Virtual Device Documentation

v2.3.0

Overview

VirtualDeviceLib is a C# library (DLL) used to interact with a Fiscal Device Management System (FDMS). This library provides several public methods to fetch device configurations, check device status, manage fiscal days, and process Fiscal Documents. This documentation offers basic examples of how to load the library in a C# application, call its methods, and handle possible error scenarios.

Referencing

To include VirtualDeviceLib in your project, reference the DLL in your Visual Studio project.

1. Right-click on the **References** section in your project.
2. Click on **Add Reference**.
3. Browse to the location (preferably your application path) of the VirtualDeviceLib DLL and add it.

Usage Example

1. Loading and Using the Library

```
using VirtualDeviceLib;
using Newtonsoft.Json.Linq;

namespace TestVirtualDeviceApp
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create an instance of the library
            var virtualDevice = new VirtualDevice();
            // Example: Fetch Device Configurations
            string configResult = virtualDevice.GetConfig();
            Console.WriteLine($"Device Config Result: {configResult}");
            // Example: Check Device Status
            string statusResult = virtualDevice.GetStatus();
            Console.WriteLine($"Device Status Result: {statusResult}");
        }
    }
}
```

2. Standard Responses

Error Response

The Error response is indicated by **Code 0** and a **Message** indicating the error message.

```
{
  "Code": "0",
  "Message": "Error Message"
}
```

Success Response

The success is indicated by **Code 1**, a **Message** indicating the error message, and the **Data** section which might be Device information (for GetStatus), Device Status Data (for GetStatus), Z Report Data (for OpenFiscalDay and CloseFiscalDay), Receipt Data (for SubmitReceipt)

```
{
  "Code": "1",
  "Message": "Success Message",
  "QRCode": "",
  "FDMSInvoiceNo": "",
  "FiscalDayNo": "",
  "VerificationCode": "",
  "VerificationLink": "https://fdmstest.zimra.co.zw",
  "DeviceID": "18489",
  "Data": {...}
}
```

3. Public Methods

`GetConfig()`

- Fetches the device configurations from the FDMS server.
- **Returns**: Success message with configuration details or an error message.

Example:

```
string config = virtualDevice.GetConfig();
```

`GetStatus()`

- Retrieves the current status of the device from the FDMS server.
- **Returns**: Success message with status details or an error message.

Example:

```
string status = virtualDevice.GetStatus();
```

`OpenFiscalDay()`

- Opens a new fiscal day on the FDMS server, provided the last fiscal day is closed.
- **Returns**: Success message with the Z Report data for the previous fiscal day or an error message

Example:

```
string openFiscalDayResult = virtualDevice.OpenFiscalDay();
```

`CloseFiscalDay()`

- Closes the current fiscal day on the FDMS server.
- **Returns**: Success message with the Z Report data for the current fiscal day or an error message.

Example:

```
string closeFiscalDayResult = virtualDevice.CloseFiscalDay();
```

Core Receipt/ Invoice Submission Methods

SubmitReceipt

Submit a Fiscal Invoice, Credit Note, or Debit Note to the FDMS server. This is the primary method for processing fiscal transactions with comprehensive validation and error handling.

Signature:

```
SubmitReceipt(
    string receiptType,
    string receiptCurrency,
    string invoiceNo,
    string referenceNumber,
    decimal invoiceAmount,
    decimal invoiceTaxAmount,
    string receiptNotes,
    string receiptLines,
    bool receiptLinesTaxInclusive,
    string moneyTypeCode,
    string receiptPrintForm,
    string buyerRegisterName,
```

```
    string buyerTradeName,
    string vatNumber,
    string buyerTIN,
    string buyerPhoneNo,
    string buyerEmail,
    string buyerProvince,
    string buyerStreet,
    string buyerHouseNo,
    string buyerCity
)
```

- Submit a Fiscal Invoice, Credit Note, or Debit Note to the FDMS server.
- **Returns**: Success message with the QR Code and Receipt Information as the Data value or an Error Message.

Parameters

Parameter	Type	Required	Description	Validation Rules	Examples
`receiptType`	string	**Yes**	Type of fiscal document being submitted	Must be: FiscalInvoice, CreditNote, DebitNote	"FiscalInvoice"
`receiptCurrency`	string	**Yes**	ISO 4217 currency code for the transaction	Valid currency codes only	"USD", "ZWL", "GBP", "EUR", "ZAR"
`invoiceNo`	string	**Yes**	Unique invoice identifier	Max 50 chars, unique per taxpayer	"INV-2025-001", "SALE-240715-0001"
`referenceNumber`	string	Conditional	Original invoice number for Credit/Debit	Required for Credit/Debit Notes, empty for invoices	"INV-2025-001" (for credit note)
`invoiceAmount`	decimal	**Yes**	Total invoice amount excluding tax	Must match sum of line totals if inclusive (for exclusive must match sum of line totals + tax of items)	215.00M

`invoiceTaxAmount`	decimal	**Yes**	Total calculated tax amount	Must match calculated tax from lines	15.00M
`receiptNotes`	string	Conditional	Additional transaction notes	Mandatory for Credit/Debit Notes	"Return of defective goods"
`receiptLines`	string	**Yes**	JSON serialized array of line items	Valid JSON, at least one line required	See ReceiptLine structure
`receiptLinesTaxInclusive`	bool	**Yes**	Whether line prices include tax	Affects tax calculation logic	true or false
`moneyTypeCode`	string	**Yes**	Payment method identifier	Must be valid payment type	"Cash", "Card", "MobileWallet"
`receiptPrintForm`	string	**Yes**	Receipt print format	Controls output format	"Receipt48", "InvoiceA4"
`buyerRegisterName`	string	**Yes**	Customer's legal/registered name (Provide for VAT-registered customers)	Max 250 chars, cannot be empty	"John Doe", "ABC Trading Ltd"

`buyerTradeName`	string	Conditional	Customer’s trading/business name(Provide for VAT-registered customers)	Max 250 chars	"ABC Supermarket"
`vatNumber`	string	Conditional	Customer’s VAT registration number(Provide for VAT-registered customers)	9 digits for VAT registered entities	"123456789"
`buyerTIN`	string	Conditional	Customer’s Tax Identification Number(Provide for VAT-registered customers)	10 digits for companies	"1234567890"
`buyerPhoneNo`	string	Conditional	Customer’s contact phone number(Provide for VAT-registered customers)	Valid phone format	"0770000000"
`buyerEmail`	string	Conditional	Customer’s email address(Provide for	Valid email format	"customer@example.com"

			VAT-registered customers)		
`buyerProvince`	string	Conditional	Customer’s province/state(Provide for VAT-registered customers)	—	"Harare"
`buyerStreet`	string	Conditional	Customer’s street address(Provide for VAT-registered customers)	—	"Bargate Road, Vainona"
`buyerHouseNo`	string	Conditional	Customer’s house/building number(Provide for VAT-registered customers)	—	"60"
`buyerCity`	string	Conditional	Customer’s city(Provide for VAT-registered customers)	—	"Harare"

Payment Types (moneyTypeCode):

- `"Cash"` - Cash payment
- `"Card"` - Card payment (credit/debit)
- `"MobileWallet"` - EcoCash, OneMoney, etc.
- `"Coupon"` - Voucher/coupon payment
- `"Credit"` - Credit sale (payment later)
- `"BankTransfer"` - Direct bank transfer

Example:

```
    var myReceiptLines = new List<receiptLine>();

    myReceiptLines.Add(new receiptLine()
    {
        receiptLineType = "Sale", //Optional
        receiptLineNo = 1, //Optional
        receiptLineHSCode = "", //Optional
        receiptLineName = "Test Item 1",
        receiptLinePrice = 100.00M,
        receiptLineQuantity = 1,
        receiptLineTotal = 100.00M,
        taxCode = "B", //NonVatable item [B] Vatable Item [A]
        taxPercent = 0M, //NonVatable item [0] Vatable Item [15]
    });
    myReceiptLines.Add(new receiptLine()
```

```
    {
        receiptLineType = "Sale", //Optional
        receiptLineNo = 3, //Optional
        receiptLineHSCode = "", //Optional
        receiptLineName = "Test Item 2",
        receiptLinePrice = 100.00M,
        receiptLineQuantity = 1,
        receiptLineTotal = 100.00M,
        taxCode = "A", //NonVatable item [B] Vatable Item [A]
        taxPercent = 15.00M, //NonVatable item [0] Vatable Item [15]
    });
    string receiptLines = JsonConvert.SerializeObject(myReceiptLines);

    var submitReceipt = virtualDevice.SubmitReceipt("FiscalInvoice", "USD", "INV001", "REF000",
110.00M, 13.04M, "Test Invoice", receiptLines, true, "CASH", "InvoiceA4", "Axis Solutions Pvt Ltd",
"Axis Solutions", "123456789", "1234567890", "0770000000", "developers@axissol.com", "Harare",
"Bargate Road, Vainona", "60", "Harare");

//ReceiptLines Class
public class receiptLine
{
    public string receiptLineName { get; set; }
    public int receiptLineNo { get; set; }
    public decimal receiptLineQuantity { get; set; }
    public string receiptLineType { get; set; }
    public decimal receiptLineTotal { get; set; }
    public int taxID { get; set; }
    public string receiptLineHSCode { get; set; }
    public decimal receiptLinePrice { get; set; }
```

```
    public string taxCode { get; set; }
    public decimal taxPercent { get; set; }
}
```

GetLicense()

- Gets the device license status.
- **Returns**: Success message with the license details or an error message

Example:

```
string getLicenseResult = virtualDevice.GetLicense();
```

SubmitReceiptExt

Description: Extended version of SubmitReceipt that enables cross-device credit note processing within the same company. This method allows creating credit notes that reference invoices from different devices belonging to the same taxpayer, enabling centralized return processing in multi-location businesses.

Key Features:

- **Cross-Device References:** Create credit notes referencing invoices from other devices in your company
- **Enhanced Validation:** Additional validation for multi-device scenarios
- **Reference Tracking:** Improved tracking of cross-device transaction relationships
- **Centralized Processing:** Enables headquarters to process returns for branch transactions

Use Cases:

- Central office processing returns for branch sales
- Multi-location businesses with shared customer service
- Warranty returns processed at different locations
- Corporate credit note management across subsidiaries

Signature:

```
SubmitReceiptExt(
```

```
    // All standard SubmitReceipt parameters, plus:
    string refDeviceID,                // Device ID that created the original invoice
    string refReceiptGlobalnumber,     // ReceiptGlobalNumber for the Invoice to be credited/ debited
    string refFiscalDay                // Fiscal day Number for the original invoice
)
```

Invoice Query and Status Methods

GetInvoice

Description: Universal invoice retrieval method that searches for any transaction by invoice number across both processed and unprocessed records. This is the primary method for checking if an invoice exists in the system, regardless of its processing state.

Signature:

```
GetInvoice(string invoiceNumber)
```

Parameters:

Parameter	Type	Required	Description
`invoiceNumber`	string	**Yes**	The exact invoice number to search for

```
string result = virtualDevice.GetInvoice("INV-2025-001");
```

```
// Parse response to check if invoice exists
var response = JObject.Parse(result);
if (response["Code"].ToString() == "1")
{
    // Invoice found - access details
}
```

Unprocessed Transaction Management

GetUnProcessedInvoices

Description: Retrieves a paginated list of unprocessed transactions for a specific fiscal day. Essential for monitoring daily transaction processing and identifying bottlenecks.

Signature:

```
GetUnProcessedInvoices(string fiscalDayNumber, int page = 1, int pageSize = 50)
```

Parameters:

Parameter	Type	Required	Default	Description
`fiscalDayNumber`	string	**Yes**	-	Fiscal day identifier (e.g., "12")
`page`	int	No	1	Page number for pagination (starts from 1)

`pageSize`	int	No	50	Records per page (max recommended: 100)

Use Cases:

- Daily reconciliation processes
- Identifying processing bottlenecks
- Monitoring fiscal day completion
- Troubleshooting specific day issues

Example Usage:

```
// Get first page of unprocessed invoices for fiscal day
string result = virtualDevice.GetUnProcessedInvoices("12", 1, 25);


// Get all unprocessed invoices for the day (multiple pages)
int currentPage = 1;
bool hasMoreData = true;
while (hasMoreData)
{
    string pageResult = virtualDevice.GetUnProcessedInvoices("12", currentPage, 50);
    var response = JObject.Parse(pageResult);
    if (response["Code"].ToString() == "1" && response["Data"]["..."].Count() > 0)
    {
```

```
        // Process records
        currentPage++;
    }
    else
    {
        hasMoreData = false;
    }
}
```

GetUnProcessedInvoicesByDate

Description: Retrieves a paginated list of unprocessed transactions for a specific calendar date. Useful when fiscal days span multiple calendar dates or for date-based reporting.

Signature:

```
GetUnProcessedInvoicesByDate(string fiscalDate, int page = 1, int pageSize = 50)
```

Parameters:

Parameter	Type	Required	Default	Description
`fiscalDate`	string	**Yes**	-	Calendar date in YYYY-MM-DD format
`page`	int	No	1	Page number for pagination
`pageSize`	int	No	50	Records per page

Use Cases:

- Calendar-based reporting
- Cross-fiscal-day analysis
- Monthly/weekly reconciliation
- Transaction monitoring

Example Usage:

```
// Get unprocessed invoices for specific date
string result = virtualDevice.GetUnProcessedInvoicesByDate("2025-07-15", 1, 30);


// Date range processing
for (DateTime date = startDate; date <= endDate; date = date.AddDays(1))
{
    string dateStr = date.ToString("yyyy-MM-dd");
    string dayResult = virtualDevice.GetUnProcessedInvoicesByDate(dateStr);
    // Process each day's unprocessed invoices
}
```

GetUnProcessedInvoicesSummary

Description: Provides summary statistics and metrics for unprocessed transactions. Can be filtered by fiscal day or date to get targeted insights into processing backlogs.

Signature:

```
GetUnProcessedInvoicesSummary(string fiscalDayNumber = null, string fiscalDate = null)
```

Parameters:

Parameter	Type	Required	Default	Description
`fiscalDayNumber`	string	No	null	Filter by specific fiscal day

`fiscalDate`	string	No	null	Filter by specific calendar date

Returns Summary Metrics:

- Total unprocessed count
- Total amount pending
- Oldest pending transaction
- Processing queue depth

Use Cases:

- Performance monitoring

Example Usage:

```
// Get summary for specific fiscal day

string daySummary = virtualDevice.GetUnProcessedInvoicesSummary("12");



// Get summary for specific date

string dateSummary = virtualDevice.GetUnProcessedInvoicesSummary(null, "2025-07-15");



// Parse and use summary data
```

```
var summary = JObject.Parse(overallSummary);

if (summary["Code"].ToString() == "1")

{

    // Process

}
```

ClearUnprocessedInvoices

Description: Performs a soft delete of all unprocessed transactions for a specific fiscal day to allow affected transactions to be processed and submitted to the FDMS. This operation marks transactions as deleted without physically removing them, allowing for data recovery if needed.

⚠️ **IMPORTANT:** This is a destructive operation that should only be used with proper authorization and confirmation of transactions affected by the FDMS outage..

Signature:

```
ClearUnprocessedInvoices(string fiscalDayNumber)
```

Parameters:

Parameter	Type	Required	Description

`fiscalDayNumber`	string	**Yes**	Fiscal day to clear (e.g., "12")

Use Cases:

- Cleanup operations to fiscalise unprocessed invoice

Example Usage:

```
// Safety check before clearing
string summary = virtualDevice.GetUnProcessedInvoicesSummary("12");
var summaryData = JObject.Parse(summary);

if (summaryData["Code"].ToString() == "1")
{
    int pendingCount = (int)summaryData["Data"]["TotalUnprocessedTransactions"];

    // Confirm clearing operation
    Console.WriteLine($"About to clear {pendingCount} unprocessed invoices for 12");
    Console.WriteLine("Are you sure? (yes/no)");

    string confirmation = Console.ReadLine();
    if (confirmation.ToLower() == "yes")
    {
```

```
        string result = virtualDevice.ClearUnprocessedInvoices("12");
        Console.WriteLine($"Clear operation result: {result}");
    }
}
```

ClearUnprocessedInvoicesByDate

Description: Performs a soft delete of all unprocessed transactions for a specific calendar date. Similar to ClearUnprocessedInvoices, but operates on calendar date rather than fiscal day.

Signature:

```
ClearUnprocessedInvoicesByDate(string fiscalDate)
```

Parameters:

Parameter	Type	Required	Description
`fiscalDate`	string	**Yes**	Calendar date to clear (YYYY-MM-DD format)

Use Cases:

- Cleanup operations to fiscalise unprocessed invoices.

Example Usage:

```
// Clear specific date with safety checks
string dateToaClear = "2025-07-15";
// First, get summary for the date
string summary = virtualDevice.GetUnProcessedInvoicesSummary(null, dateToaClear);
var summaryData = JObject.Parse(summary);
if (summaryData["Code"].ToString() == "1")
{
    // Proceed with clearing
    string result = virtualDevice.ClearUnprocessedInvoicesByDate(dateToaClear);
    // Log the operation
```

```
    Console.WriteLine($"Cleared unprocessed invoices for {dateToaClear}: {result}");
}
```

4. Error Guide

---

1. `GetConfig()`

Common Error Messages:

- **Error**: "Failed to get Device Configurations"
  - **Description**: Could not fetch the device configurations from the server.
  - **Possible Fixes**:
    - Ensure the device is connected to the FDMS server.
    - Check network connectivity between the device and the server.
    - Verify that the correct device ID is being used.

2. `GetStatus()`

Common Error Messages:

- **Error**: "Failed to get Device Status"
    - **Description**: Could not fetch the current device status from the FDMS server.
    - **Possible Fixes**:
        - Ensure the device is properly connected to the network.
        - Check if the FDMS server is reachable.
        - Verify if the device configuration is correct.

3. `OpenFiscalDay()`

Common Error Messages:

- **Error**: "Failed to open a new Fiscal Day"
    - **Description**: A fiscal day is already open, and the system is trying to open another one.
    - **Possible Fixes**:
        - Ensure the previous fiscal day is closed before attempting to open a new one.
        - Check the status of the device to verify if the day is open or closed.

- **Error**: "Failed to get Fiscal Day Status"
    - **Description**: Unable to retrieve the current fiscal day status from the server.
    - **Possible Fixes**:
        - Ensure the server is reachable and the device is connected.
        - Review the network configuration and try fetching the fiscal day status again.

- **Error**: "Fiscal Day No: {number} is already open"
    - **Description**: The system detected that a fiscal day is already open, preventing a new day from being opened.
    - **Possible Fixes**:
        - Close the currently open fiscal day before opening a new one.

4. `CloseFiscalDay()`

Common Error Messages:

- **Error**: "Failed to close Fiscal Day"
    - **Description**: The current fiscal day could not be closed due to some issues.
    - **Possible Fixes**:
        - Ensure there are no pending transactions or processes that need to be completed.
        - Check if the device is properly connected to the FDMS server.

- **Error**: "Failed to get the Current Fiscal Day Status"
    - **Description**: Unable to retrieve the current fiscal day status.
    - **Possible Fixes**:
        - Ensure the FDMS server is reachable and the device is online.

- **Error**: "Fiscal Day No: {number} is already closed"
    - **Description**: The system detected that the current fiscal day is already closed, and thus, no further action is required.
    - **Possible Fixes**:
        - Ensure that no duplicate actions are being performed, and confirm the fiscal day status before closing.

5. `SubmitReceipt(arg1,..., argn)`

Common Error Messages:

Error: "Device License Error - Device Not Licensed"

Description: The fiscalisation process failed because the device is not licensed.
Possible Fixes:

- Ensure that the device has a valid license installed.
- Verify the license payment status with the license provider.

Error: "Device License Error - License Status: {status}"

Description: The fiscalisation process failed due to an issue with the license status.
Possible Fixes:

- Check the license status for further details.
- Contact support if the license status is unclear or incorrect.

Error: "The current Fiscal Day has exceeded the maximum hours limit"

Description: Fiscalisation failed because the current fiscal day is beyond the maximum allowable time.
Possible Fixes:

- Close the current fiscal day and open a new one to continue transactions.
- Check the system clock and ensure it is correctly synchronized.

Error: "Invoice Number already exists in your fiscal records"

Description: The provided invoice number is already recorded in the system.
Possible Fixes:

- Use a unique invoice number for each transaction.
- Review existing fiscal records for the invoice number in question.

Error: "Fiscal Day is currently closed"

Description: The fiscal day has been closed, preventing further transactions until a new day is opened.
Possible Fixes:

- Open a new fiscal day to process further transactions.

- Confirm the status of the fiscal day before attempting operations.

Error: "Invalid ReceiptPrintForm {receiptPrintForm}"

Description: The specified receipt print form is invalid.
Possible Fixes:

- Verify the receipt print form against the allowed formats.
- Update the print form configuration if necessary.

Error: "Buyer TIN {buyerTIN} is not a valid TIN"

Description: The Tax Identification Number (TIN) provided for the buyer is invalid.
Possible Fixes:

- Check the TIN for accuracy and compliance with regulatory standards.
- Validate the TIN through official channels.

Error: "FDMS VAT Amount: {taxAmount} Is not equal to the invoice Tax Amount: {invoiceTaxAmount}"

Description: The VAT amount from the FDMS does not match the expected invoice tax amount.
Possible Fixes:

- Ensure that the correct VAT rates are applied to the invoice.
- Review the calculation of the invoice tax amount for accuracy.

Error: "Invalid Receipt lines provided {receiptLines}"

Description: The receipt lines provided do not meet the necessary criteria for processing.
Possible Fixes:

- Check that all required fields for receipt lines are correctly filled out.
- Ensure receipt lines correspond to valid products/services.

Error: "Amount {invoiceAmount} Exceeds the original Invoice Amount"

Description: The specified amount exceeds the original invoice amount recorded in the system.
Possible Fixes:

- Ensure that the transaction amount is within the original invoice limit.
- Review the invoice and transaction amounts for discrepancies.

Error: "Currency {receiptCurrency} does not match original invoice currency"

Description: The currency used in the current transaction does not match that of the original invoice.
Possible Fixes:

- Confirm that the correct currency is used in the transaction.
- Update the currency settings in the system if necessary.

Error: "Invoice Number {invoiceNo} does not exist in your fiscal records"

Description: The provided invoice number cannot be found in the fiscal records.
Possible Fixes:

- Double-check the invoice number for accuracy.
- Review fiscal records to ensure the invoice was recorded correctly.

Error: "Error saving transaction to Queue {Message}"

Description: The system encountered an error while trying to save the transaction to the queue.
Possible Fixes:

- Review system logs for more detailed error information.
- Ensure the transaction data is valid and complete.

Error: "Invalid Receipt Currency. Valid Currency options: {allowedCurrencies)}"

Description: The receipt currency provided is invalid or not among the allowed currencies.
Possible Fixes:

- Confirm that the receipt currency is among the supported options.
- Update the currency configuration if necessary.

Error: "Invalid Invoice Amount - Invoice Amount: {invoiceAmount} must be equal to sum of all Line Item Amounts: {totalReceiptLineTotal}"

Description: The invoice amount must equal the total of all line item amounts provided.
Possible Fixes:

- Check the calculations for line item amounts to ensure accuracy.
- Adjust the invoice amount to match the sum of line items.

Error: "Invalid Invoice Amount - Invoice Amount: {invoiceAmount} must be equal to sum of all Line Item Amounts plus sum of all Line Item VAT Amounts: {amount}"

Description: The invoice amount must equal the sum of line item amounts plus the VAT amounts.
Possible Fixes:

- Review the line item and tax calculations for accuracy.
- Adjust the invoice total to ensure it matches the expected sums.

Error: "Invalid receiptTotal - receiptTotal must be greater than 0 for FiscalInvoice and DebitNote."

Description: The total amount for the receipt must be positive for fiscal invoices and debit notes.
Possible Fixes:

- Review the receipt total and ensure it is correctly calculated.
- Verify the type of transaction being processed.

1. `GetLicense()`

Common Error Messages:

- **Error**: "License Status: Device Not Licensed"
    - **Description**: Could not fetch the device license.
    - **Possible Fixes**:
        - Ensure the device license is paid and activated.
        - Open fiscal day to load license if it has been recently paid or activated.