

# Oracle® MICROS Symphony

## Transaction Services API Document



Release 19.2  
F32377-02  
April 2022



Copyright © 2010, 2022, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

# Contents

Preface	v
<b>1 Introduction</b>	<b>1-1</b>
Symphony Architecture	1-1
Hosting Method	1-3
Quick Installation Roadmap	1-5
Error Logging	1-6
TS API Class Hierarchy	1-6
<b>2 Transaction Operations</b>	<b>2-1</b>
Calculate Totals	2-1
Post Transaction	2-2
Add to Transaction	2-4
Void Transaction	2-6
Check Print Job Status	2-6
<b>3 Guest Check Operations</b>	<b>3-1</b>
Get Check Summary	3-1
Get Check Detail	3-1
Get Printed Check	3-2
<b>4 Configuration Operations</b>	<b>4-1</b>
Get Configuration Information	4-1
<b>5 Fiscal Operations</b>	<b>5-1</b>
Sanity Check	5-1
<b>6 Structure Reference</b>	<b>6-2</b>
SymphonyPosAPI_CheckSummary	6-2
SymphonyPosAPI_CheckSummaryEx	6-3
SymphonyPosAPI_OpenChecks	6-4
SymphonyPosAPI_GuestCheck	6-4
SymphonyPosAPI_CheckRequest	6-5
SymphonyPosAPI_CheckResponse	6-6
SymphonyPosAPI_CheckDetailRequest	6-6
SymphonyPosAPI_CheckDetailResponse	6-6
SymphonyPosAPI_MenuItem	6-6
SymphonyPosAPI_MenuItemEx	6-7
SymphonyPosAPI_MenuItemDefinition	6-7
SymphonyPosAPI_MenuItemDefinitionEx	6-9
SymphonyPosAPI_CombiMeal	6-10
SymphonyPosAPI_CombiMealEx	6-11
SymphonyPosAPI_Discount	6-11
SymphonyPosAPI_DiscountEx	6-11
SymphonyPosAPI_SvcCharge	6-11
SymphonyPosAPI_SvcChargeEx	6-12
SymphonyPosAPI_TmedDetailItemEx	6-12
SymphonyPosAPI_TmedDetailItemEx2	6-12
SymphonyPosApi_Extensibility	6-13

SimphonyPosAPI_EPayment	6-13
SimphonyPosAPI_TotalsResponse	6-15
SimphonyPosAPI_TotalsResponseEx	6-15
SimphonyPosAPI_ConfigInfoRequest	6-16
SimphonyPosAPI_ConfigInfo	6-16
SimphonyPosAPI_ConfigInfoResponse	6-16
SimphonyPosAPI_CheckPrintResponse	6-19
SimphonyPosAPI_PrintJobStatus	6-19
SimphonyPosAPI_OperationalResult	6-20
SimphonyPosAPI_SanityCheckResponse	6-21
CheckTaxDataPerRate	6-21
<b>7 Example and Code Snippets</b>	<b>7-1</b>
<hr/>	
Calculate Totals of a Transaction	7-1
Create a Guest Check	7-12
Add an Item to an Open Guest Check	7-17
Void All Items of an Open Guest Check	7-18
Get Status of a Print Job	7-19
Get Summary of All Open Guest Checks	7-20
Get Open Guest Checks with RVC Object Number	7-22
Get Open Guest Checks from a Specific RVC	7-23
Get Summary and KDS Order Status of Open and Closed Guest Checks	7-24
Get Check Detail	7-26
Get Printed Texts of a Guest Check	7-27
Get Configured Information (Method 1 - GetConfigurationInfo)	7-28
Get Configured Information (Method 2 - GetConfigurationInfoEx)	7-30
<b>8 Simphony Platform Requirements</b>	<b>8-1</b>
<hr/>	
Simphony Software Version	8-1
Offline Transaction Support	8-1
Printing Services	8-1
Calling Conventions	8-1
<b>9 Demo Client for Transaction Services API</b>	<b>9-1</b>
<hr/>	
Application Path	9-1
Prerequisites	9-1
Initial Setup	9-1
Demonstration	9-2
<b>10 Extending Transaction Services Through Plug-ins</b>	<b>10-1</b>
<hr/>	

# Preface

This document is intended for use by software engineers developing applications that interface with Symphony using Transaction Services (TS) API.

## Purpose

Introduction to the document.

## Audience

This document is intended for the following audiences:

- Installers/Programmers
- Dealers
- Customer Service
- Training Personnel
- MIS or IT Personnel

## Prerequisite Knowledge

This document assumes the reader has the following knowledge or expertise:

- Operational understanding of PCs
- Understanding of basic network concepts
- Experience in configuring workstation clients in the Symphony EMC

## Glossary

Acronym/Abbreviation	Full Text
API	Application Programming Interface
CAL	Client Application Loader
CAPS	Check and Posting Service
DB	Database
EMC	Enterprise Management Console
IIS	Microsoft Internet Information Services
OPS	Operations Software (POS Client)
PC	Personal Computer
PDA	Personal Digital Assistance
POS	Point of Sale
RVC	Revenue Center
SQL	Structured Query Language
SVC	Stored Value Card
TS	Transaction Services

## Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

<https://support.oracle.com>

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)
- Detailed step-by-step instructions to re-create
- Exact error message received and any associated log files
- Screenshots of each step you take

## Documentation

Oracle Food and Beverage product documentation is available on the Oracle Help Center at <https://docs.oracle.com/en/industries/food-beverage/pos.html>

## Revision History

Date	Description of Change
November 2020	<ul style="list-style-type: none"><li>• Initial publication.</li></ul>
April 2022	<ul style="list-style-type: none"><li>• Updated the Symphony Architecture section in Chapter 1.</li></ul>

# 1

## Introduction

Transaction Services (TS) API is a web service that helps integrate a third-party application with the Symphony Point-Of-Sale (POS) system. It leverages core business functionalities of the Symphony POS and exposes them via web methods for third-party integration. This web service provides methods to perform the following POS operations.

1. Calculate total amounts of a transaction
2. Create a guest check for a transaction in the Symphony POS database
3. Add one or more items, such as menu, discount, service charge, tender, and so on to any existing open guest check
4. Void all items of an open guest check
5. Retrieve a summary of all open guest checks from a specific or all revenue centers of a property
6. Retrieve a printed version of a guest check (print receipt for a guest check)

All guest checks posted via Transaction Services API can be opened with the POS client user interface, which is configured to run on any workstation from the same property.

The following are sample business scenarios in which Transaction Services API can be used to integrate a third-party application with the Symphony POS.

- Remote ordering from a kiosk or mobile phone application
- Remote ordering or centralized order dispatch via a web application
- Guest payment approval using mobile phones or PDAs

Transaction Services web service is installed on each workstation where the Symphony CAL package is executed. The CAL package downloads the required file contents from the Symphony application server to the actual workstation and installs it. As part of the installation, the CAL package creates a shortcut to ServiceHost.exe on the desktop. Launching ServiceHost.exe ensures that the TS web service is hosted (along with other services and the POS client user interface), and is ready to process requests from any client.

## Simphony Architecture

The POS client provides a user interface for all POS operations in the Symphony system. The order taker can log on to the POS client using valid credentials and add a transaction to create a guest check. As part of a transaction, he or she can add menu items, discounts, service charges, multiple tax rates, and tender details to the guest check and save it to the POS database. The guest checks that are created on one workstation of the property can be accessed from other workstations of the same property using the CAPS (Check and Posting) service.

TS web service uses the same underlying business libraries that are used by the POS client for all POS operations. Technically TS web service is a version of the POS client without the UI.

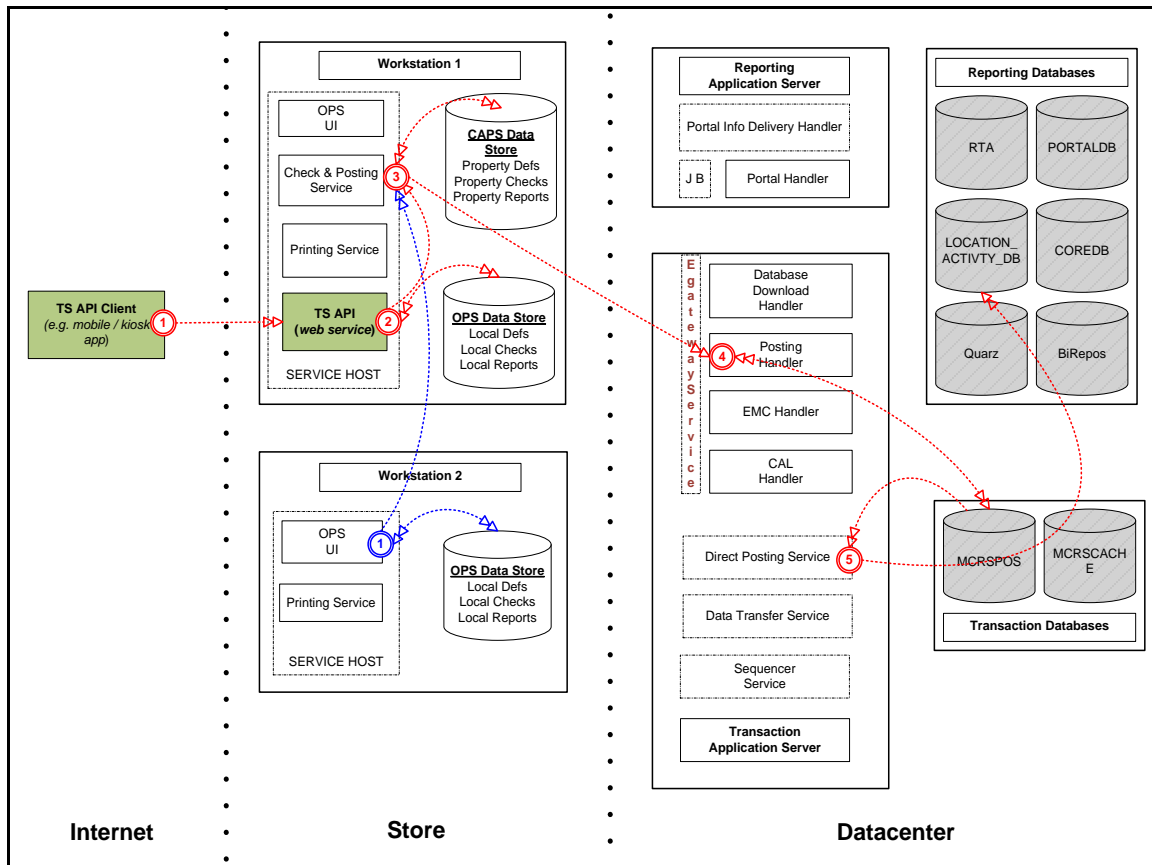
The guest check created through the Transaction Services API posts to the enterprise database (MCRSPOS) for reporting purposes via CAPS. All transactions made through the TS web service or POS client UI is first stored in the local POS database named DataStore. The CAPS which owns the CheckAndPostingDB database posts all transactional data to the enterprise database (MCRSPOS) with the help of the EGateway service, which is hosted on the Symphony application server.

All configuration changes made by the EMC application at the enterprise server are downloaded to the local POS database of each workstation using a DB Sync component. By default, the DB Sync operation runs every 30 minutes. Thus, any configuration changes made at the enterprise server for a specific property or workstation will be available for Transaction Services and the POS client within 30 minutes. The DB Sync can also be manually initiated anytime using a function button in the POS client.

The following image shows the Symphony system architecture, including Transaction Services API/web service. The image depicts how data flows from a TS client application to the reporting database.



Figure 1-1 - Simphony Architecture



Direct Posting Services is a Windows service that runs on the application server to upload data from the transaction database (MCRSPOS) to one of the reporting databases named LOCATION\_ACTIVITY\_DB. Any workstation in a store can be configured to host TS API. In this example, Workstation 1 hosts the TS API.

## Hosting Method

The Transaction Services web service is pre-installed on each workstation or application server where the Simphony installation media or CAL package was executed. Configuring a proper workstation client of type POS API from the EMC application will allow for the successful hosting of Transaction Services web service at the POS workstation.

The Service Host, which is configured via EMC to run on a workstation, hosts the Transaction Services API as a web service by default at port number 8080. The format of the web service URL is:

<http://<<WorkstationIPAddress>>:8080/EGateway/SimphonyPosApiWeb.asmx>

Any client machine that has access to a given POS workstation can consume Transaction Services by referring to the correct URL of the TS web service.

To configure a POS API client for the TS web service:

1. Log on to the EMC application.

2. Select a property to configure.
3. Click the **Setup** tab.
4. Click the **Workstation** link under Hardware/Interfaces.
5. Insert a new workstation as a POS API client (for example, TS API).
6. Double-click the record to open the created workstation in form view.
7. On the **General** tab under the General Settings section, select **3 - POSAPI Client** from the **Type** drop-down list. This enables the workstation to run only TS inside the Service Host process without the POS client.
8. If the POS client needs to run on the same workstation that is run by a Service Host, you need to create two workstations: one for the POS workstation client and another for the TS client type. From the **Service Host** link, select the Service Host of the POS client when the drop-down list appears during Pos Api client WS type selection. This ensures that there will be two workstation identities - one WorkstationID for the POS client and another API\_Workstation ID for the TS workstation.
9. Click **Save**.

The screenshot shows the configuration form for a workstation. The 'General Settings' section is highlighted with a red box around the 'Workstation ID' field, which is labeled 'Api\_WorkstationID'. The 'Service Host Fields' section is also highlighted with a red box around the 'Service Host ID' field, which is labeled '92 - WS\_OPS'. A red note above the 'Service Host ID' field states: 'OPS WS and TS WS should use the same Service Host ID'.

To confirm that the TS web service is hosted correctly:

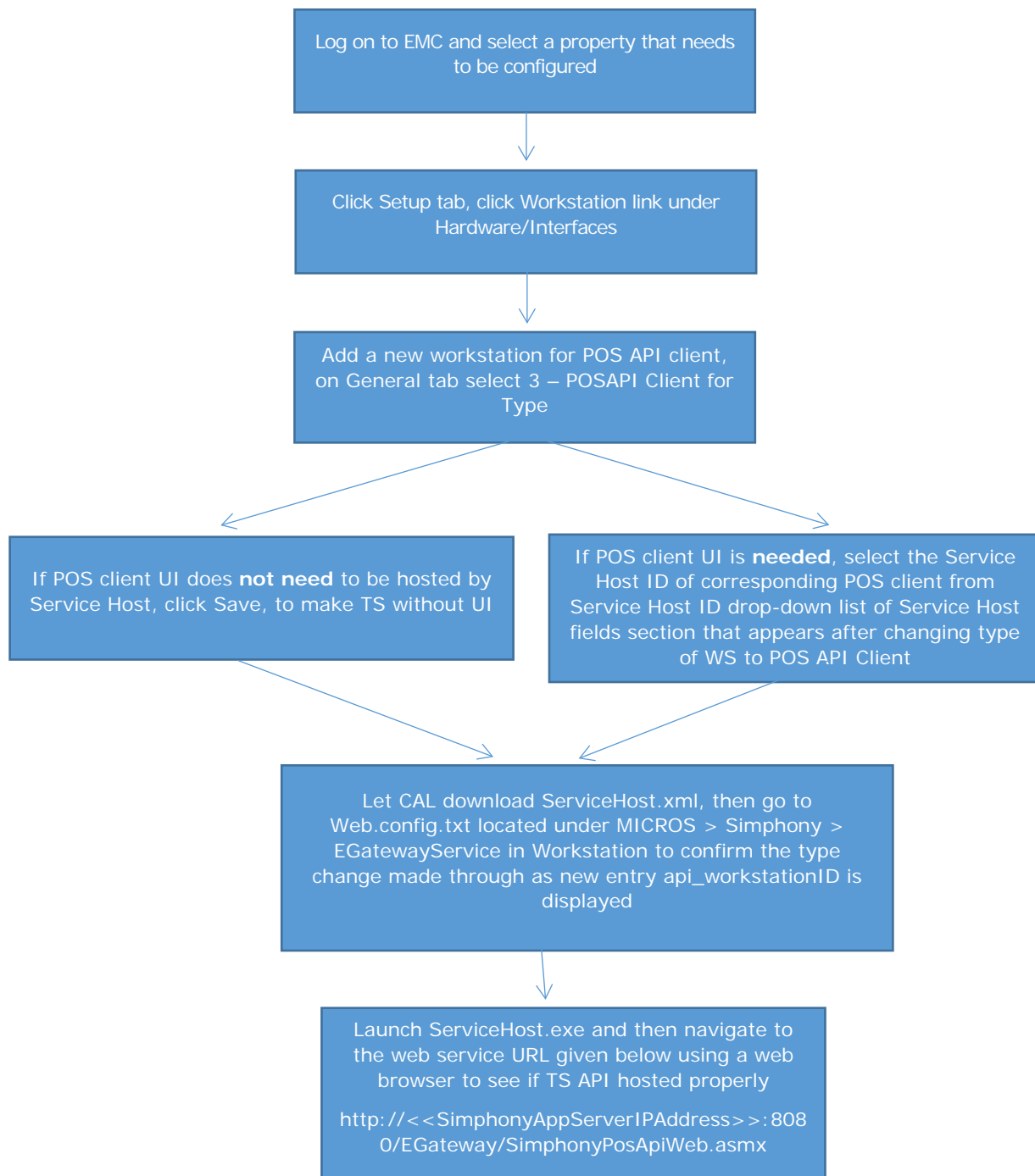
1. Launch **ServiceHost.exe** on the POS workstation.
2. Go to the web service URL above using any web browser. If WSDL details are shown in the web browser, the web service is hosted properly.

Apart from this method (that is, hosting TS web service on Service Host), TS web service can be hosted on the application server as well, but it will never be applied because the application server is beyond the boundary of third-party clients. By default, the Symphony application server has the TS web service installed and hosted after successful execution of Symphony installation media. This web service can be accessed with the right URL to the application server. It is normally hosted on the following URL. Replace the placeholder with the IP address of the application server below.

<http://<SymphonyAppServerIPAddress>:8080/EGateway/SymphonyPosApiWeb.asmx>

To create a stub or proxy or WSDL for the client application to integrate with the TS web service, software engineers can add a reference to this web service. Later, the URL can be changed to point at the instance that is hosted on the workstation.

## Quick Installation Roadmap



## Error Logging

Transaction Services API writes all errors and other informational messages to a flat file in the following folder on the POS workstation for diagnostics purposes. "Simphony Pos Api:" is appended before any logs in TS.

```
<ROOT_INSTALL_DRIVE>\MICROS\Simphony\WebServer\wwwroot\EGateway\EgatewayLog\
```

Example path: C:\MICROS\Simphony\WebServer\wwwroot\EGateway\EgatewayLog\

## TS API Class Hierarchy

**Table 1-1 - Interfaces Table**

Interface	Description
ITransactionServices Interface	The main interface that supports transaction related operations (for example, calculating transaction totals, posting a transaction to create a guest check in the POS database, adding items to an existing guest check, voiding a transaction/check, and retrieving status of any print job).
IGetCheckInfo Interface	Interface that provides support for fetching all open/closed guest checks, retrieving printed lines of guest check and configuration related information from the POS database.
SimphonyPosAPI_CheckSummary Structure	Structure that defines summary of a guest check.
SimphonyPosApi_CheckSummaryEx Structure	This structure derives from SimphonyPosAPI_CheckSummary and holds a couple of fields on KDS order status.
SimphonyPosAPI_OpenChecks Structure	Structure that represents check summary of all open checks.
SimphonyPosAPI_GuestCheck Structure	Structure that defines guest check details such as Check ID, Order Type, Guest Count, Table Number, and so on.
SimphonyPosApi_CheckRequest Structure	Structure that defines input parameters required to call GetChecks web method.
SimphonyPosApi_CheckResponse Structure	Structure that holds the response of GetChecks web method.
SimphonyPosApi_CheckDetailRequest Structure	Structure that defines input parameters required to call GetCheckDetail web method.

<b>Interface</b>	<b>Description</b>
SimphonyPosApi_CheckDetailResponse Structure	Structure that holds the response of GetCheckDetail web method.
SimphonyPosAPI_MenuItem Structure	Structure that holds the definition of menu item and its condiments.
SimphonyPosAPI_MenuItemDefinition Structure	Structure that defines details of a menu item, such as menu item object number, price, discount, and so on.
SimphonyPosAPI_ComboMeal Structure	Structure that defines a Combo Meal (main and side items).
SimphonyPosAPI_Discount Structure	Structure used to represent a discount in the Symphony POS system.
SimphonyPosAPI_SvcCharge Structure	Structure used to represent a Service Charge in the Symphony POS system. This has details such as service charge amount or percentage.
SimphonyPosAPI_EPayment Structure	Structure that defines Advanced Electronic Payment details such as credit card account, tip amount, cash back amount, and so on.
SimphonyPosAPI_TmedDetailItemEx Structure	Structure to represent a tender media which has mode of payment.
SimphonyPosAPI_TotalsResponse Structure	Structure that holds various totals, such as subtotal, due amount, tax amount and automatic service charges of a transaction.
SimphonyPosApi_ConfigInfoRequest Structure	Structure that defines input parameters required to call GetConfigurationInfoEx web method.
SimphonyPosApi_ConfigInfo Structure	Structure that holds filter criteria to be used to retrieve configuration data using GetConfigurationInfoEx method.
SimphonyPosApi_ConfigInfoResponse Structure	Structure that holds configuration details of items, such as menu item definitions, menu item price, currency, discounts, employees, order type, revenue centers, tender media, service charge, and so on.
SimphonyPosApi_CheckPrintResponse Structure	Structure that holds the response of Get Printed Check method call. This structure holds operation (success/failure) results along with printed check lines.
SimphonyPrintApi_PrintJobStatus Structure	Structure that holds the response of Get Printed Job Status method call. This structure holds the operation result (success/failure) along with details of print

---

Interface	Description
SymphonyPosAPI_OperationalResult Structure	jobs and status code and error/success message.
SymphonyPosApi_SanityCheckResponse Structure	Structure used to represent result (success or failure) of a method call. In case of failure, this structure will provide error code along with error message that tells the cause of failure.
	Structure that holds the response of the Sanity Check method call. This structure holds two sets of results (success or failure, in case of failure with error code and message). One result for the operational aspect of the method call, and the other for the sanity check validation, which is performed based on the provided sanity code.

---

# 2

## Transaction Operations

TS API provides several transaction related POS operations. As the parameters required by an operation are updated, the operation name is extended by adding to the operation name. For example *CalculateTransactionTotals* and *CalculateTransactionTotalsEx* both perform the same operation; however, the newer “Ex” method accepts different parameters to support additional functionality.

### Calculate Totals

The calculate totals operation is used to compute totals of an order without creating a guest check in the Symphony POS database.

The following Calculate Totals method is available in Symphony version 2.7 and later.

```
void CalculateTransactionTotals
(
    string                vendorCode,
    ref SymphonyPosApi_MenuItem[] ppMenuItems,
    ref SymphonyPosApi_ComboMeal[] ppComboMeals,
    ref SymphonyPosApi_SvcCharge pSvcCharge,
    ref SymphonyPosApi_Discount pSubtotalDiscount,
    int                    revenueCenter,
    short                  orderType,
    int                    employeeNumber,
    ref SymphonyPosApi_TotalsResponse pTotalsResponse
)
```

The following Calculate Totals method is available beginning with Symphony version 18.2. This version of the method has updated structures that add support for multiple discounts, the ability to specify menu item quantity and definition sequence, and specify extension data with menu items.

```
void CalculateTransactionTotalsEx
(
    ref SymphonyPosApi_MenuItemEx[] ppMenuItemsEx,
    ref SymphonyPosApi_ComboMealEx[] ppComboMealsEx,
    ref SymphonyPosApi_SvcChargeEx pSvcChargeEx,
    ref SymphonyPosApi_DiscountEx[] pSubTotalDiscountEx,
    int                    revenueCenterObjectNum,
    short                  orderType,
    int                    employeeObjectNum,
    int                    checkGuestcount,
    ref SymphonyPosApi_TotalsResponseEx pTotalsResponseEx
)
```

## Parameters

<b><i>vendorCode</i></b>	Vendor code for license validation. In Symphony version 2.7 MR3 or later, set this parameter to an empty string.
<b><i>revenueCenterObjectNum</i></b>	Object number of given revenue center.
<b><i>orderType</i></b>	Number of the order type for the transaction (for example, Dine-in or Carry-out).
<b><i>employeeObjectNum</i></b>	Object number of employee authorized to perform the operation.
<b><i>checkGuestCount</i></b>	The number of guests on the guest check. This value is used to compute automatic service charges.

The guest check is defined by creating a set of menu items, combo meals, service charge, and discount. Refer to the Structure Reference chapter for details on how they are specified.

## Response

The method calculates discounts (subtotal, item, and automatic discounts), service charges and taxes. The results of the calculations are returned in the same structures used in the request parameters and indicate the price of items and the results of applying discounts and service charges.

The *pTotalsResponse/Ex* parameter contains the success or failure result of operations, results of tax calculations, and summary totals.

## Post Transaction

The post transaction operation is used to create a new guest check.

The following post transaction method is available in Symphony version 2.7 and later.

```
void PostTransactionEx
(
    string vendorCode,
    ref SymphonyPosApi_GuestCheck pGuestCheck,
    ref SymphonyPosApi_MenuItem[] ppMenuItems,
    ref SymphonyPosApi_ComboMeal[] ppComboMeals,
    ref SymphonyPosApi_SvcCharge pServiceChg,
    ref SymphonyPosApi_Discount pSubTotalDiscount,
    ref SymphonyPosApi_TmedDetailItemEx pTmedDetail,
    ref SymphonyPosApi_TotalsResponse pTotalsResponse,
    ref string[] ppCheckPrintLines,
    ref string[] ppVoucherOutput
)
```

The following Post Transaction method is available beginning with Symphony version 18.2. This version of the method has updated structures that add support for multiple



discounts, the ability to specify menu item quantity and definition sequence, and specify extension data with menu items.

```

void PostTransactionEx2
(
  ref SymphonyPosApi_GuestCheck          pGuestCheck,
  ref SymphonyPosApi_MenuItemEx[]       ppMenuItemsEx,
  ref SymphonyPosApi_ComboMealEx[]      ppComboMealsEx,
  ref SymphonyPosApi_SvcChargeEx        pSvcChargeEx,
  ref SymphonyPosApi_DiscountEx[]       pSubTotalDiscountEx,
  ref SymphonyPosApi_TmedDetailItemEx2[] pTmedDetailEx2,
  ref SymphonyPosApi_TotalsResponseEx   pTotalsResponseEx,
  ref string[]                          ppCheckPrintLines,
  ref string[]                          ppVoucherOutput,
  ref SymphonyPosApi_Extensibility[]     checkExtensibilityDetails
)

```

## Parameters

<b>vendorCode</b>	Vendor code for license validation. In Symphony version 2.7 MR3 or later, set this parameter to an empty string.
<b>revenueCenterObjectNum</b>	Object number of given revenue center.
<b>orderType</b>	Number of the order type for the transaction (for example, Dine-in or Carry-out).
<b>employeeObjectNum</b>	Object number of employee authorized to perform the operation.

The guest check is defined by creating a set of menu items, combo meals, service charges, and discounts. The *pGuestCheck* parameter is used to specify additional attributes for the guest check, Guest Count, Check ID, Event Number, and so on.

A tender media is required. If a tender media of type Service Total is supplied, the guest check is created in the open state. If a tender media of type Payment is supplied, the check is closed.

Refer to the Structure Reference chapter for details on how they are specified.

## Response

The guest check is printed to the local and remote order devices defined for the POS API workstation. The printed check details are returned in the *ppCheckPrintLines* parameter, while credit voucher details are filled in the *ppVoucherOutput* parameter.

The price of menu items, discounts, and service charges are returned in the same structures used to create the request. The *pGuestCheck* and *pTotalsResponse/Ex* parameter contains a summary of totals.

The system does not create the guest check if payment or another interim operation fails. Inspect the *OperationalResult* property of the *pGuestCheck* structure in the response to verify that the operation has completed successfully.

When the operation completes successfully, the `CheckNum` and `CheckSeq` properties are set on the `pGuestCheck` parameter. These values are used to reference the guest check in subsequent operations.

## Add to Transaction

This operation is used to add items or payments to a guest check previously created using a post transaction operation. This operation can be used to

- add one or more menu items or combo meals to an existing open guest check
- apply partial or full payment on an existing open guest check
- applied a coupon discount to an existing open guest check

The following post transaction method is available in Symphony version 2.7 and later.

```
void AddToExistingCheckEx
(
    string                vendorCode,
    ref SymphonyPosApi_GuestCheck pGuestCheck,
    ref SymphonyPosApi_MenuItem[] ppMenuItems,
    ref SymphonyPosApi_ComboMeal[] ppComboMeals,
    ref SymphonyPosApi_SvcCharge pServiceChg,
    ref SymphonyPosApi_Discount pSubTotalDiscount,
    ref SymphonyPosApi_TmedDetailItemEx pTmedDetail,
    ref SymphonyPosApi_TotalsResponse pTotalsResponse,
    ref string[] ppCheckPrintLines,
    ref string[] ppVoucherOutput
)
```

The following post transaction method is available beginning with Symphony version 18.2. This version of the method has updated structures that add support for multiple discounts, the ability to specify menu item quantity and definition sequence, and specify extension data with menu items.

```
void AddToExistingCheckEx2
(
    ref SymphonyPosApi_GuestCheck pGuestCheck,
    ref SymphonyPosApi_MenuItemEx[] ppMenuItemsEx,
    ref SymphonyPosApi_ComboMealEx[] ppComboMealsEx,
    ref SymphonyPosApi_SvcChargeEx pSvcChargeEx,
    ref SymphonyPosApi_DiscountEx[] pSubTotalDiscountEx,
    ref SymphonyPosApi_TmedDetailItemEx2[] pTmedDetailEx2,
    ref SymphonyPosApi_TotalsResponseEx pTotalsResponseEx,
    ref string[] ppCheckPrintLines,
    ref string[] ppVoucherOutput,
    SymphonyPosApi_Extensibility[] checkExtensibilityDetails
)
```

## Parameters

<b>vendorCode</b>	Vendor code for license validation. In Symphony version 2.7 MR3 or later, set this parameter to an empty string.
-------------------	--

---

<b><i>revenueCenterObjectNum</i></b>	Object number of given revenue center.
<b><i>orderType</i></b>	Number of the order type for the transaction (for example, Dine-in or Carry-out).
<b><i>employeeObjectNum</i></b>	Object number of employee authorized to perform the operation.

---

The guest check is defined by creating a set of menu items, combo meals, service charges, and discounts. The *pGuestCheck* parameter is used to specify additional attributes for the guest check, Guest Count, Check ID, Event Number, and so on.

A tender media is required. If a tender media of type Service Total is supplied, the guest check is created in the open state. If a tender media of type Payment is supplied, the check is closed.

When this method is invoked, the guest check structure (*pGuestCheck*) is evaluated and changed where appropriate. The following properties of *SimphonyPosApi\_GuestCheck* can be modified and updated to reflect the new information during execution of this method:

- CheckID
- CheckTableObjectNum (when supported)
- CheckOrderType
- CheckEmployeeObjectNum
- CheckDateToFire
- pCheckInfoLines

The *CheckNum*, *CheckSeq* and *CheckRevenueCenterObjectNum* properties are not modified by this method.

Refer to the Structure Reference chapter for details on how they are specified.

## Response

The price of menu items, discounts, and service charges are returned in the same structures used to create the request. The *pGuestCheck* and *pTotalsResponse/Ex* parameter contains a summary of totals.

The guest check is printed to the local and remote order devices defined for the POS API workstation. The printed check details are returned in the *ppCheckPrintLines* parameter, while credit voucher details are filled in the *ppVoucherOutput* parameter.

The operation does not complete if payment or another interim operation fails. Inspect the *OperationalResult* property of the *pGuestCheck* structure in the response to verify that the operation completed successfully.

When the operation completes successfully, the *CheckNum* and *CheckSeq* properties are set on the *pGuestCheck* parameter. These values are used to reference the guest check in subsequent operations.

## Void Transaction

This operation voids all items (for example, menu items, tender media, service charge, and discount) in the given guest check and then closes the check. This method works only if the guest check is in the open state. This method fails if the check is already in the closed state.

The following post transaction method is available in Symphony version 2.7 and later.

```
void VoidTransaction
(
    string vendorCode,
    ref SymphonyPosApi_GuestCheck    pGuestCheck
)
```

### Parameters

---

<b>vendorCode</b>	Vendor code for license validation. In Symphony version 2.7 MR3 or later, set this parameter to an empty string.
-------------------	--

---

The request must specify both the *CheckNum* and *CheckSeq* properties of the *pGuestCheck* parameter. All other properties of *pGuestCheck* can be set to their default values.

### Response

Check the *OperationalResult* property of the *pGuestCheck* parameter from the response to verify that the operation succeeded.

## Check Print Job Status

This method is used to obtain the status of a specific print job (for example, guest check print and credit voucher print) of a transaction.

The following post transaction method is available in Symphony version 2.7 and later.

```
void CheckPrintJobStatus
(
    string vendorCode,
    int ppJobId,
    ref SymphonyPrintApi_PrintJobStatus    ppJobStatus
)
```

---

## Parameters

<b><i>vendorCode</i></b>	Vendor code for license validation. In Symphony version 2.7 MR3 or later, set this parameter to an empty string.
<b><i>ppJobId</i></b>	The ID of a print job for which status is retrieved. This value comes from the <i>PPrintJobIds</i> property of the <i>SimphonyPosApi_GuestCheck</i> structure.
<b><i>ppJobStatus</i></b>	The response contains the status of the print job.

## Response

This method gets the status of a specified print job. It also gets the complete list of print jobs and stores it in the *PrintJobList* field of parameter *ppJobStatus*. The following is an exhaustive list of job status:

- Job Pending
- Job Complete
- Job Aborted
- Job Sent to backup printer
- Job Failed
- Job Not found

The status of the print job is returned in the *ppJobStatus* parameter.

# 3

## Guest Check Operations

TS API provides support for two check related and one configuration related operations. One web method is exposed to support each of these operations. The following table describes the web methods.

### Get Check Summary

<pre>void <b>GetOpenChecks</b> (string vendorCode, int employeeObjectNum,                     ref SymphonyPosAPI_OpenChecks openChecks)</pre> <p><i>Gets summary of all open guest checks from all revenue centers of the property from Symphony POS database</i></p>
<pre>void <b>GetOpenChecksEx</b> (string vendorCode, int employeeObjectNum,                       ref SymphonyPosAPI_OpenChecks openChecks)</pre> <p><i>Gets summary of all open guest checks from all revenue centers of the property from Symphony POS database. The only difference between GetOpenChecks and this method is that GetOpenChecks populates <b>CheckRevenueCenterObjectNum</b> member with ID of revenue center while this method populates Object Number of revenue center.</i></p>
<pre>void <b>GetOpenChecksByRVC</b> (string vendorCode, int employeeObjectNum, int                           revenueCenterObjectNum,                           ref SymphonyPosAPI_OpenChecks openChecks)</pre> <p><i>Gets summary of open guest checks for a specific revenue center from Symphony POS database</i></p>
<pre>void <b>GetChecks</b> (SymphonyPosApi_CheckRequest ppCheckFilter, ref                  SymphonyPosApi_CheckResponse                  ppChecksResponse)</pre> <p><i>Gets summary of both open and closed guest checks after applying given filter condition</i></p>

### Get Check Detail

<pre>void <b>GetCheckDetail</b> (SymphonyPosApi_CheckDetailRequest ppCheckDetailFilter, ref                     SymphonyPosApi_CheckDetailResponse ppCheckDetailResponse)</pre> <p><i>Gets completes details of a guest check in xml format</i></p>
---

## Get Printed Check

```
void GetPrintedCheck (string vendorCode, int CheckSeq, int EmplObjectNum, int
TmedObjectNum,
    ref SimphonyPosApi\_CheckPrintResponse ppCheckPrintLines)
Gets printed texts of an open guest check
```

The following sections provide details about each operation.

## Get Summary of All Open Guest Checks

```
void GetOpenChecks
(
    string vendorCode,
    int employeeId,
    ref SimphonyPosAPI\_OpenChecks openChecks
)
```

### Business Purpose

The user wants to view a summary of all open guest checks from all revenue centers within the property.

### Method Description

This method gets a summary of all open guest checks from all revenue centers within the property from the POS database. Guest checks that are created by a specific employee can be fetched by passing the appropriate value to the *employeeId* parameter. However, when 0 is passed to *employeeId*, it fetches all open guest checks irrespective who created the check. The *CheckRevenueCenterObjectNum* field of the *openChecks.SimphonyPosApi\_CheckSummary* structure is mislabeled; it holds the value of Revenue Center ID instead of Revenue Center Object Number. If this field is expected to hold Object Number of Revenue Center, the new method named *GetOpenChecksEx* can be used instead.

### Parameters

<b><i>vendorCode</i></b>	Vendor code for license validation. In Symphony version 2.7 MR3 or later, set this parameter to an empty string.
<b><i>employeeId</i></b>	Employee ID of employees to filter open guest checks based on who created it. Pass specific employee ID to fetch open checks created by that specific employee. Pass zero to fetch all open checks irrespective of who created the check.
<b><i>openChecks</i></b>	Holds open checks retrieved from the POS database (output parameter).

### Return Value

Void. Result is encapsulated in *openChecks* reference parameter.

## Get Open Guest Checks With RVC Object Number

```

void GetOpenChecksEx
(
    string          vendorCode,
    int            employeeObjectNum,
    ref SymphonyPosAPI_OpenChecks openChecks
)
    
```

### Business Purpose

The user wants to view a summary of all open guest checks from all revenue centers within the property, and wants to have the object number of the revenue center (instead of ID) for each guest check. The object number of the revenue center is different from that of ID.

### Method Description

This method is another version of the *GetOpenChecks* method that is described in the previous section. This was introduced in Symphony version 2.7 MR5 to retrieve all open guest checks from all revenue centers within the property. The only difference compared to the *GetOpenChecks* method is that the *CheckRevenueCenterObjectNum* property of *openChecks.SymphonyPosApi\_CheckSummary* holds the Revenue Center Object Number instead of the Revenue Center ID. All open guest checks created by a specific employee can be fetched by passing the appropriate value to the *employeeObjectNum* parameter. However, when 0 is passed to *employeeObjectNum*, it fetches all open guest checks irrespective of who created them.

### Parameters

<b><i>vendorCode</i></b>	Vendor code for license validation. In Symphony version 2.7 MR3 or later, set this parameter to an empty string.
<b><i>employeeObjectNum</i></b>	Object number of employees to filter open guest checks based on who created it. Pass specific employee object numbers to fetch open checks created by that specific employee. Pass zero to fetch all open checks irrespective of who created the check.
<b><i>openChecks</i></b>	Holds open checks retrieved from the POS database (output parameter).

### Return Value

Void. Result is encapsulated in the *openChecks* reference parameter.

## Get Open Guest Checks From a Specific RVC

```

Void GetOpenChecksByRVC
(
    String          vendorCode,
    Int            employeeObjectNum,
    Int            revenueCenterObjectNum,
    ref SymphonyPosAPI_OpenChecks openChecks
)
    
```



### Business Purpose

The user wants to view a summary of open guest checks from a specific revenue center.

### Method Description

This method was introduced in Symphony version 2.7MR4 to get all open guest checks in a specific revenue center from the Symphony POS database. All open guest checks created by a specific employee in a specific revenue center can be fetched by passing the appropriate value to the *employeeObjectNum* and *revenueCenterObjectNum* parameters. However, when 0 is passed for *employeeObjectNum*, it fetches all open guest checks from the specified revenue center irrespective of who created it. Also, note that the other two related methods named *GetOpenChecks* and *GetOpenChecksEx* return a summary of all open checks from all revenue centers within the property.

### Parameters

<b><i>vendorCode</i></b>	Vendor code for license validation. In Symphony version 2.7 MR3 or later, set this parameter to an empty string.
<b><i>employeeObjectNum</i></b>	Object number of employees to filter open guest checks based on who created it. Pass specific employee object numbers to fetch open checks created by that specific employee. Pass zero to fetch all open checks irrespective of who created the check.
<b><i>revenueCenterObjectNum</i></b>	Object number of revenue center for which checks needs to be retrieved.
<b><i>openChecks</i></b>	Holds open checks retrieved from the POS database (output parameter).

### Return Value

Void. Result is encapsulated in the *openChecks* reference parameter.

## Get Printed Texts of a Guest Check

```

void GetPrintedCheck
(
    string                vendorCode,
    int                  CheckSeq,
    int                  EmplObjectNum,
    int                  TmedObjectNum,
    ref SymphonyPosAPI_CheckPrintResponse ppCheckPrintLines
)
    
```

### Business Purpose

The user wants to reprint a guest check for a customer using an external printer.

### Method Description

This method gets printed texts of an open guest check. This method works only on open guest checks and throws an exception for a closed guest check.

This method requires the tender media as input because it has several printing options that assist in the formatting of the final guest check.

### Parameters

<b><i>vendorCode</i></b>	Vendor code for license validation. In Symphony version 2.7 MR3 or later, set this parameter to an empty string.
<b><i>CheckSeq</i></b>	Check number of the guest check (and not the check sequence number as the name implies).
<b><i>EmplObjectNum</i></b>	Object number of the employee who wants to perform this operation.
<b><i>TmedObjectNum</i></b>	Object number of the tender media to print the check with.
<b><i>ppCheckPrintLines</i></b>	This holds an array of printed lines of the check along with response code.

### Return Value

Void. Result is encapsulated in the *ppCheckPrintLines* reference parameter.

## Get Summary and KDS Order Status of Open and/or Closed Guest Checks

```

Void GetChecks
(
    SymphonyPosApi_CheckRequest    ppCheckFilter,
    ref SymphonyPosApi_CheckResponse ppChecksResponse
)
    
```

### Business Purpose

The user wants to view a summary of both open and closed guest checks that satisfy one or more filter conditions. This method can also be used when the user wants to know the KDS order status of one or more guest checks.

### Method Description

This method was introduced in Symphony version 18.1 to get a summary of both open and closed guest checks that satisfy one or more filter conditions. If no filter is passed, this method will return all open guest checks that are created on the current business date in the default revenue center assigned to the POS API workstation. If closed checks need to be returned too, the field *IncludeClosedCheck* of *ppCheckFilter* should be set to True. This method applies filters based on data passed to following fields of *ppCheckFilter*:

- **CheckNumbers**  
Pass one or more check numbers to filter checks based on check numbers
- **EmployeeObjectNum**  
Pass Object Number of an employee to get only checks created by that particular employee
- **RvcObjectNum**  
Pass Object Number of an RVC that is currently assigned to a POS API workstation to get only checks created in that RVC

- **OrderTypeID**  
Pass Object Number of an Order Type to get guest checks created for that specific order type
- **KdsOrderStatus**  
Pass one or more KDS Order Status ID to retrieve checks that hold those status ID as their current KDS order status. Possible KDS order status IDs include:
  - 30 means DS\_SENT (order has been sent to the kitchen with at least 1 menu item)
  - 50 means DS\_PREP\_DONE (order has been prepared by at least one station)
  - 60 means DS\_READY (order has been expo done and is ready to be distributed to the customer)
  - 100 means DS\_CANCELLED (order cancelled)
- **LookupStartDate**  
Pass a date to retrieve only checks that were created after that given date
- **IncludeClosedCheck**  
Pass True to retrieve closed checks too. When False or value not specified, this method will return only open guest checks.

None of these fields mandates input data and the caller of this method can pass input data to one or more of these fields to filter the guest checks as needed.

The field VendorCode of ppCheckFilter is reserved for future use and is not currently used for any purpose.

**Parameters**

<b><i>ppCheckFilter</i></b>	Criteria based on which guest checks need to be filtered.
<b><i>ppChecksResponse</i></b>	Holds status of operation along with a summary of filtered guest checks. The summary includes KDS order status as well.

**Return Value**

Void. Result is encapsulated in *ppChecksResponse* reference parameter.

# 4 Configuration Operations

## Get Configuration Information

```
void GetConfigurationInfo (string vendorCode, int employeeObjectNum, int[]
configurationInfoType,
    int revenueCenter, ref SimphonyPosApi_ConfigInfoResponse configInfoResponse)
```

*Gets configuration data for one or more types from POS database.*

**Note:** This method returns all the records of specified configuration data type and it may throw "timeout" error when the POS database has a huge volume of configuration data for one or more types. The integrator can use the new method named `GetConfigurationInfoEx` (introduced in 2.9) in such cases to retrieve configuration data batch by batch by specifying ranges in the input parameter.

```
void GetConfigurationInfoEx (SimphonyPosApi_ConfigInfoRequest configInfoRequest,
    ref SimphonyPosApi_ConfigInfoResponse configInfoResponse)
```

*A new version of `GetConfigurationInfo` method to retrieve configuration data from POS database batch by batch by specifying ranges. This new method can be used instead of `GetConfigurationInfo` if the volume of configuration data is huge. Because, the other method named `GetConfigurationInfo` may throw "timeout" error when it tries to pull huge volume of records in one request.*

### Get Configured Information (*method 1*)

```
void GetConfigurationInfo
(
    string                vendorCode,
    int                   employeeObjectNum,
    ARRAY(int)            configurationInfoType,
    int                   revenueCenter,
    ref SimphonyPosAPI_ConfigInfoResponse configInfoResponse
)
```

#### **Business Purpose**

The user wants to fetch configuration data from the Simphony POS database.

#### **Method Description**

This method returns configuration data, such as menu item definition and menu item price found in the POS database. If the volume of configuration data is found to be large, this method may throw a timeout error. In such cases, the client application can call a new version of this method named `GetConfigurationInfoEx` that is explained in the next section. The new method returns configuration data for only a specified range of records. The caller has to call this new method multiple times to retrieve a complete list of records. Review the next section for more details on the new version of this method. The following list of configuration data can be retrieved using the `GetConfigurationInfo` method.

1. Menu Item Definitions
2. Menu Item Prices
3. Menu Item Classes
4. Service Charges
5. Discounts
6. Tender Media
7. Order Types
8. Family Groups
9. Major Groups
10. Revenue Center Parameters
11. Revenue Center Configurations
12. Interfaces
13. Menu Item Masters
14. Serving Periods
15. Currencies
16. Product Version
17. Employees
18. Dining Tables
19. Languages
20. Menu Level Set
21. Menu Item SLU
22. Main Menu Levels
23. Sub Menu Levels
24. Event Types
25. Event SubTypes
26. Event Definitions
27. TAX

**Parameters**

<b><i>vendorCode</i></b>	Vendor code for license validation. In Symphony version 2.7 MR3 or later, set this parameter to an empty string.
<b><i>employeeObjectNum</i></b>	Object number of employee who wants to perform this operation.
<b><i>configurationInfoType</i></b>	Array of information types for which details need to be fetched from the POS.
<b><i>revenueCenter</i></b>	Object number of revenue center.

<b><i>configInfoResponse</i></b>	Structure that holds the results. This holds configuration information along with the operation result. There is one field for each information type requested.
----------------------------------	---

**Return Value**

Void. Result is encapsulated in the *configInfoResponse* reference parameter.

## Get Configured Information (*method 2*)

```
void GetConfigurationInfoEx
(
    SymphonyPosAPI_ConfigInfoRequest    configInfoRequest,
    ref SymphonyPosAPI_ConfigInfoResponse configInfoResponse
)
```

**Business Purpose**

The user wants to fetch configuration data for a specified range of records from the Symphony POS database.

**Method Description**

As mentioned in the previous section, this is a new version of the GetConfigurationInfo method. The main difference between these two methods is that this new method returns configuration data for only a specified range of records, while GetConfigurationInfo returns all records by default. This new method can be used if the given system has a large volume of configuration data for one or more types (for example, menu item definition has more than 30,000 records).

**Parameters**

<b><i>configInfoRequest</i></b>	Request parameter that holds filter conditions to be applied while retrieving configuration data. This includes configuration data type and ranges (for example, StartIndex and MaxRecordCount).
<b><i>configInfoResponse</i></b>	Structure that holds the results. This holds configuration information along with the operation result. There is one field for each information type requested.

**Return Value**

Void. Result is encapsulated in the *configInfoResponse* reference parameter.

# 5

## Fiscal Operations

TS API provides the ability to the end-user to validate their current session to help ensure that they remain fiscally compliant in their country of operation. This API works in conjunction with an 'Extension Plugin' DLL designed to be country-specific, for which a separate documentation shall be provided by the Fiscal Development Team. The 'Extension Plugin' shall exist with TS on the workstation and operate between the '1. TS API Client' and the '2. TS API' illustrated in Figure 1-1 – Symphony Architecture.

### Sanity Check

```
void SanityCheck (string sanityCode, ref SimphonyPosApi\_SanityCheckResponse  
pSanityCheckResponse)
```

*Gets the result of the sanity check validation performed*

#### Business Purpose

The user wants to validate that check operations can be safely performed with their current session while remaining fiscally compliant in their country of operation.

#### Method Description

This method returns the result of the country-specific validation performed on the current session (Sanity Check). A code needs to be provided as a parameter to determine 'what' validation is to be performed. This 'Sanity Code' is dependent on the 'Extension Plugin' that is initiated with the TS, and would therefore be provided in the documentation accompanying the 'Extension Plugin' designed for that particular country.

#### Parameters

<b><i>sanityCode</i></b>	Code that determines the country-specific validation to be performed.
<b><i>pSanityCheckResponse</i></b>	Structure that holds the results. This holds both the operational result, and the result of the sanity check validation performed, which is defined by the <i>sanityCode</i> parameter.

#### Return Value

Void. Result is encapsulated in the *pSanityCheckResponse* reference parameter.

# 6

## Structure Reference

This section describes the various structures used for pass data for request and response.

### SimphonyPosAPI\_CheckSummary

This structure is to encapsulate summary details of a check.

#### Public Attributes

<b>int CheckSeq</b>	<i>Check sequence is a number that identifies a check in the Simphony POS database. The sequence number will be assigned to a check when it's created by the system.</i>
<b>int CheckNum</b>	<i>Check number is a number to identify a check in a particular workstation. This number will be assigned to a check by the system when the check is created. Minimum and maximum range for check number can be configured in EMC for any workstation.</i>
<b>int CheckEmployeeObjectNum</b>	<i>ID of employee who created the check</i>
<b>int CheckRevenueCenterObjectNum</b>	<i>ID of revenue center this check is currently active</i>
<b>int CheckLastWorkstationOwner</b>	<i>Object number of workstation that owned the check last time</i>
<b>int CheckCurrentlyOpenOnWorkstation</b>	<i>Object number of workstation that has this check currently opened</i>
<b>int CheckTableObjectNum</b>	<i>Object number of dining table for which the check created.</i>
<b>int CheckTableGroup</b>	<i>ID of table group in which dining table of this check falls under</i>
<b>int CheckOrderType</b>	<i>Order type ID of the check. E.g. Dine In and Eat Out</i>
<b>string CheckID</b>	<i>Name to identify a check. Duplicate check names on open checks are not allowed.</i>
<b>string CheckTotalDue</b>	<i>Due or balance amount to be paid by the customer for the check</i>
<b>DateTime CheckLastServiceTime</b>	<i>The time when the check was submitted last time to POS database</i>



<b>DateTime</b> <b>CheckOpenTime</b>
<i>The time when the check was opened/created on POS database</i>
<b>DateTime</b> <b>CheckAutoFireTime</b>
<i>The time when check will be fired</i>
<b>short</b> <b>CheckInTraining</b>
<i>A flag that indicates whether the check is opened on training mode or not. Always 0 is populated.</i>
<b>short</b> <b>CheckInsufficientBeverage</b>
<i>A flag that indicates whether insufficient beverage found on the check (i.e. beverage count is less than total guest count). Always 0 is populated.</i>
<b>short</b> <b>CheckTransferredToDriver</b>
<i>A flag that indicates status of the check as having been assigned to a driver. This is no longer in use. Always 0 is populated.</i>
<b>short</b> <b>CheckIsDelayedOrder</b>
<i>A flag that indicates status of the check as being a Delayed Order. This is no longer in use. Always 0 is populated.</i>
<b>short</b> <b>CheckIsFutureOrder</b>
<i>A flag that indicates status of this check as having been assigned to a driver. Always 0 is populated.</i>

## SimphonyPosAPI\_CheckSummaryEx

This structure is an extended version of `SimphonyPosAPI_CheckSummary` structure. This extended version is created to hold a couple of additional fields on the KDS order status. This structure inherits all fields from `SimphonyPosAPI_CheckSummary`.

### Public Attributes

<b>SimphonyPosApi_KdsOrderStatus</b> <b>LastKnownKdsOrderStatus</b>
<i>Last status reported by KDS device (e.g. DS_SENT, DS_PREP_DONE, DS_CANCELLED)</i>
<b>ARRAY(SimphonyPosApi_KdsOrderStatus)</b> <b>KdsOrderStatusHistory</b>
<i>List of order status reported by KDS for the history of the order</i>

## SimphonyPosAPI\_OpenChecks

This structure is to encapsulate details of all open checks.

### Public Attributes

<b>ARRAY(SimphonyPosAPI_CheckSummary)</b>	<b>CheckSummary</b>
<i>A structure to hold summary of all open checks</i>	
<b>SimphonyPosAPI_OperationalResult</b>	<b>OperationalResult</b>
<i>A structure that indicates whether the current operation succeeded or not. In case of failure, this will have appropriate error code and error message</i>	

## SimphonyPosAPI\_GuestCheck

This structure is to encapsulate the details of a guest check.

The guest check structure is a collection of elements that are passed as a parameter. This shared structure is used to communicate key elements of the transaction to the API and for the API to return key elements to the API consumer.

It is possible to create a future check (a.k.a. auto fire check), by mentioning the appropriate value for *CheckStatusBits* property. The system will apply all applicable automatic discounts and service charges while creating the guest check. In addition, the system calculates the tax amount based on how the order type is configured in the EMC.

The details, such as Check Number and Check Sequence Number of the created check, will be filled in the *pGuestCheck* parameter. This method prints the guest check and credit voucher if configured to do so in EMC for the given workstation.

### Public Attributes

<b>string</b> CheckID
<i>Name to identify a check. Duplicate check names on open checks are not allowed</i>
<b>int</b> CheckTableObjectNum
<i>Object number of dining table for which the check opened</i>
<b>int</b> CheckRevenueCenterID
<i>Object number of revenue center</i>
<b>int</b> CheckOrderType
<i>Order type ID of the check. E.g. Dine In and Eat Out</i>
<b>int</b> CheckEmployeeObjectNum
<i>Object number of employee who opened the check</i>
<b>int</b> CheckSeq
<i>Check sequence is a number that identifies a check in the POS database. The number is assigned to the check is opened. This is used as a parameter while adding items to an existing check.</i>
<b>int</b> CheckNum

<i>Check number is a number to identify a check in a particular workstation. This number will be assigned to a check by the system when the check is created. Minimum and maximum range for check number can be configured in EMC for any workstation.</i>
<b>DateTime CheckDateToFire</b> <i>Time when check should fire. This will permit an order to be delayed on the current business date</i>
<b>int CheckGuestCount</b> <i>Total number of guest in a transaction</i>
<b>int CheckStatusBits</b> <i>Check status identifier. E.g. "Rush Order" or "VIP".</i>
<b>ARRAY(string) PCheckInfoLines</b> <i>Information lines that are added to guest check</i>
<b>ARRAY(int) PPrintJobIds</b> <i>List of print job ID that resulted from the transaction. The method CheckPrintJobStatus can be used to get the status of any print job later</i>
<b>int EventObjectNum</b> <i>Object Number of an event definition that needs to be associated to given guest check</i>
<b>SimphonyPosAPI_OperationalResult OperationalResult</b> <i>A structure that indicates whether current operation succeeded or not. In case of failure, this will have appropriate error code and message</i>

## SimphonyPosAPI\_CheckRequest

This structure is to encapsulate the input parameters of the GetChecks method.

### Public Attributes

<b>string VendorCode</b> <i>Vendor code for license validation (pass an empty value for Symphony version 2.7 MR3 or later).</i>
<b>Int[] CheckNumbers</b> <i>Check Number(s)</i>
<b>int EmployeeObjectNum</b> <i>Object Number of an employee</i>
<b>int RvcObjectNum</b> <i>Object Number of a Revenue Center</i>
<b>int OrderTypeID</b> <i>Object Number of an Order Type</i>
<b>Int[] KdsOrderStatus</b> <i>Status ID of KDS Order</i>
<b>DateTime LookUpStartDate</b> <i>Lookup start date. All checks that were created on/after this date would be returned in the result.</i>

**bool IncludeClosedCheck**

*A boolean flag to specific whether or not closed checks need to be included in the result*

## SimphonyPosAPI\_CheckResponse

This structure is to encapsulate the response of the GetChecks method.

### Public Attributes

**SimphonyPosApi\_OperationalResult OperationalResult**

*A structure that indicates whether current operation succeeded or not. In case of failure, this will have appropriate error code and message*

**SimphonyPosApi\_CheckSummaryEx Checks**

*Extended summary of guest checks that includes KDS order status*

## SimphonyPosAPI\_CheckDetailRequest

This structure is to encapsulate the input fields of the GetCheckDetail method.

### Public Attributes

**string VendorCode**

*Vendor code for license validation (pass an empty value for Simphony version 2.7 MR3 or later).*

**int CheckNumber**

*Guest Check Number*

**int CheckSeqNumber**

*Sequence Number of Guest Check*

## SimphonyPosAPI\_CheckDetailResponse

This structure is to encapsulate the response of the GetCheckDetail method.

### Public Attributes

**SimphonyPosApi\_OperationalResult OperationalResult**

*A structure that indicates whether current operation succeeded or not. In case of failure, this will have appropriate error code and message*

**string CheckDetail**

*Check Detail in XML format*

## SimphonyPosAPI\_MenuItem

This structure is to encapsulate the details of a menu item along with its condiments.

The menu item is comprised of the desired main item and an array of condiments. An example may be a Cheeseburger (main item), Well Done, and Extra Pickles (condiment array).

#### Public Attributes

<b>SimphonyPosAPI_MenuItemDefinition</b>	<b>MenuItem</b>
<i>Structure that defines details of a main menu item. The details include object number of menu item, price, discount etc.</i>	
<b>ARRAY(SimphonyPosAPI_MenuItemDefinition)</b>	<b>Condiments</b>
<i>List of a structure that defines details of condiment added to menu item</i>	

## SimphonyPosAPI\_MenuItemEx

This structure is to encapsulate the details of a menu item along with its condiments.

The menu item is comprised of the desired main item and an array of condiments. An example may be a Cheeseburger (main item), Well Done, and Extra Pickles (condiment array).

This structure is similar to SimphonyPosAPI\_MenuItem. However, it uses the newer “Ex” structure for MenuItem and Condiments and adds support for the Extensibility attribute.

#### Public Attributes

<b>SimphonyPosAPI_MenuItemDefinitionEx</b>	<b>MenuItem</b>
<i>Structure that defines details of a main menu item. The details include object number of menu item, price, discount etc.</i>	
<b>SimphonyPosAPI_MenuItemDefinitionEx[]</b>	<b>Condiments</b>
<i>Array of a structure that defines details of condiment added to menu item</i>	
<b>SimphonyPosApi_Extensibility[]</b>	<b>Extensibility</b>
<i>Array of a structure that defines extensibility detail that can be added to the menu item.</i>	

## SimphonyPosAPI\_MenuItemDefinition

This structure is to encapsulate the details of a menu item.

#### Public Attributes

<b>int MiObjectNum</b>
<i>Object number of given menu item</i>
<b>int MiMenuLevel</b>
<i>Main level to be used while picking up a menu definition from definition list. This must be a value between 1 and 8 (if not 0). When 0 is specified, system will pick up first menu definition irrespective of whether it's active or not on given Main level</i>

<b>int MiSubLevel</b>
<i>Sub level to be used while picking up a menu definition from definition list. This must be a value between 1 and 8 (if not 0). When 0 is specified, system will pick up first menu definition irrespective of whether it's active or not on given Sub level</i>
<b>int MiPriceLevel</b>
<i>Sequence number to be used while picking up a price definition from the list. <b>This is not currently supported in Transaction Services web service.</b> That is, price definition will always be picked up based on the value of Sub level or Main level mentioned above</i>
<b>string MiOverridePrice</b>
<i>Price to override default value of the item. This field can be left empty if default price is desired. If left empty this will be populated with default price by this method.</i>
<b>string MiWeight</b>
<i>Weight of given item. <b>This is not currently supported in the API.</b></i>
<b>string MiReference</b>
<i>A text that needs to be added as reference to given menu item</i>
<b>SymphonyPosApi_Discount ItemDiscount</b>
<i>Discount that needs to applied to given menu item</i>

**Remarks**

The MiReference value can contain extra data to control additional behavior. This is used to specify a quantity, to override active tax rates, and to indicate a default condiment.

Format: <ExtraData><!-- extra data specified here --></ExtraData>reference-text

Example: <ExtraData><MiQuantity>3</MiQuantity></ExtraData>VIP

**List of Extra Data Elements**

<b>MiQuantity</b>
<i>Used in v1 of API to specify quantity other than one. If not specified, the default quantity of 1 is used.</i>
<b>Example:</b> <MiQuantity>2</MiQuantity>
<b>TaxOverride</b>
<i>Used to override tax rates active for menu item. The value is a string of up-to 64 zeros or ones. A one indicates the tax rate for the ordinal position is enable, whereas zero indicates the rate is disabled. If value are only provided for the first 10 rates, the remaining rates use the configured behavior.</i>
<b>Example:</b> <TaxOverride>101.....</TaxOverride>
<b>AsDefaultCondiment</b>
<i>Use to indicate condiment is a default condiment. This extra data only applies to Condiments.</i>
<b>Example:</b> <AsDefaultCondiment/>

Here is a more complete example:

```
"<ExtraData><MiQuantity>3</MiQuantity></ExtraData>"
```

In this example, 3 is the quantity. The regular reference text can be specified before or after the above XML. For example:

```
"<ExtraData><MiQuantity>3</MiQuantity></ExtraData>Make it spicy"
```

In the example above, the text "Make it spicy" will be treated as the reference text for the given menu item and the XML that defines quantity will not appear on the screen. Refer to the code snippet section of this document for more details.

## SimphonyPosAPI\_MenuItemDefinitionEx

This structure is to encapsulate the details of a menu item.

### Public Attributes

<b>int</b> <b>MiObjectNum</b>	<i>Object number of given menu item</i>
<b>int</b> <b>MiMenuLevel</b>	<i>Main level to be used while picking up a menu definition from definition list. This must be a value between 1 and 8 (if not 0). When 0 is specified, system will pick up first menu definition irrespective of whether it's active or not on given Main level</i>
<b>int</b> <b>MiSubLevel</b>	<i>Sub level to be used while picking up a menu definition from definition list. This must be a value between 1 and 8 (if not 0). When 0 is specified, system will pick up first menu definition irrespective of whether it's active or not on given Sub level</i>
<b>int</b> <b>MiPriceLevel</b>	<i>Sequence number to be used while picking up a price definition from the list. <b>This is not currently supported in Transaction Services web service.</b> That is, price definition will always be picked up based on the value of Sub level or Main level mentioned above</i>
<b>string</b> <b>MiOverridePrice</b>	<i>Price to override default value of the item. This field can be left empty if default price is desired. If left empty this will be populated with default price by this method.</i>
<b>string</b> <b>MiWeight</b>	<i>Weight of given item. <b>This is not currently supported in the API.</b></i>
<b>string</b> <b>MiReference</b>	<i>A text that needs to be added as reference to given menu item</i>
<b>decimal</b> <b>MiQuantity</b>	<i>The quantity of item to add to the guest check.</i>
<b>int</b> <b>MiDefinitionSeqNum</b>	<i>The number of the definition to use when ordering the item. Different definitions may have different behavior, prices depending on configuration in EMC.</i>
<b>SimphonyPosApi_DiscountEx[]</b> <b>ItemDiscount</b>	

List of discount to apply to a given menu item. In the response, this attributes indicates the discounts applied to this item by coupon and/or automatic discounts.

**Remarks**

The MiReference value can contain extra data to control additional behavior. See the Remarks section for [SymphonyPosAPI\\_MenuItemDefinition](#) for more information.

**List of Extra Data Elements****TaxOverride**

Used to override tax rates active for menu item. The value is a string of up-to 64 zeros or ones. A one indicates the tax rate for the ordinal position is enable, whereas zero indicates the rate is disabled. If value are only provided for the first 10 rates, the remaining rates use the configured behavior.

**Example:** <TaxOverride>101.....</TaxOverride>

**AsDefaultCondiment**

Use to indicate condiment is a default condiment. This extra data only applies to Condiments.

**Example:** <AsDefaultCondiment/>

## SymphonyPosAPI\_ComboMeal

This structure is to encapsulate the details of a combo meal (main and side menus).

**Public Attributes**

<b>SymphonyPosAPI_MenuItem ComboMealMenuItem</b>
<i>Combo Meal Menu Item (e.g. Burger Combo)</i>
<b>SymphonyPosAPI_MenuItem ComboMealMainItem</b>
<i>Combo Meal Main Item (e.g. Hamburger)</i>
<b>ARRAY(SymphonyPosAPI_MenuItem) SideItems</b>
<i>Combo Meal Side Items (e.g. French Fries, Coke etc.)</i>
<b>int ComboMealObjectNum</b>
<i>Combo Meal Object Number</i>

**Remarks**

When ordering combo meals, TS API is strict in checking all combo meal linkages. The combo meal menu item passed along must be linked to a combo meal object number. Additionally, the combo meal side items that are passed along must be correctly linked to a combo meal as defined in the target database. This means that side items must be passed in order. All items in orders must be filled correctly for combo meals.



## SimphonyPosAPI\_ComboMealEx

This structure is to encapsulate the details of a combo meal (main and side menus).

### Public Attributes

Inherits all attributes from [SimphonyPosAPI\\_ComboMeal](#), and includes the following additional attributes.

<b>SimphonyPosApi_Extensibility Extensibility</b>
---

*Provides extra information about detail.*

## SimphonyPosAPI\_Discount

This structure is to encapsulate the details of a discount.

### Public Attributes

<b>int DiscObjectNum</b>
--------------------------

*Discount Object Number*

<b>string DiscAmountOrPercent</b>
-----------------------------------

*Amount or Percentage to be discounted. API expects value for this property in case of "Open Discount".*

*However, in case of "Closed Discount", discount amount or percent will be taken from POS database*

*with the help of Discount Object Number.*

<b>string DiscReference</b>
-----------------------------

*Reference text to be added to given discount for reference purpose*

## SimphonyPosAPI\_DiscountEx

This structure is to encapsulate the details of a discount to be applied on a guest check.

### Public Attributes

Inherits all attributes from [SimphonyPosAPI\\_Discount](#), and includes the following additional attributes.

<b>SimphonyPosApi_Extensibility Extensibility</b>
---

*Provides extra information about detail.*

## SimphonyPosAPI\_SvcCharge

This structure is to encapsulate the details of a service charge to be applied on a guest check.

### Public Attributes

<b>int SvcChgObjectNum</b>
----------------------------

<i>Object Number of Service Charge that needs to be applied on guest check</i>
<b>string SvcChgAmountOrPercent</b>  <i>Amount or percentage to be applied as Service Charge. API expects value for this property in case of "Open Service Charge". However, in case of "Predefined Service Charge", the amount or percent will be taken from POS database with the help of Service Charge Object Number.</i>
<b>string SvcChgReference</b>  <i>Reference text to be added to given Service Charge item</i>

## SimphonyPosAPI\_SvcChargeEx

This structure is to encapsulate the details of a service charge to be applied on a guest check.

### Public Attributes

Inherits all attributes from [SimphonyPosAPI\\_SvcCharge](#), and includes the following additional attributes.

<b>SimphonyPosApi_Extensibility Extensibility</b>  <i>Provides extra information about detail.</i>
--

## SimphonyPosAPI\_TmedDetailItemEx

This structure is to encapsulate the details of tender media for a payment operation.

### Public Attributes

<b>int TmedObjectNum</b>  <i>Object number of tender media chosen for payment</i>
<b>string TmedPartialPayment</b>  <i>This indicates the amount tendered by the customer in cash for payment. This amount does not include tips.  Leave this field empty in case of paid-in-full. This field is applicable for only cash payment.</i>
<b>string TmedReference</b>  <i>Tender Media reference information</i>
<b>SimphonyPosAPI_EPayment TmedEPayment</b>  <i>Electronic Payment details</i>

## SimphonyPosAPI\_TmedDetailItemEx2

This structure is to encapsulate the details of tender media for a payment operation.

### Public Attributes

Inherits all attributes from [SimphonyPosAPI\\_TmedDetailItemEx](#), and includes the following additional attributes.

**SimphonyPosApi\_Extensibility Extensibility***Provides extra information about detail.*

## SimphonyPosApi\_Extensibility

This structure is to encapsulate the details of extension data that can be added to menu items, discounts, service charges, or tender detail items.

**Public Attributes**

<b>string DisplayName</b>	<i>A string displayed and/or printed if this items is displayed or printed.</i>
<b>string ExtensibilityAppName</b>	<i>A string indicating the name of the application associated with this data.</i>
<b>string ExtensibilityDataName</b>	<i>A string selected by the user that names the data stored by this extensibility item.</i>
<b>string ExtensibilityDataType</b>	<i>A string selected by the user that describes the type of data stored by this extensibility item.</i>
<b>string PrintOptionBits</b>	<i>This attribute is not currently used.</i>
<b>string StringData</b>	<i>This attributed is used to store the data payload for the extensibility data.</i>
<b>string DataID</b>	<i>This attribute is not currently used.</i>

## SimphonyPosAPI\_EPayment

This structure is to encapsulate the details of electronic payment on a guest check.

**Public Attributes**

<b>EPaymentDirective PaymentCommand</b>	<p><i>Enumeration on payment method (for example, credit authorization only, credit authorization and pay, debit authorization only, debit authorization and pay, SVC authorization, SVC redeem). Possible values are:</i></p> <ul style="list-style-type: none"> <li>• NO_E_PAYMENT</li> <li>• AUTHORIZE_AND_PAY</li> <li>• DEBIT_AUTHORIZE_AND_PAY</li> </ul> <p><b>Note:</b> Transaction Services only supports the <b>MCreditDebit Payment</b> driver for credit/debit card payment.</p>
<b>EAccountDataSource AccountDataSource</b>	<p><i>Enumeration on source of payment details (for example, magnetic stripe, RFID card, or manually keyed). Possible values are:</i></p> <ul style="list-style-type: none"> <li>• SOURCE_UNDEFINED</li> </ul>

<ul style="list-style-type: none"> <li>• RFID_TRACK_DATA_RULES</li> <li>• RFID_M_CHIP_RULES</li> <li>• MANUALLY_KEYED_TRACK_1_CAPABLE</li> <li>• MANUALLY_KEYED_TRACK_2_CAPABLE</li> <li>• MANUALLY_KEYED_NO_CARD_READER</li> </ul>
<p><b>EAccountType AccountType</b></p> <p><i>Type of account (for example, checking and savings). Possible values are:</i></p> <ul style="list-style-type: none"> <li>• ACCOUNT_TYPE_UNDEFINED</li> <li>• CHECKING</li> <li>• SAVINGS</li> </ul>
<p><b>string AcctNumber</b></p> <p><i>Account number of payment card.</i></p>
<p><b>string AuthorizationCode</b></p> <p><i>Authorization code of payment card.</i></p>
<p><b>DateTime StartDate</b></p> <p><i>Start date as mentioned in payment card.</i></p>
<p><b>short IssueNumber</b></p> <p><i>Issue number as mentioned in payment card.</i></p>
<p><b>string Track1Data</b></p> <p><i>Magnetic stripe data for Track 1.</i></p>
<p><b>string Track2Data</b></p> <p><i>Magnetic stripe data for Track 2.</i></p>
<p><b>string Track3Data</b></p> <p><i>Magnetic stripe data for Track 3.</i></p>
<p><b>string BaseAmount</b></p> <p><i>Base amount to be debited. This does not include tip or cash back amount.</i></p>
<p><b>string TipAmount</b></p> <p><i>Amount to be debited for tip.</i></p>
<p><b>string CashBackAmount</b></p> <p><i>Cash back amount.</i></p>
<p><b>string KeySerialNum</b></p> <p><i>Debit key serial number for given transaction. Maximum length is 20 characters.</i></p>
<p><b>string DeviceId</b></p> <p><i>Device Identifier</i></p>
<p><b>DateTime ExpirationDate</b></p> <p><i>Expiration date as mentioned in payment card.</i></p>
<p><b>string PinBlock</b></p> <p><i>Pin Number of payment card in encrypted format. This is used only with debit card payment.</i></p>
<p><b>string CVVNumber</b></p>

<i>Card Verification Value (CVV) number of payment card</i>
<b>string AddressVerification</b> <i>Address for verification</i>
<b>string InterfaceName</b> <i>Interface name of stored value card</i>
<b>string SvcResponse</b> <i>Stored value card response message. This contains descriptive error message in case of payment failure.</i>
<b>string SvcAccountType</b> <i>Stored value account. Maximum of 32 characters.</i>

## SimphonyPosAPI\_TotalsResponse

This structure is to encapsulate the details on totals of a transaction.

### Public Attributes

<b>string TotalsSubTotal</b> <i>Subtotal amount of current transaction.</i>
<b>string TotalsTaxTotals</b> <i>Total tax applied on current transaction.</i>
<b>string TotalsOtherTotals</b> <i>Service charge applied on current transaction.</i>
<b>string TotalsAutoSvcChgTotals</b> <i>Automatic service charge applied on current transaction.</i>
<b>string TotalsTotalDue</b> <i>Total amount due.</i>
<b>SimphonyPosAPI_OperationalResult OperationalResult</b> <i>A structure that indicates whether the current operation succeeded or failed. In case of failure, this will have appropriate error code and message.</i>

## SimphonyPosAPI\_TotalsResponseEx

This structure is to encapsulate the details on totals of a transaction.

Inherits all attributes from [SimphonyPosAPI\\_TotalsResponse](#), and includes the following additional attributes.

### Public Attributes

<b>CheckTaxDataPerRate[] CheckTaxDataPerRate</b> <i>List of tax rate data for each tax rate.</i>
---

## SimphonyPosAPI\_ConfigInfoRequest

This structure is to encapsulate the input parameters, such as Vendor Code, Employee Object Number, RVC Object Number, and Configuration Info Types with range conditions.

### Public Attributes

<b>string VendorCode</b>
<i>The vendor code for license validation (pass empty for Symphony version 2.7 MR3 or later).</i>
<b>int EmployeeObjectNumber</b>
<i>Employee object number for validation purposes only.</i>
<b>int RVCObjectNumber</b>
<i>Object number of the revenue center for which configuration data is needed.</i>
<b>ARRAY(SimphonyPosApi_ConfigInfo) ConfigurationInfo</b>
<i>This holds the IDs of configuration data type along with start index and maximum records to be returned for each configuration data type.</i>

## SimphonyPosAPI\_ConfigInfo

This structure is to encapsulate the input parameters, such as Configuration Info Type and ranges of records to be retrieved.

### Public Attributes

<b>EConfigurationInfoType ConfigurationInfoTypeID</b>
<i>The type of configuration data that needs to be fetched from the Symphony POS database (for example, menu item definition, service charge definition, discount definition, and so on).</i>
<b>int StartIndex</b>
<i>Index of first record to be fetched from the POS database.</i>
<b>int MaxRecordCount</b>
<i>Maximum number of records to be fetched.</i>

## SimphonyPosAPI\_ConfigInfoResponse

This structure is to encapsulate the configured details for menu, price, currency, discounts, employees, order type, revenue center, tender media, service charge, and so on.

### Public Attributes

<b>ARRAY(EConfigurationInfoType) ConfigInfoType</b>
<i>List of type of configuration information for which details need to be fetched from the Symphony POS database (for example, menu item definition, service charge definition, discount definition, and so on). Possible enumeration values are:</i>
<ul style="list-style-type: none"> <li>• UNDEFINED = 0</li> </ul>

<ul style="list-style-type: none"> <li>• MENUITEMDEFINITIONS = 1</li> <li>• MENUITEMPRICE = 2</li> <li>• MENUITEMCLASS = 3</li> <li>• SERVICECHARGE = 4</li> <li>• DISCOUNTDEFINITIONS = 5</li> <li>• TENDERMEDIA = 6</li> <li>• ORDERTYPE = 7</li> <li>• FAMILYGROUP = 8</li> <li>• MAJORGROUP = 9</li> <li>• REVENUECENTERPARAMETER = 10</li> <li>• REVENUECENTERS = 11</li> <li>• INTERFACES = 12</li> <li>• MENUITEMMASTERS = 13</li> <li>• SERVINGPERIODS = 14</li> <li>• CURRENCY = 15</li> <li>• VERSION = 16</li> <li>• EMPLOYEES = 17</li> <li>• TABLES = 18</li> <li>• LANGUAGEINFORMATION = 19</li> <li>• MENULEVEL = 20</li> <li>• MENUITEMSLU = 21</li> <li>• MAINMENULEVEL = 22</li> <li>• SUBMENULEVEL = 23</li> <li>• EVENTDEFINITIONS = 24</li> <li>• TAX = 25</li> </ul>
<p><b>string MenuItemDefinitions</b></p> <p><i>Details of all menu item definitions configured in the EMC for a given revenue center.</i></p>
<p><b>string MenuItemPrice</b></p> <p><i>Details of all menu item price records (for example, menu item definition ID, price, preparation cost, and so on) configured in the EMC.</i></p>
<p><b>string MenuItemClass</b></p> <p><i>Details of all menu item classes (for example, tax class, sales, discount and service charge itemizers, pricing calculation, and so on) as configured in the EMC.</i></p>
<p><b>string ServiceCharge</b></p> <p><i>Details of service charges (for example, service charge amount/percent, tips)</i></p>
<p><b>string Discounts</b></p> <p><i>Details of all discounts configured in the EMC at the enterprise level.</i></p>
<p><b>string TenderMedia</b></p> <p><i>Details of tender media configured in the EMC for payment (for example, cash and credit cards)</i></p> <p><b>Note:</b> Transaction Services only supports the <b>MCreditDebit Payment</b> driver for credit/debit card payment.</p>
<p><b>string OrderType</b></p> <p><i>Details of order types (for example, Dine-in and Carry out) configured in the EMC for a given property.</i></p>
<p><b>string FamilyGroups</b></p> <p><i>Details of all Family Groups (that is, category of menu items) configured in the EMC.</i></p>

<b>string MajorGroup</b>
<i>Details of all Major Groups configured in the EMC for menu items (for example, food and beverages).</i>
<b>string RevenueCenterParameter</b>
<i>Details of revenue center parameters that are configured in the EMC (for example, secondary print language, minimum and maximum check number, database update frequency, options, and so on).</i>
<b>string RevenueCenters</b>
<i>Details of revenue centers configured in the EMC for a given property.</i>
<b>string Interfaces</b>
<i>Details of all interfaces configured in the EMC at the enterprise level.</i>
<b>string MenuItemMasters</b>
<i>Details of all menu item master records (that is, property level menu item record).</i>
<b>string ServingPeriod</b>
<i>Details of serving period (for example, Breakfast 4am to 11am)</i>
<b>string Currency</b>
<i>Details of all currencies (for example, US Dollar, Peso) configured in the EMC at enterprise level.</i>
<b>string Version</b>
<i>Current version of the Transaction Services web service (for example, 2.700.0.77).</i>
<b>string Employees</b>
<i>Details of all employees configured in the EMC at the enterprise level.</i>
<b>string Tables</b>
<i>Details of dining tables configured in the EMC for a given property.</i>
<b>string LanguageInformation</b>
<i>Details of languages (for example, English, Spanish) that are configured in the EMC.</i>
<b>string MenuLevel</b>
<i>Details of menu level sets configured (for example, Main, Sub, and Custom levels).</i>
<b>string MenuItemSlu</b>
<i>Details of menu item SLU names (user can configure a maximum of 127 SLU names).</i>
<b>string MainMenuLevel</b>
<i>Details of main menu levels (user can configure a maximum of 8 main levels).</i>
<b>string SubMenuLevel</b>
<i>Details of sub menu levels (user can configure a maximum of 8 sub levels).</i>
<b>string EventDefinitions</b>
<i>Details of event definitions created at property levels.</i>
<b>string TAX</b>



*Details of tax rates configured in the EMC at enterprise and property levels (maximum of 64 tax rates).*

**SimphonyPosApi\_OperationResult OperationalResult**

*A structure that indicates whether the current operation succeeded or failed. In case of failure, this will have appropriate error code and message.*

## SimphonyPosAPI\_CheckPrintResponse

This structure is to encapsulate the details of response on printing a guest check.

### Public Attributes

**ARRAY(string) CheckPrintLines**

*Printed lines of a guest check.*

**SimphonyPosAPI\_OperationResult OperationalResult**

*A structure that indicates whether the current operation succeeded or failed. In case of failure, this will have appropriate error code and message.*

## SimphonyPosAPI\_PrintJobStatus

This structure is to encapsulate the details of response on retrieving the status of a print job.

### Public Attributes

**SimphonyPrintApi\_Status Status**

*An enumerator that indicates the current status of a specified print job. Possible values are:*

- JobPending = 0
- JobComplete = 1
- JobAborted = 2
- JobSentToBackup = 3
- JobFailed = 4
- JobNotFound = 5

**string StatusMsg**

*Current status of a specified print job in string format.*

**String SystemStatusMsg**

*This is for future use. Currently this holds the status of a specified print job in string format like StatusMsg field.*

**ARRAY(int) PrintJobList**

*List of print jobs on the POS system.*

**SimphonyPosAPI\_OperationResult OperationalResult**

*A structure that indicates whether the current operation succeeded or failed. In case of failure, this will have appropriate error code and message.*

# SimphonyPosAPI\_OperationalResult

This structure is to encapsulate the result of an operation.

## Public Attributes

### **bool** Success

*Indicates whether or not the operation has succeeded. This will be True if there is no exception or errors; otherwise False.*

### **TransactionServices\_ErrorCode** ErrorCode

*Error code that represents the reason for failure. Possible values are:*

AmountNotEntered	InvaildAuthCode
AppInitInProgress	InvalidCheckNumber
CCAuthDeclined	InvalidCreditCardExpirationDate
CCAuthDeclinedWithMessage	InvalidCreditCardHost
CCServerDown	InvalidCreditCardNumber
CheckEmployeeNumberMismatch	InvalidClientName
CheckNotFound	InvalidClosedDays
CheckListNotFound	InvalidConfigInfoRequestType
CheckOpenedOnSystem	InvalidConfigInfoType
CheckTableNumberMismatch	InvalidCustomerInfo
ComboMealNotFound	InvalidDetailLine
ConnectionDown	InvalidDetailLineType
DataOutOfRange	InvalidEmployeeNumber
DetailDoesNotSupportTriggeredEvents	InvalidGuestCount
DiscountNotFound	InvalidLineNumber
DiscountAmountRequired	InvalidMenuItemPrice
DiscountAmountTooLarge	InvalidOrderTypeNumber
DiscountAmountZero	InvalidPropertyNum
DiscountItemNotAllowed	InvalidRvcNum
DiscountNotAllowedFilterActive	InvalidServingPeriod
DiscountOnParentCombo	InvalidTableNumber
DuplicateLineNumber	InvalidTranslationSpecifier
EGatewayClientStartError	ItemDiscountNeedsParentItem
EGatewayClientStopError	LicensingFailed
EGatewayConnectionError	MenuItemOutOfOrder
EGatewayConnectionNotInPool	MissingDetailLinesElement
EGatewayWaitConnectionTimeout	MissingTransactionElement
EmployeeClockIOStatusMismatch	MissingTransactionHeaderElement
EmployeeIDMismatch	NoRequestHeader
EmployeeNotFound	NoSalesForDiscount
EmployeeRVCMismatch	NotImplemented
ErrorCreatingGuestcheck	NoSalesToApplyServiceCharge
ErrorInvalidWorkstation	NullInput
ErrorReadingCheck	PaidPartially
ErrorPickupCheck	PaymentAborted

FailedDataStoreInitialization	PriceMenuItemWithZeroAmount
FailedDbSettingLoad	SecurityInitFailed
FailedErrorTranslationInitial	ServiceChargeTaxClassNotFound
FailedPostCARequest	Success
FailedInitialization	TenderTypeNotFound
FailedLoggerInitialization	TransactionEmployeeNotFound
FailedSecurityAPIInitialization	TranslationFileNotAvailable
FailedSubmitPrintJob	UnhandledException
InternalCommunicationError	UnknownCreditCardType
InternalProcessingError	UnknownExceptionCode
InvalidArguments	TransactionLocked
AbortFromExtensionPlugin	FailedExtensionPlugin
NotFoundExtensionPlugin	SanityCodeNotFound
<b>string ErrorMessage</b>	
<i>Texts that further explains the exception and reason for failure.</i>	

## SimphonyPosAPI\_SanityCheckResponse

This structure is to encapsulate the result of the operation and the result of the sanity check performed.

### Public Attributes

<b>SimphonyPosApi_OperationalResult</b> <b>OperationalResult</b>
<i>A structure that indicates whether the current operation succeeded or failed. In case of failure, this will have appropriate error code and message.</i>
<b>bool</b> <b>SanitySuccess</b>
<i>Indicates whether or not the sanity check performed based on provided sanity code has succeeded. Acceptable sanity code is defined per 'Extension Plugin' and is therefore country-specific. A failure shall return an error code and error message.</i>
<b>string</b> <b>SanityErrorCode</b>
<i>Error code if sanity check validation failed. Error code is defined per 'Extension Plugin' and is therefore country-specific.</i>
<b>string</b> <b>SanityErrorMessage</b>
<i>Error message if sanity check validation failed. Error message is defined per 'Extension Plugin' and is therefore country-specific.</i>

## CheckTaxDataPerRate

This structure is to encapsulate the details of tax applied for a tax rate.

### Public Attributes

<b>int</b> <b>Index</b>
<i>The tax rate number (range 1 – 64).</i>
<b>decimal</b> <b>Tax</b>
<i>The total amount of tax applied for this rate.</i>

**bool Exempt**

*When True, indicates that tax for this rate has been exempted.*

**bool AnyApplied**

*When True, indicates that tax for this rate has been applied to one or more items.*

# 7

## Example and Code Snippets

### Calculate Totals of a Transaction

The following scenario describes a user who wants to find out the total amount of a transaction for items that are being ordered by the customer.

- Add two menu items
  - Add two condiments to the first menu item, and add one condiment to the second menu item
  - Quantity of 1<sup>st</sup> menu item is 3, and 2<sup>nd</sup> menu item is 1
  - Override the price of the 1<sup>st</sup> menu item
  - For the 1<sup>st</sup> menu item, instead of the default definition, pick up a specific menu item definition based on main and sub menu levels
  - Apply an open discount to the 1<sup>st</sup> menu item
  - Add a reference text to the 1<sup>st</sup> menu item
- Add a combo meal
- Apply an Open service charge
  - Add a reference text on the service charge
- Apply a Closed discount on the subtotal (that is, at the check level)

S. No	Type of Data	Parameter Name	Sample Data
1	Vendor code	<i>vendorCode</i>	<b>yzsroioq</b>
2	Menu Items and Condiments along with item level Discount	<i>ppMenuItems</i>	Object number of two menu items are <b>110003</b> and <b>110004</b> Object numbers of two condiments of first menu item are <b>41103</b> and <b>44502</b> . Object number of condiment of second menu item is <b>41103</b> . Overridden price for first menu item is <b>\$10</b> Menu levels to pick up first menu item is <b>Main Menu Level - 2 and Sub Menu Level - 3</b> Open discount percent for first menu item is <b>7%</b> Reference text for first menu item is <b>"Chef's favorite"</b>
3	Combo Meal	<i>ppComboMeals</i>	Combo meal details Object number of combo meal is <b>10</b> Object number of main item is <b>110003</b> Object numbers of side items are <b>41103</b> and <b>44502</b>

			Object number of drink is <b>110004</b>
4	Service Charge	<i>pServiceChg</i>	Open service charge is <b>6%</b> Reference text is " <b>6% service charge including tips</b> "
5	Subtotal Discount	<i>pSubTotalDiscount</i>	Open subtotal discount amount is <b>\$5</b>
6	Revenue Center object number	<i>revenueCenterObjectNum</i>	<b>3016</b>
7	Order Type	<i>orderType</i>	<b>1</b> (for example, Dine-in)
8	Employee Number	<i>employeeObjectNum</i>	<b>90001</b>
9	Total Response	<i>pTotalsResponse</i>	N/A (This parameter is to hold output.)

Order type is Dine-in

The following sample data is provided for the scenario mentioned above.

The method CalculateTransactionTotals can be used in this situation. Here is the signature of the method for quick reference.

## Calculate Transaction Totals Method Signature

```
void CalculateTransactionTotals
(
    string vendorCode,
    ref ARRAY(SimphonyPosAPI_MenuItem) ppMenuItems,
    ref ARRAY(SimphonyPosAPI_ComboMeal) ppComboMeals,
    ref SimphonyPosAPI_SvcCharge pServiceChg,
    ref SimphonyPosAPI_Discount pSubTotalDiscount,
    int revenueCenterObjectNum,
    short orderType,
    int employeeObjectNum,
    ref SimphonyPosAPI_TotalsResponse pTotalsResponse
)
```

The following code snippet demonstrates how data for input parameters of the **CalculateTransactionTotals** method can be constructed and used to invoke the method.

```
SimphonyPosAPIWebSoapClient mTSApi = new SimphonyPosAPIWebSoapClient();

string vendorCode = "lzsroioq";
int revenueCenterObjectNum = 3016;
int employeeObjectNum = 90001;
short orderType = 1; // e.g. Dine-in

public void InvokeCalculateTransactionTotalMethod()
{
    SimphonyPosApi_MenuItem[] ppMenuItems = GetMenuItemList();
    SimphonyPosApi_ComboMeal[] ppComboMeals = GetComboMealList();
    SimphonyPosApi_SvcCharge pSvcCharge = GetServiceCharge(true);
    SimphonyPosApi_Discount pSubtotalDiscount = GetSubtotalDiscount(true);
    SimphonyPosApi_TotalsResponse pTotalsResponse = new SimphonyPosApi_TotalsResponse();
```

```
mTSApi.CalculateTransactionTotals(vendorCode, ref ppMenuItems, ref ppComboMeals,
    ref pSvcCharge, ref pSubtotalDiscount, revenueCenterObjectNum, orderType,
    employeeObjectNum, ref pTotalsResponse);

if (pTotalsResponse.OperationalResult.Success)
{
    Console.WriteLine("Calculate Transaction Total succeeded...");

    Console.WriteLine("Total Due: " + pTotalsResponse.TotalsTotalDue);
    Console.WriteLine("Subtotal: " + pTotalsResponse.TotalsSubTotal);
    Console.WriteLine("Total Auto Service Charge: " +
        pTotalsResponse.TotalsAutoSvcChgTotals);
    Console.WriteLine("Total Service Charge (Manual): " +
        pTotalsResponse.TotalsOtherTotals);
    Console.WriteLine("Total Tax: " + pTotalsResponse.TotalsTaxTotals);
}
else
{
    Console.WriteLine(String.Format(
        "Calculate Transaction Total failed. Error Code: {0}, Error Message: {1}",
        pTotalsResponse.OperationalResult.ErrorCode,
        pTotalsResponse.OperationalResult.ErrorMessage));
}
}
```

The following sections explain constructing data for each input parameter with the sample data provided above.

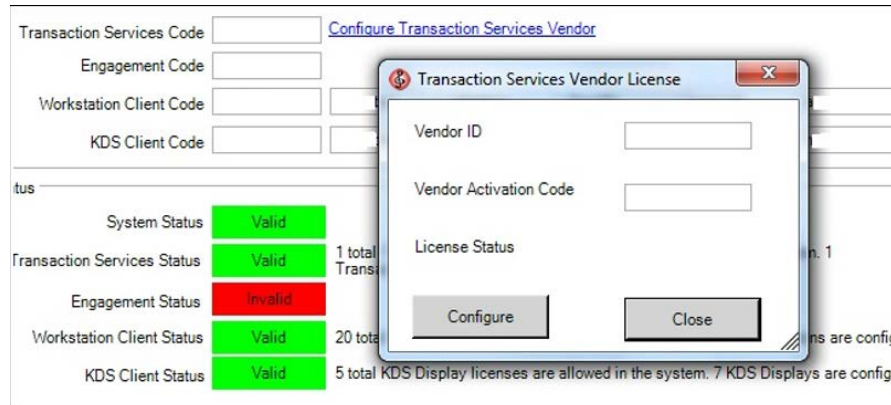
## Vendor Code

Vendor Code **is no longer supported beyond Symphony version 2.7 MR3**. This should be an empty value from clients. Vendor Code or Vendor Activation Code is a string value that uniquely identifies a vendor for Transaction Services. This was introduced to validate the license of TS API. The Vendor Activation Code should have been configured in the EMC in the following location for TS API to work properly.

**Enterprise** level, **Setup** tab, **Parameters** section, **Enterprise Parameters** module, **Licensing** tab.

This requirement was removed in Symphony version 2.7 MR3 and later. However, this parameter still exists in the latest Transaction Services API for backward compatibility. Client applications that integrate with Transaction Services API with Symphony version 2.7 MR3 or later can pass an empty value to this parameter, while prior versions can pass a valid Vendor Code that was distributed to them.

The following image shows the EMC dialog where you can configure the Vendor Activation Code for the Transaction Services API.



```

Parameter Signature

String      vendorCode

e.g.
string vendorCode = "yzsroioq";
    
```

## Menu Items and Condiments

This parameter represents the list of menu items with required condiments for those menu items. Each menu item and condiment is identified by an Object Number. Menu items and condiments are configured at the enterprise, property, or revenue center level in the EMC using the following module.

**Enterprise level, Configuration tab, Menu Items section, Menu Item Maintenance module.**

### Parameter Signature

```

ref ARRAY(SimphonyPosAPI_MenuItem)      ppMenuItems
    
```

### SimphonyPosAPI\_MenuItem Signature

```

public class SimphonyPosApi_MenuItem
{
    public SimphonyPosApi_MenuItemDefinition[] Condiments;
    public SimphonyPosApi_MenuItemDefinition MenuItem;
}
    
```

A menu item is the core foundation of all POS transactions. Everything ordered or added to the system is a menu item. In restaurant revenue centers, drinks and entrees are menu items. In retail revenue centers, shirts and hats are also considered menu items. Therefore, in Simphony, it can be said that any item being sold is a menu item.

The following code snippet demonstrates how to construct input data for the menu item list parameter. This code adds two menu items and respective condiments to the list as required. It also applies a discount (at the item level) to the first menu item.



```

private SymphonyPosApi_MenuItem[] GetMenuItemList()
{
    List<SymphonyPosApi_MenuItem> menuItemList = new List<SymphonyPosApi_MenuItem>();

    SymphonyPosApi_MenuItem firstMenuItem = new SymphonyPosApi_MenuItem();
    firstMenuItem.MenuItem = GetFirstMenuItem();
    firstMenuItem.MenuItem.ItemDiscount = GetItemDiscount(true);
    firstMenuItem.Condiments = new SymphonyPosApi_MenuItemDefinition[2];
    firstMenuItem.Condiments[0] = GetFirstCondimentItem();
    firstMenuItem.Condiments[1] = GetSecondCondimentItem();
    menuItemList.Add(firstMenuItem);

    SymphonyPosApi_MenuItem secondMenuItem = new SymphonyPosApi_MenuItem();
    secondMenuItem.MenuItem = GetSecondMenuItem();
    secondMenuItem.Condiments = new SymphonyPosApi_MenuItemDefinition[1];
    secondMenuItem.Condiments[0] = GetFirstCondimentItem();
    menuItemList.Add(secondMenuItem);

    return menuItemList.ToArray();
}

```

The following code demonstrates how to construct two menu items with given input data. Each menu item and condiment is identified by a unique identifier called Menu Item Object Number. The first and second menu items have Object Numbers 110003 and 110004 respectively. In this example, the price of the first menu item is overridden by \$7. It is possible that any menu item or condiment is configured to have more than one definition with a different price record for each definition. When no menu levels are specified, it picks up the first definition by default. In this example, both main and sub menu levels are specified for the first menu item in order to pick up a particular definition instead of the default. A reference text is also added to the first menu item for reference.

```

private SymphonyPosApi_MenuItemDefinition GetFirstMenuItem()
{
    SymphonyPosApi_MenuItemDefinition menuItemDefn = new
        SymphonyPosApi_MenuItemDefinition();
    menuItemDefn.MiObjectNum = 110003;
    menuItemDefn.MiOverridePrice = "7";
    menuItemDefn.MiMenuLevel = 2;
    menuItemDefn.MiSubLevel = 3;

    // Specify 3 as quantity and 'Make it spicy' as reference text
    menuItemDefn.MiReference = "<ExtraData><MiQuantity>3</MiQuantity></ExtraData>Make
it spicy";
    return menuItemDefn;
}

private SymphonyPosApi_MenuItemDefinition GetSecondMenuItem()
{
    int menuItemObjectNum = 110004;
    SymphonyPosApi_MenuItemDefinition menuItemDefn = new
        SymphonyPosApi_MenuItemDefinition();
    menuItemDefn.MiObjectNum = menuItemObjectNum;
    return menuItemDefn;
}

```

The following code demonstrates how to construct an object for two condiment menu items with given input data.

```
private SymphonyPosApi_MenuItemDefinition GetFirstCondimentItem()
{
    int menuItemObjectNum = 41103;
    SymphonyPosApi_MenuItemDefinition menuItemDefn = new
        SymphonyPosApi_MenuItemDefinition();
    menuItemDefn.MiObjectNum = menuItemObjectNum;
    return menuItemDefn;
}

private SymphonyPosApi_MenuItemDefinition GetSecondCondimentItem()
{
    int menuItemObjectNum = 44502;
    SymphonyPosApi_MenuItemDefinition menuItemDefn = new
        SymphonyPosApi_MenuItemDefinition();
    menuItemDefn.MiObjectNum = menuItemObjectNum;
    return menuItemDefn;
}
```

The following code demonstrates how to construct a discount object for given input data. Each discount configured in the EMC is identified by a unique identifier called Discount Object Number. For a preset discount, the amount or percentage of the discount is taken from a value configured in the EMC. However, for an open discount, the amount or percentage of the discount should be supplied by the caller. The property DiscAmountOrPercent could be an amount or percent based on how the given discount is configured using the EMC. This example demonstrates that 10 is the Discount Object Number of an open discount and the caller is applying a 7% discount to a menu item. All manual discounts should be added to applicable menu items explicitly in this way, while API applies an automatic discount implicitly by itself.

```
private SymphonyPosApi_Discount GetItemDiscount(bool isOpenDiscount)
{
    SymphonyPosApi_Discount discount = new SymphonyPosApi_Discount();
    discount.DiscObjectNum = 10;

    // percentage or amount based on how it's configured in EMC
    if (isOpenDiscount)
        discount.DiscAmountOrPercent = "7";

    discount.DiscReference = "Mother's day discount";
    return discount;
}
```

## Combo Meal

A combo meal is a combination meal (for example, a burger with fries and a drink, or pancake with ham and coffee) offered at a lower price than the menu item's cost individually. Configure a combo meal in the EMC before adding it through the TS API.

Combo meals can be found in EMC at the **Property** level, **Configuration** tab, **Sales** section, **Combo Meals** module.

To configure a combo meal:

1. Create a menu item.
2. Create a Combo Meal Menu Item class.
3. Add the menu item to the Combo Meal class.
4. Create a Combo Meal Group.
5. Add a Main, Drink, and Side Item.

#### Parameter Signature

```
ref ARRAY(SymphonyPosAPI_ComboMeal) ppComboMeals
```

#### SimphonyPosApi\_ComboMeal Signature

```
public class SimphonyPosApi_ComboMeal
{
    public SimphonyPosApi_MenuItem ComboMealMainItem;
    public SimphonyPosApi_MenuItem ComboMealMenuItem;
    public int ComboMealObjectNum;
    public SimphonyPosApi_MenuItem[] SideItems;
}
```

The following code snippet demonstrates how to construct a combo meal object for given input data. This example adds a main menu item, two side items, and a drink to form a combo meal. Each combo meal is identified by a unique identifier called Combo Meal Object Number. Log in to the EMC to obtain the object numbers of the combo meal and related items.

```
private SimphonyPosApi_ComboMeal[] GetComboMealList()
{
    SimphonyPosApi_ComboMeal[] comboMeal = new SimphonyPosApi_ComboMeal[1];

    SimphonyPosApi_ComboMeal comboMeal1 = new SimphonyPosApi_ComboMeal();
    comboMeal1.ComboMealObjectNum = 10;

    // Add a Main item
    SimphonyPosApi_MenuItem mainItem = new SimphonyPosApi_MenuItem();
    mainItem.MenuItem = new SimphonyPosApi_MenuItemDefinition();
    mainItem.MenuItem.MiObjectNum = 110003;
    comboMeal1.ComboMealMainItem = mainItem;

    // Add 2 Side items
    SimphonyPosApi_MenuItem[] sideItemList = new SimphonyPosApi_MenuItem[2];
    SimphonyPosApi_MenuItem firstSideItem = new SimphonyPosApi_MenuItem();
    firstSideItem.MenuItem = new SimphonyPosApi_MenuItemDefinition();
}
```

```

firstSideItem.MenuItem.MiObjectNum = 41103;
sideItemList[0] = firstSideItem;
SymphonyPosApi_MenuItem secondSideItem = new SymphonyPosApi_MenuItem();
secondSideItem.MenuItem = new SymphonyPosApi_MenuItemDefinition();
secondSideItem.MenuItem.MiObjectNum = 44502;
sideItemList[1] = secondSideItem;
comboMeal1.SideItems = sideItemList;

// Add a Drink
SymphonyPosApi_MenuItem menuItem = new SymphonyPosApi_MenuItem();
menuItem.MenuItem = new SymphonyPosApi_MenuItemDefinition();
menuItem.MenuItem.MiObjectNum = 110004;
comboMeal1.ComboMealMenuItem = menuItem;

comboMeal[0] = comboMeal1;
return comboMeal;
}

```

## Service Charge

A service charge is an amount that is added to a sales transaction for a service rendered. There are two ways to add a service charge to the transaction:

- Automatic Service Charge
- Manual Service Charge

An automatic service charge is a service charge that applies to all items in the Menu Item Class with the **Add to Automatic Service Charge Itemizer** option enabled, without entry by operator intervention.

A manual service charge should be added to the input parameter of the TS API.

Configure a service charge in the EMC at the **Enterprise** or **Property** level, **Configuration** tab, **Sales, Service Charges** module

### Parameter Signature

```
ref SymphonyPosAPI_SvcCharge pServiceChg
```

### SymphonyPosApi\_SvcCharge Signature

```

public class SymphonyPosApi_SvcCharge
{
    public string SvcChgAmountOrPercent;
    public int SvcChgObjectNum;
    public string SvcChgReference;
}

```

The following code demonstrates how to construct a service charge object for a given input data. Each service charge configured in the EMC is identified by a unique identifier called Service Charge Object Number. For a preset service charge, the amount or percentage of the service charge is taken from a value configured in the EMC. For an open service charge, the amount or percentage of the service charge should be supplied

by the caller. The field `SvcChgAmountOrPercent` can be an amount or percent based on how it is configured in EMC. This example demonstrates that 12 is the Service Charge Object Number of an open service charge and the caller is applying a 6% service charge on the guest check. Any manual service charge should be added to the guest check explicitly in this way, while API applies an automatic service charge implicitly by itself.

```
private SymphonyPosApi_SvcCharge GetServiceCharge(bool isOpenServiceCharge)
{
    SymphonyPosApi_SvcCharge serviceCharge = new SymphonyPosApi_SvcCharge();
    serviceCharge.SvcChgObjectNum = 12;

    if (isOpenServiceCharge)
        serviceCharge.SvcChgAmountOrPercent = "6"

    serviceCharge.SvcChgReference = "6% service charge including tips";

    return serviceCharge;
}
```

## Subtotal Discount

A discount reduces the price of an item or items on a check. Discounts are generally used for promotional purposes (for example, a coupon for a free dessert) or for customer satisfaction. Discounts can be configured as Subtotal Discounts or Item Discounts. An Item Discount is used to discount a single item, whereas Subtotal Discounts apply to one or more items on the check based on the configuration of the discount in the EMC.

By default, all discounts are Subtotal Discounts, which means that the discount applies to all items on a check that belong to a Menu Item Group or Itemizer Group affected by the discount. A discount is a subtotal discount when the **This is an Item Discount** option is disabled in EMC.

There are three different types of activation for discounts:

1. Manual

A manual discount is applied by the user to a check. This type of discount is a traditional discount.

2. Automatic

An automatic discount is applied by the discount engine when certain criteria of the transaction are met. As a user adds items, the workstation continually looks for items that will trigger an automatic discount, and then the award amount is applied to the check if necessary.

3. Coupon

An automatic coupon discount is an automatic discount with one difference: the user must first apply the discount to the check, letting the discount engine know that the discount is available for the check.

## Parameter Signature

```
ref SymphonyPosAPI_Discount pSubTotalDiscount
```

## SymphonyPosApi\_Discount Signature

```
public class SymphonyPosApi_Discount
{
    public string DiscAmountOrPercent;
    public int DiscObjectNum;
    public string DiscReference;
}
```

The following code snippet demonstrates how to construct a Subtotal discount object with the given input data. Each discount configured in EMC is identified by a unique identifier called Discount Object Number. For a Preset discount, the amount or percentage of the discount will be taken from a value configured in EMC. For an Open discount, the amount or percentage of the discount should be supplied by the caller. The property DiscAmountOrPercent could be an amount or percent based on how the discount is configured in EMC. This example demonstrates that 11 is the Discount Object Number of an open Subtotal discount and the caller is applying a 5% discount to the guest check for all triggered menu item groups. All manual Subtotal discounts should be added to the guest check explicitly in this way, while API applies automatic Subtotal discounts implicitly.

```
private SymphonyPosApi_Discount GetSubtotalDiscount(bool isOpenDiscount)
{
    SymphonyPosApi_Discount subTotalDiscount = new SymphonyPosApi_Discount();
    subTotalDiscount.DiscObjectNum = 11;

    // percentage or amount based on how it's configured in EMC
    if (isOpenDiscount)
        subTotalDiscount.DiscAmountOrPercent = "5";

    subTotalDiscount.DiscReference = "Weekend discount";
    return subTotalDiscount;
}
```

## Revenue Center Object Number

Revenue centers are a distinctly identifiable department, division, or unit of a firm that generates revenue through the sale of goods and/or services. For example, the rooms department and food-and-beverages department of a hotel are revenue centers. Each revenue center of a property is identified by a unique identifier called Revenue Center Object Number.

Revenue center can be found in the EMC at the **Property** level, **Setup** tab, **Property Configuration, RVC Configuration** module. In the following example, **3016** is the Object Number of a revenue center.

**Parameter Signature**

```
int revenueCenterObjectNum
e.g.,
int revenueCenterObjectNum = 3016;
```

## Order Type ID

An order type is a configurable menu item sales category. Order types can be used to control tax rates that are active during a transaction. Dine-out and Dine in are common-order types.

**Parameter Signature**

```
short orderType
e.g., short orderType = 1; //e.g., Dine-in
```

## Employee Object Number

Each employee at a property is identified by a unique identifier called Employee Object Number. This object number of an employee should be passed to the API for this operation to associate an employee for a given transaction.

Employee details, including their object number can be found in EMC:

**Property level, Configuration tab, Personal, Employee Maintenance** module.

In this example, **90001** is the object number of an employee called David and is assigned to revenue center 3016.

**Parameter Signature**

```
int employeeObjectNum
e.g., int employeeObjectNum = 90001;
```

## API Response

The response of the method call can be found in the ***pTotalsResponse*** parameter. The value of ***OperationalResult*** property of pTotalsResponse object indicates whether or not the operation succeeded. If the operation succeeded, data for subtotal, total due, tax amount, auto and manual service charge amounts can be found in the respective properties of ***pTotalsResponse***. If the operation failed, ***OperationalResult.ErrorCode*** property will hold the error code while the detailed error message can be found in ***OperationalResult.ErrorMessage*** property.

**Parameter Signature**

```
ref SymphonyPosAPI_TotalsResponse pTotalsResponse
```

### SimphonyPosApi\_TotalsResponse Signature

```
public class SimphonyPosApi_TotalsResponse
{
    public SimphonyPosApi_OperationalResult OperationalResult;
    public string TotalsTotalDue
    public string TotalsSubTotal
    public string TotalsTaxTotals
    public string TotalsAutoSvcChgTotals
    public string TotalsOtherTotals
}
```

## Create a Guest Check

After the user calculates and reviews the total amount of a transaction, he or she may want to post that transaction and create a guest check in the Simphony POS database by providing tender/payment details. The tender can be of any type (for example, cash or credit/debit).

#### NOTE:

Transaction Services only supports the **MCreditDebit Payment** driver for credit/debit card payment. There is no Stored Value Card (SVC) support for Transaction Services.

The method `PosTransactionEx` can be used for this purpose.

TS API supports the auto-fire feature on guest checks. Auto-fire means that the guest check will be immediately created in the system, but it will fire only when the time specified to fire at the time of guest check creation is attained. This example demonstrates the auto-fire feature by firing the guest check only after 12 hours from when the guest check is posted. The auto-fire feature is used in hotels where a guest wants to order food in the morning to be consumed at dinner. In this case, the guest check will be created in the system as soon as the transaction posts, but will fire only at the specified time in the evening, so that the chef can prepare ordered food for dinner for the specific customer.

The input data for menu items, combo meals, discounts, and service charges are described in the previous section for calculating totals. The example is taken for this method as well. However, this method needs data for the following additional parameters.

### Tender/Payment Details

The `Post Transaction` method posts current transactions to the Simphony POS database to create a guest check. Like `CalculateTransactionTotals`, this method calculates the transaction totals. If the payment/tender media is of `Service Total`, the system will create the guest check and keep it in the `Open` state. When the tender media with appropriate payment details (cash, credit/debit) are passed for full payment, the check will be created and changed to a `Closed` state at the end of the call. A tender with partial payment will still have the created check in the `Open` state only. Another tender with payment for the balance amount can be added later to that check using a method called `AddToExistingCheckEx` to close the check.



**Post Transaction Method Signature**

```

void PostTransactionEx
(
    String                vendorCode,
    ref SymphonyPosAPI_GuestCheck    pGuestCheck,
    ref ARRAY(SymphonyPosAPI_MenuItem) ppMenuItems,
    ref ARRAY(SymphonyPosAPI_ComboMeal) ppComboMeals,
    ref SymphonyPosAPI_SvcCharge    ServiceChg,
    ref SymphonyPosAPI_Discount    pSubTotalDiscount,
    ref SymphonyPosAPI_TmedDetailItemEx    pTmedDetailEx,
    ref SymphonyPosAPI_TotalsResponse    pTotalsResponse,
    ref ARRAY(string)    ppCheckPrintLines,
    ref ARRAY(string)    ppVoucherOutput
)

```

The following code snippet demonstrates how data for input parameters of the PostTransactionEx method can be constructed and used to invoke the method. This example demonstrates creating a guest check with the same input data mentioned in the previous section for calculate totals.

```

SymphonyPosAPIWebSoapClient mTSApi = new SymphonyPosAPIWebSoapClient();

string vendorCode = "lzsroioq";
int revenueCenterObjectNum = 3016;
int employeeObjectNum = 90001;
short orderType = 1; // e.g. Dine-in

public void InvokePostTransactionEx()
{
    bool isAutoFireCheck = true;

    SymphonyPosApi_GuestCheck guestCheck = new SymphonyPosApi_GuestCheck();
    guestCheck.CheckOrderType = orderType;
    guestCheck.CheckEmployeeObjectNum = employeeObjectNum;
    guestCheck.CheckRevenueCenterID = revenueCenterObjectNum;

    // Optional parameters
    guestCheck.CheckGuestCount = 2; // Number of guests
    guestCheck.CheckTableObjectNum = 5; // Dining table number

    if (isAutoFireCheck)
    {
        // 0x10 is the status bit for auto fire (a.k.a. future order)
        // Note: 0x1 - Rush Order, 0x2 - VIP Order, 0x10 - Auto fire Order
        guestCheck.CheckStatusBits |= 0x10;
        // Check fires after 12 hours
        guestCheck.CheckDateToFire = DateTime.Now.AddHours(1);
    }

    string[] ppCheckPrintLines = new string[] { "" }; // Output parameter
    string[] ppVoucherOutput = new string[] { "" }; // Output parameter
}

```

```

SimphonyPosApi_MenuItem[] ppMenuItems = GetMenuItemList();
SimphonyPosApi_ComboMeal[] ppComboMeals = GetComboMealList();
SimphonyPosApi_SvcCharge pSvcCharge = GetServiceCharge(true);
SimphonyPosApi_Discount pSubtotalDiscount = GetSubtotalDiscount(true);
SimphonyPosApi_TmedDetailItemEx tenderMedia = GetTenderMedia(
    TenderMediaType.CreditCard);
SimphonyPosApi_TotalsResponse pTotalsResponse = new
    SimphonyPosApi_TotalsResponse();

mTSApi.PostTransactionEx(vendorCode, ref guestCheck, ref ppMenuItems,
    ref ppComboMeals, ref pSvcCharge, ref pSubtotalDiscount, ref tenderMedia,
    ref pTotalsResponse, ref ppCheckPrintLines, ref ppVoucherOutput);

if (guestCheck.OperationalResult != null && guestCheck.OperationalResult.Success)
{
    Console.WriteLine("Post Transaction operation has succeeded...");

    Console.WriteLine("Guest Check ID: " + guestCheck.CheckID);
    Console.WriteLine("Guest Check Number: " + guestCheck.CheckNum);
    Console.WriteLine("Guest Check Sequence Number: " + guestCheck.CheckSeq);

    Console.WriteLine("Total Due: " + pTotalsResponse.TotalsTotalDue);
    Console.WriteLine("Subtotal: " + pTotalsResponse.TotalsSubTotal);
    Console.WriteLine("Total Auto Service Charge: " +
        pTotalsResponse.TotalsAutoSvcChgTotals);
    Console.WriteLine("Total Service Charge (Manual): " +
        pTotalsResponse.TotalsOtherTotals);
    Console.WriteLine("Total Tax: " + pTotalsResponse.TotalsTaxTotals);
}
else
{
    Console.WriteLine(String.Format("Post Transaction operation has failed.
        Error Code: {0}, Error Message: {1}",
            pTotalsResponse.OperationalResult.ErrorCode,
            pTotalsResponse.OperationalResult.ErrorMessage));
}
}

```

The following sections explain the parameters that are not listed in the Calculate Totals Transaction method in the previous section. Refer to the previous section for all other parameters.

## Guest Check

For post transaction operations, the caller of the method is expected to pass data for the following mandatory properties of the guestCheck parameter:

- CheckEmployeeObjectNum
- CheckRevenueCenterID
- CheckOrderType

The following properties are optional:

- CheckGuestCount

- CheckTableObjectNum
- CheckStatusBits
- CheckDateToFire

The following properties will be populated by the API when the operation succeeds. In case of failure, only OperationalResult will populate to hold data for error code and error message:

- CheckID
- CheckNum
- CheckSeq
- OperationalResult
- PCheckInfoLines
- PPrintJobIds

Check Number and Check Sequence Number are used to identify the created guest check uniquely and they can be used in the future for updating a specific guest check. For example, this method can create a guest check with a partial payment, and another method called AddToExistingCheckEx can be invoked later to add another tender for the balance amount to the same check by specifying the Check Number and Check Sequence Number of that check. An example of the AddToExistingCheckEx method is provided in the next section.

## Tender/Payment

Tender media is a form of payment or a service total used on a guest check. Each tender media is identified by a unique identifier called Tender Media Object Number. Tender media must be configured in EMC before it can be used in the TS API.

The following code snippet demonstrates how tender media with sample data can be constructed based on the tender media type. This code indicates the data expected by the payment driver while creating a guest check.

```
private SymphonyPosApi_TmedDetailItemEx GetTenderMedia(TenderMediaType tmType)
{
    SymphonyPosApi_TmedDetailItemEx tenderMedia = new SymphonyPosApi_TmedDetailItemEx();
    SymphonyPosApi_EPayment ePayment = new SymphonyPosApi_EPayment();
    switch (tmType)
    {
        case TenderMediaType.Cash:
        {
            tenderMedia.TmedObjectNum = 12;
            tenderMedia.TmedPartialPayment = "30"; // tendered amount excluding tip

            // indicates cash payment
            ePayment.PaymentCommand = EPaymentDirective.NO_E_PAYMENT;
            ePayment.TipAmount = "5"; // tendered amount for tip

            tenderMedia.TmedEPayment = ePayment;
            tenderMedia.TmedReference = "Total amount tendered (including tip) is $35";
            break;
        }
    }
}
```

```
}
case TenderMediaType.CreditCard:
{
    tenderMedia.TmedObjectNum = 30;
    ePayment.PaymentCommand = EPaymentDirective.CREDIT_AUTHORIZE_AND_PAY;

    // Track2 has most of the data required by the payment driver
    ePayment.Track2Data = "7777666655554444=00010002000370783149";
    ePayment.BaseAmount = "30"; // Base amount excluding tip
    ePayment.TipAmount = "5"; // Amount for tip
    ePayment.CashBackAmount = "15";

    tenderMedia.TmedReference =
        "Total amount to be deducted from CREDIT CARD is $50";
    break;
}
case TenderMediaType.DebitCard:
{
    tenderMedia.TmedObjectNum = 31;
    ePayment.PaymentCommand = EPaymentDirective.DEBIT_AUTHORIZE_AND_PAY;

    // Track2 has most of the data required by the payment driver
    ePayment.Track2Data = "8888777766665555=00020003000470783249";
    ePayment.BaseAmount = "30"; // Base amount excluding tip
    ePayment.TipAmount = "5"; // Amount for tip
    ePayment.CashBackAmount = "15";

    tenderMedia.TmedReference =
        "Total amount to be deducted from DEBIT CARD is $50";
    break;
}
case TenderMediaType.StoredValueCard:
{
    tenderMedia.TmedObjectNum = 35;
    ePayment.PaymentCommand = EPaymentDirective.STORED_VALUE_CARD_REDEEM;

    // Track2 has most of the data required by the payment driver
    ePayment.Track2Data = "9999888877776666=10020003000470783249";
    ePayment.BaseAmount = "30"; // Base amount excluding tip
    ePayment.TipAmount = "5"; // Amount for tip
    ePayment.CashBackAmount = "15";

    tenderMedia.TmedReference =
        "Total amount to be deducted from SVC CARD is $50";
}
case TenderMediaType.ServiceTotal:
{
    tenderMedia.TmedObjectNum = 49;
    tenderMedia.TmedReference = "Payment is not done yet";
    break;
}
}
return tenderMedia;
}
```

For details on other input parameters of this method, refer to the [Calculate Transaction Totals Method Signature](#) section of this document.

## API Response

If the post transaction operation succeeded, the output details (for example, Check ID, Check Number, and Check Sequence Number) of the created check will be populated in the appropriate fields of the pGuestCheck parameter. The operation also populates the print receipt of the guest check in the ppCheckPrintLines property of pGuestCheck, while it populates the credit voucher of the current transaction in the ppVoucherOutput property of the same parameter. If this method encounters a problem (for example, payment failure), it will not create a guest check and will throw an appropriate exception to the caller. In addition, the OperationalResult property of pTotalsResponse parameter will hold the appropriate error code and message in that case.

## Add an Item to an Open Guest Check

After the user posts a transaction or check to the Symphony POS database, sometimes, he or she may need to update it. For example, when a transaction was posted to create a guest check with a Service Total as tender media. That is, payment has not been made yet for the transaction. In this case, the user may want to add a tender to the guest check later, in order to make payment for the amount due. In such cases, the AddToExistingCheckEx method can be used.

The following code snippet demonstrates adding a tender media to an existing open guest check to make payment so the check can be closed. Similar to a tender media, other items such as menu item, combo meal, service charge, or discount can be added to an open guest check.

```
SimphonyPosAPIWebSoapClient mTSApi = new SimphonyPosAPIWebSoapClient();
string vendorCode = "lzsroioq"; // This can be empty from 2.7MR3 onwards

public void AddTenderToExistingGuestCheck()
{
    SimphonyPosApi_GuestCheck guestCheck = new SimphonyPosApi_GuestCheck();
    guestCheck.CheckEmployeeObjectNum = 90001;
    guestCheck.CheckRevenueCenterID = 3016;
    guestCheck.CheckNum = 1043; // Check Number of Guest Check to which tender needs to applied

    guestCheck.CheckSeq = 534418293; // Sequence Number of Guest Check

    string[] ppCheckPrintLines = new string[] { "" };
    string[] ppVoucherOutput = new string[] { "" };

    SimphonyPosApi_MenuItem[] ppMenuItems = new SimphonyPosApi_MenuItem[0];
    SimphonyPosApi_ComboMeal[] ppComboMeals = new SimphonyPosApi_ComboMeal[0];
    SimphonyPosApi_SvcCharge pSvcCharge = new SimphonyPosApi_SvcCharge();
    SimphonyPosApi_Discount pSubtotalDiscount = new SimphonyPosApi_Discount();
    SimphonyPosApi_TmedDetailItemEx tenderMedia =
        GetTenderMedia(TenderMediaType.DebitCard);
    SimphonyPosApi_TotalsResponse pTotalsResponse = new SimphonyPosApi_TotalsResponse();

    mTSApi.AddToExistingCheckEx(vendorCode, ref guestCheck, ref ppMenuItems,
```

```
ref ppComboMeals, ref pSvcCharge, ref pSubtotalDiscount, ref tenderMedia,
ref pTotalsResponse, ref ppCheckPrintLines, ref ppVoucherOutput);

if (pTotalsResponse.OperationalResult.Success)
{
    Console.WriteLine("Add To Existing Check succeeded...");

    Console.WriteLine("Total Due: " + pTotalsResponse.TotalsTotalDue);
    Console.WriteLine("Subtotal: " + pTotalsResponse.TotalsSubTotal);
    Console.WriteLine("Total Auto Service Charge: " +
        pTotalsResponse.TotalsAutoSvcChgTotals);
    Console.WriteLine("Total Service Charge (Manual): " +
        pTotalsResponse.TotalsOtherTotals);
    Console.WriteLine("Total Tax: " + pTotalsResponse.TotalsTaxTotals);
}
else
{
    Console.WriteLine(String.Format("Add To Existing Check failed.
        Error Code: {0}, Error Message: {1}",
        pTotalsResponse.OperationalResult.ErrorCode,
        pTotalsResponse.OperationalResult.ErrorMessage));
}
}
```

Like the tender media given in this example, one or more items such as a menu item, combo meal, service charge, and subtotal discount can be added to an existing open guest check using this method. However, this method can operate only on open guest checks and will throw an exception if the caller tries to add an item a closed check.

For details on all input parameters of this method, see the [Calculate Transaction Totals Method Signature](#) and [Create a Guest Check](#) sections of this document.

## API Response

Adding an item to an existing open guest check will close the original check and create a new check with all items internally. When this method is invoked, `pGuestCheck` parameter will be interrogated and fields such as Check ID, Check Number, and Check Sequence Number will be updated with details of the new guest check.

## Void All Items of an Open Guest Check

Sometimes a user may need to cancel a posted transaction or check due to incorrect order entry or other reasons. In such cases, the `VoidTransaction` method can be used to achieve it. This method can operate on only open guest checks. It voids each item in the check and keeps the check open. To identify the check, the caller is expected to pass both Check Number and Check Sequence Number to this method.

## VoidTransaction Method Signature

```
void VoidTransaction  
(  
    string vendorCode,  
    ref SymphonyPosAPI_GuestCheck pGuestCheck  
)
```

The following code snippet demonstrates invoking the VoidTransaction method with sample input data.

```
SimphonyPosAPIWebSoapClient mTSApi = new SimphonyPosAPIWebSoapClient();  
string vendorCode = "lzsroioq"; // This can be empty from 2.7MR3 onwards  
  
public void InvokeVoidTransaction()  
{  
    SimphonyPosApi_GuestCheck guestCheck = new SimphonyPosApi_GuestCheck();  
    guestCheck.CheckNum = 1008; // Check Number of the Guest Check  
    guestCheck.CheckSeq = 315015863; // Sequence Number of the Guest Check  
  
    mTSApi.VoidTransaction(vendorCode, ref guestCheck);  
  
    if (guestCheck.OperationalResult.Success)  
    {  
        Console.WriteLine("Void Transaction succeeded...");  
    }  
    else  
    {  
        Console.WriteLine(String.Format("Void Transaction failed. Error Code: {0},  
            Error Message: {1}", guestCheck.OperationalResult.ErrorCode,  
            guestCheck.OperationalResult.ErrorMessage));  
    }  
}
```

## API Response

If the operation succeeded, the OperationalResult.Success field will hold True. In addition the, OperationalResult.ErrorCode will hold the error code, while OperationalResult.ErrorMessage holds the reason for failure.

## Get Status of a Print Job

When a transaction is posted to the POS database using the PostTransactionEx method, at the end of posting, the method creates a print job to print the guest check. The ID of the print job is stored and returned to the caller via the PPrintJobIds field of the SimphonyPosApi\_GuestCheck parameter. The job IDs are accumulated in the PPrintJobIds parameter. The last job ID present in the PPrintJobIds array indicates the print job ID of the last guest check that was posted to the Simphony POS database.

If a guest check did not print due to any reason, the status of the corresponding print job can be retrieved using the *CheckPrintJobStatus* method of TS API. This method accepts the ID of the print job as one of the parameters and returns the status of that job.

The following code snippet demonstrates how to retrieve the status of a print job.

```
SimphonyPosAPIWebSoapClient mTSApi = new SimphonyPosAPIWebSoapClient();
string vendorCode = "lzsroioq"; // This can be empty from 2.7MR3 onwards

public void InvokeCheckPrintJobStatus()
{
    // Print Job Id corresponding to a Guest Check that didn't print
    int printJobId = 1052;
    SimphonyPrintApi_PrintJobStatus printJobStatus =
        new SimphonyPrintApi_PrintJobStatus();

    mTSApi.CheckPrintJobStatus(vendorCode, printJobId, ref printJobStatus);

    if (printJobStatus.OperationalResult.Success)
    {
        Console.WriteLine("Checking status of print job succeeded...");
        Console.WriteLine("Print job status:" + printJobStatus.Status.ToString());
    }
    else
    {
        Console.WriteLine("Checking status of print job failed. Error Code: {0},
            Error Message: {1}", printJobStatus.OperationalResult.ErrorCode,
            printJobStatus.OperationalResult.ErrorMessage);
    }
}
```

## API Response

If the operation succeeds, the `OperationalResult.Success` field will hold `True`. The `printJobStatus` will hold the status of queries print job. The following enumerator has potential status of any print job.

```
public enum SimphonyPrintApi_Status
{
    JobPending = 0,
    JobComplete = 1,
    JobAborted = 2,
    JobSentToBackup = 3,
    JobFailed = 4,
    JobNotFound = 5,
}
```

## Get Summary of All Open Guest Checks

At times, the user may want to get a summary of all open guest checks from all revenue centers of the property.



## GetOpenChecks Method Signature

```
void GetOpenChecks
(
    string                vendorCode,
    int                   EmployeeObjectNum,
    ref SymphonyPosAPI_OpenChecks openChecks
)
```

The following code snippet demonstrates retrieving a summary of all open guest checks created by a specific employee whose Employee Object Number is 90001. The caller has to pass 0 (zero) to the Employee Object Number parameter if he or she wishes to fetch all open checks irrespective of the owner who created the check.

```
SimphonyPosAPIWebSoapClient mTSApi = new SimphonyPosAPIWebSoapClient();
string vendorCode = "lzsroioq"; // This can be empty from 2.7MR3 onwards
int employeeObjectNum = 90001; // Pass 0 to fetch all open checks irrespective of Check
Owner

public void InvokeGetOpenChecks()
{
    SimphonyPosApi_OpenChecks openChecks = new SimphonyPosApi_OpenChecks();

    mTSApi.GetOpenChecks(vendorCode, employeeObjectNum, ref openChecks);

    if (openChecks.OperationalResult.Success)
    {
        Console.WriteLine("Get Open Check succeeded...");

        foreach (SimphonyPosApi_CheckSummary check in openChecks.CheckSummary)
        {
            Console.WriteLine("Check Number:" + check.CheckNum);
            Console.WriteLine("Check Sequence Number:" + check.CheckSeq);
            Console.WriteLine("Check Total Due:" + check.CheckTotalDue);
            // The field CheckRevenueCenterObjectNum returns RVC ID (not Object Number)
            Console.WriteLine("RVC ID:" + check.CheckRevenueCenterObjectNum);
        }
    }
    else
    {
        Console.WriteLine("Get Open Check failed. Error Code: {0},
            Error Message: {1}",
            openChecks.OperationalResult.ErrorCode,
            openChecks.OperationalResult.ErrorMessage);
    }
}
```

## API Response

If the operation succeeds, the `OperationalResult.Success` field will hold `True` and a summary of open guest checks will be populated in the `openChecks` parameter. The field `CheckRevenueCenterObjectNum` of `SimphonyPosApi_CheckSummary` structure is mislabeled and it will return Revenue Center ID and not Revenue Center Object Number as the name suggests.

In case of failure, the `OperationalResult.Success` field will hold `False`, while the `OperationalResult.ErrorCode` holds the error code and `OperationalResult.ErrorMessage` holds reason for failure.

## Get Open Guest Checks with RVC Object Number

If the caller expects the field `CheckRevenueCenterObjectNum` of `SimphonyPosApi_CheckSummary` to hold RVC Object Number (instead of RVC ID), the `GetOpenChecksEx` method can be used instead of `GetOpenChecks`, which is explained in the previous section. `GetOpenChecks` populates `CheckRevenueCenterObjectNum` with the ID of the revenue center, while `GetOpenChecksEx` populates the same field with Object Number.

### GetOpenChecksEx Method Signature

```
void GetOpenChecksEx
(
    string          vendorCode,
    int             employeeObjectNum,
    ref SimphonyPosAPI_OpenChecks openChecks
)
```

The following code snippet demonstrates retrieving a summary of all open guest checks created by a specific employee whose Employee Object Number is 90001. The caller has to pass 0 (zero) to the Employee Object Number parameter if he or she wishes to fetch all open checks irrespective of the owner who created the check. The property `CheckRevenueCenterObjectNum` of response object returns the object number of RVC instead of ID. Object number is different from ID.

```
SimphonyPosAPIWebSoapClient mTSApi = new SimphonyPosAPIWebSoapClient();
string vendorCode = "lzsroioq"; // This can be empty from 2.7MR3 onwards
int employeeObjectNum = 90001; // Pass 0 to fetch all open checks irrespective of Check
Owner

public void InvokeGetOpenChecks()
{
    SimphonyPosApi_OpenChecks openChecks = new SimphonyPosApi_OpenChecks();
    mTSApi.GetOpenChecksEx(vendorCode, employeeObjectNum, ref openChecks);
    if (openChecks.OperationalResult.Success)
    {
        Console.WriteLine("Get Open Check succeeded...");
        foreach (SimphonyPosApi_CheckSummary check in openChecks.CheckSummary)
        {
            Console.WriteLine("Check Number:" + check.CheckNum);
            Console.WriteLine("Check Sequence Number:" + check.CheckSeq);
            Console.WriteLine("Check Total Due:" + check.CheckTotalDue);
            Console.WriteLine("RVC Object Number:" + check.CheckRevenueCenterObjectNum);
        }
    }
    else
```

```
{  
    Console.WriteLine("Get Open Check failed. Error Code: {0}, Error Message: {1}",  
        openChecks.OperationalResult.ErrorCode,  
        openChecks.OperationalResult.ErrorMessage);  
}
```

## API Response

If the operation succeeds, the `OperationalResult.Success` field will hold `True` and a summary of open guest checks will be populated in the `openChecks` parameter. The field `CheckRevenueCenterObjectNum` of `SimphonyPosApi_CheckSummary` structure will return Revenue Center Object Number as the name implies.

In case of failure, `OperationalResult.Success` field will hold `False`, while `OperationalResult.ErrorCode` holds error code and `OperationalResult.ErrorMessage` holds the reason for failure.

## Get Open Guest Checks from a Specific RVC

Sometimes a user may want to view a summary of all open guest checks from a specific revenue center within the property. The following method can be used.

### GetOpenChecksByRVC Method Signature

```
void GetOpenChecksByRVC  
(  
    string vendorCode,  
    int EmployeeObjectNum,  
    int revenueCenterObjectNum,  
    ref SimphonyPosAPI_OpenChecks openChecks  
)
```

The following code snippet demonstrates retrieving a summary of all open guest checks from RVC #3016 that are created by a specific employee whose Employee Object Number is 90001. The caller has to pass 0 (zero) to the Employee Object Number parameter if he or she wishes to fetch all open checks irrespective of the owner who created the check.

```
SimphonyPosAPIWebSoapClient mTSApi = new SimphonyPosAPIWebSoapClient();
string vendorCode = "lzsroioq"; // This can be empty from 2.7MR3 onwards
int employeeObjectNum = 90001; // Pass 0 to fetch all open checks irrespective of Check
Owner
int revenueCenterObjectNum = 3016;

public void InvokeGetOpenChecksByRVC()
{
    SimphonyPosApi_OpenChecks openChecks = new SimphonyPosApi_OpenChecks();

    mTSApi.GetOpenChecksByRVC(vendorCode, employeeObjectNum, revenueCenterObjectNum,
        ref openChecks);

    if (openChecks.OperationalResult.Success)
    {
        Console.WriteLine("Get Open Check succeeded...");

        foreach (SimphonyPosApi_CheckSummary check in openChecks.CheckSummary)
        {
            Console.WriteLine("Check Number:" + check.CheckNum);
            Console.WriteLine("Check Sequence Number:" + check.CheckSeq);
            Console.WriteLine("Check Total Due:" + check.CheckTotalDue);
            // The field CheckRevenueCenterObjectNum returns RVC ID (not Object Number)
            Console.WriteLine("RVC ID:" + check.CheckRevenueCenterObjectNum);
        }
    }
    else
    {
        Console.WriteLine("Get Open Check failed. Error Code: {0}, Error Message: {1}",
            openChecks.OperationalResult.ErrorCode,
            openChecks.OperationalResult.ErrorMessage);
    }
}
```

## API Response

If the operation succeeds, the `OperationalResult.Success` field will hold `True` and a summary of open guest checks will be populated in the `openChecks` parameter. In case of failure, the `OperationalResult.Success` field will hold `False`, while `OperationalResult.ErrorCode` holds the error code and `OperationalResult.ErrorMessage` holds the reason for failure.

## Get Summary and KDS Order Status of Open and Closed Guest Checks

If a user wants to view a summary of guest checks that satisfies few filter conditions, the following method can be used.

## GetChecks Method Signature

```
void GetChecks
(
    SymphonyPosApi_CheckRequest ppCheckFilter,
    SymphonyPosApi_CheckResponse ppChecksResponse
)
```

The following code snippet demonstrates how to retrieve both open and closed guest checks that were created in the last 5 days by an employee whose Employee Object Number is 90001.

```
SimphonyPosAPIWebSoapClient mTSApi = new SimphonyPosAPIWebSoapClient();

public void InvokeGetChecks()
{
    const int NUMBER_OF_DAYS = 5;

    // Construct request object
    SymphonyPosApi_CheckRequest request = new SymphonyPosApi_CheckRequest();
    request.LookUpStartDate = DateTime.Today.AddDays(-1 * NUMBER_OF_DAYS);
    request.IncludeClosedCheck = true; // get closed checks too
    request.EmployeeObjectNum = 90001;

    // Construct response object
    SymphonyPosApi_CheckResponse response = new SymphonyPosApi_CheckResponse();

    // Call web service method
    mTSApi.GetChecks(request, ref response);
    if (response.OperationalResult.Success)
    {
        foreach (SimphonyPosApi_CheckSummaryEx check in response.Checks)
        {
            Console.WriteLine("Check Number:" + check.CheckNum);
            Console.WriteLine("Check Sequence Number:" + check.CheckSeq);

            Console.WriteLine("KDS Order Status Code:" + check.LastKnownKdsOrderStatus);
        }
    }
    else
    {
        Console.WriteLine("GetChecks failed. Error Code: {0}, Error Message: {1}",
            response.OperationalResult.ErrorCode,
            response.OperationalResult.ErrorMessage);
    }
}
```

## API Response

If the operation succeeds, the `OperationalResult.Success` field will hold `True` and a summary of filtered guest checks will be populated in the `response.Checks` parameter. In case of failure, `OperationalResult.Success` field will hold `False`, while `OperationalResult.ErrorCode` holds the error code and `OperationalResult.ErrorMessage` holds the reason for failure.

## Get Check Detail

If a user wants to view complete details of a guest check or most recent status of the order, the following method can be used.

### GetCheckDetail Method Signature

```
void GetCheckDetail
(
    SymphonyPosApi_CheckDetailRequest ppCheckDetailFilter,
    SymphonyPosApi_CheckDetailResponse ppChecksDetailResponse
)
```

The following code snippet demonstrates retrieving complete details of a guest check in XML format and printing the most recent status of the order.

```
SimphonyPosAPIWebSoapClient mTSApi = new SimphonyPosAPIWebSoapClient();

public void InvokeGetCheckDetail()
{
    SymphonyPosApi_CheckDetailRequest request = new SymphonyPosApi_CheckDetailRequest();
    request.CheckNumber = 104;
    request.CheckSeqNumber = 123456789;

    SymphonyPosApi_CheckDetailResponse response = new SymphonyPosApi_CheckDetailResponse();

    mTSApi.GetCheckDetail(request, ref response);
    if (response.OperationalResult.Success)
    {
        Console.WriteLine("Check XML:" + response.CheckDetail);

        XmlDocument checkXML = new XmlDocument();
        checkXML.LoadXml(response.CheckDetail);

        XmlNode checkNumber = checkXML.SelectSingleNode("CheckNumber");
        Console.WriteLine("Check Number - " + checkNumber.InnerText);

        XmlNode dueAmount = checkXML.SelectSingleNode("Due");
        Console.WriteLine("Check Due - " + dueAmount.InnerText);

        // Read the most recent status of the order from extensibility data
        string extensibilityAppName = "OIS"; // this name is configured in EMC
        XmlNodeList xnl = checkXML.SelectNodes(string.Format(
            "//extensibility_data[ExtensibilityAppName = \"{0}\"]", extensibilityAppName));

        if (xnl != null && xnl.Count > 0)
        {
            string mostRecentOrderStatus = string.Empty;
            DateTime mostRecentTimeStamp = DateTime.MinValue;

            foreach (XmlNode node in xnl)
            {
                XmlNode stringData = node.SelectSingleNode("StringData");
                // Read the value of 'Timestamp' property from stringData
                DateTime timeStamp = GetOrderTimeStamp(stringData.InnerText);
            }
        }
    }
}
```

```

        if (timeStamp > mostRecentTimeStamp)
        {
            mostRecentOrderStatus = node.SelectSingleNode("DisplayName").InnerText;
            mostRecentTimeStamp = timeStamp;
        }
    }

    Console.WriteLine("Most Recent Order Status - " + mostRecentOrderStatus);
    Console.WriteLine("Timestamp - " + mostRecentTimeStamp);
}
else
{
    Console.WriteLine("GetCheckDetail failed. Error Code: {0}, Error Message: {1}",
        response.OperationalResult.ErrorCode,
        response.OperationalResult.ErrorMessage);
}
}
}

```

## API Response

If the operation succeeds, the `OperationalResult.Success` field will hold `True` and the check XML will be populated in the `response.CheckDetail` parameter. In case of failure, `OperationalResult.Success` field will hold `False`, while `OperationalResult.ErrorCode` holds error code and `OperationalResult.ErrorMessage` holds reason for failure.

## Get Printed Texts of a Guest Check

The print lines of an open guest check can be retrieved by specifying the Check Number and a few other required details. This is often required where an external printer is used to print an open guest check. The method `GetPrintedCheck` can be used for this purpose. This method will retrieve the print lines in the output parameter without actually printing the guest check on a printer. This method works only on open guest checks.

### GetPrintedCheck Method Signature

```

void GetPrintedCheck
(
    string                vendorCode,
    int                   CheckSeq,
    int                   EmplObjectnum,
    int                   TmedObjectNum,
    ref SymphonyPosApi_CheckPrintResponse ppCheckPrintLines
)

```

The following code snippet demonstrates retrieving print lines of a guest check.

```

SymphonyPosAPIWebSoapClient mTSApi = new SymphonyPosAPIWebSoapClient();
string vendorCode = "lzsroioq"; // This can be empty from 2.7MR3 onwards
int employeeObjectNum = 90001; // for authentication

public void InvokeGetPrintedCheck()
{

```

```
int checkNumber = 1052; // Check Number for which print lines required
int tenderMediaObjNum = 49; // Tender Media Object Number of Service Total
SymphonyPosApi_CheckPrintResponse checkPrintLines =
    new SymphonyPosApi_CheckPrintResponse();

mTSApi.GetPrintedCheck(vendorCode, checkNumber, employeeObjectNum,
    tenderMediaObjNum, ref checkPrintLines);

if (checkPrintLines.OperationalResult.Success)
{
    Console.WriteLine("Get Printed Check succeeded...");

    Console.WriteLine("Printed check lines:");
    foreach (string printLine in checkPrintLines.CheckPrintLines)
    {
        Console.WriteLine(printLine);
    }
}
else
{
    Console.WriteLine("Get Printed Check failed. Error Code: {0},
        Error Message: {1}", checkPrintLines.OperationalResult.ErrorCode,
        checkPrintLines.OperationalResult.ErrorMessage);
}
}
```

## API Response

If the operation succeeds, the `OperationalResult.Success` field will hold `True`, and print lines of the guest check will populate in the `checkPrintLines` parameter. In case of failure, the `OperationalResult.Success` field will hold `False`, while the `OperationalResult.ErrorCode` holds error code and `OperationalResult.ErrorMessage` holds reason for failure.

## Get Configured Information (Method 1 - GetConfigurationInfo)

Some of the data configured in the EMC can be retrieved from the POS database via TS API.

The client application can retrieve the configuration data such as menu item definition, family group, interfaces, menu item master, menu item price, major group, revenue center parameter, tender media, currency, employees, menu item class, serving periods, service charge, discounts, dining tables, order types, revenue centers, menu levels, language information, application version, menu levels, menu item SLU, main menu levels, sub menu levels, event types, event sub types and event definition from the Symphony POS database using the `GetConfigurationInfo` method. This method returns all records of the specified configuration data type. The integrator can use another version of this method (explained in the next section) named `GetConfigurationInfoEx` to retrieve records batch by batch when the volume of configuration data is large.



## GetConfigurationInfo Method Signature

```
void GetConfigurationInfo
(
    String                vendorCode,
    Int                   employeeObjectNum,
    ARRAY(int)            configurationInfoType,
    Int                   revenueCenter,
    ref SymphonyPosAPI_ConfigInfoResponse configInfoResponse
)
```

The following code snippet demonstrates retrieving configured data for menu item definition, menu item price, tender media, currency, and service charge.

```

SymphonyPosAPIWebSoapClient mTSApi = new SymphonyPosAPIWebSoapClient();
string vendorCode = "lzsroioq"; // This can be empty from 2.7MR3 onwards
int employeeObjectNum = 90001; // for authentication
int revenueCenterObjectNum = 3016;

public void InvokeGetConfigurationInfo()
{
    int[] configurationInfoType = new int[] {
        (int)EConfigurationInfoType.MENUITEMDEFINITIONS,
        (int)EConfigurationInfoType.MENUITEMPRICE,
        (int)EConfigurationInfoType.TENDERMEDIA,
        (int)EConfigurationInfoType.CURRENCY,
        (int)EConfigurationInfoType.SERVICECHARGE };

    SymphonyPosApi_ConfigInfoResponse configInfoResponse = new
        SymphonyPosApi_ConfigInfoResponse();

    mTSApi.GetConfigurationInfo(vendorCode, employeeObjectNum, configurationInfoType,
        revenueCenterObjectNum, ref configInfoResponse);

    if (configInfoResponse.OperationalResult.Success)
    {
        Console.WriteLine("Get Configuration Info succeeded...");
        Console.WriteLine("Menu item definitions: " +
            configInfoResponse.MenuItemDefinitions);
        Console.WriteLine("Menu item price: " + configInfoResponse.MenuItemPrice);
        Console.WriteLine("Tender media: " + configInfoResponse.TenderMedia);
        Console.WriteLine("Currency: " + configInfoResponse.Currency);
        Console.WriteLine("Service Charge: " + configInfoResponse.ServiceCharge);
    }
    else
    {
        Console.WriteLine("Get Configuration Info failed. Error Code: {0},
            Error Message: {1}", configInfoResponse.OperationalResult.ErrorCode,
            configInfoResponse.OperationalResult.ErrorMessage);
    }
}

```

## API Response

If the operation succeeds, the `OperationalResult.Success` field will hold `True`, and configured data for queried type can be found in the appropriate fields of the `configInfoResponse` object. In case of failure, the `OperationalResult.Success` will hold `False`, while `OperationalResult.ErrorCode` holds error code and `OperationalResult.ErrorMessage` holds reason for failure.

## Get Configured Information (Method 2 - GetConfigurationInfoEx)

`GetConfigurationInfoEx` is a new version of the `GetConfigurationInfo` method to retrieve configuration data batch by batch or in full. This method is useful when the POS database has a large volume of configuration data for one or more types, such as Menu Item Definition, Menu Item Master, and so on. This method is designed to return all records for a configuration data type when no ranges are specified in the input parameters. In other words, this method will behave exactly like the old method (`GetConfigurationInfo`) when no ranges (start index and maximum records count) are specified.

### GetConfigurationInfoEx Method Signature

```
void GetConfigurationInfoEx
(
    SymphonyPosApi_ConfigInfoRequest      configInfoRequest,
    ref SymphonyPosAPI_ConfigInfoResponse configInfoResponse
)
```

The following code snippet demonstrates retrieving configured data for the menu item definition and menu item price.

```

SymphonyPosAPIWebSoapClient mTSApi = new SymphonyPosAPIWebSoapClient();

string vendorCode = "lzsroioq"; // This can be empty from 2.7MR3 onwards
int employeeObjectNum = 90001; // for authentication
int revenueCenterObjectNum = 3016;

private const int MAX_RECORD_COUNT= 5000; // Indicates the maximum number of records to be retrieved

private static string[] CONFIG_DATA_NODE_NAMES = new string[] { "DbMenuItemMaster",
    "DbMenuItemPrice" };

public void InvokeGetConfigurationInfo()
{
    SymphonyPosApi_ConfigInfoRequest request = new SymphonyPosApi_ConfigInfoRequest();
    request.EmployeeObjectNumber = employeeObjectNum;
    request.RVCObjectNumber = revenueCenterObjectNum;

    List<SymphonyPosApi_ConfigInfo> configTypeList = new List<SymphonyPosApi_ConfigInfo>();

    // Menu Item Master

```

```
SimphonyPosApi_ConfigInfo miMasterType = new SimphonyPosApi_ConfigInfo();
miMasterType.ConfigurationInfoTypeID = EConfigurationInfoType.MENUITEMMASTERS;
miMasterType.MaxRecordCount = MAX_RECORD_COUNT;
miMasterType.StartIndex = 1; // 1 is just an initial value; this will be incremented each
                             // time GetConfigurationInfoEx method is called.
configTypeList.Add(miMasterType);

// Menu Item Price
SimphonyPosApi_ConfigInfo miPriceType = new SimphonyPosApi_ConfigInfo();
miPriceType.ConfigurationInfoTypeID = EConfigurationInfoType.MENUITEMPRICE;
miPriceType.MaxRecordCount = MAX_RECORD_COUNT;
miPriceType.StartIndex = 1; // 1 is just an initial value; this will be incremented each
                             // time GetConfigurationInfoEx method is called.
configTypeList.Add(miPriceType);

request.ConfigurationInfo = configTypeList.ToArray();
int miMasterRecordsCount = 0;
int miPricecRecordsCount = 0;

while (true) // call GetConfigurationInfoEx method in a loop until it return all
             // the records of requested configuration data type
{
    SimphonyPosApi_ConfigInfoResponse response = new SimphonyPosApi_ConfigInfoResponse();

    // Call the TS method and check the response
    mTSApi.GetConfigurationInfoEx(request, ref response);
    if (response.OperationalResult.Success)
    {
        bool bFoundData = false;

        // Menu Item Master
        if (!string.IsNullOrEmpty(response.MenuItemMasters))
        {
            // add it to the output list
            XmlDocument tempXmlDoc = new XmlDocument();
            tempXmlDoc.LoadXml(response.MenuItemMasters);
            XmlNodeList miMasterList =
                tempXmlDoc.GetElementsByTagName(CONFIG_DATA_NODE_NAMES[0]);

            if (miMasterList != null && miMasterList.Count > 0)
            {
                bFoundData = true;
                miMasterRecordsCount += miMasterList.Count;

                // TODO: Save these records to a file or memory for consolidation
            }
        }

        // Menu Item Price
        if (!string.IsNullOrEmpty(response.MenuItemPrice))
        {
            // add it to the output list
            XmlDocument tempXmlDoc = new XmlDocument();
            tempXmlDoc.LoadXml(response.MenuItemPrice);
```

```
XmlNodeList miPriceList =
    tempXmlDoc.GetElementsByTagName(CONFIG_DATA_NODE_NAMES[1]);

if (miPriceList != null && miPriceList.Count > 0)
{
    bFoundData = true;
    miPricecRecordsCount += miPriceList.Count;

    // TODO: Save these records to a file or memory for consolidation
}
}

if (bFoundData == false)
{
    break; // all records are already returned by TS, so come out of the loop now.
}

// increment the start index so that GetConfigurationInfoEx can be
// called again to get next set of records.
IncrementStartIndex(configTypeList.ToArray(), MAX_RECORD_COUNT);
}
}

System.Console.WriteLine("Operation completed successfully.");
System.Console.WriteLine("MI Master count - " + miMasterRecordsCount);
System.Console.WriteLine("MI Price count - " + miPricecRecordsCount);

System.Console.ReadLine();
}

private static void IncrementStartIndex(SymphonyPosApi_ConfigInfo[] configTypeList,
    int maxRecordCount)
{
    foreach(SymphonyPosApi_ConfigInfo configInfo in configTypeList)
    {
        configInfo.StartIndex += maxRecordCount;
    }
}
```

## API Response

If the operation succeeds, the `OperationalResult.Success` field will hold `True`, and configured data for queried type can be found in the appropriate fields of the `configInfoResponse` object. In case of failure, `OperationalResult.Success` will hold `False`, while `OperationalResult.ErrorCode` holds error code and `OperationalResult.ErrorMessage` holds reason for failure.

# 8

## Simphony Platform Requirements

### Simphony Software Version

This release of the Simphony POS API requires Simphony version 2.7 or later.

### Offline Transaction Support

The Simphony Transaction Services can never be in an offline state. It does not have an offline feature. As it is hosted by either a ServiceHost or IIS, the lazy playback mechanism posts the checks to the Check and Posting Server (CAPS). If CAPS is offline, checks will not be posted to the CAPS machine unless it is restarted.

### Printing Services

The API supports printing to remote and local order devices. When a check is opened through the TS API and posted to the Simphony database, the menu items will print on the remote and local devices based on the default workstation definition and the assigned Menu Item Print Class. There should be no difference between how an API check prints versus a check opened directly by the user on the POS devices. Local guest check printing is not supported at this time through the API. The Print Controller service must be running for printing to work.

### Calling Conventions

There are two types of parameters passed to the API: ref and non-ref parameters. All parameters are mandatory. However, if you do not wish to use one of the parameters, simply create the structure and set all of its members to zero.

For example, if a check does not contain a Subtotal discount, you can pass the address of this structure to the API; everything will be zero. To add a discount, fill in the appropriate members of the discount object.

# 9 Demo Client for Transaction Services API

The Demo client is a Windows application used to demonstrate or test the features of the TS API. This application is distributed with Symphony install media. This application builds data for input parameters based on values provided by the user, and sends a request to TS API and displays the response in the UI.

## Application Path

The demo client application is located in the following folder of install media:

`<InstallMediaFolder>\Install\Symphony2\Tools\PosAPIDemoClient`

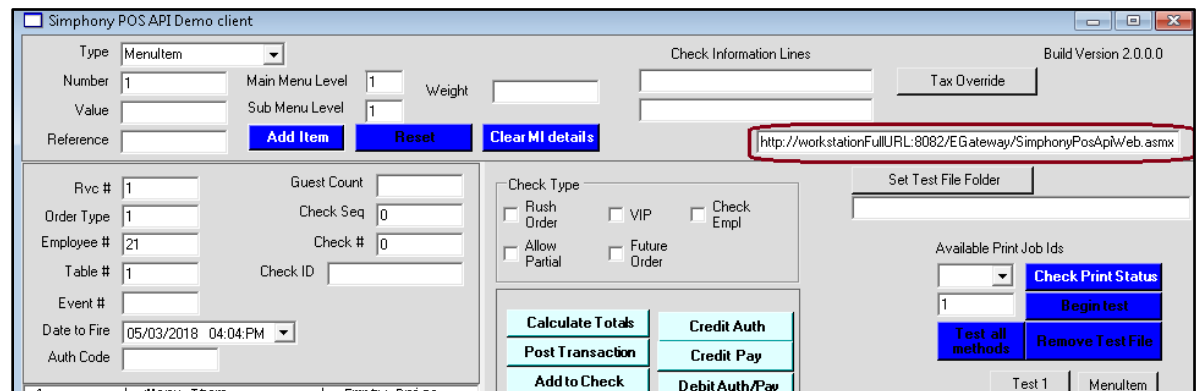
## Prerequisites

A workstation is configured using the EMC and is runs Service Host application to host the Transaction Services web service. Navigate to the URL of the TS web service using a web browser to see if the TS web service is running.

## Initial Setup

To configure and run the demo client application:

1. Copy the *PosAPIDemoClient* folder from install media to a local folder.
2. Launch **POSAPI\_WebClient.exe**.
3. Enter the full address of the workstation hosting the TS service in the text box (highlighted in the following image).



## Demonstration

### Calculate Totals of a Transaction

The Demo client has a **Calculate Totals** button to invoke the *CalculateTransactionTotals* method of TS API.

To pass input data through the UI and invoke the method:

1. Select the **MenuItem** option from the **Type** drop-down list.
2. Enter the menu item's Object Number **110003** in the **Number** field, and then click the **Add Item** button.  
You can obtain the Menu Item Object Number from the EMC, at the **Property** level, **Configuration** tab, **Menu Items**, **Menu Item Maintenance** module.
3. Enter 3016 in the **RVC #** textbox.  
You can obtain the RVC number from the EMC, **Property** level, **Setup** tab, **Property Configuration**, **RVC Configuration** module.
4. Enter 90001 in the **Employee #** textbox.  
You can obtain the Employee Number from the EMC, **Property** level, **Configuration** tab, **Personnel**, **Employee Maintenance** module.
5. Enter a valid vendor code in the **License Activation Code** field. This can be left blank for Symphony version 2.7 MR3 or later.
6. Click the **Calculate Totals** button to send the request to the TS API.

The status of the operation will appear in the lower area of the UI, while results appear in the block highlighted in green as shown in the following image.

The screenshot shows the 'Symphony POS API Demo client' window. The 'Type' dropdown is set to 'MenuItem'. The 'Number' field contains '110003'. The 'Rvc #' field contains '3016'. The 'Employee #' field contains '90001'. The 'License Activation Code' field contains 'jdyrkjgc'. The 'Calculate Totals' button is highlighted in green. Below the button is a table showing the results of the calculation:

8.7900	Subtotal
0.68	Tax Total
0	Other Total
0	Auto Svc Charge
9.4700	Total Due

At the bottom of the window, the status bar shows the following log messages:

```
11:30:38.780: Finished CalculateTransactionTotals()
11:29:56.918: Calling CalculateTransactionTotals() ...
```

## Create a Guest Check

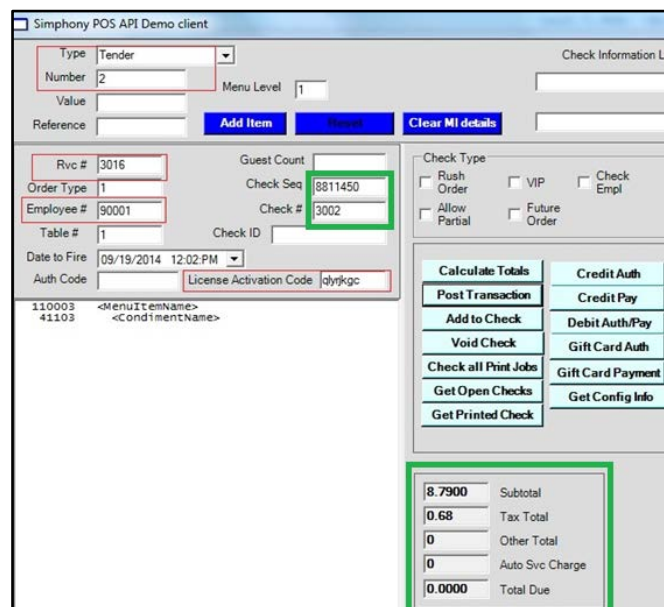
Use the **Post Transaction** button to send a request to create a new guest check in the Symphony database.

1. Select **MenuItem** from the **Type** drop-down list.
2. Enter the menu item's Object Number **110003** in the **Number** field, and then click the **Add Item** button.
3. Select **RequiredCondiment** from c.
4. Enter the Object Number **41103** in the **Number** textbox, and then click the **Add Item** button.
5. For payment, select **Tender** from the **Type** drop-down list.
  - a. For a cash payment, enter the Tender Media Object Number of Cash (for example, 2) in the **Number** field. Then enter the amount (for example, 10) in the **Value** textbox.
  - b. For a debit or credit payment, enter the Tender Media Object Number of Credit/Debit, and then click the **Credit Auth** button to provide payment card details in the popup.

 **NOTE:**

Transaction Services only supports the **MCreditDebit Payment** driver for credit/debit card payments.

6. To add tender details, click the **Add Item** button.
7. If you are using versions earlier than Symphony 2.7 MR3, enter values in the **RVC #**, **Employee #**, and **License Activation Code** fields.
8. To send a request to the TS API, click the **Post Transaction** button. The result is populated in the fields highlighted in green as shown in the following image.



The screenshot shows the 'Symphony POS API Demo client' window. The form is divided into several sections:

- Top Section:** Includes a 'Type' dropdown menu set to 'Tender', a 'Number' field with '2', a 'Value' field, and a 'Menu Level' field with '1'. There are buttons for 'Add Item', 'Reset', and 'Clear MI details'.
- Middle Section:** Contains fields for 'Rvc #' (3016), 'Guest Count', 'Order Type' (1), 'Check Seq' (8811450), 'Employee #' (90001), 'Check #' (3002), 'Table #' (1), 'Check ID', 'Date to Fire' (09/19/2014 12:02 PM), 'Auth Code', and 'License Activation Code' (jdykkgc).
- Right Section:** Features a 'Check Type' section with checkboxes for 'Rush Order', 'VIP', 'Check Empl', 'Allow Partial', and 'Future Order'. Below this are two columns of buttons: 'Calculate Totals' (Post Transaction, Add to Check, Void Check, Check all Print Jobs, Get Open Checks, Get Printed Check) and 'Credit Auth' (Credit Pay, Debit Auth/Pay, Gift Card Auth, Gift Card Payment, Get Config Info).
- Bottom Section:** A summary table with a green border showing:

8.7900	Subtotal
0.68	Tax Total
0	Other Total
0	Auto Svc Charge
0.0000	Total Due



## Add an Item to an Open Guest Check

Use the **Add to Check** button to add one or more items to an existing guest check.

This example creates a guest check using **Post Transaction** first and then adds one menu item to the check using the **Add To Check** button.

1. Select the **MenuItem** option from the **Type** drop-down list.
2. Enter the menu item's Object Number **110003** in the **Number** field, and then click the **Add Item** button.
3. Select **RequiredCondiment** from the **Type** drop-down list.
4. Enter the Object Number **44502** in the **Number** field, and click then the **Add Item** button.
5. Select **Tender** from the **Type** drop-down list.
6. Enter the Tender Media Object Number of **Service Total** in the **Number** field, and then click the **Add Item** button.
7. If you are using versions earlier than Symphony 2.7 MR3, enter values in the **RVC #**, **Employee #**, and **License Activation Code** fields.
8. Click the **Post Transaction** button to create the guest check. As the tender type is **Service Total**, the check that is created will be in the Open state. One or more items can be added to the open check by following further steps below.

The screenshot shows the Symphony POS API Demo client interface. The window title is "Symphony POS API Demo client". The form is divided into several sections:

- Top Section:** Includes a "Type" dropdown menu set to "Tender", a "Number" field with "49", a "Menu Level" field with "1", and a "Reference" field. There are buttons for "Add Item", "Reset", and "Clear MI details".
- Check Information Section:** Includes fields for "Rvc #", "Guest Count", "Order Type", "Check Seq", "Employee #", "Check #", "Table #", and "Check ID". There are also checkboxes for "Check Type" options: "Rush Order", "VIP", "Check Empl", "Allow Partial", and "Future Order".
- Auth and License Section:** Includes fields for "Date to Fire" (09/19/2014 02:02:PM), "Auth Code", and "License Activation Code" (qlyrkgc).
- Item List Section:** A table with two rows:
 

110003	<MenuItemName>
44502	<CondimentName>
- Buttons Section:** A grid of buttons including "Calculate Totals", "Post Transaction" (highlighted with a red box), "Add to Check", "Void Check", "Check all Print Jobs", "Get Open Checks", "Get Printed Check", "Credit Auth", "Credit Pay", "Debit Auth/Pay", "Gift Card Auth", "Gift Card Payment", and "Get Config Info".
- Totals Section:** A summary of financial values:
 

8.7900	Subtotal
0.68	Tax Total
0	Other Total
0	Auto Svc Charge
9.4700	Total Due

9. Click the **Clear MI details** button to clear current details.

10. Select **Menuitem** from the **Type** drop-down list, and then add a menu item to the existing guest check.
11. Enter the menu item's Object Number **110004** in the **Number** field and then click the **Add Item** button.
12. Select **RequiredCondiment** from the **Type** drop-down list.
13. Enter the Object Number **41103** in the **Number** field, and then click the **Add Item** button.
14. Ensure that the value of Check Sequence Number and Check Number of original check still appears in the **Check Seq** and the **Check #** fields.
15. Click the **Add to Check** button.

Simphony POS API Demo client

Type: Tender  
Number: 2  
Menu Level: 1  
Value:   
Reference:   
Add Item   
Reset   
Clear MI details

Rvc #: 3016  
Guest Count:   
Order Type: 1  
Check Seq: 676895748  
Employee #: 90001  
Check #: 3007  
Table #: 1  
Check ID:   
Date to Fire: 09/19/2014 02:04:PM  
Auth Code:   
License Activation Code: qlyrjkgc

110004 <MenuItemName>  
41103 <CondimentName>

Check Type  
 Rush Order  VIP  Check Empl  
 Allow Partial  Future Order

Calculate Totals   
Post Transaction   
Add to Check   
Void Check   
Check all Print Jobs   
Get Open Checks   
Get Printed Check

Credit Auth   
Credit Pay   
Debit Auth/Pay   
Gift Card Auth   
Gift Card Payment   
Get Config Info

18.3800	Subtotal
1.42	Tax Total
0	Other Total
0	Auto Svc Charge
0.0000	Total Due

## Combo Meal Ordering

The following steps describe how to add a combo meal to the check. Ensure that a combo meal is already configured in the EMC so that it can be added on the POS API.

1. Select **ComboMeal** from the **Type** drop-down list, enter the combo meal Object Number in the **Number** field, enter the Object Number in the **Combo Menu Item Number** field, and then click the **Add Item** button.

Simphony POS API Demo client

Type: ComboMeal  
Number: 1  
Combo Menu Item Number: 110006  
Menu Level: 1  
Value:   
Reference:   
Add Item   
Reset   
Clear MI details

Rvc #: 3016  
Guest Count:   
Order Type: 1  
Check Seq: 0  
Employee #: 90001  
Check #: 0  
Table #: 1  
Check ID:   
Date to Fire: 09/19/2014 03:45:PM  
Auth Code:   
License Activation Code: qlyrjkgc

1 <Combo Meal>  
110006 <Combo Menu Item>

Check Type  
 Rush Order  VIP  Check Empl  
 Allow Partial  Future Order

Calculate Totals   
Post Transaction   
Add to Check   
Void Check   
Check all Print Jobs   
Get Open Checks   
Get Printed Check

Credit Auth   
Credit Pay   
Debit Auth/Pay   
Gift Card Auth   
Gift Card Payment   
Get Config Info

18.3800	Subtotal
1.42	Tax Total
0	Other Total
0	Auto Svc Charge
0.0000	Total Due

2. To add a main item, select **ComboMain** from the **Type** drop-down list.

3. Enter the combo meal's Object Number in the **Number** field, and then click the **Add Item** button.
4. To add side items, select **ComboSide** from the **Type** drop-down list.
5. Enter the side item's Object Number in the **Number** field, and click the **Add Item** button.
6. Click the **Calculate Totals** button to calculate the price of the combo meal or add a tender, and then click the **Post Transaction** button to create a guest check.

The screenshot shows the 'Symphony POS API Demo client' window. It contains several input fields and buttons. The 'Type' dropdown is set to 'Tender'. The 'Number' field contains '2'. The 'Menu Level' field contains '1'. The 'Rvc #' field contains '3016'. The 'Order Type' field contains '1'. The 'Employee #' field contains '90001'. The 'Table #' field contains '1'. The 'Date to Fire' field shows '09/19/2014 04:14:PM'. The 'License Activation Code' field contains 'qlyrkgc'. The 'Check Seq' field contains '764618844'. The 'Check #' field contains '3011'. The 'Check ID' field is empty. The 'Check Type' section has several checkboxes: 'Rush Order', 'VIP', 'Check Empl', 'Allow Partial', and 'Future Order'. The 'Calculate Totals' button is highlighted with a red box. The 'Post Transaction' button is also highlighted with a red box. The 'Add to Check' button is highlighted with a red box. The 'Void Check' button is highlighted with a red box. The 'Check all Print Jobs' button is highlighted with a red box. The 'Get Open Checks' button is highlighted with a red box. The 'Get Printed Check' button is highlighted with a red box. The 'Credit Auth' button is highlighted with a red box. The 'Credit Pay' button is highlighted with a red box. The 'Debit Auth/Pay' button is highlighted with a red box. The 'Gift Card Auth' button is highlighted with a red box. The 'Gift Card Payment' button is highlighted with a red box. The 'Get Config Info' button is highlighted with a red box. The 'Subtotal' field contains '18.7900'. The 'Tax Total' field contains '0'. The 'Other Total' field contains '0'. The 'Auto Svc Charge' field contains '0'. The 'Total Due' field contains '18.7900'. The 'Item List' section shows the following items:

1	<Combo Meal>
110006	<Combo Menu Item>
124001	<Combo Main Item>
41101	<Combo Side Item>
2	<Tender>

## Void All Items of an Open Guest Check

Use the **Void Check** button to send a request to void all items of a guest check.

Enter the **Check Sequence Number** and the **Check Number** of the guest check in the relevant fields (highlighted in red in the following image) to void all items of the guest check. No other input is required to perform this operation.

To send a void request to TS API, click the **Void Check** button.

## Get Summary of All Open Guest Checks

Use the **Get Open Checks** button to send a request to TS API to retrieve a summary of all open guest checks from all or a specific revenue center of the property from the Simphony POS database. This button calls different method (*GetOpenChecks*, *GetOpenChecksEx*, *GetOpenChecksByRVC*) of TS API based on input given to the **Revenue Center** field of the *FormGuestCheckParams* dialog, as described below.

1. If you are using versions earlier than Simphony 2.7 MR3, enter the value for the **License Activation Code** field, and then click the **Get Open Checks** button.

The following dialog appears:

FormGuestCheckParams

Enter Params for Get Open Checks method

Enter a valid Employee Object Number. Enter 0 to view all open Checks.

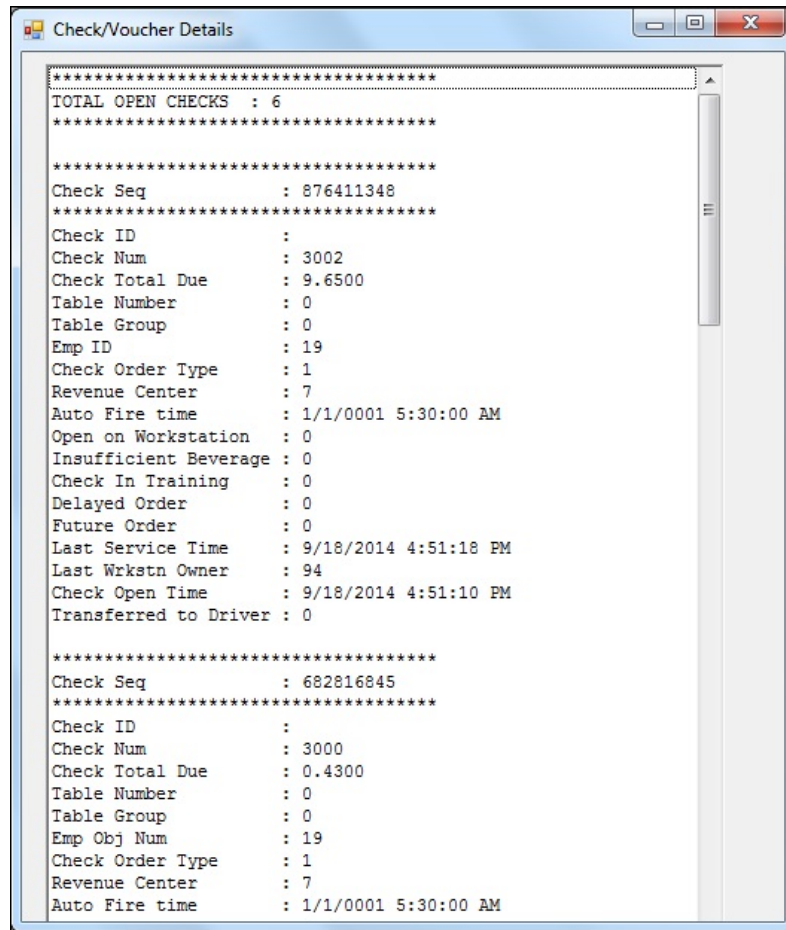
Employee number 90001

Revenue Center

OK

2. Enter the **Employee Object Number** in the **Employee number** field to filter checks based on the employee who created it (or enter **0** to get all open checks irrespective of who created them).
3. Enter the appropriate value in the **Revenue Center** field, and enable the check box:
  - a. Enter **-1** to retrieve open checks from all revenue centers of the property and display **ID** (instead of Object Number) of Revenue Center for each check in the output window.
  - b. Enter **-2** to retrieve open checks from all revenue centers of the property and display **Object Number** (instead of ID) of Revenue Center for each check in the output window.
  - c. Enter the **Object Number of any RVC** to retrieve all open checks from that specific revenue center and display **ID** (instead of Object Number) of Revenue Center for each check in the output window.

The following image shows a summary of all open checks.



## Get Summary of Open and Closed Guest Checks

Use the **Get Checks** button to send a request to TS API to retrieve a summary of all open and closed guest checks from default or a specific revenue center of the property from the Symphony POS database. This button calls the GetChecks method of TS API.

1. Click the **Get Checks** button from the main window to open the following window.

The screenshot shows a dialog box titled "Get Checks". At the top, it says "Please enter data for filters as needed:". Below this are several input fields: "Check Number(s)" with a note "Note: Separate by comma", "Check Sequence Number" (disabled), "Employee Object Number", "RVC Object Number", "Order Type ID", and "Kds Order Status ID(s)" with a note "Note: Separate by comma". There is a "Lookup Start Date" dropdown menu showing "Thursday, March 10, 2016". A checkbox "Include Closed Check" is unchecked. A "Send Request" button is located to the right of the checkbox. Below the input fields is a large text area labeled "Extensibility Detail". At the bottom of the dialog is an "Extension Detail App Name" input field.

2. Enter data for one or more filters as needed (for example, Check Number(s), Employee Object Number, RVC Object Number, Order Type ID, KDS Order Status ID(s), LookupStartDate). Select the **Include Closed Check** checkbox if closed checks need to be retrieved.

The **Check Sequence Number** field is not applicable to this operation, so it is disabled.

3. Click the **Send Request** button.



Please enter data for filters as needed:

Check Number(s)

Note: Separate by comma

Check Sequence Number

Employee Object Number

RVC Object Number

Order Type ID

Kds Order Status ID(s)

Note: Separate by comma

Lookup Start Date

Include Closed Check

Extensibility Detail

```
==== Success | Success [LastResponseTransaction.xml]
Chk# 125 Seq# 765124172 closed KdsOrderState unknown
Chk# 48 Seq# 287824895 open KdsOrderState unknown
Chk# 49 Seq# 687901674 open KdsOrderState unknown
Chk# 47 Seq# 292853520 open KdsOrderState unknown
```

Extension Detail App Name

4. The summary of returned checks appears in the area highlighted in green in the image above.
5. To view the complete details of returned checks, open the LastResponseTransaction.xml file that is created by this demo client application in the folder where this application is installed.

## Get Check Detail

Use the **Get Check Detail** button to send a request to TS API to retrieve the full detail of any open or closed guest check. This button calls the `GetCheckDetail` method of TS API.

1. Click the **Get Check Detail** button from the main window to open the following window.

Get Check Detail

Please enter data for filters as needed:

Check Number

Check Sequence Number

Employee Object Number

RVC Object Number

Order Type ID

Kds Order Status ID(s)

Note: Separate by comma

Lookup Start Date

Include Closed Check

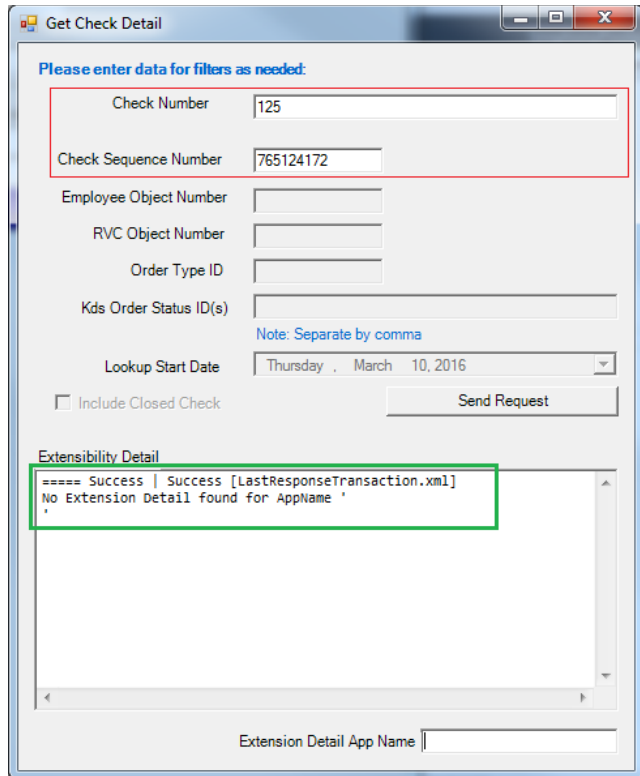
Send Request

Extensibility Detail

Extension Detail App Name

2. Enter the **Check Number** and **Check Sequence Number** fields. Other fields, such as Employee Object Number, RVC Object Number, and so on are not applicable to this operation.
3. Click the **Send Request** button.

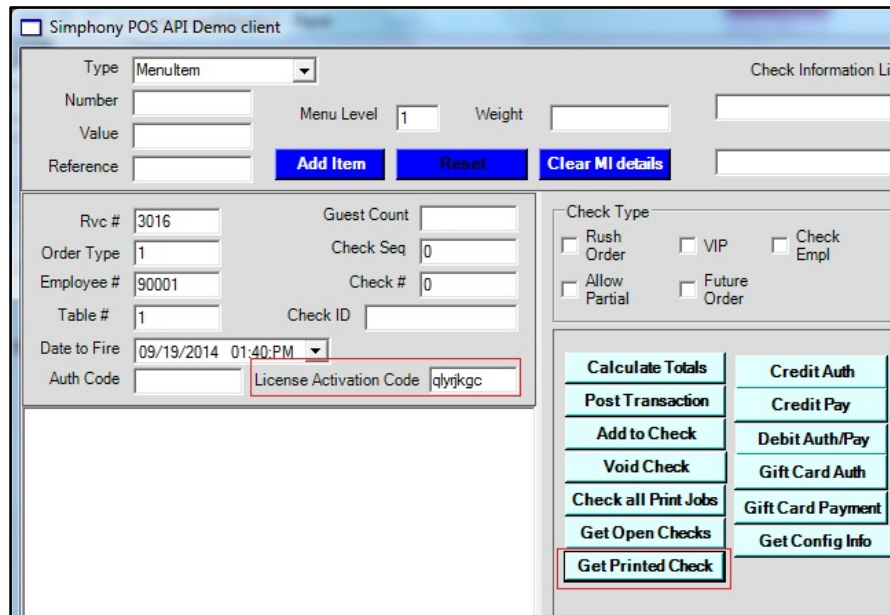
The area highlighted in green appears if the check has extensibility data. To see the entire check detail, open the `LastResponseTransaction.xml` file that is created by this demo client application in the folder where this application is installed.



## Get Printed Texts of a Guest Check

Use the **Get Printed Check** button to open posted checks by supplying the check number and the tender number.

1. If you are using versions earlier than Symphony 2.7 MR3, enter the **License Activation Code**.
2. Click the **Get Printed Check** button.



- 3. Enter the **Employee number**, **Check Number** (or Check Sequence), and **Tender Media Object Number**, and then click **OK**.

FormGuestCheckParams

Enter Params for Get Printed Check method

Please enter valid values for the following :

Employee number	90001
Check Number	3002
Tender/Media Obj Num	2

OK

Check/Voucher Details

Blue Ridge Tavern and Trading Post  
Asheville Regional Airport

90001 MICROS	1
CHK 3002	GST 1
1 BACON EGG CHZ BISCUIT	7.79
Food	7.79
Tax	0.60
Total Due	8.39

## Get Configured Information

1. Enter the RVC Object Number value in the **Rvc #** field.
2. If you are using versions earlier than Symphony 2.7 MR3, enter the **License Activation Code**.
3. Click the **Get Config Info** button.

The screenshot shows the 'Symphony POS API Demo client' window. It contains several input fields and buttons. The 'Rvc #' field is highlighted with a red box and contains the value '3016'. The 'Employee #' field is also highlighted with a red box and contains the value '90001'. The 'License Activation Code' field is highlighted with a red box and contains the value 'qlrjkgc'. Other fields include 'Type' (Menuitem), 'Number' (1), 'Menu Level' (1), 'Weight', 'Order Type' (1), 'Check Seq' (0), 'Table #' (1), 'Check ID', 'Date to Fire' (09/19/2014 01:24:PM), and 'Auth Code'. There are buttons for 'Add Item', 'Reset', 'Clear MI details', 'Calculate Totals', 'Post Transaction', 'Add to Check', 'Void Check', 'Check all Print Jobs', 'Get Open Checks', 'Get Printed Check', 'Credit Auth', 'Credit Pay', 'Debit Auth/Pay', 'Gift Card Auth', 'Gift Card Payment', and 'Get Config Info'. The 'Get Config Info' button is highlighted with a red box.

4. Enter the Employee Object Number in the **Employee number** field, and then enter configuration data type IDs in the **Configuration Number** textbox. The configuration numbers should be separated by a comma when data for more than one configuration data type needs to be fetched.
5. If a specific range of records are required for given configuration data type, select the **Get records by index?** checkbox. Specify the start index and maximum number of records required in the **Start Index** and **Max Records Count** textboxes. When the checkbox **Get records by index?** is not selected, the TS API returns all records for the specified configuration data types.
6. Click the **OK** button to send a request to the TS API.

FormGuestCheckParams

Enter Params for Get Config Info method  
Enter number separated by comma in configuration box

**Enter a valid Employee Object Number.**

Employee number

Configuration Number

Get records by index?

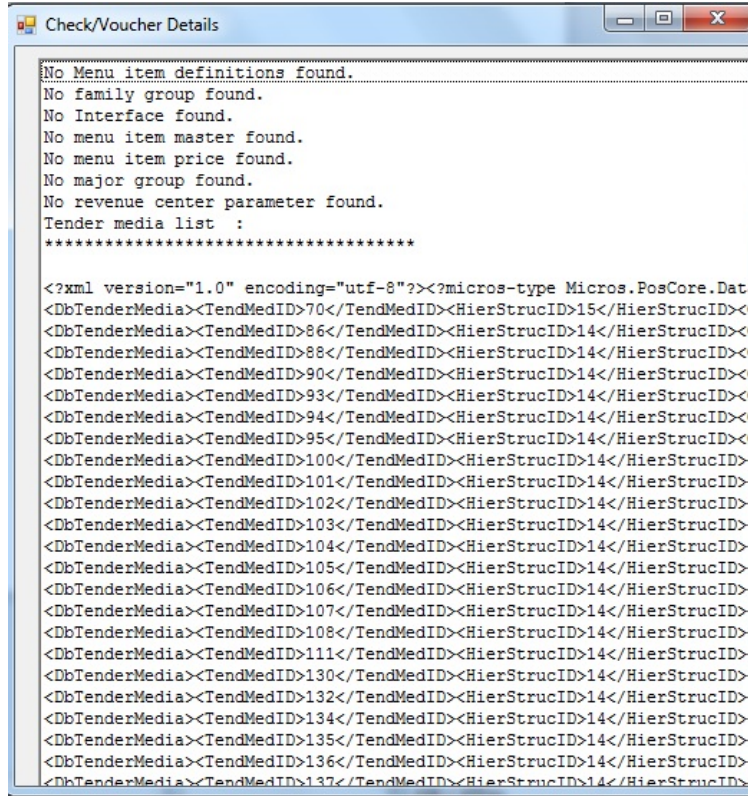
Start Index

Max Records Count

**OK**

- 1 - MENUITEMDEFINITIONS
- 2 - MENUITEMPRICE
- 3 - MENUITEMCLASS
- 4 - SERVICECHARGE
- 5 - DISCOUNTDEFINITIONS
- 6 - TENDERMEDIA
- 7 - ORDERTYPE
- 8 - FAMILYGROUP
- 9 - MAJORGROUP
- 10 - REVENUECENTERPARAMETER
- 11 - REVENUECENTERS
- 12 - INTERFACES
- 13 - MENUITEMMASTERS
- 14 - SERVINGPERIODS

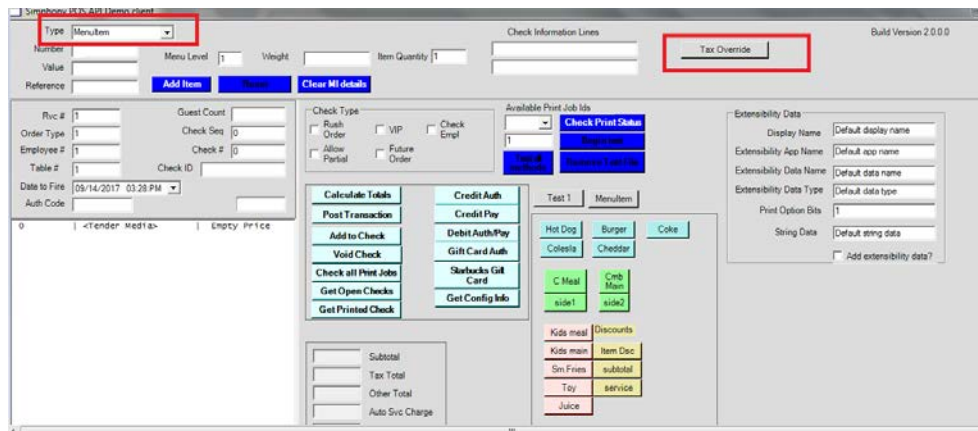
The following image shows the configuration data returned by TS API.



## Tax Override

Use the **Tax Override** button to override a menu item’s existing tax rate. These items include menu items, combo meals, combo sides, combo main, required condiment, and allowed condiments. It is assumed that there are one or more tax rates preconfigured in EMC at the enterprise or property level.

1. After adding a menu item to the check, click the **Tax Override** button.



2. Place a checkmark next to the tax rate to apply to this item.





Finally, the user should be able to tender the check and the check printout should display the correct taxed amount.

CHK 8516	GST 0
92102937 Temporary	
TRN 9/1	OCT23'2017 6:47AM
	Pier 55
	Pier 55
-----	
92102937 Temporary	
1 TL PIKE PLACE	1.95
Check	30.00
Cash	-27.66
Subtotal	1.95
Tax	0.39
Total	2.34
Change Due	27.66
-----	

## Perform Sanity Check

Use the **SanityCheck** button to perform a sanity check validation by providing a Sanity Code for the 'Extension Plugin' currently initiated with TS.

1. Click on the **Fiscal** tab to display the Fiscal API section containing the user interface for performing Sanity Check.

The screenshot shows the 'Symphony POS API Demo client' window. The 'Fiscal' tab is selected, and the 'Fiscal API' section is visible. The 'Sanity Code' field is highlighted with a red box, and the 'SanityCheck' button is also highlighted. The 'General' tab is also visible, showing 'Config/Def Type' set to 'UNDEFINED' and 'Ref. Quantity' set to '1'. The 'Order Entry' section shows 'Config/Def Type' set to 'UNDEFINED' and 'Ref. Quantity' set to '1'. The 'Fiscal API' section shows 'Sanity Code' set to '0' and the 'SanityCheck' button.

- Consult the documentation of the country-specific 'Extension Plugin' being initiated on the workstation, and enter a valid Sanity Code value in the text box (e.g. '999').

The screenshot shows the Symphony POS API Demo client interface. The 'Fiscal API' section is active, and the 'Sanity Code' field is set to '999'. The 'SanityCheck' button is highlighted in cyan. The 'Order Entry' section shows 'Ref. Quantity' set to '1'. The 'Reference' field is empty.

- Press the **SanityCheck** button and the results shall display in the output textbox found at the bottom area of the application window.

## Menu Item Quantity for PostTransactionEx API

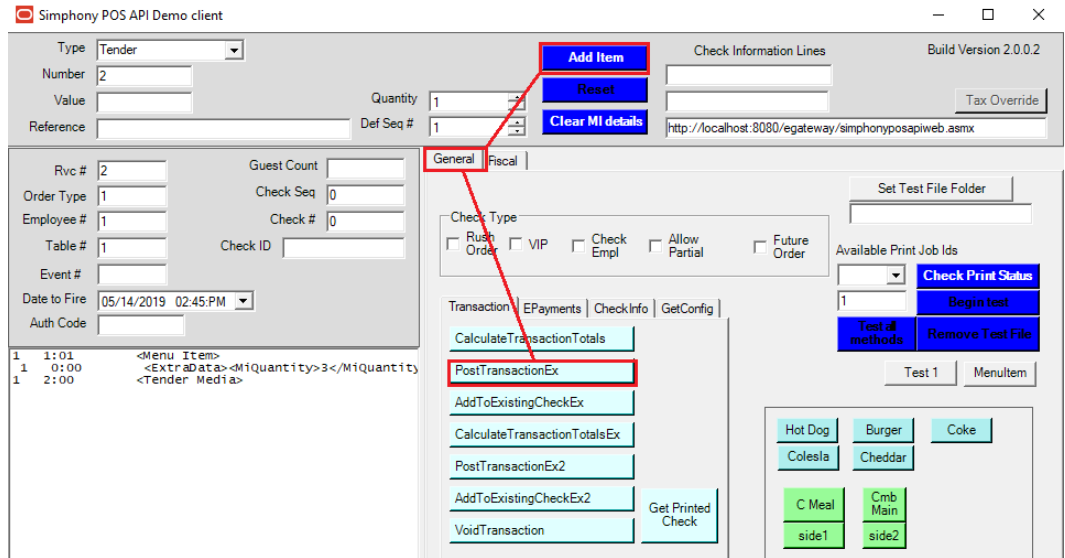
Setting the quantity of menu item, combo meal, combo main, and combo side when testing **PostTransactionEx** API required manual entry of a case-sensitive xml snippet in the **Reference** field. A utility function is provided to avoid human error.

- Enter the quantity in the **Ref. Quantity** field and press **Insert Ref.** button. The `<MiQuantity>` xml snippet will be inserted in the **Reference** field.

The screenshot shows the Symphony POS API Demo client interface. The 'Reference' field contains the XML snippet '<ExtraData><MiQuantity>3</MiQuantity></ExtraData>'. The 'Ref. Quantity' field is set to '3' and the 'Insert Ref.' button is highlighted in cyan. The 'Sanity Code' field is set to '999'.

- Set the Menu Item by entering the object number in **Number** field and proceed to add menu item using **Add Item** button.
- Press **Clear Ref.** button to clear the **Reference** field.
- Set the Tender Media by selecting **Tender** in **Type** dropdown and enter the object number in **Number** field. Press **Add Item** button.

5. Press the **General** tab to display standard functions, then proceed to press **PostTransactionEx** API button.

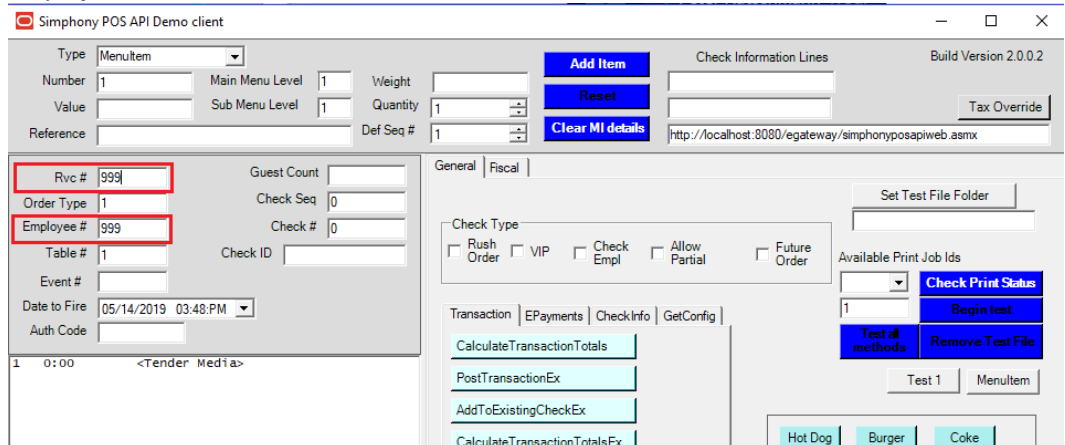


6. Confirm in the print out that the menu item quantity is correctly registered in the system.

## Check Detail Definition Selector

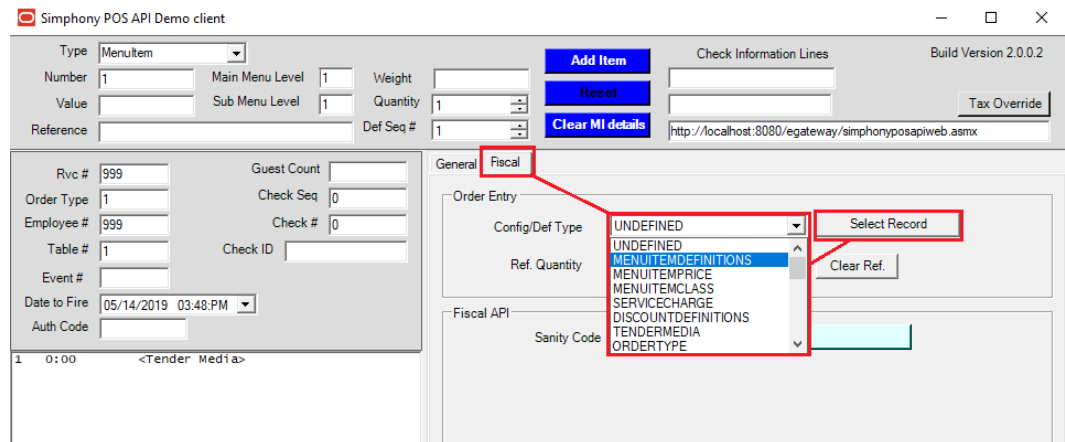
This feature fills the **Number** field with the object number of Menu Item, Service Charge, Discount, and Tender Media definition records that has been selected from a list. The check detail definition records are obtained using the **GetConfigurationInfo** / **GetConfigurationInfoEx** API.

1. Ensure appropriate value is entered in Rvc# and Employee# fields. Selected employee must have access to the definition records.



2. Press **Fiscal** tab and select the configuration/definition type from the **Config/Def Type** dropdown. Then proceed to press the **Select Record** button. Note: Only Configuration/Definition Types MENUITEMDEFINITION, SERVICECHARGE,

DISCOUNTDEFINITION and TENDERMEDIA are currently supported.



3. A window with a list of definition records will display. From this list, select one definition record and press **Set Number Field** button. The window will close and the **Number** field will be updated with the object number of the definition record selected. Note: the **Type** field will not be adjusted automatically, please ensure the **Type** field matches the configuration/definition type defined at step 2 before pressing the **Add Item** button.

# 10

## Extending Transaction Services Through Plug-ins

Transaction Services operations do not raise Extension Application events. There is, however, a mechanism for integrators to extend core behavior of receipt printing, order device output, and check numbering.

When ServiceHost.exe initializes Transaction Services, it looks in the web.config.txt configuration file to see if a plug-in is defined that it can load.

The plug-in can implement one or more of the following interfaces:

- `SimphonyPosApi.ICustomReceiptPlugin`
- `SimphonyPosApi.ICupLabelPlugin`
- `SimphonyPosApi.IAlternateCheckNumberPlugin`

The Visual Studio project attached to this page can be used as a reference on how to implement these interfaces.

Once the project is built, the following steps are used to install and configure the Transaction Services extensions.

1. Build DLL and deploy to the Handlers folder
2. Register the plug-in by editing the web.config.txt file
3. If the `ICupLabelPlugin` interface is implemented, set the Custom Instruction to a value for the order device (image below)

### Web.config.txt Updates

```
<add key="CupLabelClassFile"
value="TransactionServicesEventHandlers.dll,TransactionServicesEventHandlers.CupLabelPluginImplementation" />
```

```
<add key="CustomReceiptClassFile"
value="TransactionServicesEventHandlers.dll,TransactionServicesEventHandlers.CustomReceiptPluginImplementation" />
```

```
<add key="CustomReceipt2ClassFile"
value="TransactionServicesEventHandlers.dll,TransactionServicesEventHandlers.CustomReceipt2PluginImplementation" />
```

```
<add key="AlternateCheckNumberFile"
value="TransactionServicesEventHandlers.dll,TransactionServicesEventHandlers.AlternateCheckNumberPluginImplementation" />
```

If only the `ICupLabelPlugin` interface is implemented/used then only the first line is needed.

The key names are special and recognized by Simphony.

The key value is the name of the DLL, followed by a comma, followed by the Namespace qualified name of the class that implements the interface.

Once this is done, restart service host.

The Transaction Services client will load the libraries on the first transaction. The ICupLabelPlugin interface is called each time a print job is sent by the API.

```
// interface implemented for CupLabelClassFile
public interface ICupLabelPlugin
{
    string Name { get; }
    object CustomFormatting(EXTOPS::DeviceType target,
List<EXTOPS::ExtensibilityOpsDisplayDetail> dtl, string customName,
Dictionary<string, string> currentState);
}

// interface implemented for CustomReceiptClassFile
public interface ICustomReceiptPlugin
{
    string Name { get; }
    bool CustomReceiptPrintFormat(EventMonitorArgs args, List<SortedDetailBase> list);
}

// interface implemented for CustomReceipt2ClassFile
public interface ICustomReceiptPlugin2
{
    string Name { get; }
    bool CustomReceiptPrintFormat(EventMonitorArgs args, List<SortedDetailBase> list,
Dictionary<string, string> currentState);
}

// interface implemented for AlternateCheckNumberFile
public interface IAlternateCheckNumberPlugin
{
    string Name { get; }
    bool GetAlternateCheckNumber(EventMonitorArgs args);
}
```

## Current State Dictionary

The currentState Dictionary passed by the ICupLabelPlugin and ICustomerReceiptPlugin2 interface contains the following keys:

**CheckNumber:**

number of current check

**WorkstationNum:**

workstation number

**CheckID:**

Check ID added/set by the transaction-service client application

**CheckInfoLine[n]:**

[n] is identifier of check-information-line, can have one or more of these

**[ExtensionDetailName]:**

key is the data-name assigned to the extension detail, value is the string-data of the extension detail

**RevenueCenterNum:**

revenue center number

**RevenueCenterName:**

revenue center name

**OrderType:**

order type number of the transaction

**OrderTypeName:**

order type name

## Sample Project

This sample project shows how to implement interfaces in .NET C# assembly:

[TransactionServicesEventHandlers.zip](#)