User Guide

Terminal Service Management (TSM) for *Android*™

Overview

The purpose of this document is to outline the new Terminal Service Management (TSM) features for CommerceDriver™ for the Android operating system. All CommerceDriver™ integrators are required to support the new TSM features in order to support updates to the Ingenico line of terminals, including the iCMP, iPP320, and iPP350 terminals.

Terminals will receive updates periodically, and if an update is not applied to a terminal by the associated deadline date, the terminal will be unable to transact until the update is installed.

Initialize Terminal

The `InitializeTerminal` method of the CommerceDriver™ object now provides information if an update is available for the terminal currently in use. Users must be signed on to their instance of CommerceDriver™ in order for the initialize terminal process to begin and for the terminal to begin checking for updates. This sign on procedure can be found for each operating system in their respective Quick Start Guides. After performing the steps to authenticate and add a terminal, check the response from the `initializeTerminal()` method to determine if updates are available.

Code Snippets

To initialize a terminal in C#, call the `initializeTerminal()` method in CommerceDriver™. This method runs asynchronously and returns an `ConnectionResult`, which contains a Boolean `isUpdateAvailable()` as well as a list of available updates via `getUpdates()`.

Example code to call and handle the completion of the `initializeTerminal()` method can be found below:

```
class TerminalConnectCallable implements Callable<ConnectResponse> {

    private final CommerceDriver commerceDriver;

    public TerminalConnectCallable(CommerceDriver commerceDriver) {
        this.commerceDriver = commerceDriver;
    }

    @Override
    public ConnectResponse call() throws Exception {
        return commerceDriver.connectTerminal();
    }
}

// MainActivity
 @Override
 public void connectSelectedTerminal() {
            TerminalConnectCallable callable = new TerminalConnectCallable(commerceDriver);
        CallableTask<ConnectResponse> task = new CallableTask<ConnectResponse>(callable);
        task.setUiCallback(this);
        task.setResultCallback(new CallableTaskResultCallback<ConnectResponse>() {
            @Override
            public void onReturnException(Exception exception) {
                // Exception handling here
            }

            @Override
            public void onReturnResult(ConnectResponse response) {
                if (response.isUpdateAvailable()) {
                      // Terminal has an update. Can prompt dialog here and move on, or start update flow.
                }
                // Connected with no updates available.
            }
        });
        task.execute();
}
```

InitializeTerminalResult

```
package com.evosnap.commercedriver.terminal;

import com.evosnap.commercedriver.cws.terminal.AvailableUpdateInfo;

import java.util.List;

public interface ConnectResponse {

    Result getResult();

    String getErrorMessage();

    List<AvailableUpdateInfo> getUpdates();

    boolean isUpdateAvailable();

    enum Result {
        CONNECTED,
        INVALID_TERMINAL_ID,
        SESSION_REQUIRED,
        TERMINAL_ERROR,
    }
}
```

If the `isUpdateAvailable` property of the `TerminalUpdate` object is true, call the `DownloadAndApplyUpdate` method as described below before the terminal update deadline date.

IMPORTANT! If a terminal has not downloaded the available terminal updates by the associated deadline date, the terminal will be deactivated, preventing any future transactions. Notification of this should be given to the merchant in the form of a "pop-up" message on the terminal. The suggested message can be found below:

"Terminal updates not installed by their deadline will result in a deactivation of your terminal. Would you like to begin the updates?"

Suggested action buttons and their suggested corresponding actions can be found below:

Action Button Message	Suggested Resulting Action
"Begin updates now"	Begins updates immediately after selecting the button. Note that this process can take up to 30 minutes, so this action should only be selected when the terminal can be inactive for approximately half an hour. **Remind merchants that they cannot disrupt the terminal while it is in the process of applying the update; this could break the terminal completely and requires a new terminal to be sent to the merchant.**
"Remind me later"	The pop-up is shown every time the terminal initializes; it is recommended that ISVs give merchants the option to start updates at any time, such as in a dropdown menu.

Downloading and Applying a Terminal Update

This section outlines the steps for using the `downloadAndApplyUpdate` call. This call can be made at any time, meaning merchants do not need to wait for an update to be available in order to make this call.

Note that, normally, there will only be one terminal update to be applied at any given time, but it is possible for multiple updates to need to be applied at a time.

The process of downloading and applying a terminal update can take up to 30 minutes, so it is recommended that these updates be started with ample time to complete. **Do not disrupt the**

terminal while it is in the process of applying the update, otherwise you risk breaking the terminal completely. This will require a new terminal to be sent to the merchant applying the update.

See the table below for an outline of the steps that occur during this process and how long they could possibly take:

Enum Name	Description	Step Speed
`Validation`	This step validates that the terminal can and is ready to be updated.	Very fast
`CheckForUpdates`	This step checks for any available updates for the terminal.	Fast
`DownloadFile`	This step downloads the update file.	Speed depends on file size and internet speed of the downloader.
`WriteFileToTerminal`	This step takes the update and breaks it down into smaller chunks so that the file can be written to the terminal. Then, it writes the broken up file pieces to the terminal.	Speed depends on the file size.
`TerminalReboot`	This step reboots the terminal after the update has been applied.	Slow – this is the longest step in the terminal update process.
`ConfirmUpdate`	This step reports that the terminal has been successfully updated to the Snap* platform.	Fast

Updates are installed one at a time and the terminal is restarted at the end of each update. After a terminal restart, the `InitializeTerminal` method must be called again. It is possible for multiple updates to be available for the terminal at the same time, but they are installed one at a time. For this reason, when calling `InitializeTerminal` after restart, check the property `isUpdateAvailable` again. If it is true, call the `DownloadAndApplyUpdate()` method again.

When using a terminal which connects via Bluetooth, it may be necessary to go through the Bluetooth pairing process after an update. This only happens when the firmware of the device is updated. This process needs to be performed before you can reconnect to the device.

Code Snippets

If `InitializeTerminal()` returns updates as outlined above, call the `DownloadAndApplyUpdate()` method in CommerceDriver™ before the specified deadline date. This method runs asynchronously and will return an `ApplyUpdateContext` object when finished.

DownloadAndApplyUpdate

`DownloadAndApplyUpdate` will return an `ApplyUpdateContext`. This context can then be used to set update callback, file accessor, and executing the update as shown in the example below. When finished, `downloadAndApplyResultType` enum can be used for seeing the status of the invoked update.

```
class ApplyUpdateCallable implements Callable<DownloadAndApplyResultType> {
    private final ApplyUpdateContext context;

    public ApplyUpdateCallable(ApplyUpdateContext context) {
        this.context = context;
    }

    @Override
    public DownloadAndApplyResultType call() throws Exception {
        return context.beginUpdate();
    }
}
```

ApplyUpdateContext

```
package com.evosnap.commercedriver.terminal.update;

public interface ApplyUpdateContext {

    String TAG = "ApplyUpdateContext";

    DownloadAndApplyResultType beginUpdate();

    void setUpdateCallback(TerminalUpdateCallback callback);

    void setFileAccessor(FileAccessor fileAccessor);

    DownloadAndApplyResultType getDownloadAndApplyResultType();

    String getErrorMessage();
}

// In MainActivity (or fragment)

public void applyUpdate(ApplyUpdateContext context) {

        Timber.tag(BuildConfig.TAG).d("[%s] applyUpdate()", TAG);

        FileAccessor fileAccessor = new ContextFileAccessor(getContext());

        context.setFileAccessor(fileAccessor);

        TerminalUpdateCallback updateCallback = new RecyclerUpdateCallback(recyclerViewAdapter);

        context.setUpdateCallback(updateCallback);
```

ApplyUpdateContext (cont.)

```
        Callable<DownloadAndApplyResultType> callable = new ApplyUpdateCallable(context);
        CallableTask<DownloadAndApplyResultType> task = new CallableTask<>(callable);
        task.setResultCallback(new CallableTaskResultCallback<DownloadAndApplyResultType>() {
            @Override
            public void onReturnException(Exception exception) {
                Timber.tag(BuildConfig.TAG).w("[%s] Exception Thrown : %s", TAG, exception);
                recyclerViewAdapter.addMessage("Exception...");
                recyclerViewAdapter.addMessage(exception.getMessage());
            }
            @Override
            public void onReturnResult(DownloadAndApplyResultType downloadAndApplyResultType) {
                Timber.tag(BuildConfig.TAG).w("[%s] ApplyUpdate Chain Completed : %s", TAG,
downloadAndApplyResultType);
                recyclerViewAdapter.addMessage(String.format("Completed - %s", downloadAndApplyResultType));
            }
        });
        task.execute();
    }
```

DownloadAndApplyResultType

```
public enum DownloadAndApplyResultType {
    CANCELLED,
    TERMINAL_UP_TO_DATE,
    VALIDATION_ERROR_NOT_LOGGED_IN,
    VALIDATION_ERROR_TERMINAL_NOT_SELECTED,
    VALIDATION_ERROR_NOT_INITIALIZED,
    VALIDATION_ERROR_NOT_ENOUGH_SPACE,
    DOWNLOAD_FAILED,
    HASH_CHECK_FAILED,
    TERMINAL_WRITE_FAILED,
    CONFIRM_UPDATE_FAILED,
    SUCCESS,
}
```

ProgressMessage

A `progressMessage` enum is also set throughout the update chain, which can be checked throughout the update for additional update context.

```
public enum ProgressMessage {
    Validation,
    CheckForUpdates,
    DownloadFile,
    WriteFileToTerminal,
    TerminalReboot,
    ConfirmUpdate,
}
```

New Merchants

New merchants have a slightly different workflow when getting started with TSM. New terminals become auto-registered when used to take their first transaction, after which merchants have three days to apply the expired available updates, no matter how long after the update expiration date, to their terminals in order to remain compliant. Multiple updates will most likely be needed.

Testing and Certification

To begin testing and certification of TSM, please contact EVO Snap*, and we will begin putting together sample files and an environment for you to test in. To properly set up these files, we will need the merchant ID to test with, the current RBA version of the terminal, the terminal serial number, and terminal type being used to test.

Troubleshooting Common Errors

The following list details common problems encountered while using TSM. If a merchant encounters one of these errors, direct them to contact your technical support, who should then contact EVO Snap* support for further assistance if needed.

* The terminal keeps rebooting.
* The terminal loads in LLT mode.
* Terminal loads but all text is distorted, or the all the text is absent.
* Terminal loads with a blank default screen.
* Terminal loads and shows 'Waiting for download...' message, along with file names and versions.

* Terminal loads correctly, but all transactions are declining before prompting for a card.

Final Steps and Best Practices

Eventually, there will be no more terminal updates available for download and application. After all the terminal updates have been applied to the terminal, the terminal will disconnect and the `InitializeTerminal` call will need to be run again. Check the response from the `InitializeTerminal` call to ensure no other updates are needed; optionally, run the `DownloadAndApplyUpdate()` method.

IMPORTANT! The information contained in this document outlines the best practices for handling the implementation of TSM, as determined by EVO Snap*. Though users are able to implement TSM in whatever manner they choose, it is strongly encouraged that ISVs follow the directions in this document to ensure ease of use and quality of service for merchants.