp folder exists and that you can write to it.
ERROR_INSTALL_ PLATFORM_UNSUPPORTED	1633	This installation package is not supported on this platform. Contact your application vendor.
ERROR_INSTALL_NOTUSED	1634	Component is not used on this machine.
ERROR_PATCH_PACKAGE_ OPEN_FAILED	1635	This patch package could not be opened. Verify that the patch package exists and is accessible, or contact the application vendor to verify that this is a valid Windows Installer patch package.
ERROR_PATCH_PACKAGE_ INVALID	1636	This patch package could not be opened. Contact the application vendor to verify that this is a valid Windows Installer patch package.
ERROR_PATCH_PACKAGE_ UNSUPPORTED	1637	This patch package cannot be processed by the Windows Installer service. You must install a Windows service pack that contains a newer version of the Windows Installer service.
ERROR_PRODUCT_VERSION	1638	Another version of this product is already installed. Installation of this version cannot continue. To configure or remove the existing version of this product, use Add/Remove Programs in Control Panel.
ERROR_INVALID_ COMMAND_LINE	1639	Invalid command line argument. Consult the Windows Installer SDK for detailed command-line help.
ERROR_INSTALL_REMOTE_ DISALLOWED	1640	The current user is not permitted to perform installations from a client session of a server running the Terminal Server role service.
ERROR_SUCCESS_REBOOT_ INITIATED	1641	The installer has initiated a restart. This message is indicative of a success.
ERROR_PATCH_TARGET_ NOT_FOUND	1642	The installer cannot install the upgrade patch because the program being upgraded may be missing or the upgrade patch updates a different version of the program. Verify that the program to be upgraded exists on your computer and that you have the correct upgrade patch.
ERROR_PATCH_PACKAGE_ REJECTED	1643	The patch package is not permitted by system policy.
ERROR_INSTALL_ TRANSFORM_REJECTED	1644	One or more customizations are not permitted by system policy.

Error Code	Value	Description
ERROR_INSTALL_REMOTE_PROHIBITED	1645	Windows Installer does not permit installation from a Remote Desktop Connection.
ERROR_PATCH_REMOVAL_UNSUPPORTED	1646	The patch package is not a removable patch package. Available beginning with Windows Installer version 3.0.
ERROR_UNKNOWN_PATCH	1647	The patch is not applied to this product. Available beginning with Windows Installer version 3.0.
ERROR_PATCH_NO_SEQUENCE	1648	No valid sequence could be found for the set of patches. Available beginning with Windows Installer version 3.0.
ERROR_PATCH_REMOVAL_DISALLOWED	1649	Patch removal was disallowed by policy. Available beginning with Windows Installer version 3.0.
ERROR_INVALID_PATCH_XML	1650	The XML patch data is invalid. Available beginning with Windows Installer version 3.0.
ERROR_PATCH_MANAGED_ADVERTISED_PRODUCT	1651	Administrative user failed to apply patch for a per-user managed or a per-machine application that is in advertise state. Available beginning with Windows Installer version 3.0.
ERROR_INSTALL_SERVICE_SAFEBOOT	1652	Windows Installer is not accessible when the computer is in Safe Mode. Exit Safe Mode and try again or try using System Restore to return your computer to a previous state. Available beginning with Windows Installer version 4.0.
ERROR_ROLLBACK_DISABLED	1653	Could not perform a multiple-package transaction because rollback has been disabled. Multiple-Package Installations cannot run if rollback is disabled. Available beginning with Windows Installer version 4.5.
ERROR_INSTALL_REJECTED	1654	The app that you are trying to run is not supported on this version of Windows. A Windows Installer package, patch, or transform that has not been signed by Microsoft cannot be installed on an ARM computer.
ERROR_SUCCESS_REBOOT_REQUIRED	3010	A restart is required to complete the install. This message is indicative of a success. This does not include installs where theForceReboot action is run.

For more information about these supported return codes, go to *http://msdn.microsoft.com/en-us/library/windows/desktop/aa368542%28v=vs.85%29.aspx*.