|  |
| --- |
| **Design Document KB\_ME\_01** |
| **Label Printing** |
| **Knorr Bremse**  Flassak, Martin <Martin.Flassak@knorr-bremse.com> |
| Sebastian Erler (IoT Senior Consultant)  serler@ptc.com |

**Document Properties**

|  |  |
| --- | --- |
| File Name | Status |
| KB\_ME\_01\_Solution\_Design.docx | Draft |

**Change History**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Description |
| 07.05.2018 | Sebastian Erler | 0.1 | Initial Draft |
| 28.11.2018 | Sebastian Erler | 0.6 | Updating Installation and Usage Instructions |
| 21.06.2019 | Sebastian Erler-Yates | 1.0.11 | Version according to update 1.0.11 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**PTC Product Version**

|  |  |
| --- | --- |
| PTC Product | Version |
| Thingworx | 8.2 |
|  |  |
|  |  |

**3rd Party Products**

|  |  |
| --- | --- |
| PTC Product | Version |
| ERP S400 |  |
| Testbench… |  |
| Printing… |  |
| Zebra… |  |

**Approvers**

|  |  |  |
| --- | --- | --- |
| Approver | Approval Date | Comments |
|  |  |  |
|  |  |  |

Content

[Content 4](#_Toc534641565)

[Introduction 6](#_Toc534641566)

[Purpose 6](#_Toc534641567)

[Scope 6](#_Toc534641568)

[Document Overview 6](#_Toc534641569)

[Delivery Approach 8](#_Toc534641570)

[Project Scope 9](#_Toc534641571)

[1.1 Use Case Design /User Story Design 10](#_Toc534641572)

[1.2 Existing KB Thingworx architecture Modell 10](#_Toc534641573)

[1.2.1 Server setup 10](#_Toc534641574)

[1.2.2 Thingworx Basic Setup 10](#_Toc534641575)

[1.3 ERP Connection 12](#_Toc534641576)

[1.3.1 Overview 12](#_Toc534641577)

[1.3.2 Illustration 13](#_Toc534641578)

[1.3.3 Use Cases 13](#_Toc534641579)

[1.3.4 Implementation Details 15](#_Toc534641580)

[*1.3.4.1* *ERP Connector – TWX implementation view* 19](#_Toc534641581)

[1.4 Testbench Connection 20](#_Toc534641582)

[1.4.1 Overview 20](#_Toc534641583)

[1.4.2 Illustration 21](#_Toc534641584)

[1.4.3 Use Cases 21](#_Toc534641585)

[1.4.4 Implementation Details 23](#_Toc534641586)

[*1.4.4.1* *Test Bench point of view* 23](#_Toc534641587)

[*1.4.4.2* *Thingworx internal point of view* 27](#_Toc534641588)

[*1.4.4.3* *Testbench Connection – TWX implementation view* 30](#_Toc534641589)

[1.5 Conventional Printing 30](#_Toc534641590)

[1.5.1 Overview 30](#_Toc534641591)

[1.5.2 Illustration 31](#_Toc534641592)

[1.5.3 Use Cases 32](#_Toc534641593)

[1.5.4 Implementation Details 33](#_Toc534641594)

[*1.5.4.1* *Conventional Printing – TWX implementation view* 33](#_Toc534641595)

[1.6 Zebra Printing 34](#_Toc534641596)

[1.6.1 Introduction: Current Zebra Printing Process 34](#_Toc534641597)

[1.6.2 Overview 35](#_Toc534641598)

[1.6.3 Illustration 36](#_Toc534641599)

[1.6.4 Use Cases 36](#_Toc534641600)

[1.6.5 Implementation Details 37](#_Toc534641601)

[*1.6.5.1* *Zebra printing – TWX implementation view* 37](#_Toc534641602)

[Installation Instructions 37](#_Toc534641603)

[1.7 Package 37](#_Toc534641605)

[1.8 Installation procedure 38](#_Toc534641606)

[Third Party Components 38](#_Toc534641607)

[1.9 PDFtoPrinter 38](#_Toc534641609)

[1.10 The Apache™ FOP Project 39](#_Toc534641610)

**Overview**

This document contains the design documentation for **the Labeling Process of KB Microelletrica**. Development of the component is contingent upon the approval of this document.

**The following Solution Design is based on a PoC conducted by KB Rail at the Microelettrica Manufacturing Plant. Target of the PoC is to support the Labelling Process, which is currently a manual process. At the end of the PoC labels and protocols for parts should be automatically printed based on data coming from a testbench and an ERP System.**

**Introduction**

**Purpose**

**Instructions**

Define the role of this document in the overall project documentation; Describe the structure of this document in brief; Identify the specific audiences for this document and indicate how they are expected to use it.

This document describes the low-level design for **the Labeling Process of KB Microelletrica** Development of the component is contingent upon the approval of this document.

**Scope**

**Instructions**

Define the scope of this document, with a specific caveat that anything that is not specifically defined as being within the scope is out of the scope.

The scope of this document is limited to low-level design of the component.

Use cases, requirements, process descriptions, project plans, and other documents listed under internal and external references may be referenced, paraphrased, or interpreted to help develop, explain, or evaluate the domain model; however, this document is not the authoritative reference for such information. Accordingly, creating or modifying use cases, requirements, process descriptions, etc. are out of the scope of this document.

**Document Overview**

**Instructions**

Explain how the document is organized and what it contains.

**Section 2:**

* Explains the Use Case in Detail and breaks it down into different smaller pieces.
* Contains a list of different modules (in an ideal world, that would be user stories, but since it is an automated scenario, it will be broken in different modules)

**Section 3:**

* Contains Use Cases, which could be transformed into test cases by the customer
* The transformation from Use Cases into Test Cases and conducting the Tests is not part of the PTC responsibility within the PoC

**Section 4:**

* Contains a list of third party Software components
* Contains all necessary license agreements for third party software

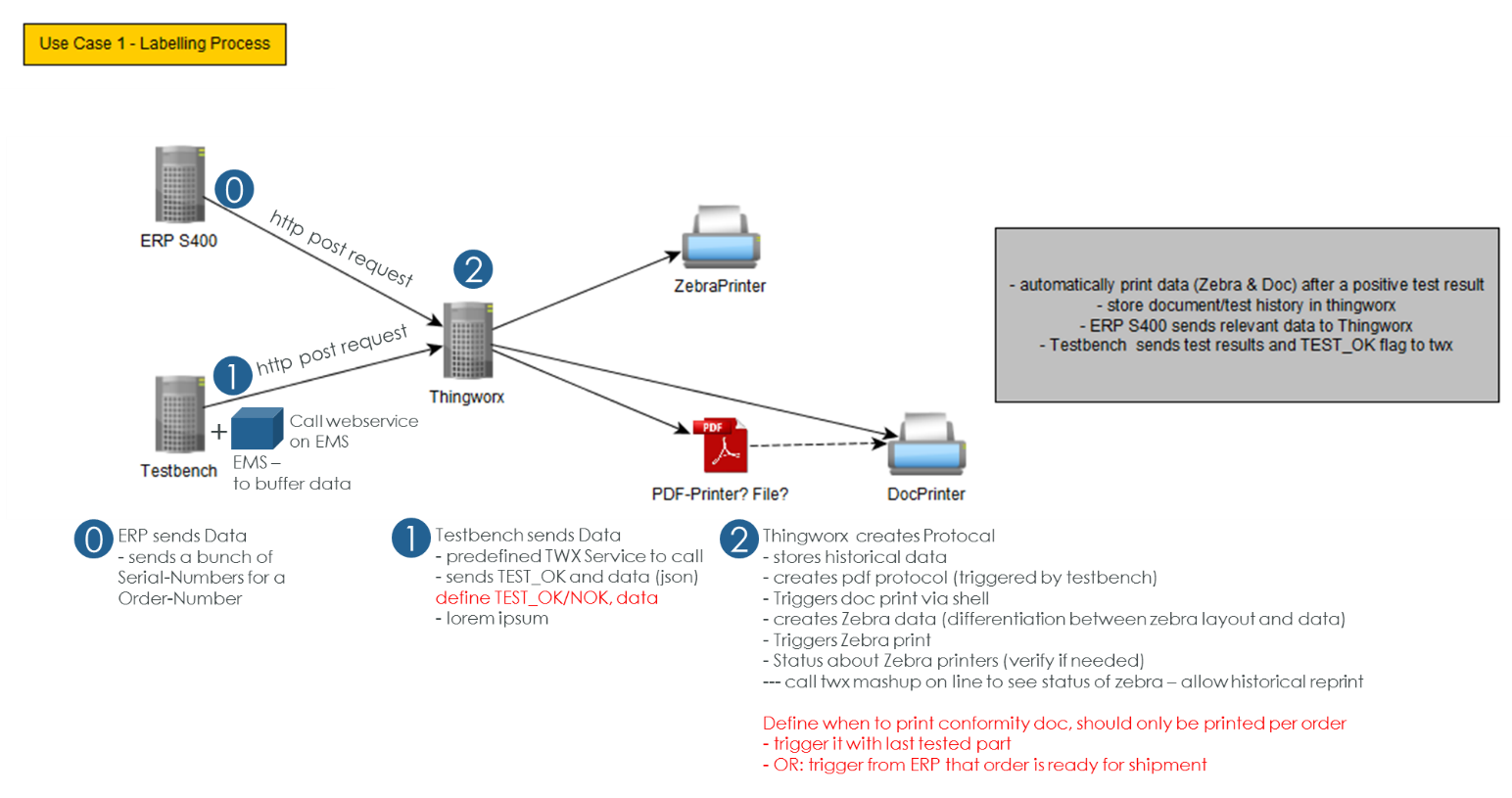
**Delivery Approach**

**Instructions**

Explain with which methodology the project target will be achieved.

The full functionality will be developed based on an waterfall approach having conducted a Kick-Off Workshop to gather the requirements. The Kick-Off Workshop took place on 24.4.2018 in Milano at Microelettrica.

The implementation of the PoC will be broken down in single modules, described in the document. Each module will be delivered with test data and conducted basic implementation tests.



**Project Scope**

**Instructions**

Explain the Use Case in Detail and write down a break down of the full functionality into different user stories, or at least modules.

As Label-Responsible, my job today is to prepare early enough the Zebra-Labels for the products, which are created on the production line. I need to have the labels ready for the workers, whenever a label is necessary. Labels are always related to a specific Order number, so the first thing I need to do is, looking in ERP S400 for an order number and extract all serial numbers for the order. With this serial number, I can start the label printing, which I provide afterwards to the worker. When a product is tested successfully the worker places the next available label on the product itself. For each successfully tested product I’m also responsible to print out the document and place it next to the product. After a complete order is processed, I also need to print out the order of conformity.

Target of the PoC here is to automate the process.

In the near-term future, the ERP System S400 should automatically inform Thingworx about order number and their belonging serial numbers. The testbench will be connected to Thingworx as well, and once there is a “successful” test signal from the testbench to Thingworx, the next available serial number for the current order should be applied to the tested good. This ultimately triggers the whole printing process, consisting of Zebra-Label printing, protocol creation and printing.

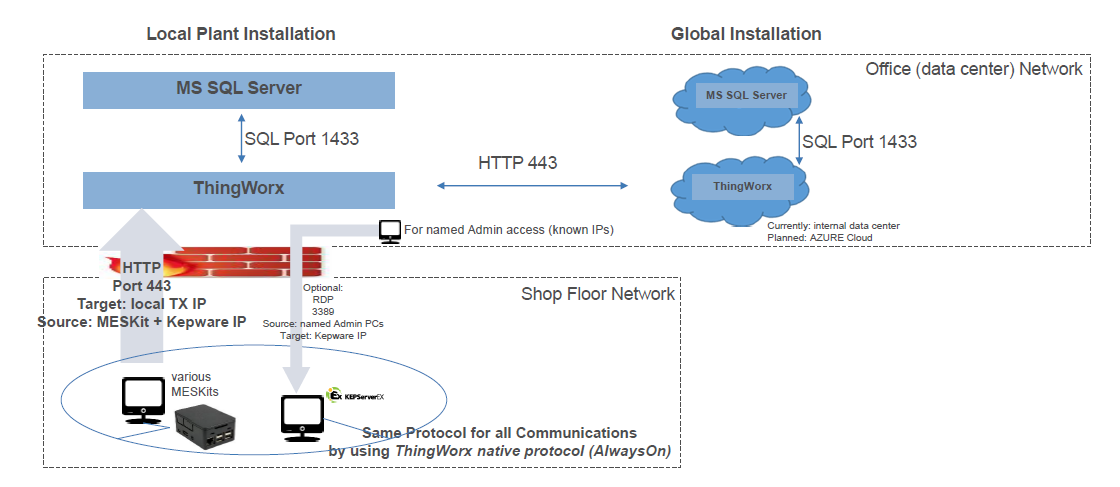
Once the last serial number is assigned, the order of conformity should be printed as well.

To get the tasks better structured the above described process will be broken down in different moduls. Each module is descriped in detail in the next chapter.

* Module 1: ERP Connection  
  Order Number and Serial Number collection from ERP S400
* Module 2: Testbench Connection  
  Testbench sending data to Thingworx and creation of Protocol in XML
* Module 3: Conventional Printing  
  Printing of Protocol, including conversion of xml into pdf and conventional printing
* Module 4: Zebra Printing  
  Printing of Zebra-Label using a Zebra Printer
  1. Use Case Design /User Story Design
  2. Existing KB Thingworx architecture Modell
     1. *Server setup*

At KB Thingworx is implemented on global level, where e.g. all KPI calculations are done and as well on local plant level to have the possibility to integrate Thingworx in production processes. For failure safety e.g. in scope of network failures it’s needed to have the local system as near as possible to the shop floor.

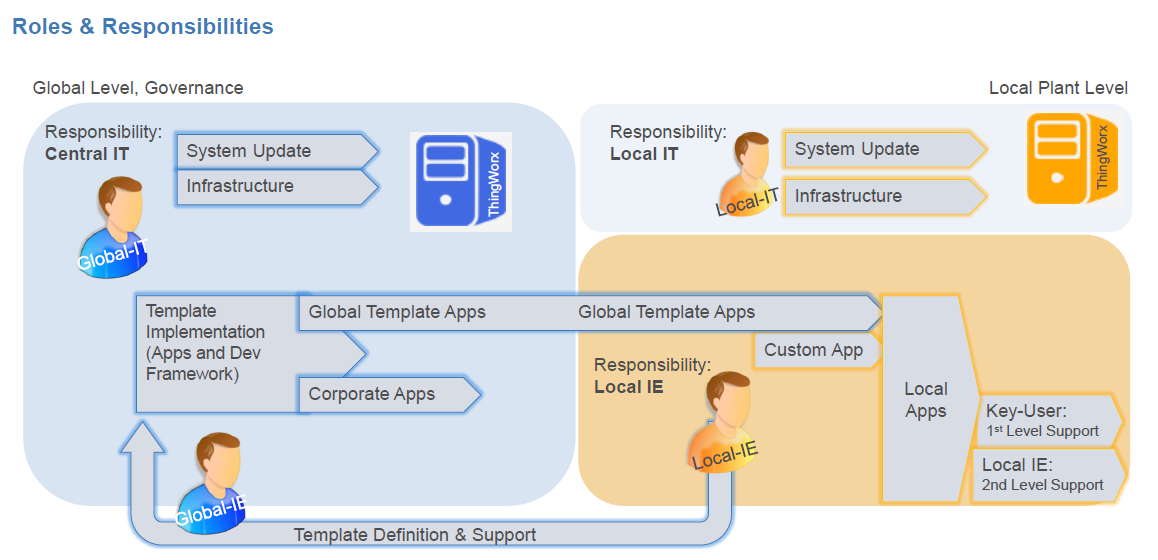
Global system is designed as a 3 staging system (development-, test- and productive system), local system as 2 staging system (test- and productive system).



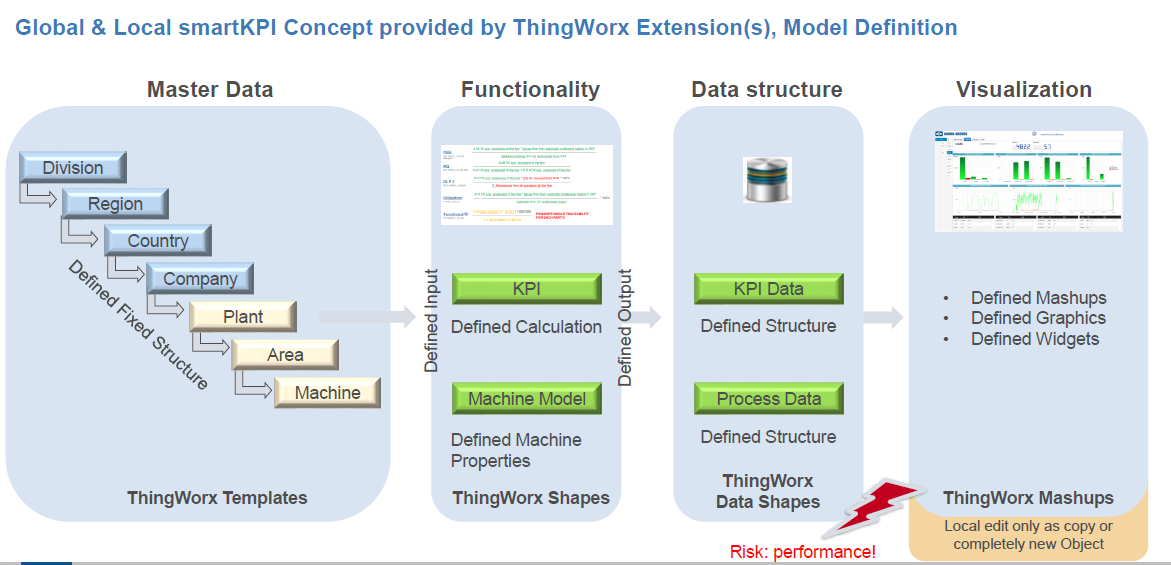
Production equipment is fully separated in an own V-LAN (Shop-floor Network). Communication to thingworx must be implemented as Webservice or Thingworx-AlwaysOn protocol. Because source, target (Thingworx) and port (HTTPS: 443) is known firewall openings can be reduced to an absolute minimum.

* + 1. *Thingworx Basic Setup*

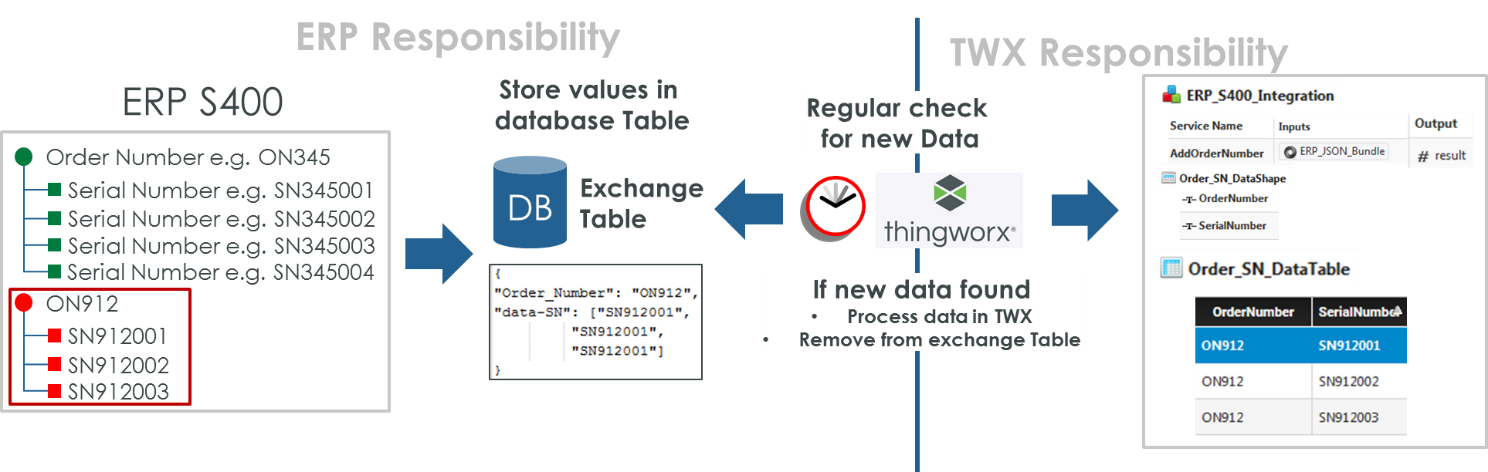
At KB basic Thingworx architecture is defined by global IT. Objects (Things, Templates…) must fit to it and can only be created by global admins. With this common architecture all data content is defined. Mashups (visualizations) can be also created by local admins or local developers. Responsibility to maintain the system especially in times of production lies on local IT. More details are given by the definition of the operation model.



As mentioned before basic Thingworx architecture is defined in a fixed structure from division level down to machines and stations of a machine or line. Special equipment (e.g. test benches) are handled by defined Templates, also functionality and data structure is defined. Because of this later on described Properties and/or Services ***must*** be used! Additional needed functionality must be approved by DM Team (I/APP1.3).



* 1. ERP Connection
     1. Overview
* The ERP System running on IBM AS/400 should store the relevant data (here mentioned for now order number and serial numbers) in 2 new database tables. Thingworx will read in a regular time interval (every 15-30 min) the data from these tables (via JDBC) and process it in Thingworx.  
  It is ERP responsibility to use a database type where JDBC driver for Thingworx is available!
* **Table 1: order number exchange table**   
  Contains information about order numbers and meta data information. **The absolute minimum as information within the table is an order number, a timestamp, Drawing Number and a Result-Identifier.** More attributes could be added, which are related to one single order number. (Please see concrete attribute list in implementation details)
* **Table 2: serial number exchange table**  
  Contains information about order number related serial numbers. **The absolute minimum as information within the table is an order number, a serial number, a timestamp and a Result-Identifier.** More attributes could be added, which are related to one single serial number. (Please see concrete attribute list in implementation details). Only Serial Numbers with a related Order Number entry in Table 1 will be processed! Typically, this is secured by using a Foreign Key relationship to Table 1
* Each time data is read from the tables information will be stored in internal order table structure by using given Services**.**
* The same order number and/or Serial Numbers can be send multiple times. If new data is available in one of the 2 tables, it will either overwrite existing information or add an entry in the data table. For **serial number exchange table** the Serial number will serve as a primary, for **order number exchange data table** the order number will serve as a primary key.
* As long the order is not in production (first test result stored) the information in the Thingworx internal data table are always updated. As soon production is started data will not be updated anymore! See related Result Identifier! Later on added Serial numbers are transferred (new ones added but existing ones not updated or deleted!)
* It is the responsibility of the ERP system to write data into the exchange database table for Thingworx. The ERP must just ensure availability of data at the defined data location at the right time. Furthermore, it is then the responsibility of Thingworx to regularly read the data out of the exchange tables. No data is removed automatically in TWX or in exchange data tables, error handling will be provided via Result-Identifier. Data which are processed (Result-Identifier != null) can be deleted by ERP system.
* It’s essential to have an exchange data table in between because of the following reasons:
  + Clear defined responsibility of the data
  + Only approved/released data is pushed to Thingworx
  + Historical data are not affected
  + Thingworx can provide return values about the process result
    1. Illustration



* + 1. Use Cases

|  |  |
| --- | --- |
| Use Case |  |
| UC\_M01\_01 | New Data is provided in exchange database Tables by ERP400  Expected result:   * Added data in database table with result identifier = “null” * Thingworx is not involved in this use case * Triggering event for ERP 4000 need to be defined, only approved/released orders should be transferred to exchange table |
| UC\_M01\_02 | Thingworx collects new data from “**serial number** exchange table” with result identifier = “null”, the **serial number is nonexistent** in the datable yet  Expected result:   * After a regular timeframe the exchange database table column for the **result-identifier is filled** with related return code, so not null any more. * **The data was integrated** into the twx data table * result identifier = 10 |
| UC\_M01\_03 | Thingworx collects new data from “**order number** exchange table” with result identifier = “null”, the **order number is nonexistent** in the datable yet  Expected result:   * After a regular timeframe the exchange database table column for the **result-identifier is filled** with related return code, so not null any more. * **The data was integrated** into the twx data table * result identifier = 10 |
| UC\_M01\_04 | Thingworx collects new data from “**serial number** exchange table” with result identifier = “null”, the **serial number is existent** in the datable yet, **production not yet started**  Expected result:   * After a regular timeframe the exchange database table column for the **result-identifier is filled** with related return code, so not null any more. * **The data was updated** in the twx data table * result identifier = 10 |
| UC\_M01\_05 | Thingworx collects new data from “**order number** exchange table” with result identifier = “null”, the **order number is existent** in the datable yet, **production not yet started**  Expected result:   * After a regular timeframe the exchange database table column for the **result-identifier is filled** with related return code, so not null any more. * **The data was updated** in the twx data table * result identifier = 10 |
| The UseCases UC\_M01\_06 and UC\_M01\_07, will be implemented with a future version of the SolutionPackage. | |
| ~~UC\_M01\_06~~ | ~~Thingworx collects new data from “~~**~~serial number~~** ~~exchange table” with result identifier = “null”, the~~ **~~serial number is existent~~** ~~in the datable yet,~~ **~~production started~~**  ~~Expected result:~~   * ~~After a regular timeframe the exchange database table column for the~~ **~~result-identifier is filled~~** ~~with related return code, so not null any more.~~ * **~~No data where updated~~** ~~in the twx data table~~ * ~~result identifier = 12~~ |
| ~~UC\_M01\_07~~ | ~~Thingworx collects new data from “~~**~~order number~~** ~~exchange table” with result identifier = “null”, the order~~ **~~number is existent~~** ~~in the datable yet,~~ **~~production started~~**  ~~Expected result:~~   * ~~After a regular timeframe the exchange database table column for the~~ **~~result-identifier is filled~~** ~~with related return code, so not null any more.~~ * **~~No data where updated~~** ~~in the twx data table~~ * ~~result identifier = 12~~ |

* + 1. Implementation Details

**Instructions**

This section will be added during the actual implementation and reflects, what actually was created on the system at the end.

All Thingworx ERP related implementations must be done inside “MSerpConnector” extension which is available on KB Integrity environment. Normally no dependencies to Things defined outside of the extension are allowed, if needed they have clearly to be documented.

As we use external database to store order data do ***not*** use directly the related database Thing because databases are separated for dev/test/prod system. Therefor a Resource Service is available to always get right DB Connection:

*var DB = Resources["****KBThingWorxServerSystem****"].GetKPILocalThingWorxDatabaseThing();*

*var params = {*

*your service params*

*};*

*var result = Things[DB].YOUR\_DB\_SERVICE\_TO\_CALL(params);*

Order Data are stored mostly as Key/Value pairs, so no change in DB design has to be done if additional properties should be stored. Till now float (incl. integers), text and date-time values can be stored.

For Microlettrica Orders are stored on local system, to store data use the *“Resources["KBThingWorxServerSystem"].GetKPILocalThingWorxDatabaseThing()”* Thing and call “InsertOrUpdateOrderData” Service. Basic data is implemented as service property (like Order Number and System). The Key Value pairs are to be defined in related “KBOrderKeyValueDataShape”.

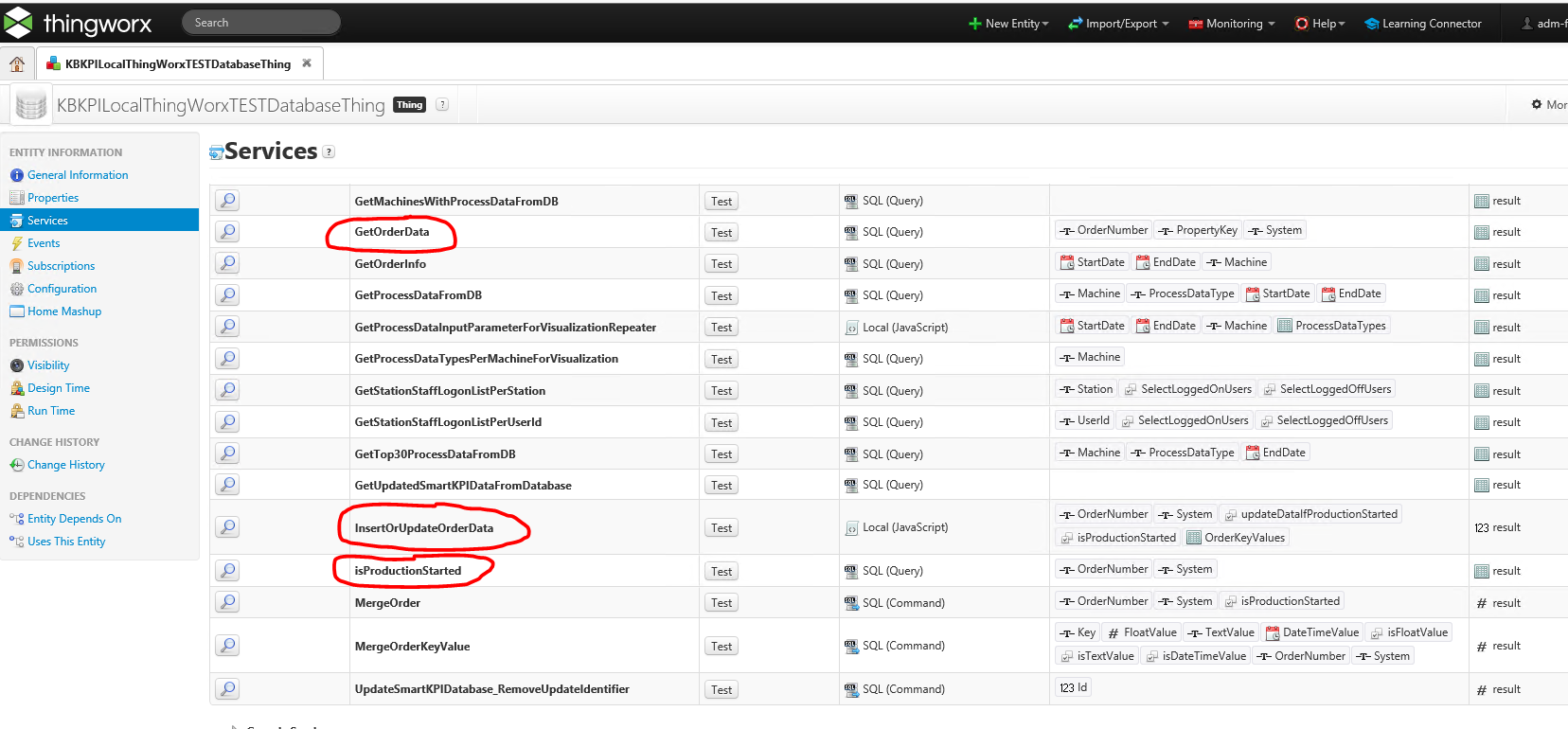
**System name for this project will be “AS/400@MS”**

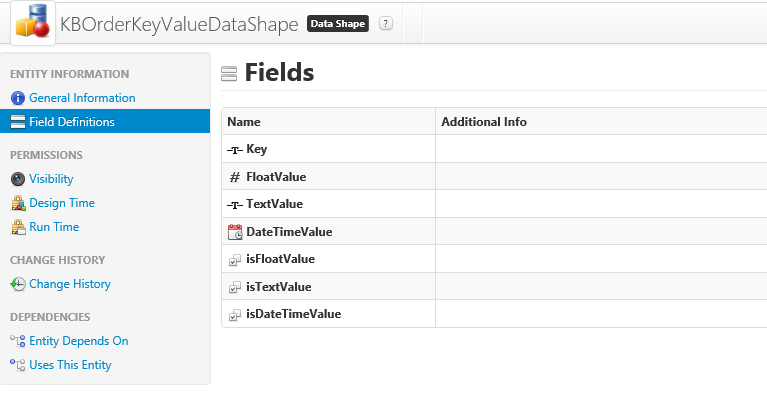
The Service directly gives the possibility to check if production was started so that data are not updated if true. So set the service property “updateDataIfProductionStarted” always to false.

The return code of the service directly fits to the result-identifier which was described before.

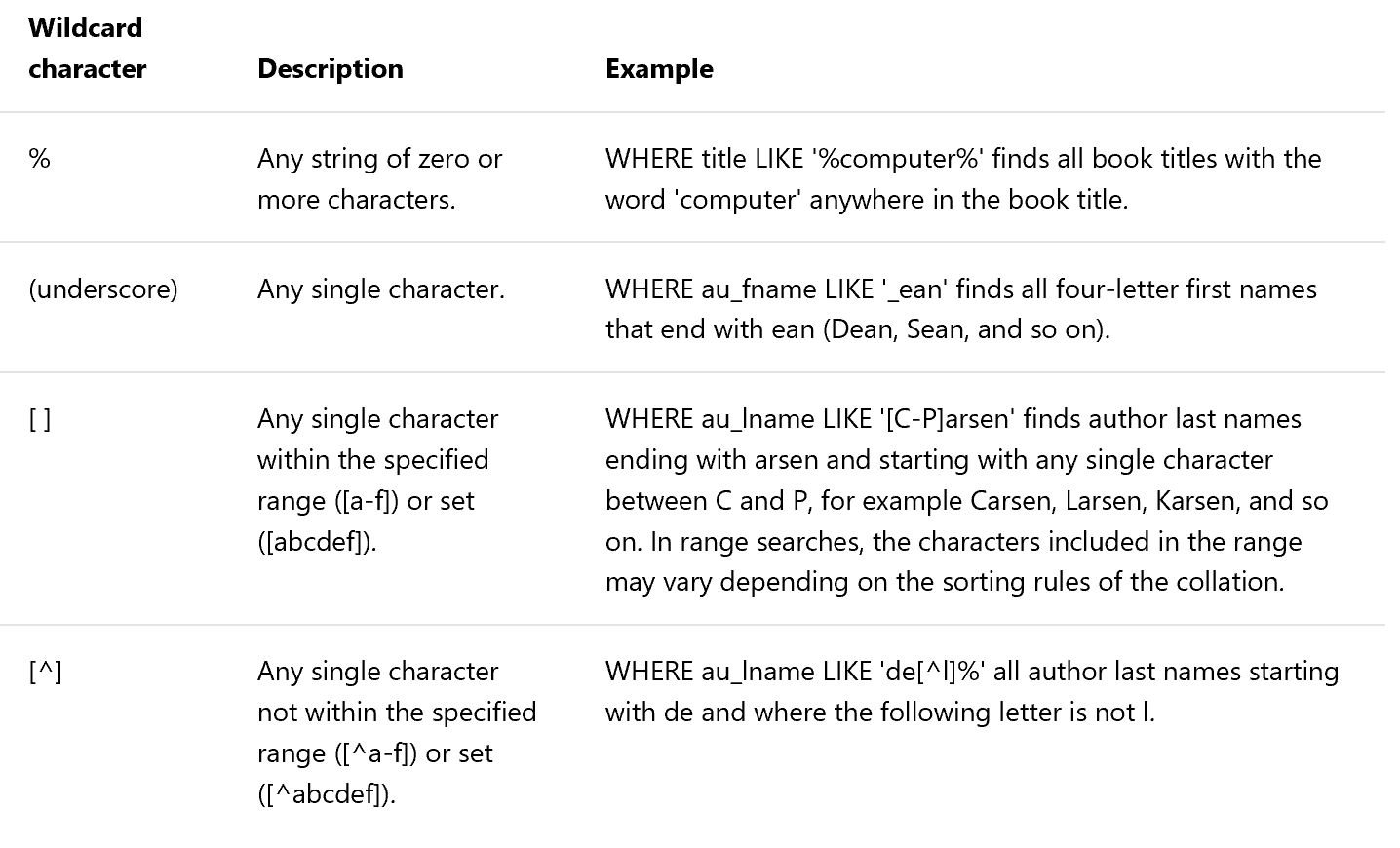
Use same service for Orders and Serial numbers:

* Order information
  + OrderNumber = [OrderNumber]
  + System = ‘AS/400@MS’
* Serial Number Information
  + OrderNumber = [OrderNumber]-[SerialNumber]
  + System = ‘AS/400@MS’
    - Key/Value Entry
      * Key = ‘SerialNumber’
      * Value = [SerialNumber]

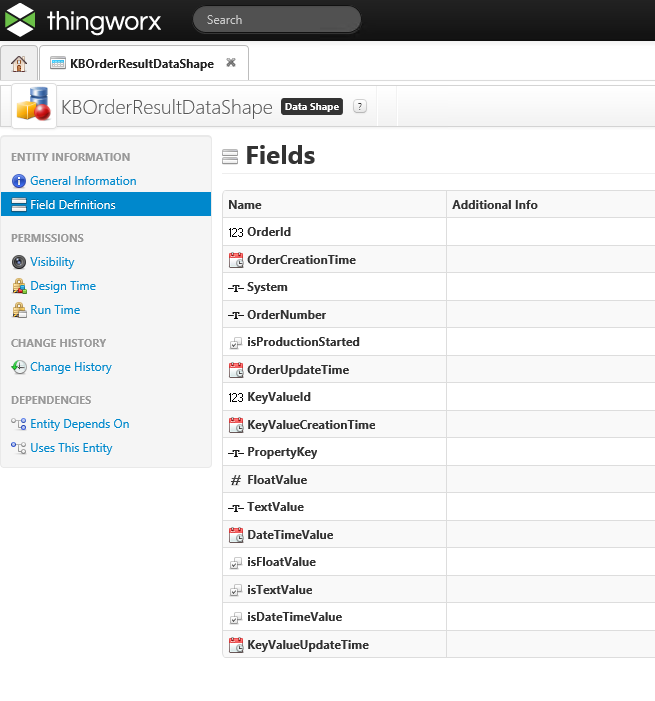




To read order data the Service “GetOrderData” is defined. Here as well the properties OrderNumber and System are required. For OrderNumber the MS SQL Server “like” functionality can be used:



The “key” service property is optional, per default it’s set to “undefined” so all Key/Value Pairs will be selected, if “Key” property is filled only the related Key/Value Pairs are selected. Result-set of the “GetOrderData” Service is the “KBOrderResultDataShape”:

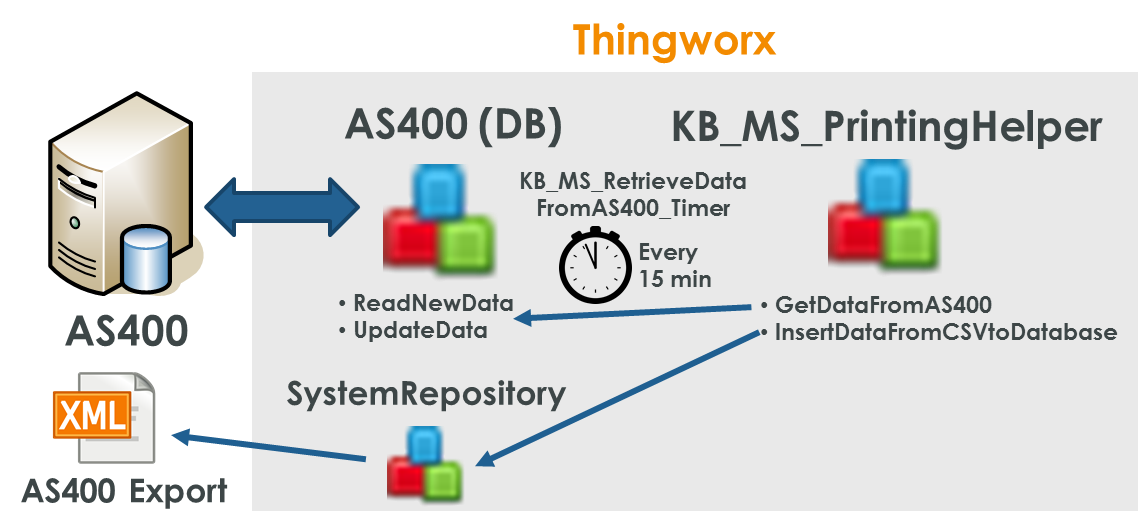


PTC will implement the use cases UC\_M01\_02, UC\_M01\_03, UC\_M01\_04, UC\_M01\_05, UC\_M01\_06 and UC\_M01\_07, responsible for use cases UC\_M01\_01 is ERP supplier of Microlettrica.

List of Result Identifiers inside exchange Table

|  |  |  |
| --- | --- | --- |
| Result-Identifier | Meaning | Comment |
| Null | New line inserted from ERP | Only lines with result-identifier null will be imported to Thingworx! Please note the meaning of null in database environment! Null <> 0! |
| 10 | Import OK |  |
| The different result identifiers below, will be implemented with a future version of the SolutionPackage. | | |
| 11 | Import NOK, undefined reason |  |
| 12 | Import NOK, Production started |  |
| 21 | Processing Error, no Serial Numbers available | New inserted line from Thingworx or existing line updated if order number still available |

* + - 1. *ERP Connector – TWX implementation view*

****

Core of the TWX Implementation for the Printing Solution is the KB\_MS\_PrintingHelper. This Thing provides basic functionalities to get and process data from AS400.

To establish the actual connectivity to AS400, a Thing called AS400 is used. It is derived from Database-Template uses the driver com.ibm.as400.access.AS400JDBCDriver and contains also the user and password for interacting with the database using sql.

One major obstacle to make the connection work was the connection string. The following connection string will make the interaction work: **jdbc:as400://10.6.4.11:446/AQS05DAT;naming=sql;errors=full;transaction isolation=none;date format=iso**

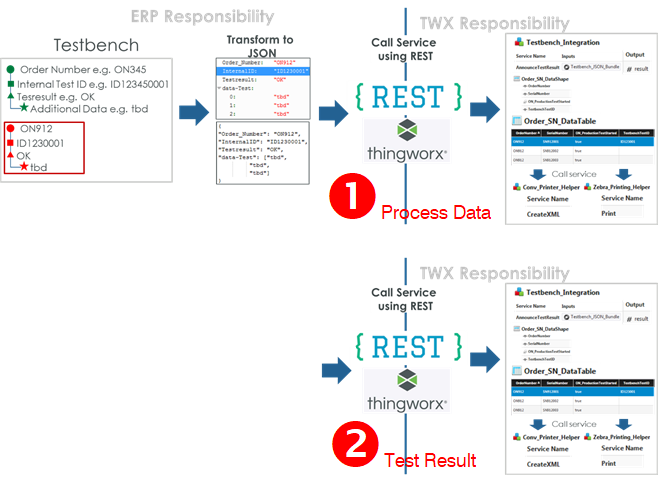
The AS400 Thing implements basic read and write features to the DB. This is e.g. ReadNewData and UpdateData [for specific serial number].  
**With Release 1.0.0 of the Solution only a 10 will be written back to AS400 after processing a dataset. The following UseCases are not implemented as part of Release 1.0.0 - UC\_M01\_06, UC\_M01\_07.**

These functions are called by the KB\_MS\_PrintingHelper by the function: GetDataFromAS400.  
The function GetDataFromAS400 will also incorporate the data from AS400 into the TWX database, as described above.

To regularly incorporate data from AS400 into the TWX DB a Timer KB\_MS\_RetrieveDataFromAS400\_Timer is part of the Solution. This Timer calls the GetDataFromAS400 function every 15 minutes (900000 milliseconds).

For testing purposes, a second way to incorporate data from AS400 into the TWX DB was implemented, using an xml-export and the function: InsertDataFromCSVtoDatabase

* 1. Testbench Connection
     1. Overview
* The Testbench Software is based on LabView. An integration concept is already available online https://sites.google.com/site/thingworxlabviewdocumentation/labview-and-thingworx
* The bare minimum Thingworx need to process the Testbench data is the following (user for transferring test result):
  + Current Order Number which is processed
  + Internal Test ID of tested product 🡪 to be mapped with serial in Thingworx
  + Test Result OK/nOK
* Additional data (e.g. process data) can be sent to Thingworx (***before sending the test result***). These data are probable necessary to create proper test protocols and/or labels. The details will be documented in the Implementation Details section.
* The Testbench is directly sending a request to Thingworx after the test result is announced.  
  Each test result will be send in a single request to Thingworx.
* Sending test result is always the last interaction from test bench to Thingworx, if available process data must be sent before!
* Thingworx will answer the request by the test bench with defined ok/error codes.
* If a new order number is sent to Thingworx for the first time, the order will be marked as “In Production”, and so no updates will be loaded from ERP for this order any more.
* If the order which is sent from test bench is not available in Thingworx a related error code will be sent. As Thingworx does not have user interaction on line side the test bench has to implement a user interaction to get a valid order number! **Without a valid order number no label will be triggered!**
* The Internal Test ID from the test bench will be written into the lowest available Serial Number of the related order.
* If no “free” serial number is available in Thingworx a related error code will be sent. **Without a valid serial number no label will be triggered!**
* After updating the Order/Serial Number DataTable a Service within TWX is called to create the xml file. The related data will be forwarded to this service.
* The OK test result from test bench will trigger the label printing
* **The order of conformity document is not part of the Release 0.0.4.**
  + 1. Illustration



* + 1. Use Cases

|  |  |
| --- | --- |
| Use Case |  |
| UC\_M02\_01 | Perform and retrieve well-formed REST-Call, with existing Order Number and Test result **OK**. The **Order Number was not sent before**. Thingworx still has an unused Serial for the related Order Number.  Expected result:   * The Order will be marked as “in Production” * The Internal Test ID will be set to the next free serial number of this order * The createXML service for conv Printing will be triggered * The Print service for Zebra Printer will be triggered * 10 as a Feedback for the service call |
| UC\_M02\_02 | Perform and retrieve well-formed REST-Call, with existing Order Number and Test result **OK**. The **Order Number was already sent earlier**. Thingworx still has an unused Serial for the related Order Number.  Expected result:   * The Internal Test ID will be set to the next free serial number of this order * The createXML service for conv Printing will be triggered * The Print service for Zebra Printer will be triggered * 10 as a Feedback for the service call |
| The UseCases UC\_M02\_02, and UC\_M02\_03 are implemented slightly different. If the test result is nOK the result identifier will always be 11. The here written behavior will be implemented with a future version of the SolutionPackage. | |
| UC\_M02\_03 | Perform and retrieve well-formed REST-Call, with existing Order Number and Test result **nOK**. Expected result:   * **Nothing will be triggered** * 11 as a Feedback for the service call |
| UC\_M02\_04 | Perform and retrieve **mal-formed Rest-Call**  Expected result:   * **Nothing will be triggered** * No data added in data table * 29 as result of the service |
| UC\_M02\_05 | Perform and retrieve well-formed REST-Call, with **non-existing Order** Number.  Expected result:   * **Nothing will be triggered** * 20 as a Feedback for the service call |
| UC\_M02\_06 | Perform and retrieve well-formed REST-Call, with existing Order Number and Test result OK. Thingworx still has **no unused Serial Number** for the related Order Number.  Expected result:   * **Nothing will be triggered** * 21 as a Feedback for the service call |

* + 1. Implementation Details

**Instructions**

This section will be added during the actual implementation and reflects, what actually was created on the system at the end.

In Thingworx each test bench will be represented by a Thing which is derived from **KBLocalStationThingTemplate**. Things will be created by global IT. In addition, the thing shape **KBStationDataReceiverThingShape** (integrated in **MESKit** Extension) must be added which has all the needed default Services implemented to receive data from Thingworx point of view. From test bench point of view an additional Thing Shape has to be added where Microelettrica specific tasks are implemented (special return codes): **KBMILTestBenchReceiverThingShape** (integrated in **KBLocalsmartKPIConfigurationMIL** Extension)

* + - 1. *Test Bench point of view*

Especially in case of error the Test Bench has to react on the different use cases because Thingworx does not have the possibility of user interaction or information visualization on line side. For pushing the test result to Thingworx an additional (extended) service has to be implemented to push back the result to test bench.

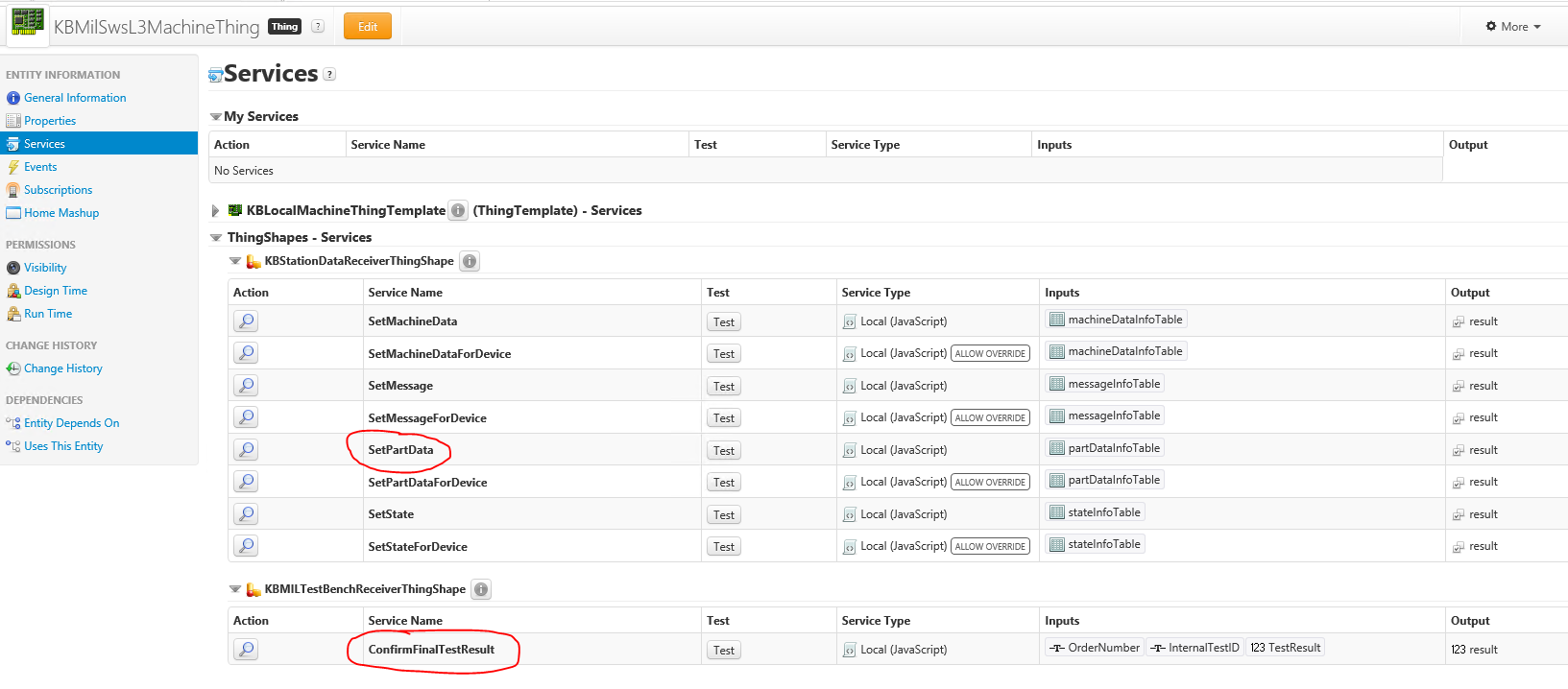
From test bench point of view there are two use cases to transfer data.

1. Process data
2. Final test result (with related return codes)

Process data always have to be transferred before the final test result is sent. Reason is that possibly process data are printed on labels or certificates. Process data can be transferred as list of data in a single step but it’s also possible to call it multiple times.

See Use Case 2 in the following chapter (Thingworx internal point of view) to get a description how to call the **SetPartData** Service.

As Thingworx has to provide a more detailed return code in case of transferring the final test result than it is available by the default implementation a new service is implemented for that. Call the **ConfirmFinalTestResult** Service to give final decision of the test result to Thingworx:



**For retests (of the same physical part) the same test bench internal identifier has to be used.** Only then it can be guaranteed that a part which is tested twice as ok has not two serial numbers assigned!

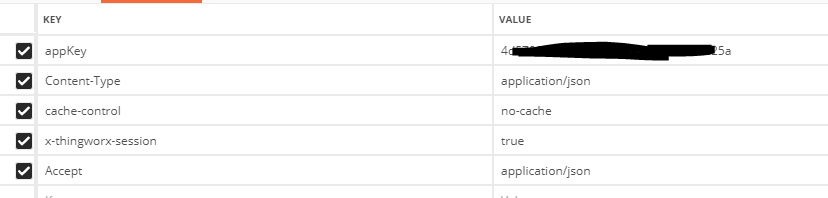
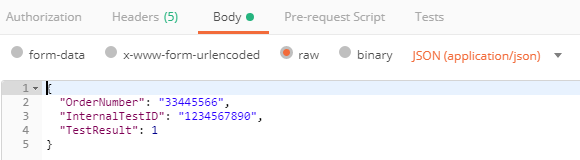
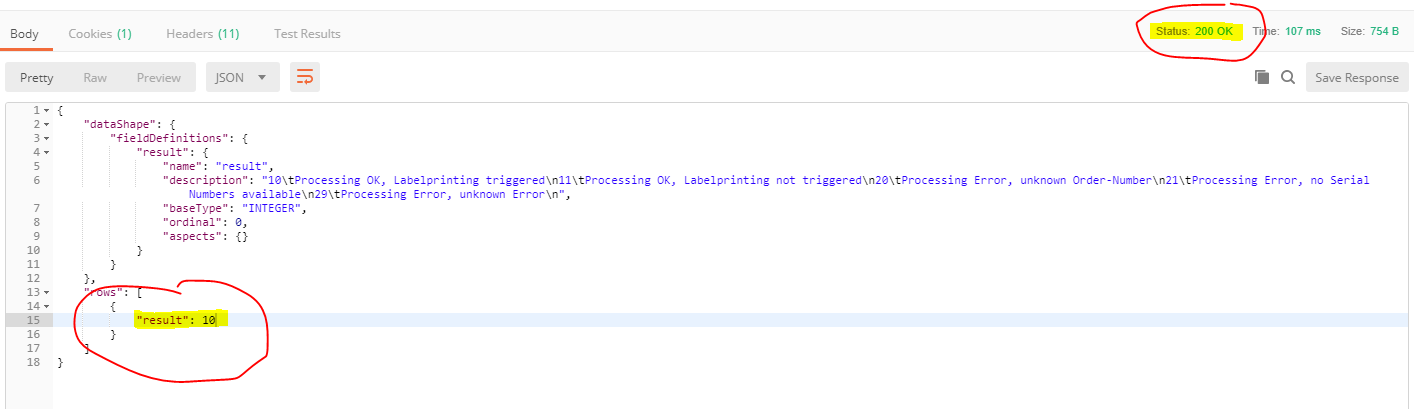
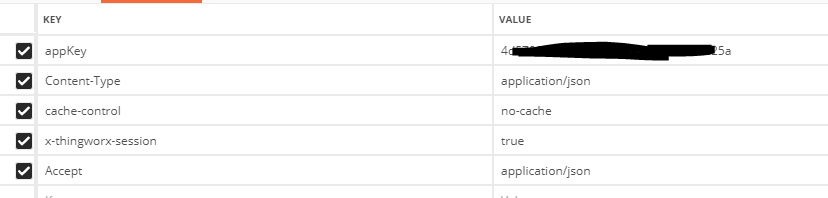
It has to be checked from process side (test bench and ERP) that all data needed for printing is transferred before! Thingworx will not check during print processing if all data are available. If related data are not available fields will be printed as empty ones!

The here written behavior will be implemented with a future version of the SolutionPackage.

List of Result Identifiers

|  |  |  |
| --- | --- | --- |
| Result-Identifier | Meaning | Comment |
| As Implemented with release 1.0.0 all NOK results will result in the result identifier 12. | | |
| 10 | Processing OK, Labelprinting triggered | Test Bench Result is OK |
| ~~11~~ | ~~Processing OK, Labelprinting for Error-Label is triggered~~ | ~~Test Bench Result is NOK~~ |
| 12 | Processing OK, Labelprinting ***not*** triggered | Test Bench Result is NOK, and it was NOK for the same physical part already earlier |
| 20 | Processing Error, unknown Order-Number |  |
| 21 | Processing Error, no Serial Numbers available |  |
| 29 | Processing Error, unknown Error |  |

Using the “Postman” application to show a working Web-Service-Call with SWS L3:

* ConfirmFinalTestResult
  + URL: <https://smartproductionmiltest.corp.knorr-bremse.com/Thingworx/Things/KBMilSwsL3TestBenchStationThing/Services/ConfirmFinalTestResult>
  + Type: Post
  + Headers: 
  + Body: 
  + Result: 
* SetPartData
  + URL: <https://smartproductionmiltest.corp.knorr-bremse.com/Thingworx/Things/KBMilSwsL3TestBenchStationThing/Services/SetPartData>
  + Type: Post
  + Headers: 
  + Body:

*{*

*"partDataInfoTable":*

*{*

*"dataShape":*

*{"fieldDefinitions":*

*{*

*"Description":{"name":"Description","aspects":{},"description":"","baseType":"STRING","ordinal":3},*

*"PartNumber":{"name":"PartNumber","aspects":{},"description":"","baseType":"STRING","ordinal":15},*

*"TrackingNumber":{"name":"TrackingNumber","aspects":{},"description":"","baseType":"STRING","ordinal":13},*

*"OrderNumber":{"name":"OrderNumber","aspects":{},"description":"","baseType":"STRING","ordinal":14},*

*"ShortName":{"name":"ShortName","aspects":{"isPrimaryKey":true},"description":"","baseType":"STRING","ordinal":7},*

*"Unit":{"name":"Unit","aspects":{},"description":"","baseType":"STRING","ordinal":8},*

*"UpperLimit":{"name":"UpperLimit","aspects":{},"description":"","baseType":"STRING","ordinal":11},*

*"DateTime":{"name":"DateTime","aspects":{"isPrimaryKey":true},"description":"","baseType":"DATETIME","ordinal":5},*

*"Line":{"name":"Line","aspects":{"isPrimaryKey":true},"description":"","baseType":"STRING","ordinal":1},*

*"Station":{"name":"Station","aspects":{"isPrimaryKey":true},"description":"","baseType":"STRING","ordinal":2},*

*"Type":{"name":"Type","aspects":{"isPrimaryKey":true},"description":"","baseType":"STRING","ordinal":4},*

*"Identifier":{"name":"Identifier","aspects":{},"description":"","baseType":"STRING","ordinal":6},*

*"SerialNumber":{"name":"SerialNumber","aspects":{},"description":"","baseType":"STRING","ordinal":12},*

*"Value":{"name":"Value","aspects":{},"description":"","baseType":"STRING","ordinal":9},*

*"LowerLimit":{"name":"LowerLimit","aspects":{},"description":"","baseType":"STRING","ordinal":10}*

*}*

*},*

*"rows":*

*[*

*{*

*"Description":"sample-desc",*

*"PartNumber":"4444444444",*

*"TrackingNumber":"2222222222",*

*"OrderNumber":"3333333333",*

*"ShortName":"sample-shortName",*

*"Unit":"sample-unit",*

*"UpperLimit":"3",*

*"DateTime":1529918532000,*

*"Line":"SWSL3",*

*"Station":"TestBench",*

*"Type":"PARTDATA",*

*"Identifier":"sample-Id",*

*"SerialNumber":"111111111",*

*"Value":"1",*

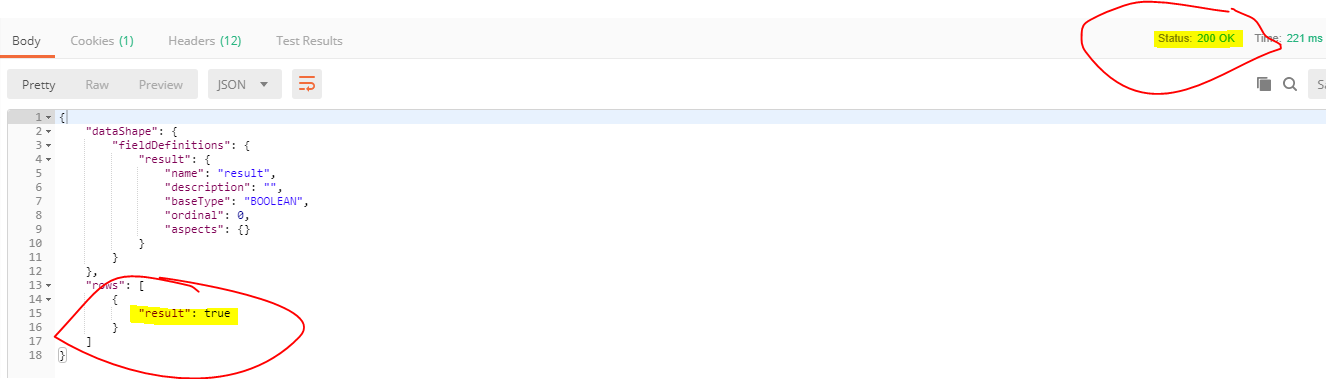
*"LowerLimit":"2"*

*}*

*]*

*}*

*}*

* + Result: 
    - 1. *Thingworx internal point of view*

**Use Case 1: Test bench is pushing OK/NOK result:**

**SetPartData** Service must be used

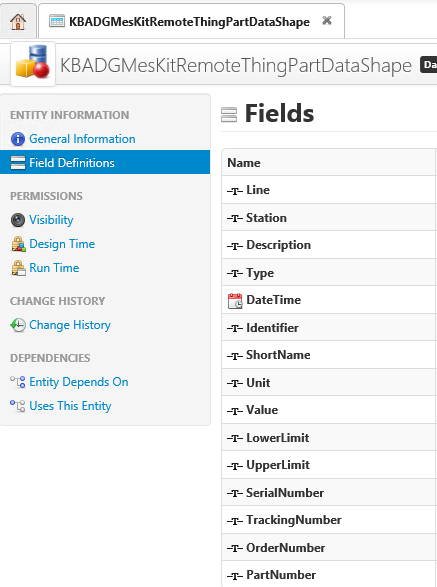
* Line Line Name
* StationName Station (Test Bench) Name
* ShortName “Testresult”
* Description optional, e.g. “Test result from Station xyz”
* Type “PARTDATA”
* DateTime Time of test result at test bench
* Identifier optional
* Unit empty, not needed
* **Value 0 if NOK, 1 if OK**
* LowerLimit empty, not needed
* UpperLimit empty, not needed
* Serial Serial Number of the Part
* Tracking Test bench Id
* **OrderNumber Order-Number must be sent!**
* PartNumber Part-Number

The integration of label printing can be done inside the SetPartDataForDevice Service which can be overridden.

**Use Case 2: Test bench is pushing test results (measurements):**

**SetPartData** Service must be used (parameters are a list/infotable of the following values, see also KBADGMesKitRemoteThingPartDataShape)

* Line Line Name
* StationName Station (Test Bench) Name
* ShortName **Type of Measurement**, not allowed is to use “Testresult” because is occupied by use case 1 (e.g. Current, Windings, Response Time…)
* Description optional, e.g. “Test result from Station xyz”
* Type “PARTDATA”
* DateTime Time of test result at test bench
* Identifier optional or if text data
* Unit Measurement Unit
* **Value Measurement result**
* LowerLimit Measurement Lower Limit
* UpperLimit Measurement Upper Limit
* Serial Serial Number, is not known by test bench because it’s assigned later on 🡪 empty
* Tracking **Test Bench Test Id**
* OrderNumber **Order-Number**
* PartNumber Part-Number if known



**Attention: With sending the OK/NOK signal (use case 1) the label printing will be triggered. With sending this information to Thingworx all needed measurements (use case 2) must been sent! It’s absolutely needed to keep the right sequence of sending results!**

**Right sequence is:**

1. **Sending all measurements from use case 2**
2. **Sending final OK/NOK signal (use case 1)**

**Optional:**

**Status Information from Test Bench:**

**SetStateData** Service must be used

* Line Line Name
* StationName Station (Test Bench) Name
* ShortName “STATE”
* Description optional
* Type “STATE”
* DateTime Time of test result at test bench
* Identifier “IDLE” / “WORKING” / “SERVICE” / “ERROR”

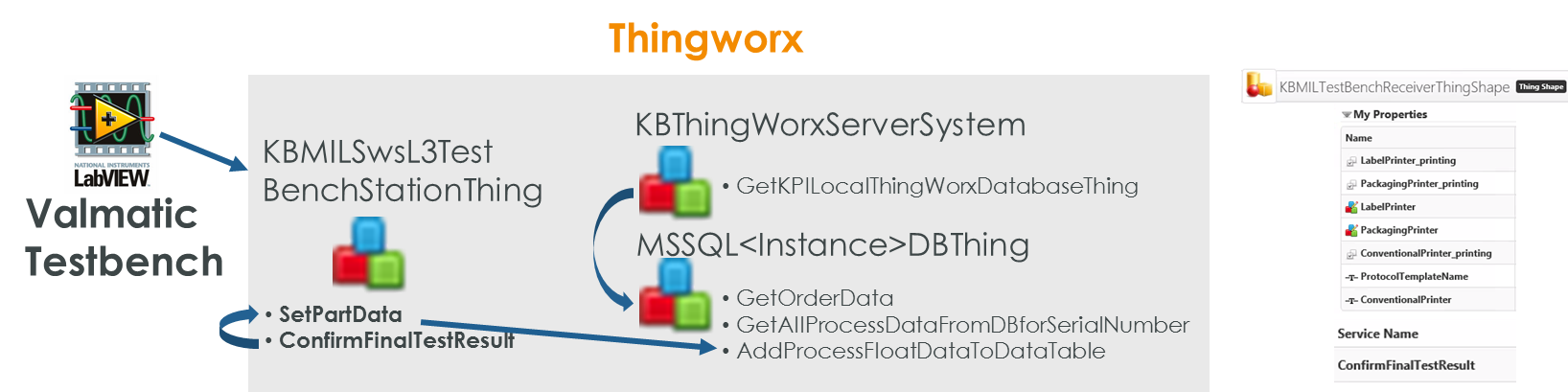
**Machine (Test Bench) related data:**

**SetMachineData** Service must be used

* Line Line Name
* StationName Station (Test Bench) Name
* ShortName Type of Measurement
* Description optional
* Type “MACHINEDATA”
* DateTime Time of test result at test bench
* Identifier optional or if text data
* Unit Measurement Unit
* **Value Measurement result**
* LowerLimit Measurement Lower Limit
* UpperLimit Measurement Upper Limit

The SetMessage Service is used to send not part related and no measurement data to Thingworx. Is there’s the need to do please contact Digital Manufacturing team.

* + - 1. *Testbench Connection – TWX implementation view*



The Valmatic Testbench is calling the digital representation of the Testbench. The digital representation of the Testbench is part of the plants assets in Thingworx. The speciality of the Testbench-Thing is the implementation of the ThingShape: KBMILTestBenchReceiverThingShape

This provides the Thing in addition to the standard SetPartData function also with the ConfirmFinalTestResult and all necessary information for the line regarding the printing solution.

The Labview application will call SetPartData for all process data and once the Test on the Testbench is finished the Labview application will call ConfirmFinalTestresult.

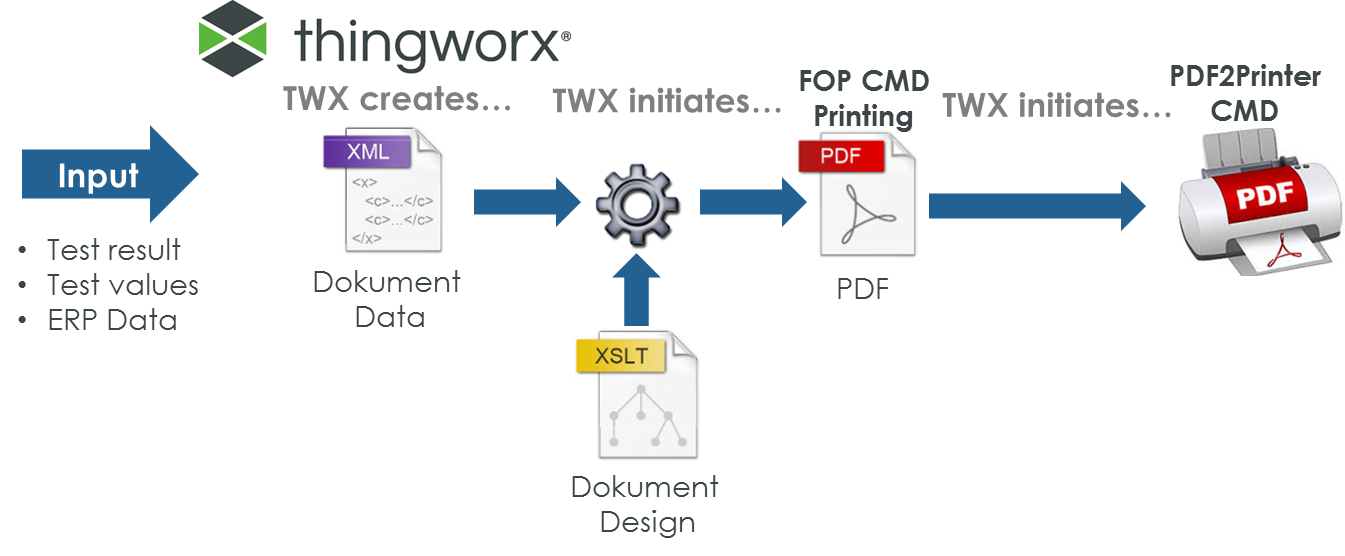
All Data is stored in the TWX DB according to the concept described above.

The ConfirmFinalTestresult function will do the error handling, assembling of data from AS400 copy (TWX DB), and trigger printing at the right devices using the KB\_MS\_Printing Helper.

ConfirmFinalTestresult will also call SetPartData to document all the testresults in the TWX DB.

**As part of the Release 0.0.4 the solution identifier for a NOK result is set to 12. This is implemented, because the Testbench is already prepared for a Error-Label process. The Error Label process should be part of future implementation, as mentioned towards the end of the document.**

* 1. Conventional Printing
     1. Overview
* The conventional Printing is usually triggered from within Thingworx. However, since this is a separate module, it can be used separate.
* As prerequisite for the conventional printing at least 1 xsl file per Dokument type needs to be placed on the server. The xsl files are manually genererated and can be iterated and replaced on the server
* The module is providing different services, which are mostly called in sequence:
  + Create a XML protocol file
    - A xml fill be created by thingworx and stored in the thingworx storage
    - The file will follow the following naming convention:
      * Test Protocols: **Protocol\_<Line>\_<OrderNr>-<SerialNr>.**xml
      * Order of Conformity: **<OrderConf\_YYYYMMDDHHMMSS>**.xml  
        With Release 1.0.0 the order of conformity was not implemented, and will be considered in a future phase.
    - Each time the service will be executed, it will create a new xml-file.  
      There will be no check for existing files.
    - When executing the service, all the input parameters will be checked for completeness, the completeness will be confirmed by the service with result = 1, otherwise result = 0
  + Build a pdf protocol file based on single xml-file and xsl-file
    - The creation of the pdf will be initiated by Thingworx and executed via runtime
    - The command looks like the following:
      * **cd /locationoffop/fop**
      * **fop -xml /localizationofXMLfile/fileName.xml -xsl /localizationofXSLFO/xslfoName.xsl -pdf /outputLocalization/fileName.pdf**
    - The conversion to pdf is based on The Apache™ FOP Project
    - All created pdfs are stored in Thingworx storage and will follow the naming of the xml-file.
    - The service will return 1 if input parameters are complete, and 0 if input parameter are not complete. There will be no additional error handling in Thingworx, if files are available at input location, or xml and xsl fit together
  + Print the pdf file to a conventional printer
    - The printing will be initiated by Thingworx
    - Printing will be executed by the command line tool PDFtoPrinter.exe
    - The Printer needs to be installed as network printer
    - Thingworx checks if the pdf-file exists, otherwise it returns 0
    - No errorhandling or logging will be created, when executing PDFtoPrinter.exe
* All Created pdf will be stored in the Thingworx storage. Long time archiving is not part of the POC but highly recommended as a future implementation.
* **The same technique will be used for testbench protocol data, as for order of conformity.**
  + 1. Illustration



* + 1. Use Cases

|  |  |
| --- | --- |
| Use Case |  |
| UC\_M03\_01 | Execute the xml-creation-service with a complete set of input parameters.   * Expected result:   + A xml file was created following the naming convention **Protocol\_<Line>\_<OrderNr>-<SerialNr>**.xml   + The service will execute and send 1 as a Feedback for the service call |
| UC\_M03\_02 | Execute the xml-creation-service with an incomplete set of input parameters.   * Expected result:   + No xml-file will be created   + The service will execute and send 0 as a Feedback for the service call |
| UC\_M03\_03 | Initiate the PDF creation from Thingwork (service call) and provide a xml and a xsl file   * Expected result:   + The service will send the result 1   + Ideally the pdf is created and stored in Thingworx location, following the naming convention of the xml   + There will be no error handling for FOP processes, if no pdf is created, logs from FOP may help |
| UC\_M03\_04 | Initiate the PDF creation from Thingwork (service call) and provide only a xml or a xsl file   * Expected result:   + The service will send the result 0   + No pdf will be created |
| UC\_M03\_05 | Initiate the conventional printing from Thingworx (service call) via PDFtoPrinter.exe. Use as parameters the pdf-name and printer name.   * Expected result:   + The file is printed   + 1 as a Feedback for the service call |
| UC\_M03\_06 | Initiate the conventional printing from Thingworx (service call) via PDFtoPrinter.exe. Use as parameters the pdf-name and printer name. The pdf-name provided in the service call, does not exist.   * Expected result:   + Printing will not be started.   + 0 as a Feedback for the service call |

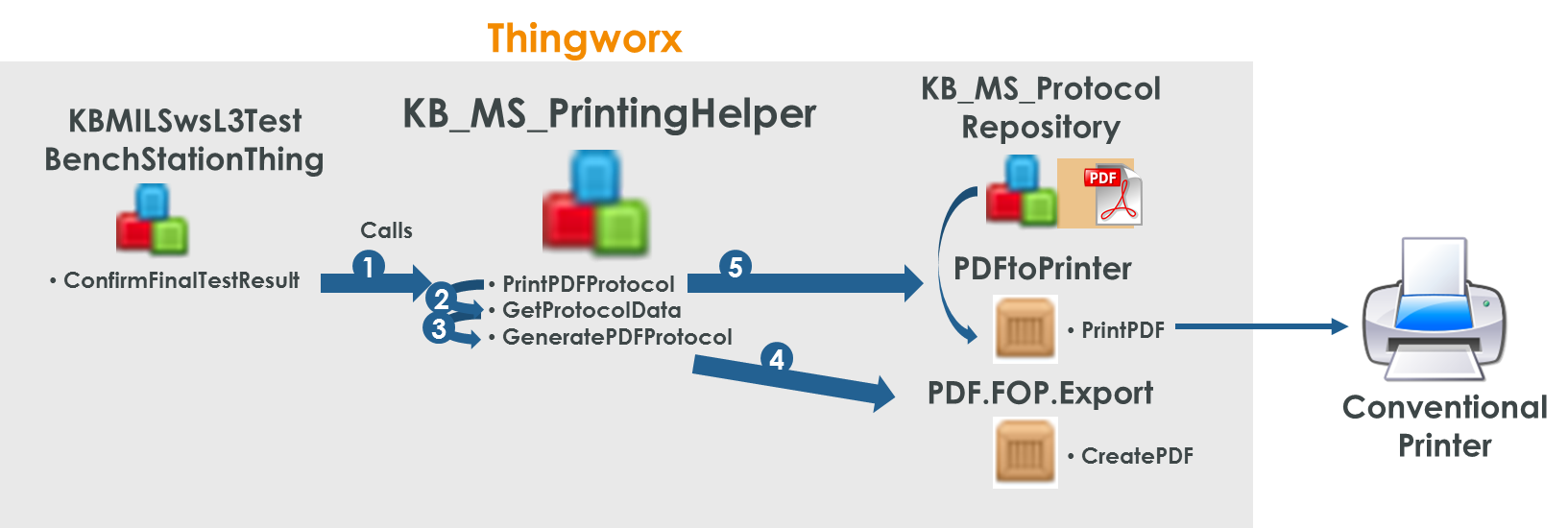
* + 1. Implementation Details

**Instructions**

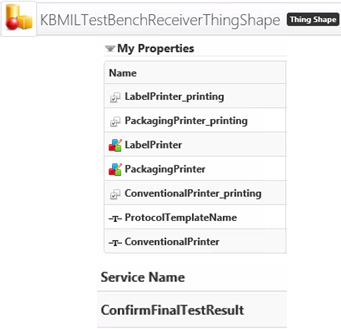
This section will be added during the actual implementation and reflects, what actually was created on the system at the end.

Calling a shell command from Thingworx:   
<https://www.ptc.com/en/support/article?n=CS284592> not visible for any customer at the moment.

* + - 1. *Conventional Printing – TWX implementation view*



The Conventional Printing modules requires multiple Ressources. One Ressource is the PDFtoPrinter Ressource, which is part of the PDFtoPrinter\_extension (subextension of solution package) and the PDF.FOP.Export, which is part of the PDF\_FOP\_Extension (subextension of solution package).

Whilst the Line specific TestBenchStationThing is responsible for triggering the printing and providing the information about printer name, it uses Helper Classes from the KB\_MS\_PrintingHelper to actually create the PDF and then print the PDF.

Properties at the testbench, implemented by the KBMILTestBenchReceiverThingShape offer Line specific information about the Printer (ConventionalPrinter Attribute), the xsl-Template (ProtocolTemplateName) and the property if physical printing of the protocol should be enabled or not (ConventionalPrinter\_printing Attribute).

The xsl-Template (for the layout of the PDF-protocol) as well as the ConventionalPrinter Name are specific for each line.

The xsl-Template for each line needs to be placed inside the KB\_MS\_Protocol Repository, where the Protocols of all Lines are stored.

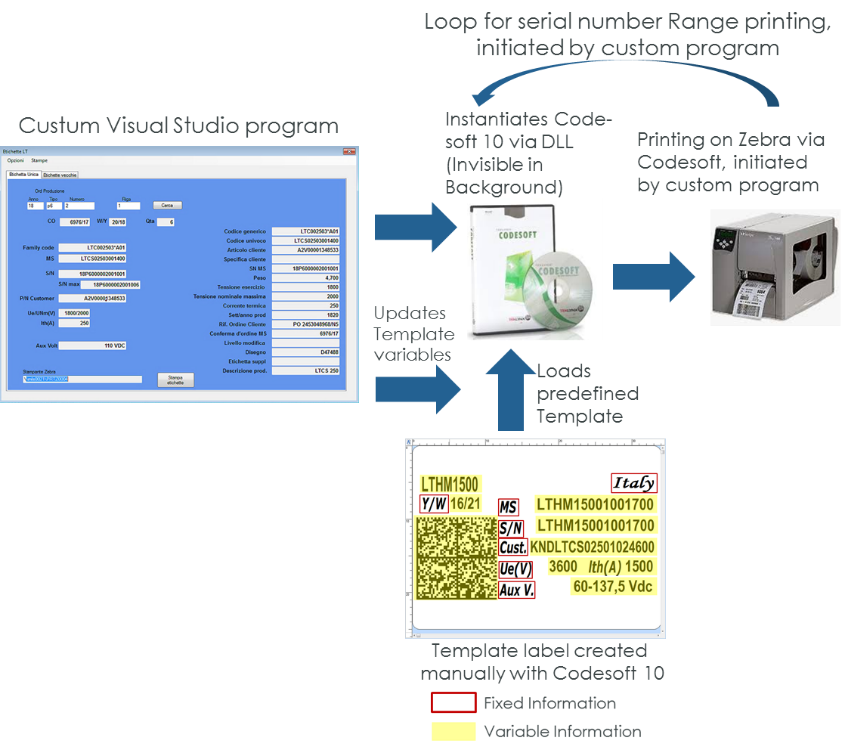
* 1. Zebra Printing
     1. Introduction: Current Zebra Printing Process

This chapter explains the current printing process of Zebra-Labels, to get a better introduction into the topic. As of today, 2 Labels are printed. One is the product label itself, the other one is the packaging label. In the discussion here we will only mention the product label, but need to keep in mind, that at the end also the packaging label need to be printed.

All Labels in the current process are printed on a Zebra S4M printer. (Just as a remark: This model does not support ZPL storage on Printer, remote printing via command line and is not based on Link-OS. Furthermore, this printer has discontinued support since 2016 [based on telephone call with Zebra Customer Support 15.05.2018]  
<https://www.barcodegraphics.com/datasheets/printers/Zebra_s4m.pdf>)  
The current Printing solution consists of 2 different software components and Zebra Printer S4M installed as a network printer.

The 2 software components are:

* + Codesoft Enterprise 10
    - RFID and Barcode Label Software
    - ZPL template creation including variables within the software
    - Triggers printing: Sends data (ZPL file) for each barcode to printer
    - Provides DLL to instantiate and connect to software from externally
  + Custom Visual Studio Program created @ Microelettrica
    - Program retrieves data from ERP AS400 based on Order Number
    - Program instantiates Codesoft Enterprise
    - Program transfers content for variables and also supports iterations for serial number increments ending in different labels



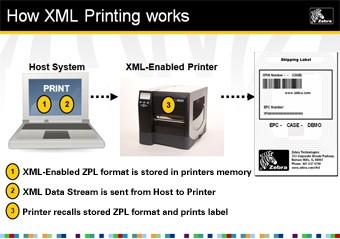
* + 1. Overview
* Printing labels are used for 2 different purposes, one is the product label itself, the other label is printed for the packaging. Due to different requirements, the labels are printed on different Zebra printers.
* The printers **ZT230** for the product label, and **GX420** for the packaging label.  
  Both printers are not available yet, but will be in the production of the proof of concept.  
  **ZT230:**<https://www.zebra.com/de/de/products/printers/industrial/zt200-series.html>   
  <https://www.zebra.com/content/dam/zebra_new_ia/en-us/solutions-verticals/product/Printers/Industrial%20Printers/ZT200%20Series%20Industrial%20Printers/GENERAL/zt200-spec-sheets/ZT200-Datasheet-German.pdf>

<https://www.zebra.com/content/dam/zebra/product-information/en-us/brochures-datasheets/industrial/zt200-technical-specifications-external-en-us-gl-a4.pdf>

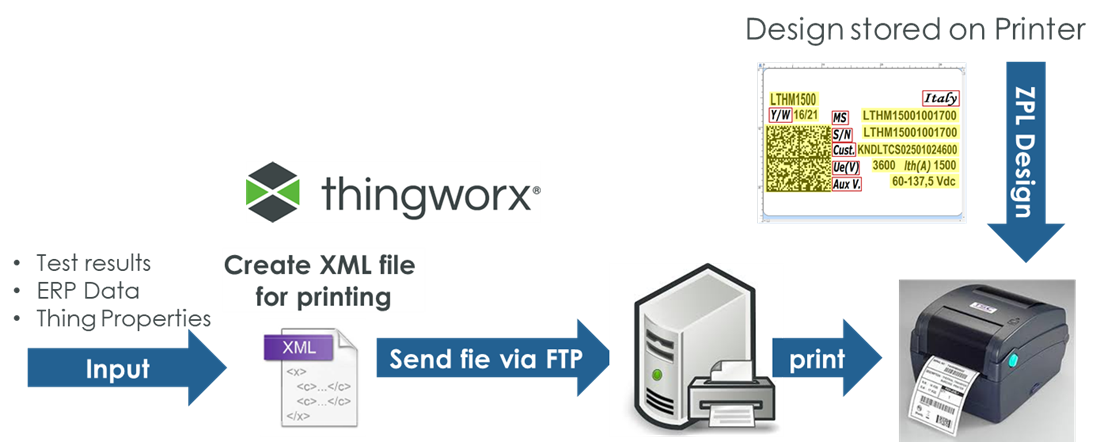
<https://www.zebra.com/de/de/support-downloads/printers/industrial/zt230.html>

**GK420T:**  
<https://www.zebra.com/content/dam/zebra_new_ia/en-us/solutions-verticals/product/Printers/Desktop%20Printers/GENERAL/spec-sheets/G-Series-Datasheet-APAC-English.pdf>

* The Zebra supplier confirmed, that both printers ZT230 and GX420 are capable of printing via xml, and fulfill the basic requirements for the use case
* The Zebra printing is initiated by the “testbench OK” signal.
* Zebra Printing Solution Approach
  + Store a ZPL Template file on the Zebra Printer with defined Variables
  + Send Variables to the printer related to the stored ZPL file.
    - This will be done using xml & sending it via ftp to the print server

  
<https://www.zebra.com/us/en/resource-library/getting-started/barcode-printing/xml-enabled-printing.html>

* As of Version 1.0.11 the printing for the product label will be done directly via ZPL. Thingworx will create the ZPL and sends it directly to the printer via ftp. (The reason for that is, that the hidden characters cannot be added in HYBRID-Mode – GS, RS, EOT)
  + 1. Illustration

****

* + 1. Use Cases

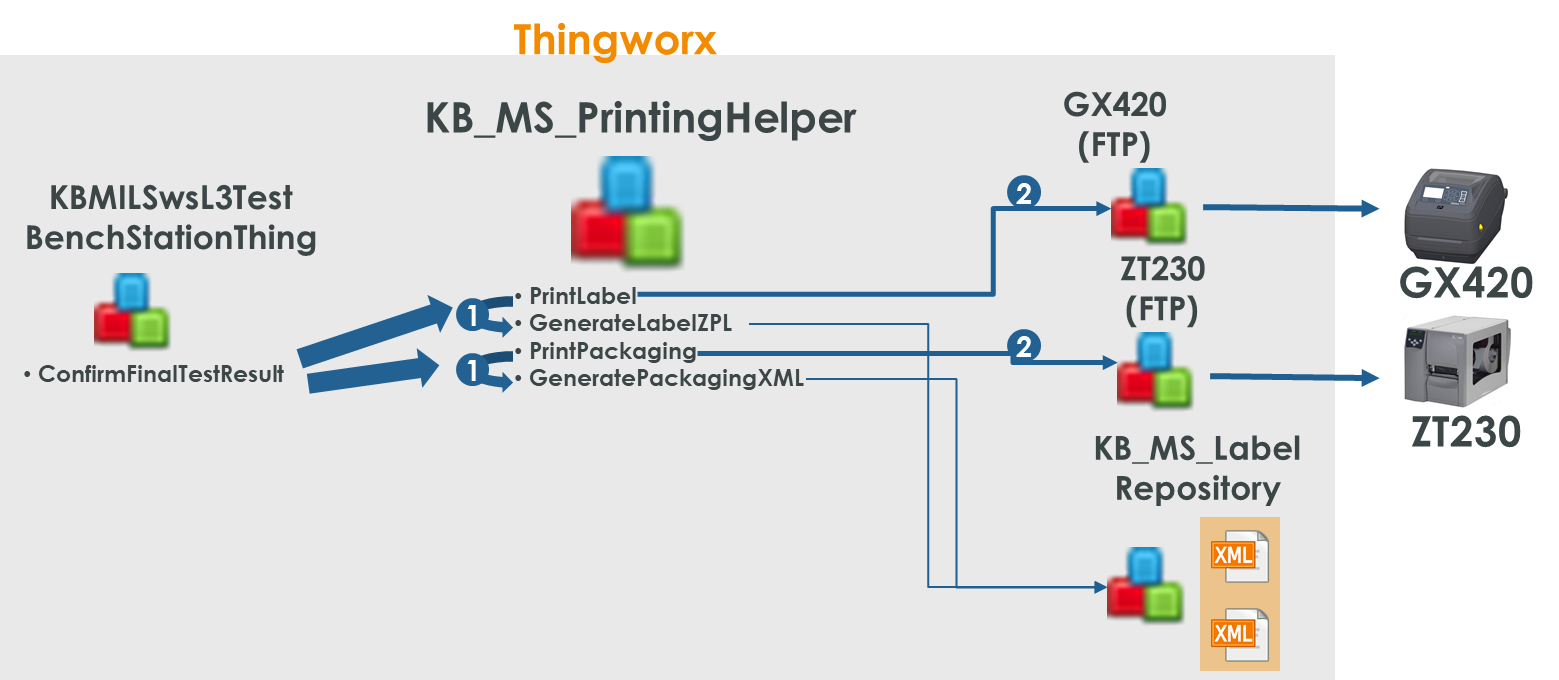
|  |  |
| --- | --- |
| Use Case |  |
| UC\_M04\_01 | Execute the xml-creation-service with a complete set of input parameters. (valid for both PackagingLabel and ProductLabel)   * Expected result:   + A xml file was created following the naming convention **Packaging\_<Line>\_<OrderNr>-<SerialNr>. xml**   or **Packaging\_<Line>\_< OrderNr>-<SerialNr>.xml** |
| UC\_M04\_01 | Execute the print-service with a complete set of input parameters. (valid for both PackagingLabel and ProductLabel)   * Expected result:   + The Packaging respectively the Product Label is printed on the printer |

* + 1. Implementation Details

**Instructions**

This section will be added during the actual implementation and reflects, what actually was created on the system at the end.

* + - 1. *Zebra printing – TWX implementation view*



To get the Label printing running it is necessary to have a virtual representation of the zebra printers. This is a FTP-Thing, because the Zebra-Printer itself acts as FTP-Server. Once we push a xml-file to the printer, it gets immediately printed, with the ZPL stored on the printer.

The trigger for printing is again coming from the TestBenchStation, as well as all the specific line information, e.g. the printer objects. All information is forwarded to the KB\_MS\_PrintingHelper to first create the xml Document and then send it to the printer. The xml-files of all Lines will be stored in the same Repository: KB\_MS\_Label Repository.

**Installation Instructions**

1. 1. Package

The necessary files for the print solution which are Thingworx objects are flagged with the following project: KB\_MS\_LinePrinting

Three Resources – PDFtoPrinter & PDF.FOP.Export & JSONtoXML.Export – which cannot be assigned to projects are also part of the solution. The PDFtoPrinter is a cmd Line utility which is used to call the PDFtoPrinter.exe application. The PDF.FOP.Export comes with a lot of Java-Classes from the Apache FOP project. They are used to create PDF Documents for the protocols. The JSONtoXML.Export to XML export is used, when a JSON needs to be transformed into XML, which is for example the case for the protocol creation, as an intermediate step.

Additionally the PDFtoPrinter.exe, the Apache FOP Files and the fop.xconf are part of the PrintingPackage and need to be placed at the System\_Repository.

For the layout of the protocol file a line specific xsl-File should be placed in the System Repository. It will define the layout for the pdf protocol.

**The version of the first Release, which is also deployed to production need to be**

* 1. Installation procedure

**Installation prerequisite:**

The following extensions need to be installed before installing the PackageForLinePrinting:

* + KBBasics
  + KBGlobalSmartKPI
  + FTP-Extension

To install the package an administration user need to import the extension package (zip file). All necessary files will be deployed to the right location.

The installation procedure/order is the following:

* + **Import KB\_MS\_LinePrinting\_Extension.zip as extension**

This installs the base data for the printing solution, independent from the first line, which was rolled out (Line 3)

* + **Import KB\_MS\_LinePrintingAdditionalEntities\_Extension.zip as extension**

This installs the objects required for Line 3 – based on the setup and IP addresses defined during the development of the project.

* + **Place the content of ToFileRepository.zip into the KB\_MS\_ProtocolRepository**This will place the “PDFtoPrinter.exe” application, the configuration File for FOP and the line specific protocol layout “protocolPDF\_Linea3.xsl” into the Thingworx space
  + **The last needed action is to import manually the Things\_AS400\_with\_queries.xml**  
    by using import from file – entity  
    The reason for the manual import is the failing upload as it was part of an extension. The additional step here will ensure a smooth installation
  + **Set up parameters on AS400**

To activate the AS 400 connection, the following parameters need to be set under configuration  
JDBC Driver Class Name = com.ibm.as400.access.AS400JDBCDriver

JDBC Connection String =  
jdbc:as400://10.6.4.11:446/AQS05DAT;naming=sql;errors=full;transaction isolation=none;date format=iso

connectionValidationString = AS400

Database Password = AS400

**Usage Instructions**

1. 1. Configuration

**If the solution should be rolled to different Lines, the following steps are necessary:**

1. **Creation/Configuration of the 3 asset objects**
   * **Things\_KB\_MS\_<Line>\_LabelPrinter**This Thing needs to be newly created and derived from the FTP-ThingTemplate. Therefore, the FTP-Extension is required. The Thing need to be configured with the IP address of the ZebraPrinter for ProductLabels.
   * **Things\_KB\_MS\_<Line>\_PackagingPrinter**

This Thing needs to be newly created and derived from the FTP-ThingTemplate. Therefore, the FTP-Extension is required. The Thing need to be configured with the IP address of the ZebraPrinter for the PackagingLabels.

* + **Things\_KBMILSwsL<LineNr>TestBenchStationThing**

Ideally this thing should already exist, because it is part of the Plant Asset Structure. For a valid testbench this Thing is based on a **KBLocalStationThingTemplate** and implements the **KBMILTestBenchReceiverTemplate**

The following properties at the TestBenchThing, need to be set accordingly:

A screenshot of a social media post

Description generated with very high confidence

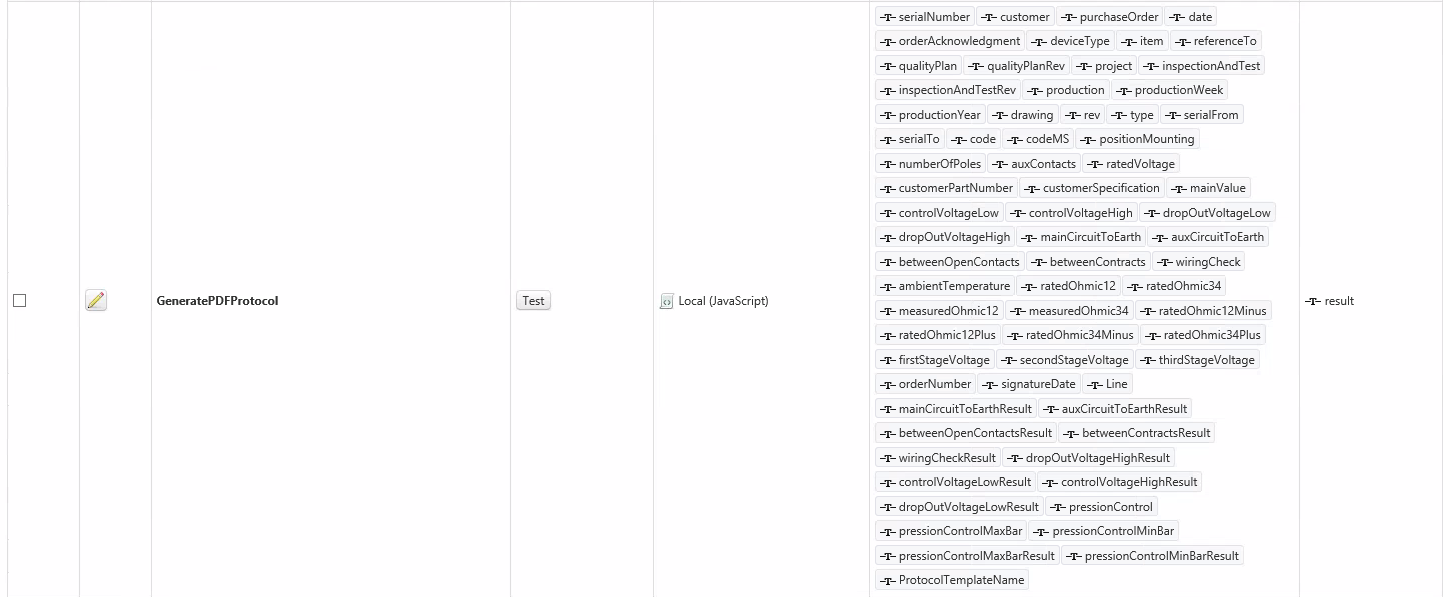
1. **Creation of a Layout XLS for the new Line**

Finally, a new protocol layout need to be developed by writing a line specific xsl file and link in the properties of the …TestBenchStationThing.

The design of the xsl is documented in this document: **PDF Printing - XSL templates.0.1.pdf**

The following properties can be used for the protocol, if the testbench sends data for them. Additional attributes need to be added in a newer version of the solution package.

Attributes available with solution package 1.0.0



1. **Upload Pack.ZPL on Packaging Printer and Upload Prod.ZPL on Label Printer**

This action will provide the printers with the right templates to print the barcode labels.

1. **Install the new printer on the Thingworx Server as network printer**

Make sure, that the printer name is unique. Set the printer name in the <Line>…TestBenchThing, so that the Line is connected to the right printer.

**Third Party Components**

**Instructions**

This section contains a list of third party modules, which will be used in the final delivery. The section cotains all relevant details about the used software pieces, including the version and the license agreement.

1. 1. PDFtoPrinter

**Name:** PDFtoPrinter: Command-line PDF printing

**Version:**

**Release Information:** <http://www.columbia.edu/~em36/pdftoprinter.html>

**License Agreement:** Email 09.05.2018 from Edward Mendelson

|  |
| --- |
| As far as I am concerned, you should feel free to use PDFtoPrinter for any purpose, including commercial ones, without any license or restriction. |

* 1. The Apache™ FOP Project

**Name:** The Apache™ FOP Project

**Version:** 2.2

**Release Informiation:** <https://xmlgraphics.apache.org/fop/download.html>

**License Agreement:** excerpt from FAQ

|  |
| --- |
| It **allows** you to:   * freely download and use Apache software, in whole or in part, for personal, company internal, or commercial purposes; * use Apache software in packages or distributions that you create. |

**Annex – Definition of the Product-Label out of the Exchange Column from AS400**

