ARS – eCommerce integration

TSC

Documentation

# Revisions

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Author** | **Description** |
| 1.0 | 26/05/2020 | Stefano Bertarello | Initial Draft |
| 1.2 | 03/06/2020 | Stefano Bertarello | Revision |
| 1.3 | 20/06/2020 | Stefano Bertarello | Integrated Return workflow |
| 1.4 | 25/06/2020 | Stefano Bertarello | Revision |
| 1.5 | 21/09/2020 | Simone Sambruni | Revision with JSON + API |
| 1.6 | 26/10/2020 | Stefano Bertarello / Simone Sambruni | Revision |
| 1.7 | 15/11/2020 | Stefano Bertarello | Revision after meeting |
| 1.8 | 02/12/2020 | Simone Sambruni | Revision after clarification 24/11 |
| 1.9 | 18/01/2021 | Stefano Bertarello | Revision |
| 1.10 | 29/01/2021 | Stefano Bertarello | Revision |

# Table of Contents

Revisions 2

Table of Contents 3

1 Overview 1

2 Architecture 2

2.1 eCommerce sends a Basket Request 2

2.1.1 Basket Type 2

2.1.2 Unique Basket Identifier 3

2.2 ARS POS receives basket and performs transaction 3

2.2.1 Return Transaction 4

2.3 eCommerce receives ARS Receipt 4

2.4 Rest Basket messages JSON structure 5

2.4.1.1 Note on Items 6

2.4.2 Basket Request (PUT) 6

2.4.3 Basket Polling (GET) with basket id 8

2.4.4 Basket Polling (GET) with terminal id 10

2.4.5 Basket Update (POST) 11

2.4.6 Basket Erase (DELETE) 13

3 Business Cases 14

3.1 Requests 14

3.1.1 eCommerce sends a Basket Request 14

3.1.2 ARS POS receives basket 14

3.1.3 eCommerce receives ARS Receipt 14

3.1.4 eCommerce receives Delete 14

3.2 Flows 15

3.2.1 Full Basket flow 15

4 Heartbeat and Check Status 16

4.1 Heartbeat 16

4.2 Status 16

5 List of Web Services 18

5.1 Basket WS 18

5.2 Heartbeat WS 18

5.3 Status WS 18

6 POS Configuration 19

7 Current Limitations 20

# Overview

This document describes the integration between AS@R / ARS and an eCommerce solution.

# Architecture

The following schemas represent the general architecture of the system.

**eCommerce Interface** – it is a Web-based application residing on ARS store Server (i.e. in every store that needs eCommerce integration). It is responsible for collecting requests coming from eCommerce software and delivering them to propery configured ARS POSes.

**eCommerce + Picking App** – is to be intended as a general client external to ARS environment. It can be either eCommerce solution or app that must have the ability to invoket eCommerce Interface rest WS.

## eCommerce sends a Basket Request

When an eCommerce transaction is being finalized, the Order picked using a native TSC application, a request is sent to eCommerce ARS Interface software (residing on each store NCR Server and therefore must be reacheable from network).

The **Basket Request** (after the order is picked by native TSC application) is sent to a Rest Web Service of the eCommerce ARS Interface invoking the WS with JSON body (to be detailed). The request must contain a unique identifier that will be used to perform all subsequent operations.

Immagine che contiene screenshot

Descrizione generata automaticamente

The request is stored locally for further use.

An ARS POS configured as eCommerce device is constantly polling eCommerce ARS Interface to ask for new baskets. The polling request will also contain POS identifier (terminal number). This will allow for different uses depending on POS identifier.

The polling request can be initiated also by pressing a configured button on ARS POS.

### Basket Type

Basket can be typed to handle either normal sale or return or further changes. Items in basket may contain price. If they do, the price sent in the basket will be used (either for sale or for return).

### Unique Basket Identifier

Each basket must have its unique basket identifier, but in case of return transaction, an additional field with **reference basket** can be used. This field will be printed on receipt.

## ARS POS receives basket and performs transaction

When ARS POS receives a basket from eCommerce ARS Interface, it performs the automatic sale or return of the whole transaction. The following events may occur:

* There’s an error during automatic item sales (or return) – in this case the ARS POS continues the sale (or return) of the remaining items and will cancel the transaction without warning the operator at the end of the process (a parameter can enable either message box or list of unsold items on receipt). If there is an error the **Status** field will be set to “Canceled”.
* If the customer ‘rewards card’ is not found the transaction will continue and the status will not be set as “Canceled”. The **ErrorCode** field will be set to 1001.
* The basket also contains payment – the ARS POS performs automatic payment at the end of transaction thus finalizing it completely (ARS tenders will be properly mapped to tenders found in the eCommerce basket). These payments do not need to have a matching payment on the external eCommerce + picking system. They are needed to provide different behavior to POS, thus allowing for automatic tendering or manual one. The tender to be used can be set using the field **Tender**. The content of the field will be properly mapped to ARS tenders using a map.
* The basket does not contain any payment – in this case the ARS POS will set the **Status** field will to “Canceled” and the **ErrorCode** will be set to 103. The transaction will not be finished. There will be an option for an automatic default tender.

Immagine che contiene screenshot

Descrizione generata automaticamente

User can start a normal transaction on a POS configured as being eCommerce device, but polling will be suspended until the end of the transaction. This is intended as parametric and allows for the reuse of an ARS POS in case of need. If the parameter will be disabled, the ARS POS will behave only as eCommerce terminal.

### Return Transaction

If the basket represents a Return Transaction, then POS will perform an ARS return transaction using prices sent in basket. If prices are not sent for some or all items, then prices currently present in ARS PLU file will be used **without any intervention by the operator**.

## eCommerce receives ARS Receipt

At the end of the sale or return the ARS POS sends a **Receipt Message** to eCommerce ARS Interface containing the following information:

* List of the items sold or returned on POS
* List of the items that cannot be sold or returned
* Total amount of the transaction (either positive or negative)
* Receipt (ARS POS Electronic Journal for the current transaction)

In this case too the unique identifier is sent to correctly match the pending transactions on eCommerce ARS Interface with the ones sent by POSes. Unique identifier is mandatory because it is the only way of identifying a basket in various steps of the basket handling.

In the meantime, eCommerce will be polling for the outcome of the sale (or return). When this will be made available by the eCommerce ARS Interface, it will be downloaded and use properly. The architecture is a three- element system. The one-shot handling cannot be guaranteed because the ARS POS is an entity on its own, there can be multiple reasons for which the response is not immediate (paper out, operator intervention needed and so on). Therefore, the communication between eCommerce + picking app and eCommerce ARS Interface must be done in two steps:

* eCommerce + picking app sends basket using PUT verb (GET CANNOT be used because JSON must be in the body) and receives general acknowledge response.
* eCommerce + picking app sends periodical requests (polling) to receive the receipt back. This polling request must contain the same basket id used for original PUT request.

Immagine che contiene screenshot

Descrizione generata automaticamente

eCommerce ARS Interface will send a Receipt Message (JSON) where the items not parsed by ARS POS for reasons like (item/barcode not available on POS) will be clearly identifiable.

## Rest Basket messages JSON structure

The following is the JSON structure used for communication between eCommerce Interface and its clients (either eCommerce / picking app or ARS POS).

{

"BasketId": "",

"Status": "",

"TerminalId": "",

"CustomerId": "",

"Type": "",

"Receipt": "",

"TotalAmount”:,

"EarnedLoyaltyPoints":,

"TransactionId": "",

"BarcodeId": "",

"Items": [

{

"Code":

"Price":

}

],

"SoldItems": [

{

"Code":

"Price":

}

],

"NotSoldItems": [

{

"Code":

"Price":

}

],

"OriginalBasketId": "",

"TenderType": "",

"ErrorCode":,

}

Fields description:

* **BasketId:** Unique identifier of the basket. The basketId is unique in a sense that it uniquely represents an order placed on eCommerce. There can be situations in which the same baskedId is used, fior example when a Return is issued for a BasketId. Must be provided by eCommerce / Picking application to retrieve the correct info. *BaskeId length is 20 numeric characters maximum*.
* **Status:** Indicates the processing status. It may have the following values:
  + “**Received**” – the basket has been received by eCommerce interface.
  + “**Processing**” – the basked has been retrieved by ARS POS and it is being processed.
  + “**Ready**” – the basked has been processed by ARS and sent back to eCommerce interface.
  + “**Canceled**” – the basked has been not processed by ARS and sent back to eCommerce interface.
* **TerminalId:** Identifier of the terminal ARS POS
* **CustomerId:** Identifier the Customer card number
* **Type:** Indicates the type of transaction. It may have the following values:
  + “**Sale**” – a sale must be performed on ARS POS
  + “**Return**” – a return must be performed on ARS POS
* **Receipt:** Contains the receipt data
* **TotalAmount:** Indicates the total amount of the transaction
* **EarnedLoyaltyPoints**: Points earned during transaction
* **TransactionId:** Code containing store, terminal, transaction, date and time of ARS system in the format SSSSTTTttttYYMMDDHHmmss with:
  + **SSSS** – store
  + **TTT** – terminal
  + **tttt** – transaction
  + **YYMMDD** – date
  + **HHmmss** – hour
* **BarcodeId:** Code containing store, terminal and transaction of ARS system in the same format of printed barcode (with header and check digit)
* **Items:** Contains the set of products. Each Item may have the following fields:
  + **Price** – Total Price of item sale (Qty \* UnitPrice)
  + **Qty** – Quantity sold or returned
  + **UnitPrice** – Unit price of an item
  + **Code** – SKU code of the Item
  + **Barcode** – Barcode of the Item
* **SoldItems:** Contains the set of products sold
* **NotSoldItems:** Contains the set of products not sold
* **OriginalBasketId**: This will contain the Id of the original basket in case of **Return** operation
* **TenderType**: Type of tender to be used for automatic tenderization
* **ErrorCode**: Possible error code for ARS processing:
  + **101** –PLU problem (either PLU not found or department or wrong parameterization)
  + **102** – Printer problem
  + **103** – Missing tender
  + **0** – no problem
  + **1001** – Customer reward card not found (this is a warning and not an error).

#### Note on Items

For each item in the basket:

* If the **Qty** field is supplied, it will be used as quantity for the item sale. If decimal quantity, it will be ignored.
* If **UnitPrice** field and **Qty** field are supplied they will be used to determine final price.
* If **Barcode** with embedded price is supplied, embedded price must have same value as **Price** field. **If, for some reason, the price embedded is not corresponding to the content of Price field, the Price field will be the one to be trusted**.
* If **Item** is a weight item (either price embedded or weight embedded label), **UnitPrice** will be used to determine missing info (either weight or final price).

### Basket Request (PUT)

This request is used to create a new Basket entry in the eCommerce interface. The required fields are shown. The other fields (previously described) are not required.

Type is either **Sale** or **Return**.

If the same BasketId and same Type is used in a PUT request, an error is returned.

Case Type **Sale:**

**Json Input:**

{

"BasketId": "12345678901234567890",

"Type": "Sale",

"CustomerId": "4444000000000",

"Items": [

{

"Code": 3005121630,

"Price": 0.750

}

],

"TenderType": "Online",

}

The response sent by eCommerce interface is the following. Status is set to **Received** stating that the basket hase been received by eCommerce interface. Fields not yet filled (this will be done by ARS POS) are sent back, but empty.

**Json Output:**

{

"BasketID": "12345678901234567890",

"Status": "Received",

"Type": "Sale",

"CustomerId": "4444000000000",

"TerminalId": "",

"Receipt": "",

"TotalAmount": 0,

"EarnedLoyaltyPoints": 0,

"TransactionId": "",

"BarcodeId": "",

"Items": [

{

"Code": 3005121630,

"Price": 0.750

}

],

"SoldItems": [],

"NotSoldItems": [],

"TenderType": "Online",

"ErrorCode": 0

}

Case Type **Return:**

**Json Input:**

{

"BasketId": "12345678901234567890",

"Type": "Return",

"CustomerId": "4444000000000",

"Items": [

{

"Code": 3005121630,

"Price": 0.750

}

],

"TenderType": "Online",

}

The response sent by eCommerce interface is the following.

**Json Output:**

{

"BasketID": "12345678901234567890",

"Status": "Return",

"Type": "Sale",

"CustomerId": "4444000000000",

"TerminalId": "",

"Receipt": "",

"TotalAmount": 0,

"EarnedLoyaltyPoints": 0,

"TransactionId": "",

"BarcodeId": "",

"Items": [

{

"Code": 3005121630,

"Price": 0.750

}

],

"SoldItems": [],

"NotSoldItems": [],

"TenderType": "Online",

"ErrorCode": 0

}

### Basket Polling (GET) with basket id

This request is sent by eCommerce / Picking to check if basket is ready. There is no Json (being the request a GET). The BasketId field is supplied in query string:

https://<hostname>:<port>/api/Basket?BasketId=*12345678901234567890*

There are 3 possible situations at this point:

1. Basket has not yet been processed by ARS POS (**Status** = “Received”) or is being processed (**Status** = “Processing”.
2. Basket has been successfully processed by ARS POS
3. Basket has been processed by ARS POS, but with errors

**Json Output (Case 1 – Basket not yet processed):**

{

"BasketId": "12345678901234567890",

"Status": "Processing",

"TerminalId": "",

"Type": "Sale",

"CustomerId": "4444000000000",

"Receipt": "",

"TotalAmount": 0,

"EarnedLoyaltyPoints": 0,

"TransactionId": "",

"BarcodeId": "",

"Items": [

{

"Code": 3005121630,

"Price": 0.750

}

],

"SoldItems": [ ],

"NotSoldItems": [ ],

"TenderType": "Online",

"ErrorCode": 0

}

In this case there is no **Receipt** yet, **TotalAmount** and **EarnedLoyaltyPoints** are 0 and **SoldItems** and **NotSoldItems** are empty lists. **TransactionId** and **TerminalId** are empty too.

The only meaningful field is **Status** that, being set to either **Received** or **Processing**, states that Basket is not ready yet. eCommerce must retry later.

**Json Output (Case 2 – Basket successfully processed):**

{

"BasketId": "12345678901234567890",

"Status": "Ready",

"TerminalId": "001",

"Type": "Sale",

"CustomerId": "4444000000000",

"Receipt": " 3005121630 /r/n بيبسي ستاربكس /r/n Pepsi Strbck Frpcn 0.750 /r/n Queue 1 /r/n ITEMS 1 /r/n الت /r/n >T O T A L 0.750 /r/n نقدأ /r/n Cash KWD 20.000 /r/n الباقي /r/n CHANGE KWD -19.250 /r/n\*0056 0045/008/102 26.10.20 10:35 AC-00 ",

"TotalAmount": 0.750,

"EarnedLoyaltyPoints": 10,

"TransactionId": "12340080045201013120000"

"BarcodeId": "9123400800452",

"Items": [

{

"Code": 3005121630,

"Price": 0.750

}

],

"SoldItems": [

{

"Code": 3005121630,

"Price": 0.750

}

],

"NotSoldItems": [],

"TenderType": "Online",

"ErrorCode": 0

}

In this case **Receipt** has been filled, **TotalAmount** and **EarnedLoyaltyPoints** are set to the actual values and **SoldItems** is filled with items that have been sold while **NotSoldItems** is empty because no error in selling has occurred. **TerminalId** contains the terminal on which the transaction occurred.

The **Status** field is set to **Ready**, stating that Basket is ready. eCommerce can process it.

**Json Output (Case 3 – Basket processed with errors):**

{

"BasketId": "12345678901234567890",

"Status": "Canceled",

"TerminalId": "001",

"Type": "Sale",

"CustomerId": "4444000000000",

"Receipt": " 3005121630 /r/n بيبسي ستاربكس /r/n Pepsi Strbck Frpcn 0.750 /r/n Queue 1 /r/n ITEMS 1 /r/n الت /r/n >T O T A L 0.750 /r/n نقدأ /r/n Cash KWD 20.000 /r/n الباقي /r/n CHANGE KWD -19.250 /r/n\*0056 0045/008/102 26.10.20 10:35 AC-00 ",

"TotalAmount": 0,

"EarnedLoyaltyPoints": 0,

"TransactionId": "12340080045201013120000"

"BarcodeId": "9123400800452",

"Items": [

{

"Code": 3005121630,

"Price": 0.750

}

],

"SoldItems": [],

"NotSoldItems": [

{

"Code": 3005121630,

"Price": 0.750

}

],

"TenderType": "Online",

"ErrorCode": 101

}

**NOTE:**

If a terminal is down and the available terminals for eCommerce are more than one, the remaining will take care of baskets.

In this case **Receipt** is not meaningful, **TotalAmount** and **EarnedLoyaltyPoints** are set to 0 and **SoldItems** is filled with items that have been sold while **NotSoldItems** may contain items that couldn’t be sold for some reason. **TerminalId** contains the terminal on which the transaction occurred.

The **Status** field is set to **Canceled**, stating that Basket has been canceled. eCommerce is informed. Someone must take action.

### Basket Polling (GET) with terminal id

This method will **not** be invoked by eCommerce, but only by ARS POS to retrieve the first available basket. It is detailed here for completeness. The TerminalId field is supplied in query string:

https://<hostname>:<port>/api/Basket?TerminalId=*001*

**Json Output:**

{

"BasketID": "12345678901234567890",

"Status": "Received",

"Type": "Sale",

"CustomerId": "4444000000000",

"TerminalId": "",

"Receipt": "",

"TotalAmount": 0,

"EarnedLoyaltyPoints": 0,

"TransactionId": "",

"BarcodeId": "",

"Items": [

{

"Code": 3005121630,

"Price": 0.750

}

],

"SoldItems": [ ],

"NotSoldItems": [ ],

"TenderType": "Online",

"ErrorCode": 0

}

### Basket Update (POST)

This method is invoked generally by ARS POS but can be invoked also by eCommerce / Picking if an update must be performed on an existing basket. It is detailed here for completeness. The BasketId doesn’t represents in an unique way an entity on ARS + eCommerce Interface, because for example the same BasketId is used for both sale and Return. When a change must be applied on a BasketId also the Type must be supplied. I.e. eCommerce may change the content of the basket for a return and not for the corresponding sale.

There are 3 possible situations at this point:

1. Basket has been successfully processed by ARS POS
2. Basket has been processed by ARS POS, but with errors
3. Basket is updated by eCommerce

Json is echoed back to client that has sent request (outpus is same as input).

**Json Input (Case 1 – ARS POS sends ready basket):**

{

"BasketId": "12345678901234567890",

"Status": "Ready",

"TerminalId": "001",

"Type": "Sale",

"CustomerId": "4444000000000",

"Receipt": " 3005121630 /r/n بيبسي ستاربكس /r/n Pepsi Strbck Frpcn 0.750 /r/n Queue 1 /r/n ITEMS 1 /r/n الت /r/n >T O T A L 0.750 /r/n نقدأ /r/n Cash KWD 20.000 /r/n الباقي /r/n CHANGE KWD -19.250 /r/n\*0056 0045/008/102 26.10.20 10:35 AC-00 ",

"TotalAmount": 0.750,

"EarnedLoyaltyPoints": 10,

"TransactionId": "12340080045201013120000"

"BarcodeId": "9123400800452",

"Items": [

{

"Code": 3005121630,

"Price": 0.750

}

],

"SoldItems": [

{

"Code": 3005121630,

"Price": 0.750

}

],

"NotSoldItems": [],

"TenderType": "Online",

"ErrorCode": 0

}

**Json Input (Case 2 – ARS POS sends canceled basket):**

{

"BasketId": "12345678901234567890",

"Status": "Canceled",

"TerminalId": "001",

"Type": "Sale",

"CustomerId": "4444000000000",

"Receipt": " 3005121630 /r/n بيبسي ستاربكس /r/n Pepsi Strbck Frpcn 0.750 /r/n Queue 1 /r/n ITEMS 1 /r/n الت /r/n >T O T A L 0.750 /r/n نقدأ /r/n Cash KWD 20.000 /r/n الباقي /r/n CHANGE KWD -19.250 /r/n\*0056 0045/008/102 26.10.20 10:35 AC-00 ",

"TotalAmount": 0,

"EarnedLoyaltyPoints": 0,

"TransactionId": "12340080045201013120000"

"BarcodeId": "9123400800452",

"Items": [

{

"Code": 3005121630,

"Price": 0.750

}

],

"SoldItems": [],

"NotSoldItems": [

{

"Code": 3005121630,

"Price": 0.750

}

],

"TenderType": "Online",

"ErrorCode": 101

}

**Json Input (Case 3 – eCommerce wants to update an existing basket):**

{

"BasketId": "12345678901234567890",

"Type": "Sale",

"CustomerId": "4444000000000",

"Items": [

{

"Code": 3005121630,

"Price": 0.750

}

]

}

**NOTE:**

This method is just a possibility. There’s no need to call it if the eCommerce does not require it.

### Basket Erase (DELETE)

This method is invoked generally by eCommerce interface. There is no Json (being the request a DELETE).

The BasketId and Type fields are supplied in query string:

https://<hostname>:<port>/api/Basket?BasketId=*12345678901234567890&Type=Return*

Il BasketId is set to “ALL”, all the baskets with status Received will be deleted.

The response to all DELETE commands successfully processed will be a JSON corresponding to the deleted basket. If “ALL” is used the JSON will be an array of deleted baskets.

Possible cases:

* A delete is sent with BasketId=**12345678901234567890** and Type=**Sale**, eCommerce Interface has the BaskedId in Received status. The basket will be deleted.
* A delete is sent with BasketId=**12345678901234567890** and Type=**Return**, eCommerce Interface has both the BaskedId with Type=**Sale** and Type=**Return** in **Received** status. The return basket will be deleted, the sale one will be processed.
* A delete is sent with BasketId=**12345678901234567890** and Type=**Return**, eCommerce Interface has both the BaskedId with Type=**Sale** and Type=**Return** in **Ready** status. No basket will be deleted. Error 400 will be sent.

# Business Cases

Following the description of business cases.

## Requests

### eCommerce sends a Basket Request

eCommerce creates a new basket and sends a PUT message [Basket Request (PUT)](#_Basket_Request_(PUT)) filling in the fields like described in previous section.

The eCommerce Interface will perform the following steps:

* Check the JSON format correctness
* Check if the basket is already present into database and if so, it sends a HTTP Response **400 [Bad request]**
* Fills the field Status with value *“****Received****”* only in the case of Type **Sale**(the basket has been received by eCommerce interface)
* Send an HTTP Response **200 [Ok]** to eCommerce.

### ARS POS receives basket

ARS POS request basket availability to eCommerce Interface through a GET message [Basket Polling (GET) with basket id](#_Basket_Polling_(GET)) provide the BasketId.

The eCommerce Interface will perform the following steps:

* Check if a basketis available into database.
* If a basket is present, then updates the field Status with value *“Processing”* (the basked has been retrieved by ARS POS and it is being processed) and sends a HTTP Response **200 [Ok]** to ARS POS.
* Otherwise, if a basket is not present, then it sends a HTTP Response **404 [Not found**] to ARS POS.

### eCommerce receives ARS Receipt

When the sale transaction ends the ARS POS send a POST message [Basket Update (POST)](#_Basket_Update_(POST)) filling in the fields like described in previous section.

The eCommerce Interface will perform the following steps:

* Check the Status value.
* If value is *“Ready”* or *“Canceled”* then update the record into database, then send an HTTP Response **200 [Ok]**.
* If value is “Processing” then send a HTTP Response **400 [Bad request].**

### eCommerce receives Delete

When the eCommerce sends a DELETE message [Basket Erase (DELETE)](#_Basket_Erase_(DELETE)) like described in previous section.

The eCommerce Interface will perform the following steps:

* Check if the BasketId and Type exist into database.
* In case of yes, then send an HTTP Response **200 [Ok]** to eCommerce.
* In case of no, then send an HTTP Response **400 [Bad request]** to eCommerce.

## Flows

In this section the flow for invoking eCommerce interface WS is described from the eCommerce point of view.

### Full Basket flow

* eCommerce generates basket and invokes PUT method.
* if no answer is returning (HTTP timeout), the eCommerce interface is not active at the given address and port.
* if answer is returned with error 400, the basket is not accepted (existing one).
* if answer is returned with status 200 and a JSON with status field set to **Received** the eCommerce must assume that basket is being processed and must start a polling of GET method invocation for a parametric timeout (application timeout not HTTP) or until a JSON with status field set to **Ready** is returned.
* if timeout occurs, i.e. eCommerce performs a number X of GET calls always getting a JSON with status field set to **Received** then there is no ARS POS active to process requests from eCommerce. Human check must be performed.
* if timeout occurs, i.e. eCommerce performs a number X of GET calls always getting a JSON with status field set to **Processing** then the POS has received basket but for some reason has not replied with basket (for example paper end on printer). Human check must be performed.
* If JSON with status field set to **Ready** is received, basket has been successfully printed by ARS POS.
* If JSON with status field set to **Canceled** is received, basket has been processed with errors. If NotSoldItems has some entry, then there has been a problem selling those items (either PLU not found or PLU-depending errors). Also, **ErrorCode** field is set to 101. eCommerce must fix PLU problems and either generate new basket or modify the existing one invoking POST.
* If JSON with status field set to **Canceled** is received, basket has been processed with errors. If NotSoldItems hasn’t any entry, then there has been a problem not related to PLUs. If **ErrorCode** field is set to 102 there has been a printer problem. Human must fix the problem and eCommerce must either generate new basket or modify the existing one invoking POST.

**NOTE:**

Every X second eCommerce may icall the Status web service to know the general status of the architecture. If for example there is any active terminal for basket processing. If the Status WS replies with no available terminals or with all the terminals in error state, that will mean that basket will not be processed and will be enqueued to eCommerce Interface.

# Heartbeat and Check Status

The eCommerce Interface will provide a Heartbeat message and a Status message to check the availability of ARS POS and to know possible issues (e.g. The receipt printer is not working).

## Heartbeat

Every X second all ARS POS configured as eCommerce POSes will send a POST message to Heartbeat WS in eCommerce interface.

https://<hostname>:<port>/api/Heartbeat

Fields description:

* **TerminalId:** Identifier of the terminal ARS POS
* **ErrorCode**: Possible error code for ARS processing:
  + **102** – Printer problem
  + **0** – no problem

**Json Input:**

{

"TerminalId": "001",

"ErrorCode": 0

}

eCommerce sends an HTTP Response **200 [Ok]** to ARS POS.

eCommerce will keep a list of available terminals.

## Status

When needed eCommerce may send a Get message to Status WS in eCommerce interface to retrieve status of various POSes (as it has been set by Heartbeat messages) and active BasketID items; no JSON input required.

https://<hostname>:<port>/api/Status

Fields description for each terminal:

* **TerminalId:** Identifier of the terminal ARS POS
* **ErrorCode**: Possible error code for ARS processing:
  + **102** – Printer problem
  + **0** – no problem
* **ActiveBaskets: Basket** JSON structure

**Json Output:**

{

"Terminals": [

{

"TerminalId": "001",

"ErrorCode": 0

},

{

"TerminalId": "002",

"ErrorCode": 102

},

],

“ActiveBaskets”: [

{

"BasketId": "",

"Status": "",

"TerminalId": "",

"CustomerId": "",

"Type": "",

"Receipt": "",

"TotalAmount”:,

"EarnedLoyaltyPoints":,

"TransactionId": "",

"BarcodeId": "",

"Items": [

{

"Code":

"Price":

}

],

"SoldItems": [

{

"Code":

"Price":

}

],

"NotSoldItems": [

{

"Code":

"Price":

}

],

"OriginalBasketId": "",

"TenderType": "",

"ErrorCode":,

}

]

}

# List of Web Services

The following is the initial list of available Web Services provided by eCommerce ARS Interface:

## Basket WS

* Basket Request (PUT) – Will insert a new basket in eCommerce ARS Interface availability. This verb will be used initially by the eCommerce + picking app to send basket
* Basket Polling (GET) – Will allow ARS POS or eCommerce to ask for the availability of:
  + a specific basket (providing the unique identifier in the request). This one is used by the eCommerce + picking app to retrieve basket information with receipt
  + whatever basket available for specific terminal (providing terminal identifier in the request). This one is used by ARS POS to search for new baskets to be processed
* Basket Update (POST) – Will allow ARS POS to update basket status with result of the transaction. This one will be used by ARS POS when the transaction has been generated and the basket needs to be updated on eCommerce Interface.

A basket status, date and unique identifier will allow specific request from clients (e.g. retrieve basket if completed, retrieve all completed basket in a specific day, …).

## Heartbeat WS

* Status Update (POST) – Will allow ARS POS to update eCommerce interface on status. This web service is used only by ARS POS.

## Status WS

* Status Polling (GET) – Will allow eCommerce to know the status of various terminals; JSON input not required.

# POS Configuration

* eComerce POS – a parameter will state if POS is devoted to eCommerce processing. Only POSes thus configured will be known by eCommerce interface and will be processing orders (baskets)
* Printed Copies – a configuration with number of receipts and additional line of text to be printed at the end of each receipt. A typical configuration is:
  + 4 copies
  + Copy 1 and 2 with “CUSTOMER COPY” priunted at the end
  + Copy 3 with “DRIVER COPY” printed at the end
  + Copy 4 with “STORE COPY” printed at the end

# Current Limitations

The current solution **does not include** any web page to operate manually on baskets or see the list of pending baskets.