USER MANUAL

InventoryControl SDK

Wasp Barcode Technologies
1400 10th St.
Plano, TX 75074

Terms, conditions, features, hours and contact information in this document are subject to change without notice. Wasp is committed to providing great products and exceptional customer service. Occasionally we may decide to update our selection and change our service offerings so please check www.waspbarcode.com for the latest information.

Wasp InventoryControl SDK Install Key _____
(Printed on shipped material)

Wasp InventoryControl SDK Registration Key _____
(Obtained from http://www.waspbarcode.com/Register/default.asp)

Table of Contents

Chapter 1: Introduction to the Wasp InventoryControl SDK

1.1 Overview

The Wasp SDK (software development kit) provides you with the means to create a custom application that shares data with your Wasp Database using standard (should this be "our" or maybe "our proprietary") functions and calls via the Wasp Service. This enables you to create applications to manipulate your data in ways not available in the InventoryControl application. All of the changes made to your data will be updated in real time to your Wasp Database.

The Wasp SDK allows you to create applications that enhance the InventoryControl functionality and customize it for your industry. Utilizing the SDK gives you broad customization options without the cost and effort of creating a stand-alone application that would require the re-entry of the data already contained in InventoryControl.

1.2 Technical Data

Wasp InventoryControl SDK version 1.0.

Chapter 2: Getting Started

2.1 Getting Started FAQ

What is included in the Wasp InventoryControl SDK?

You will receive a patch to upgrade InventoryControl 6.x or later to the latest version and the Wasp InventoryControl SDK installer.

What InventoryControl Functions are not available through the SDK?

At this time the following functions are not available in the SDK:

- Assemble
- Disassemble
- Add or Update User Security
- Reconcile Audit
- End Audit

Does the SDK need to be installed on the same machine as the Wasp Inventory Windows Service?

No. It can be installed on any machine, and can even be installed on a Virtual Machine. The only requirement is that the system must have network access to the machine running the Wasp Inventory Windows Service.

What programming code should I use to access the Wasp Inventory Windows Service?

Any programming language can be used to access the interface. The SDK package includes sample code in C# using Visual Studio .NET; however, developers are free to use any language and toolkit that they choose.

Keep in mind the Wasp will test the Sample Application, but will not debug custom applications you develop.

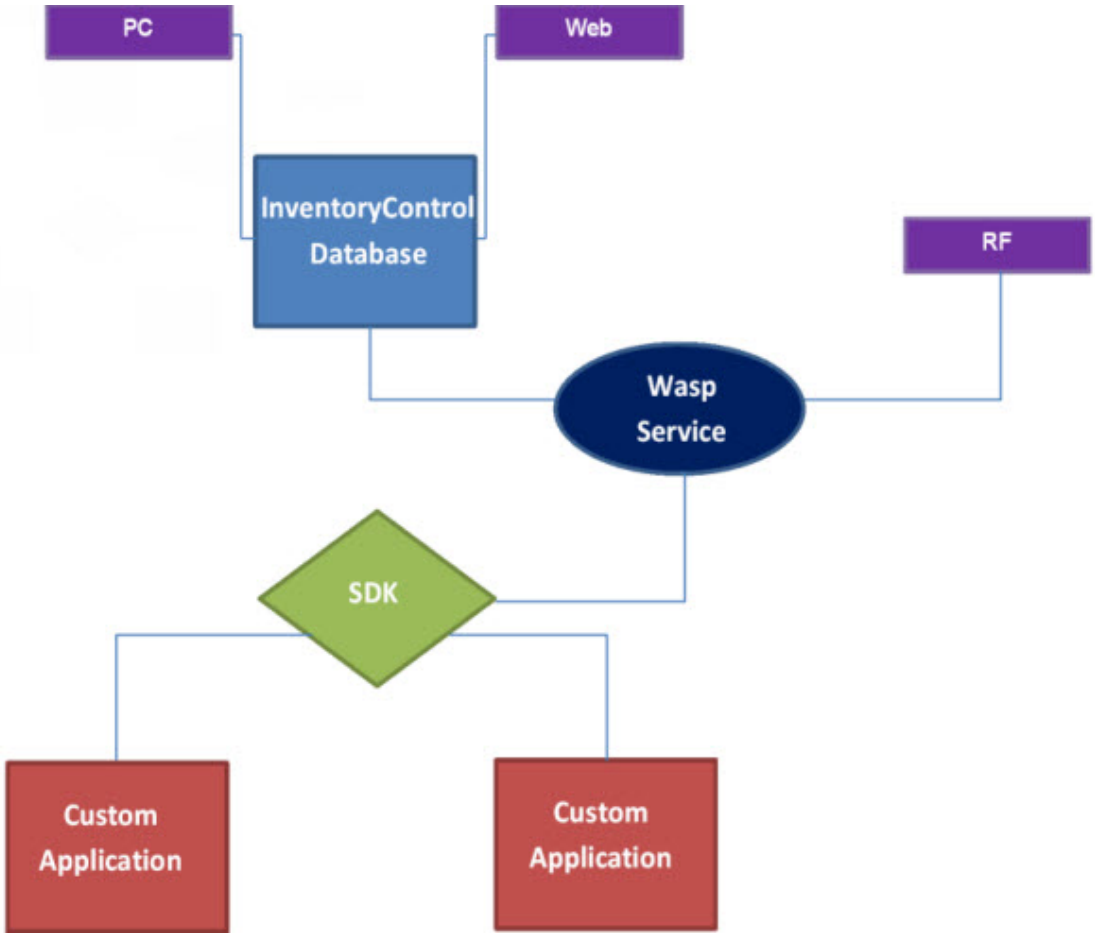
Can I use the Wasp sample code in my own programs?

You are free to create your own applications using the sample code, but Wasp Barcode Technologies does not support the sample code and makes it available on an "as-is" basis.

2.2 Communicating with InventoryControl

2.2.1 Relationship Diagram

The diagram below illustrates the relationship between the SDK and the other InventoryControl components.



2.2.2 Establishing a connection with the Wasp_Inventory_Windows_Service

You will communicate with InventoryControl by creating a connection to the Wasp_Inventory_Windows_Service. Any functions you call from your application will go through the Wasp_Inventory_Windows_Service to the InventoryControl Database.

In order to establish a connection with the Wasp_Inventory_Windows_Service, you must have installed the SDK on a machine that has network access to the service.

Below is an example of the code to connect to the Wasp_Inventory_Windows_Service.

```
     WaspServiceRequest initReq;
ServiceRequestResult ret;
ConnectToHost(out initReq, out ret);

if (ret.ServiceCallReturnVal)
{
// successfully connected
}
else
{
// connection failed. Check error
}

private void ConnectToHost(out WaspServiceRequest initReq, out ServiceRequestResult ret)
{

    initReq = new WaspServiceRequest();
    initReq.WaspServiceUrl = txtConnectionAddress.Text;       // URL of the
InventoryControl service, i.e.
http://localhost:10005/WaspInventoryService/WindowsService
    initReq.LogLevel = WaspServiceRequest.WaspLogLevels.VERBOSE;
    initReq.WaspSdkLicenseKey = txtLicense.Text;       // activated SDK license key, it
starts with 0778-

    initReq.LoginUsername = txtUsername.Text;   // Valid InventoryControl user name
    initReq.LoginPassword = txtPassword.Text;

    ret = initReq.ConnectToService();
}
```

Chapter 3: Requests

3.1 Building Requests

3.1.1 Overview

A Request is the query you are making to the database via the Wasp_Inventory_Windows_Service, either to retrieve data or perform some action on the data. There are four types of requests you can make:

- **Update Requests** - These requests create new data or update existing data.
- **Delete Requests** - Delete existing data. Keep in mind that, as in InventoryControl, when you delete data, you are not removing it permanently from the database. Deleted data is retained, but does not appear in lists or reports and does not affect inventory totals.
- **Get Requests** - Get requests retrieve data stored in the database tables
- **Perform Transaction Requests** - These requests call functions that perform a transaction on data such as Move, Remove, Add, etc.

3.1.2 Connecting to the Service

Prior to requesting information from the database, you must create a connection to the Wasp Inventory Service. See the topic **Communicating with InventoryControl** for details on creating a connection to the service.

3.2 Update Requests

Update requests create new or update existing data.

To view how an update request works in conjunction with other functions, and to copy source code, please see the Sample App. You can view detailed instructions for running **Use Cases** in the Sample App in the topic **Using the Sample Applications**.

Example Update Request
Specific Functions Available

3.2.1 Example Update Request

This example creates a new item.

Note: The order of data in the data dictionary does not matter.

```
private Dictionary<string, string> UpdateItemData(string number)
{
    Dictionary<string, string> dictItem = new Dictionary<string, string>();

    dictItem.Add("item_number", number);
    //dictItem.Add("item_number", "auto_generated");          // if you want item
number to be auto generated
    dictItem.Add("item_type_name", "Inventory");
    dictItem.Add("alt_item_number", "alt_no_1");
    dictItem.Add("category", "monitor");
    dictItem.Add("checkout_duration", "100");
    dictItem.Add("cost", "199.99");
    dictItem.Add("auto_sn", "");
    dictItem.Add("auto_sn_value", "");
    dictItem.Add("cost_method", "FIFO");
    dictItem.Add("description", "this is my description for item 1");
    dictItem.Add("dimension_unit", "Inches");
    dictItem.Add("height", "15");
    dictItem.Add("length", "20");
    dictItem.Add("linked_file", "");
    dictItem.Add("list_price", "300");
    dictItem.Add("manager", "Smith");
    dictItem.Add("manufacturer", "");
    dictItem.Add("max_stock_level", "5");
    dictItem.Add("min_stock_level", "0");
    dictItem.Add("note_text", "Item Note");
    dictItem.Add("reorder_qty", "5");
    dictItem.Add("sale_price", "250");
    dictItem.Add("supplier", "");
    dictItem.Add("supplier_number", "");
    dictItem.Add("require_customers", "false");
    dictItem.Add("track_date_codes", "false");
    dictItem.Add("track_lots", "true");
    dictItem.Add("track_pallets", "true");
    dictItem.Add("require_pos", "false");
    dictItem.Add("track_serial_numbers", "false");
```

```
    dictItem.Add("require_suppliers", "false");
    dictItem.Add("unit_of_measure", "each");
    dictItem.Add("weight", "20");
    dictItem.Add("weight_unit", "pounds");
    dictItem.Add("width", "3");
    dictItem.Add("external_key", "0");
    dictItem.Add("usrdefdate1", "4/15/1963");
    dictItem.Add("usrdefdate2", "8/20/1992");
    dictItem.Add("usrdefnumber1", "10");
    dictItem.Add("usrdeftext1", "USPS");
    dictItem.Add("usrdeftext2", "American Eagle One Ounce Gold Proof ");

    return dictItem;
}
```

3.2.2 Specific Functions Available

List of specific functions

- UpdateItem
- UpdateCustom*
- UpdateSupplier*
- UpdatePurchaseOrder
- UpdatePickOrder
- UpdateSite
- UpdateLocation*
- UpdateManufacturer*
- UpdateItemSupplierSettings
- UpdateItemLocationSettings
- UpdateShippingMethod
- UpdatePaymentMethod
- UpdatePriceLevel
- UpdateSalesTax
- UpdateCompanyInfo*

* Indicates this function is multi-tiered to accommodate multiple addresses or other layered data.

3.3 Delete Requests

Delete requests delete existing data. Keep in mind that, as in InventoryControl, when you delete data you are not removing it permanently from the database. Deleted data is retained, but does not appear in lists or reports and does not affect inventory totals.

To view how a delete request works in conjunction with other functions, and to copy source code, please see the Sample App. You can view detailed instructions for running **Use Cases** in the Sample App in the topic **Using the Sample Application**.

3.3.1 Example Delete Request

The code example below deletes item uc1_1010.

```
ServiceRequestResult ret = null;

ret = m_connector.DeleteItem("uc1_1010", true);
```

3.3.2 Specific Functions Available

- DeleteItem
- DeleteCustomer
- DeleteSupplier
- DeletePurchaseOrder
- DeletePickOrder
- DeleteSite
- DeleteLocation
- DeleteManufacturer
- DeleteItemSupplier
- DeleteItemLocation
- DeleteShippingMethod
- DeletePaymentMethod
- DeletePriceLevel
- DeleteSalesTax

3.4 Get Requests

Getting Information requests encompass all calls to retrieve information contained in the InventoryControl database. No data is changed in the database using Get requests.

To view how a Get request works, and to copy source code, please see the Sample App. You can view detailed instructions for running **Use Cases** in the Sample App in the topic **Using the Sample Application**.

Example Get Request
Available Data

3.4.1 Example Get Request

The code snippet below requests all new Pick Orders.

```
ServiceRequestResult ret = null;

QueryWhereCriteria[] commands = new QueryWhereCriteria[1];

Dictionary<string, string>[] data = null;
Dictionary<string, string>[][] details = null;

commands = SpecifyCriteriaForPickOrderData();
ret = m_connector.GetPickOrder(commands, out data, out details);

private QueryWhereCriteria[] SpecifyCriteriaForPickOrderData()
{
       QueryWhereCriteria[] list = new QueryWhereCriteria[1];
       list[0] = new QueryWhereCriteria("po_status", "=", "New");
       return list;
}
```

3.4.2 Available Data

The available tables from which you can extract data are:

- Item
- Customer
- Supplier
- Purchase Order
- Pick Order
- Transactions
- Site
- Location
- Manufacturer
- User Security Privileges
- Next Serial Number
- Addresses
- Attachments
- Price Level
- Sales Tax
- Shipping Method
- Payment Method
- Item Supplier Settings
- Item Location Settings
- Item UOMs (Unit of Measure)
- Form Labels

3.5 Perform Transaction Requests

Transaction requests perform some action on the data, such as adding inventory, checking an item in or out or closing a purchase order.

To view how a Perform Transaction request works, and to copy source code, please see the Sample App. You can view detailed instructions for running **Use Cases** in the Sample App in the topic **Using the Sample Application**.

Example Perform Transaction Request
Specific Transactions Available

3.5.1 Example Perform Transaction Request

The code below adds a quantity of 5 to item uc5_1010.

Note: If you provide a serial number as a Trackby field, make sure the Quantity is set to 1. If the quantity is other than 1, the call will fail.

```
ServiceRequestResult ret = null;

Dictionary<string, string> dictData;

dictData = GetAddTranxData("uc1_1010","uc1_s1", "uc1_loc1", "uc1_c1");
ret = m_connector.PerformAdd(dictData);


private Dictionary<string, string> GetAddTranxData(string number, string site, string
location, string customer)
{
    Dictionary<string, string> dictData = new Dictionary<string, string>();

    // fields needed for add
    dictData.Add("item_number", number);
    dictData.Add("site_name", site);
    dictData.Add("location", location);
    dictData.Add("lot", "1");
    dictData.Add("date_code", "");
    dictData.Add("serial_number", "");
    dictData.Add("pallet", "1");
    dictData.Add("po", "");
    dictData.Add("supplier", "");
    dictData.Add("customer_number", customer);
    dictData.Add("quantity", "5");
    dictData.Add("print_label_no", "");
    dictData.Add("cost", "5");
    dictData.Add("date_acquired", "10/1/2011");
    dictData.Add("record_source", "sdk_inf150");
    dictData.Add("printer_name", "");

    return dictData;
```

3.5.2 Specific Transactions Available

- PerformAdd
- PerformRemove
- PerformMove
- PerformCheck In
- PerformCheck Out
- PerformAdjust
- PerformReceive
- PerformPick
- PerformStart Audit
- PerformReconcile Audit
- PerformEnd Audit
- PerformIsPOComplete
- PerformClosePurchaseOrder
- PerformIsPickOrderComplete
- PerformClosePickOrder

Chapter 4: Handling Responses

Responses to requests your application makes to InventoryControl are handled in one of two ways. Either an action is performed on the data (such as UpdateItem, UpdateLocation, UpdateSite, etc.) or data is retrieved in list form but not altered (Get requests).

ErrorCode, if applicable, would be one of the codes in the enumeration. But there would be cases that there is no error code returned. An error message will **ALWAYS** be populated and returned with meaningful information.
If you receive an ErrorCode, check the StackTrace as it may provide detailed error data.

The Error Structure is listed below.

```
public class WaspErrorObj
{
    public int ErrorCode;
    public string ErrorMsg;
    public string Source;
    public string StackTrace;

    public WaspErrorObj(WaspError error)
    {
        ErrorCode = error.ErrorCode;
        ErrorMsg = error.ErrorDesc;
        Source = error.Source;
        StackTrace = error.StackTrace;
    }
}

public enum Errors
{
       BadXml = 2,
       GeneralSyncError = 3,
       InvoiceMissingCustomer = 4,
       InvoiceCustomerNotFound = 5,
       InvoiceLineItemMissingItem = 6,
       InvoiceItemNotFound = 7,
       InvoiceMissingBatchNumber = 8,
       ItemNotAssignedToSite = 9,
       ObjectTypeNotSearchableByName = 10,
       InvoiceExists = 11,
       ExpenseMissingBatchNumber = 13,
       ExpenseVendorNotFound = 14,
       ExpenseMissingVendor = 15,
       ExpenseExists = 16,
       AccountMissingNumber = 17,
       AccountMissingName = 18,
       AccountMissingType = 19,
       InvalidAccountType = 20,
       ExpenseLineItemMissingAccount = 21,
       ExpenseLineItemInvalidAccountName = 22,
       ExpenseMissingRefNumber = 23,
       ItemPricingInfoNotFound = 24,
       ExpenseAPAccountInvalidId = 25,
```

```
        ExpenseAPAccountInvalidName = 26,
        EmployeeMissingDepartment = 27,
        EmployeeMissingJobTitle = 28,
        EmployeeInvalidJobTitle = 29,
        EmployeeInvalidDepartment = 30,
        EmployeeMissingSSN = 31,
        EmployeeMissingFirstName = 32,
        EmployeeMissingLastName = 33,
        EmployeeMissingId = 34,
        EmployeePayCodeDoesNotExist = 35,
        EmployeeIncorrectPayRate = 36,
        EmployeeDeductionCodeDoesNotExist = 37,
        EmployeeIncorrectPayMultiplier = 38,
        EmployeeCouldNotAddPayCode = 39,
        PayCodeDoesNotExist = 40,
        EmployeeHasConflictingPayCodes = 41,
        EmployeeMissingBasePayCode = 42,
        EmployeePayCodeRequiresSUTAState = 43,
        EmployeeCouldNotAddDeductionCode = 44,
        TimeEntryMissingPayCode = 45,
        TimeEntryPayCodeDoesNotExist = 46,
        InvoiceItemInvalidPriceLevel = 47,
        ItemInvalidUofMSchedule = 48,
        ItemInvalidPriceLevel = 49,
        PayCodeMissingDescription = 50,
        PayCodeMissingCode = 51,
        TimeEntryMissingEmployee = 52
}
```

Chapter 5: Using the Sample Application

5.1 Sample Application Overview

The Sample Application that comes packaged with the SDK materials provides you with an easy way to see the code that handles all the available InventoryControl functions. You can view the code from the Sample Application and/or copy it to your clipboard.

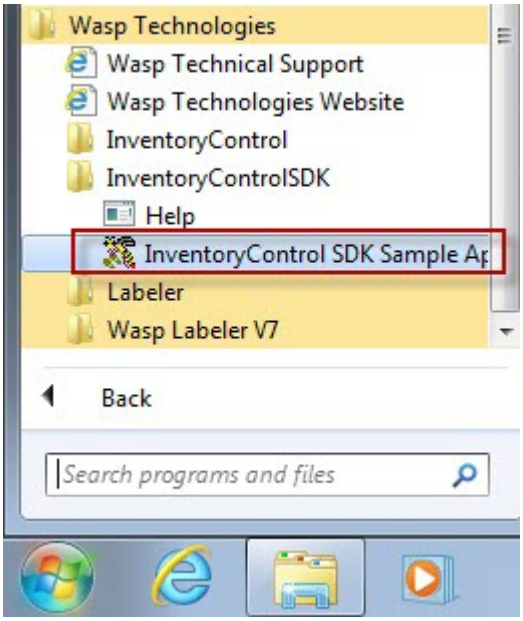
All of the InventoryControl requests available in the SDK are contained in the Sample Application. The requests are grouped together into Use Cases which you can run to see how the code works. For details on each Use Case, please see the topic **Use Case Overview**.

Important: Keep in mind that the Use Cases perform actual functions, such as creating items, moving items, creating locations, updating company information, etc. ***If you are using the Sample App as your test application, make sure that you are NOT connecting to your live InventoryControl service, which would perform the functions on your production database.*** Instead, create a test service and test database and connect the Sample App to those to retain the integrity of your production database.

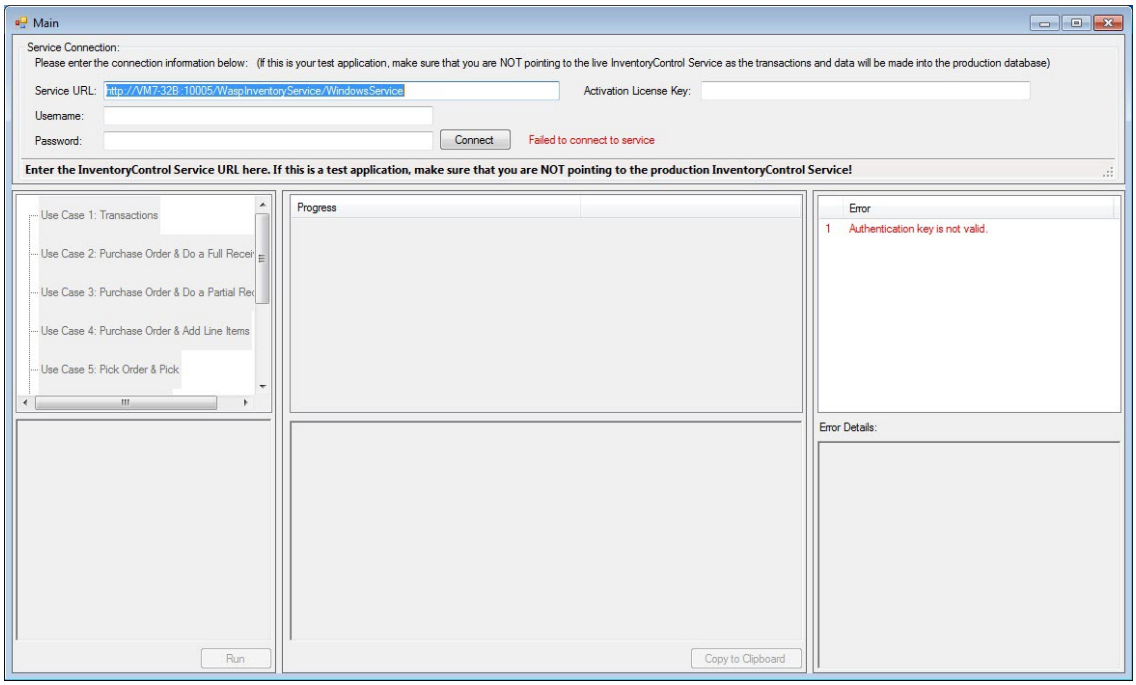
In this chapter:

5.2 Running the Sample Application for the First Time

1. Run the Sample Application by clicking on the InventoryControl SDK Sample App icon in your start menu or navigate to the install folder and click **InventoryControl SDK Sample App**.



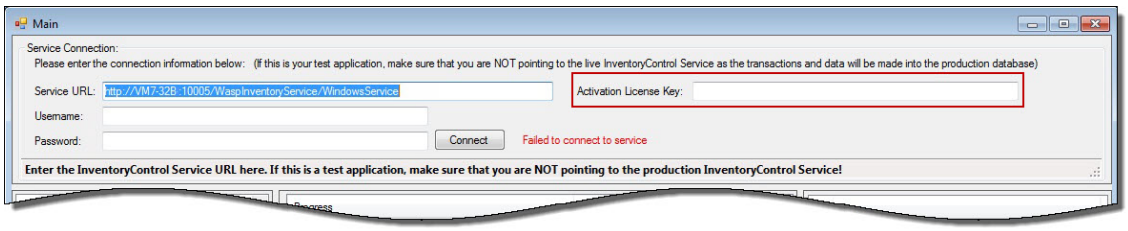
The Sample App will appear:



2. Note that the first time you run the Sample App after installation you will not be connected to the service. The error message in the Error pane indicates "Authentication key is not valid".

   You can obtain an Activation Key (this is different from your Installation Key) by registering your product on the Wasp Product Registration page: **http://waspbarcode.com/Register/default.asp**.

   When you have obtained your Activation Key, enter it in the **Activation License Key** field..

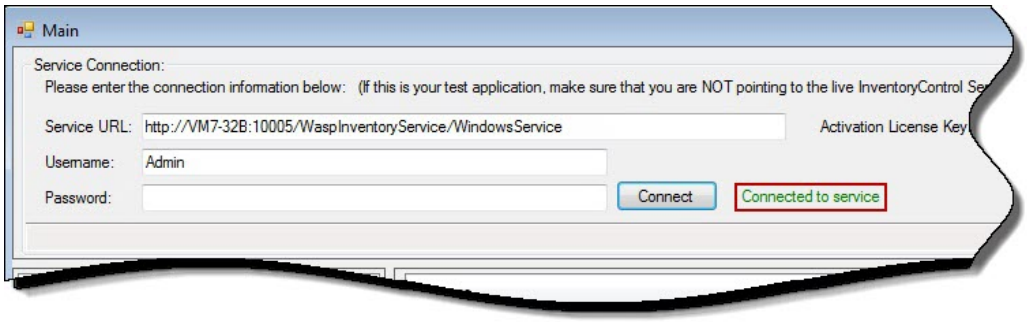


3. After entering a valid Activation Key, you will be able to connect to your service.

   a. Make sure the correct **Service URL** is entered. This field will pre-populate with the service URL if one was detected during installation.

   b. Enter the **Username** and **Password** for your service. If you are connecting to a test installation of InventoryControl and you have not setup any passwords, the **Username** is Admin and the **Password** is blank.

   c. Click the **Connect** button. *Connected to service* will appear as shown below when you have successfully connected.

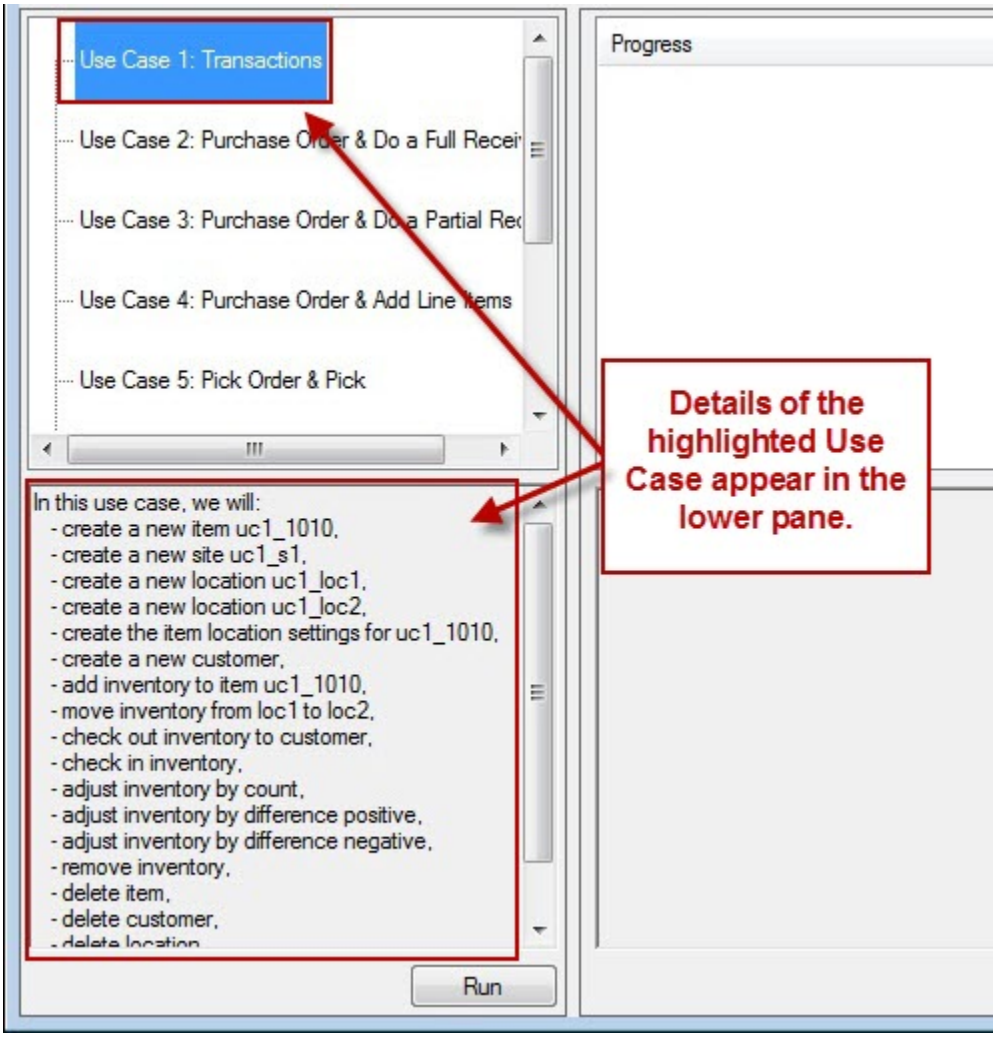


   You are now ready to run Use Cases in the Sample Application. After entering an Activation License Key, the Sample App will automatically connect to the service at opening.

5.3 Running Use Cases

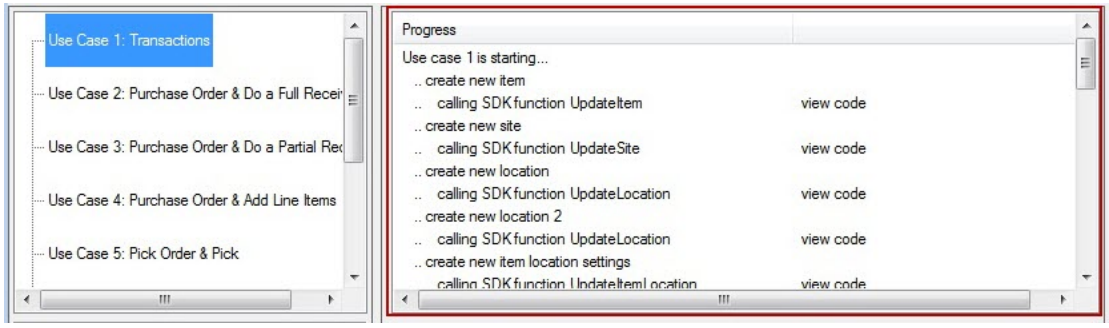
Every InventoryControl function available in the SDK is contained in one or more of the Use Cases. You can see how the functions work together and view code for each function by running the various Use Cases.

1. Highlight a Use Case in the **Use Case** pane. The functions contained in the selected Use Case appear in the window directly below the **Use Case** pane:

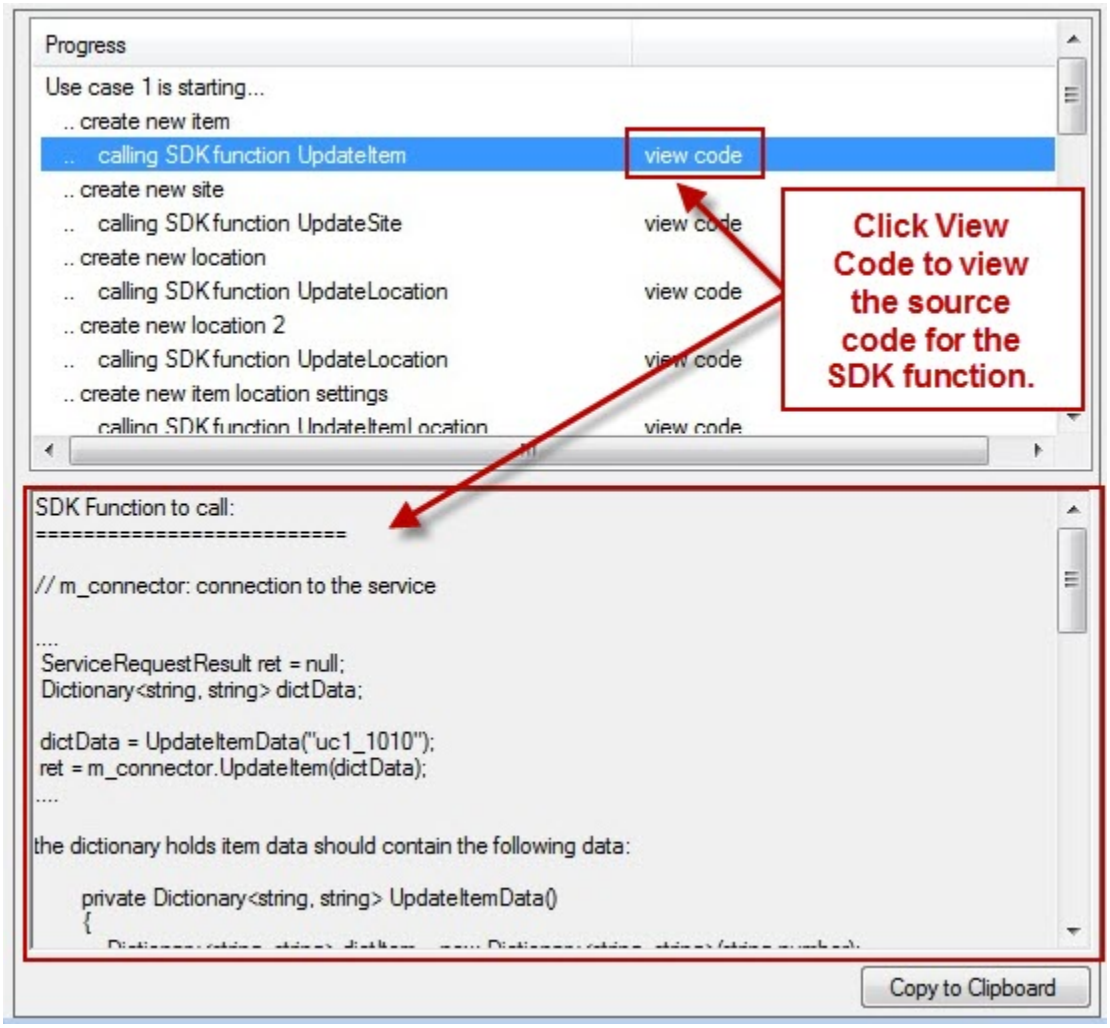


For a full list of Use Cases and their associated functions, please refer to the topic **Use Cases Overview**.

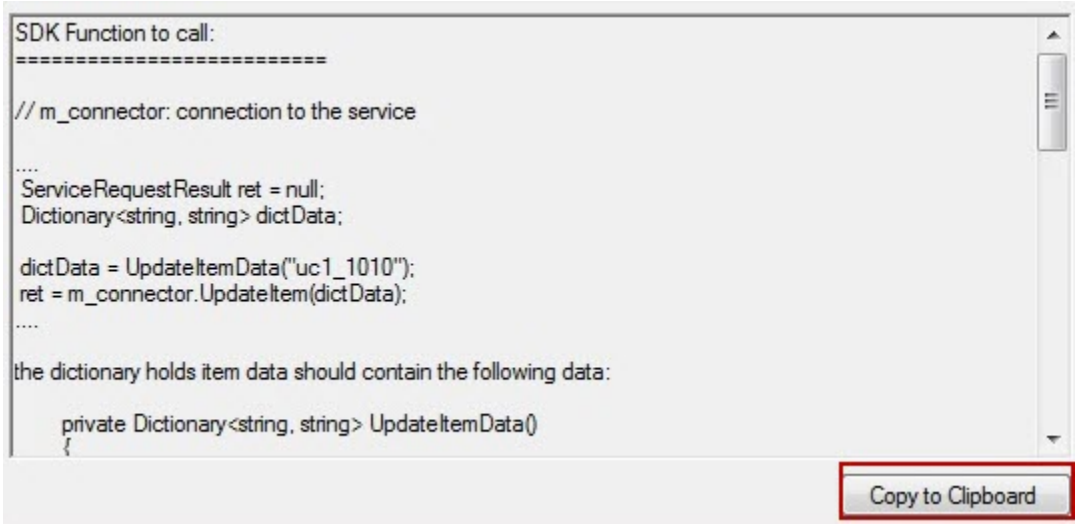
2. To run a Use Case, click the **Run** button. The behind-the-scenes calls for all functions will appear in the **Progress** pane.



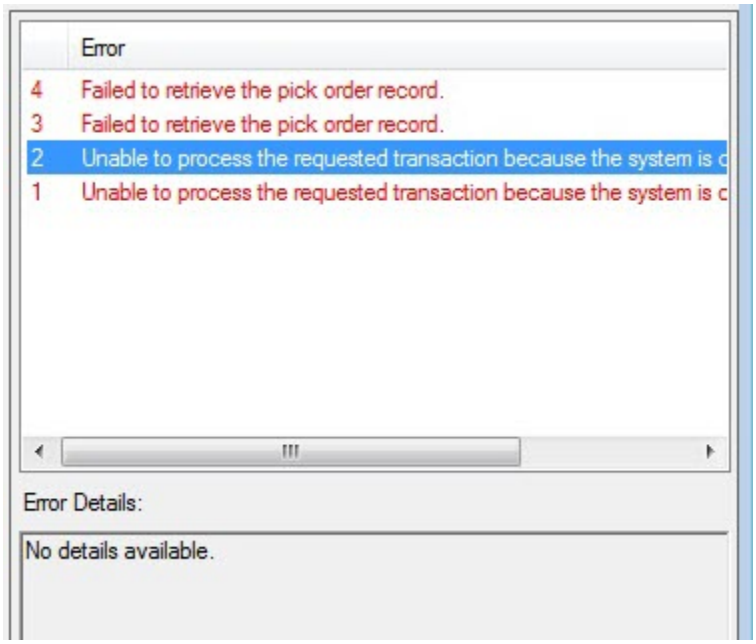
3. To view the source code for any of the functions in the **Progress** pane click the **View Code** link. The source code for the function, along with instructional data, appears in the pane directly below the **Progress** pane.



4. You can copy the source code to your clipboard by clicking the **Copy to Clipboard** button.



5. If any errors occur during the running of a Use Case, they will appear in the **Error** pane. You can click on an error to see details if any are available.



5.4 Use Cases

5.4.1 Use Case Overview

The functions available in the InventoryControl SDK are grouped into nine Use Cases in the Sample Application. Using the Sample Application, you can run these use cases to see how the components work together and to view/copy the source code.

Please note that you can copy the source code directly from the Sample App using the **Copy to Clipboard** button. Please see the topic **Using the Sample Application** for instructions.

Use Case 1: Transactions
Use Case 2: Purchase Order and Do a Full Receive
Use Case 3: Purchase Order and Do a Partial Receive
Use Case 4: Purchase Order and Add Line Items
Use Case 5: Pick Order and Pick
Use Case 6: Get List of Data
Use Case 7: Audit
Use Case 8: Update Company Information
Use Case 9: Errors

5.4.2 Use Case 1: Transactions

The **Transactions** use case goes through the process of creating a new item, site, locations, customer, adding inventory, checking in/out, etc. This use case is a full cycle in that all new data that is created is subsequently deleted.

To view how a request works in conjunction with other functions, and to copy source code for the individual requests, please see the Sample App. You can view detailed instructions for running the Sample App in the topic Using the Sample App.

This use case performs the following functions:

SDK Function	Result
UpdateItem	Creates a new item labeled uc1_1010.
UpdateSite	Creates a new site labeled us1_s1.
UpdateLocation	Creates a new location labeled uc_1_loc1.
UpdateLocation	Creates a new location labeled uc1_loc2.
UpdateItemLocation	Creates the item location settings for uc1_1010.
UpdateCustomer	Creates a new customer.
PerformAdd	Adds inventory to item uc1_1010.
PerformMove	Moves inventory from loc1 to loc2.
PerformCheckOut	Checks out inventory to the customer.
PerformCheckIn	Check in inventory.
PerformAdjust	Adjusts inventory by count.
PerformAdjust	Adjusts inventory by difference positive.
PerformAdjust	Adjusts inventory by difference negative.
PerformRemove	Removes inventory.
DeleteItem	Deletes item uc1_1010.
DeleteCustomer	Deletes customer.
DeleteLocation	Deletes location uc1_loc1.
DeleteLocation	Deletes location us1_loc2.
DeleteSite	Deletes site uc1_1010.

5.4.3 Use Case 2: Purchase Order & Do a Full Receive

This use case runs through a cycle of creating a new item, customer, supplier, shipping method, payment method, purchase order, etc. and fully receiving/closing the purchase order.

To view how a request works in conjunction with other functions, and to copy source code for the individual requests, please see the Sample App. You can view detailed instructions for running the Sample App in the topic Using the Sample App.

This use case performs the following functions:

SDK Function	Result
UpdateItem	Creates a new item labeled uc2_1010.
UpdateCustomer	Creates a new customer labeled uc2_c1.
UpdateSupplier	Creates a new supplier labeled uc2_1010.
UpdateShippingMethod	Creates a new shipping method labeled uc2_1010.
UpdatePaymentMethod	Creates a new payment method labeled uc2_1010.
UpdateSupplier	Creates the item supplier settings for uc2_1010.
UpdatePurchaseOrder	Creates a new purchase order for uc2_1010.
UpdateSite	Creates a new site labeled uc2_s1.
UpdateLocation	Creates a new location labeled uc2_loc1.
UpdateLocation	Creates a new location labeled uc2_loc2.
PerformReceive	Fully receives the purchase order uc2_1010s
IsPoClosed	Checks if the purchase order uc2_1010 is complete.
PerformClosePO	Closes the purchase order uc2_1010.

5.4.4 Use Case 3: Purchase Order & Do a Partial Receive

This use case runs through a cycle of creating a new item, customer, supplier, shipping method, payment method, purchase order, etc. and partially receiving/closing the purchase order.

To view how a request works in conjunction with other functions, and to copy source code for the individual requests, please see the Sample App. You can view detailed instructions for running the Sample App in the topic Using the Sample App.

This use case performs the following functions:

SDK Function	Result
UpdateItem	Creates a new item labeled uc3_1010.
UpdateCustomer	Creates a new customer labeled uc3_c1.
UpdateSupplier	Creates a new supplier labeled uc3_1010.
UpdateShippingMethod	Creates a new shipping method labeled uc3_1010.
UpdatePaymentMethod	Creates a new payment method labeled uc3_1010.
UpdateItemSupplier	Creates the item supplier settings for uc3_1010.
UpdatePurchaseOrder	Creates a new purchase order uc3_1010.
UpdateSite	Creates a new site labeled uc3_s1.
UpdateLocation	Creates a new location labeled uc3_loc1.
UpdateLocation	Creates a new location labeled uc3_loc2.
PerformReceive	Partially receives the purchase order uc3_1010.
IsPoClosed	Checks if the purchase order is complete uc3_1010.
PerformClosePO	Closes the purchase order uc3_1010.

5.4.5 Use Case 4: Purchase Order and Add Line Items

This use case runs through a cycle of creating a new item, customer, supplier, shipping method, payment method, purchase order, and adding line items to the purchase order.

To view how a request works in conjunction with other functions, and to copy source code for the individual requests, please see the Sample App. You can view detailed instructions for running the Sample App in the topic Using the Sample App.

This use case performs the following functions:

SDK Function	Result
UpdateItem	Creates a new item labeled uc4_1010.
UpdateCustomer	Creates a new customer labeled uc4_c1.
UpdateSupplier	Creates a new supplier labeled uc4_1010.
UpdateShippingMethod	Creates a new shipping method labeled uc4_1010.
UpdatePaymentMethod	Creates a new payment method labeled uc4_1010.
UpdateItemSupplier	Creates the item supplier settings for uc4_1010.
UpdatePurchaseOrder	Creates a new purchase order uc4_1010.
UpdatePurchaseOrderAddLineItems	Adds 14 line items to purchase order uc4_1010.
UpdatePurchaseOrderAddLineItems	Adds 20 line items to purchase order uc4_1010.
DeletePurchaseOrder	Deletes purchase order uc4_1010.

5.4.6 Use Case 5: Pick Order & Pick

This use case runs through a cycle of creating a new item, customer, supplier, shipping method, payment method, pick order, site and location and then adding and picking inventory.

To view how a request works in conjunction with other functions, and to copy source code for the individual requests, please see the Sample App. You can view detailed instructions for running the Sample App in the topic Using the Sample App.

This use case performs the following functions:

SDK Function	Result
UpdateItem	Creates a new item labeled uc5_1010.
UpdateCustomer	Creates a new customer labeled uc5_1010.
UpdateSupplier	Creates a new supplier labeled uc5_1010.
UpdateShippingMethod	Creates a new shipping method labeled uc5_1010.
UpdatePaymentMethod	Creates a new payment method labeled uc5_1010.
UpdateItemSupplier	Creates the item supplier settings for uc5_1010.
UpdatePickOrder	Creates a new pick order uc5_1010.
UpdateSite	Creates a new site labeled uc5_s1.
UpdateLocation	Creates a new location labeled uc5_loc1.
UpdateLocation	Creates a new location labeled uc5_loc2.
PerformAdd	Adds inventory to item uc5_1010.
PerformPick	Fully receives the pick order uc5_1010.
IsPickOrderClosed	Checks if the pick order is complete uc5_1010.
PerformClosePickOrder	Closes the pick order uc5_1010.

5.4.7 Use Case 6: Get List of Data

This use case performs a get on requested data and returns the data as a text file.

To view how a request works in conjunction with other functions, and to copy source code for the individual requests, please see the Sample App. You can view detailed instructions for running the Sample App in the topic Using the Sample App.

The available tables from which you can extract data are:

- Item
- Customer
- Supplier
- Purchase Order
- Pick Order
- Transactions
- Site
- Location
- Manufacturer
- Addresses
- Attachments
- Price Level
- Sales Tax
- Shipping Method
- Payment Method
- Item Supplier Settings
- Item Location Settings
- Item UOMs (Unit of Measure)

5.4.8 Use Case 7: Audit

The Audit use case creates an item, site, location, customer, adds inventory and then starts an audit. This Use Case does not perform a reconcile or stop the audit. You will need to manually reconcile and turn the audit off from InventoryControl on the PC. If you have setup a test service and database, you should also have an install of InventoryControl pointing to that service/database. You will need to manually turn off the audit on your InventoryControl test install in order to run other use cases in the Sample Application.

To view how a request works in conjunction with other functions, and to copy source code for the individual requests, please see the Sample App. You can view detailed instructions for running the Sample App in the topic Using the Sample App.

This use case performs the following functions:

SDK Function	Result
UpdateItem	Creates a new item labeled uc7_1010.
UpdateSite	Creates a new site labeled uc7_s1.
UpdateLocation	Creates a new location labeled uc7_loc1.
UpdateLocation	Creates a new location labeled uc7_loc2.
UpdateItemLocation	Creates item location settings for uc7_1010.
UpdateCustomer	Creates a new customer labeled uc7_c1.
PerformAdd	Adds inventory to uc7_1010.
PerformStartAudit	Starts Audit.

5.4.9 Use Case 8: Update Company Information

This use case performs the UpdateCompanyInfo request. It adds the company "Test Company" along with address information, a website address and email information.

To view how a request works in conjunction with other functions, and to copy source code for the individual requests, please see the Sample App. You can view detailed instructions for running the Sample App in the topic Using the Sample App.

This use case performs the following functions:

SDK Function	Result
UpdateCompanyInfo	Updates the company Test Company and associated website, email, address information, etc.

5.4.10 Use Case 9: Errors

The Errors use case produces results with errors so you can see how these errors are handled by the service. The Wasp SDK DLL and the Wasp InventoryControl service log both progresses and errors. You can find the Wasp SDK log files on your local machine's common data folder. On a Windows 7 machine, the location is: C:\ProgramData\Wasp Barcode Technologies\WaspInventorySDK

The InventoryControl service's log can be found on the machine from which the service is running. This log is also located under the common data folder. On a Windows 7 machine, the location is: C:\ProgramData\Wasp Barcode Technologies\WaspInventoryWindowsService\6.0.0.0

This use case performs the following functions:

SDK Function	Result
UpdateItem	Creates a new item labeled uc9_1010.
UpdateSite	Creates a new site labeled uc9_s1.
UpdateLocation	Creates a new location labeled uc9_loc1.
UpdateCustomer	Creates a new customer
PerformAdd	Adds inventory to item uc9_1010.
DeleteItem	Deletes item uc9_1010.
DeleteCustomer	Deletes customer.
DeleteLocation	Deletes location uc9_loc1.

Appendix A - Functions in the SDK DLL

```
public class ServiceRequestResult
{
        public bool ServiceCallReturnVal { get; set; }
        public List<WaspErrorObj> ErrorDetails { get; set; }
}
```

In this topic:

ConnecttoService

```
/// <summary>
/// Authenticates with InventoryControl Service and logins in with specified
InventoryControl user name and password.
/// Prior to calling this function, must create an instance of WaspInitializeRequest and
specify all the following:
/// - WaspServiceUrl
/// - WaspSdkLicenseKey
/// - LoginUsername
/// - LoginPassword
/// The following two items are optional:
/// - LogLevel: defaults to log only warnings
/// - LogFilePath: defaults to [Common Data Folder]\Wasp Barcode
Technologies\WaspInventorySDK\WaspSDK.log
/// </summary>
/// <returns>a ServiceRequestResult structure with a return value of whether the call is
successful or not, and error details if the call is not successful.</returns>
public ServiceRequestResult ConnectToService()
```

UpdateItem

```
/// <summary>
/// Create a new item or update an existing item record with provided info
/// </summary>
/// <param name="dictItem">A dictionary of strings for item data. Each pair of data
includes the key of the data field and the value. </param>
/// <returns>a ServiceRequestResult structure with a return value of whether the call is
successful or not, and error details if the call is not successful.</returns>
public ServiceRequestResult UpdateItem(Dictionary<string, string> dictItem)
```

UpdateCustomer

```
/// <summary>
/// Create a new customer or update an existing customer record with provided info
/// </summary>
/// <param name="dictCustomer"></param>contains general customer information
/// <param name="dictAddress"></param>contains lists of addresses for the customer
/// <returns>a ServiceRequestResult structure with a return value of whether the call is
successful or not, and error details if the call is not successful.</returns>
public ServiceRequestResult UpdateCustomer(Dictionary<string, string> dictCustomer,
Dictionary<string, string>[] dictAddress)
```

UpdateSupplier

```
/// <summary>
/// Create a new supplier or update an existing ssupplier record with provided info
/// </summary>
/// <param name="dictSupplier"></param>contains general information
/// <param name="dictAddress"></param>contains lists of addresses associated with
supplier
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult UpdateSupplier(Dictionary<string, string> dictSupplier,
Dictionary<string, string>[] dictAddress)
```

UpdatePurchaseOrder

```
/// <summary>
/// create a new or update an existing purchase order
/// </summary>
/// <param name="dictPurchaseOrder">contains general purchase order information</param>
/// <param name="dictItemLineItem">contains line item information</param>contains line
item information
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult UpdatePurchaseOrder(Dictionary<string, string>
dictPurchaseOrder, Dictionary<string, string>[] dictItemLineItem)
```

UpdatePurchaseOrderAddLineItems

```
/// <summary>
/// add specified line items to an existing purchase order
/// </summary>
/// <param name="dictPurchaseOrder">contains general purchase order information</param>
/// <param name="dictItemLineItem">contains line item information</param>contains line
item information
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult UpdatePurchaseOrderAddLineItems(Dictionary<string, string>
dictPurchaseOrder, Dictionary<string, string>[] dictItemLineItem)
```

UpdatePickOrder

```
/// <summary>
/// create a new or update an existing pick order
/// </summary>
/// <param name="dictPickOrder">contains general pick order information</param>
/// <param name="dictItemLineItem">contains line item information</param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult UpdatePickOrder(Dictionary<string, string> dictPickOrder,
Dictionary<string, string>[] dictItemLineItem)
```

UpdateSite

```
/// <summary>
/// create a new or update an existing site record
/// </summary>
/// <param name="dictSite">site information</param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult UpdateSite(Dictionary<string, string> dictSite)
```

UpdateLocation

```
/// <summary>
/// create a new or update an existing location record
/// </summary>
/// <param name="dictLocation">location data</param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult UpdateLocation(Dictionary<string, string> dictLocation)
```

UpdateManufacturer

```
/// <summary>
/// Create a new manufacturer or update an existing manufacturer record with provided
info
/// </summary>
/// <param name="dictManufacturer"></param>contains general information
/// <param name="dictAddress"></param>contains lists of addresses associated with
manufacturer
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult UpdateManufacturer(Dictionary<string, string>
dictManufacturer, Dictionary<string, string>[] dictAddress)
```

UpdatePriceLevel

```
/// <summary>
/// create a new or update an existing price level
/// </summary>
/// <param name="dictPriceLevel"></param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult UpdatePriceLevel(Dictionary<string, string> dictPriceLevel)
```

UpdateSalesTax

```
/// <summary>
/// create a new or update an existing sales tax record
/// </summary>
/// <param name="dictSalesTax"></param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult UpdateSalesTax(Dictionary<string, string> dictSalesTax)
```

UpdateItemSupplier

```
/// <summary>
/// create new or update existing item supplier settings along with order unit settings
/// </summary>
/// <param name="dictItem"></param>
/// <param name="dictUOMs"></param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult UpdateItemSupplier(Dictionary<string, string> dictItem,
Dictionary<string, string>[] dictUOMs)
```

UpdateItemLocation

```
/// <summary>
/// create new or update existing item location settings
/// </summary>
/// <param name="dictItem"></param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult UpdateItemLocation(Dictionary<string, string> dictItem)
```

UpdateShippingMethod

```
/// <summary>
/// Create a new shipping method
/// </summary>
/// <param name="method">name of the shipping method</param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult UpdateShippingMethod(string method)
```

UpdatePaymentMethod

```
/// <summary>
/// Create a new payment method
/// </summary>
/// <param name="method"></param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult UpdatePaymentMethod(string method)
```

Transaction Requests

```
/// <summary>
/// The following is a list of transaction requests to complete corresponding inventory
transactions
/// </summary>
/// <param name="dictTranx">transaction data with location, track by information and
quantity specification</param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult PerformAdd(Dictionary<string, string> dictTranx)
public ServiceRequestResult PerformRemove(Dictionary<string, string> dictTranx)
public ServiceRequestResult PerformMove(Dictionary<string, string> dictTranx)
public ServiceRequestResult PerformAdjust(Dictionary<string, string> dictTranx)
public ServiceRequestResult PerformCheckIn(Dictionary<string, string> dictTranx)
public ServiceRequestResult PerformCheckOut(Dictionary<string, string> dictTranx)
public ServiceRequestResult PerformAudit(Dictionary<string, string> dictTranx)
public ServiceRequestResult PerformStartAudit()
public ServiceRequestResult PerformEndAudit()
public ServiceRequestResult PerformClosePO(Dictionary<string, string> dictTranx)
public ServiceRequestResult PerformClosePickOrder(Dictionary<string, string> dictTranx)
// the following two methods have an additional parameter bCloseIfComplete to instruct
whether to close the purchase/pick order if it is fully received/picked.
public ServiceRequestResult PerformReceive(Dictionary<string, string> dictTranx, bool
bCloseIfComplete)
public ServiceRequestResult PerformPick(Dictionary<string, string> dictTranx, bool
bCloseIfComplete)
```

IsPOClosed/IsPickOrderClosed

```
/// <summary>
/// check to see if specified purchase order is closed
/// </summary>
/// <param name="dictTranx">specify po_number in the dictionary</param>
/// <returns></returns>
public ServiceRequestResult IsPoClosed(Dictionary<string, string> dictTranx)
/// <summary>
/// check to see if specified pick order is closed
/// </summary>
/// <param name="dictTranx">specify order_number in the dictionary</param>
/// <returns></returns>
public ServiceRequestResult IsPickOrderClosed(Dictionary<string, string> dictTranx)
```

Get Requests

```
/// <summary>
/// The following is a list of transaction requests to retrieve a list of the subject,
matching the specified criteria
/// </summary>
/// <param name="commands">contains the list of criteria to get the items</param>
/// <param name="data">stores the data that is returned. </param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful.
/// </returns>
public ServiceRequestResult GetItem(QueryWhereCriteria[] commands, out
Dictionary<string,string>[] data)
public ServiceRequestResult GetCustomer(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data)
public ServiceRequestResult GetSupplier(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data)
public ServiceRequestResult GetManufacturer(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data)
public ServiceRequestResult GetPurchaseOrder(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data, out Dictionary<string, string>[][] details)
public ServiceRequestResult GetPickOrder(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data, out Dictionary<string, string>[][] details)
public ServiceRequestResult GetTransactions(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data)
public ServiceRequestResult GetInventoryDetailsList(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data)
public ServiceRequestResult GetSite(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data)
public ServiceRequestResult GetLocation(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data)
public ServiceRequestResult GetAddresses(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data)
public ServiceRequestResult GetItemUOMs(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data)
public ServiceRequestResult GetItemSupplier(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data)
public ServiceRequestResult GetItemLocation(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data)
public ServiceRequestResult GetSalesTax(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data)
public ServiceRequestResult GetPriceLevel(out Dictionary<string, string>[] data)
public ServiceRequestResult GetShippingMethod(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data)
public ServiceRequestResult GetPaymentMethod(QueryWhereCriteria[] commands, out
Dictionary<string, string>[] data)
```

UpdateCompanyInfo

```
/// <summary>
/// Update company information, as well as the addresses info
/// </summary>
/// <param name="info">specifies the company general information</param>
/// <param name="addresses">company address info</param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful.
/// </returns>
public ServiceRequestResult UpdateCompanyInfo(Dictionary<string,string> info,
Dictionary<string,string>[] addresses)
```

GetUserSercurityPrivileges

```
/// <summary>
/// retrieve specified user's security privileges and accessible site list
/// </summary>
/// <param name="dataPrvileges">retrieve specified user's security privileges</param>
/// <param name="dataSites">retrieve specified user's accessible site list</param>
/// <param name="sUser">specify user logon name</param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful.
/// </returns>
public ServiceRequestResult GetUserSecurityPrivileges(out Dictionary<string, string>[]
dataPrvileges, out Dictionary<string, string>[] dataSites, string sUser)
```

GetFormLabels

```
/// <summary>
/// retrieve all field labels for the specified form
/// </summary>
/// <param name="commands"></param>
/// <param name="data">list of field labels matching the specified criteria and form
name</param>
/// <param name="sForm">form name</param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful.
/// </returns>
public ServiceRequestResult GetFormLabels(QueryWhereCriteria[] commands, out
Dictionary<string,string>[] data, string sForm)
```

GetNextSerialNumber

```
/// <summary>
/// Get the next serial number for the specified item, given that this item is set to
automatically assign serial number
/// </summary>
/// <param name="data">serial number</param>
/// <param name="sItem">item number</param>
/// <returns>a ServiceRequestResult structure with
///     - a return value of whether the call is successful or not, and
///     - error details if the call is not successful. </returns>
public ServiceRequestResult GetNextSerialNumber(out Dictionary<string, string>[] data,
string sItem)
```

Delete Requests

```
/// <summary>
/// The following is a list of transaction requests to delete the specified record
/// </summary>
/// <param name="itemNumber">the number of the item to be deleted</param>
/// <param name="deleteAssociation">determines whether the user wants to delete
associated data</param>
public ServiceRequestResult DeleteItem(string itemNumber, bool deleteAssociation)
public ServiceRequestResult DeleteCustomer(string customerName)
public ServiceRequestResult DeleteSupplier(string supplierName, bool deleteAssociation)
public ServiceRequestResult DeletePurchaseOrder(string poNum)
public ServiceRequestResult DeletePickOrder(string pickOrderNum)
public ServiceRequestResult DeleteSite(string siteName)
public ServiceRequestResult DeleteLocation(string locationName, string siteName)
public ServiceRequestResult DeleteManufacturer(string manufacturerName, bool
deleteAssociation)
public ServiceRequestResult DeleteShippingMethod(string shippingMethod)
public ServiceRequestResult DeletePaymentMethod(string paymentMethod)
public ServiceRequestResult DeletePriceLevel(string sPriceLevel)
public ServiceRequestResult DeleteSalesTax(string sSalesTax)
public ServiceRequestResult DeleteItemSupplierSetting(string sItemNumber, string
sSupplier)
public ServiceRequestResult DeleteItemLocationSetting(string sItemNumber, string sSite,
string sLocation)
```

Index