Data Exchange Service

User Documentation

Valid from EWA Version 1.18.164.0

Content

1. Introduction

1.1 Description

EWA is a portal hosting the aftersales applications WIS/ASRA and EPC. WIS/ASRA is used to search and display workshop literature as well as service information. To search and display part information EPC is used.

Both applications provide a file based mechanism to export parts lists and job order data.

While this mechanism is efficient for single installations in a single workshop context, it doesn't work well for bigger installations with 80000+ users hosted on a single farm like Daimler's RetailFactory.

This document introduces the handling of shopping lists and job order data via web services and provides information of how to use and integrate the web service.

1.2 Abbreviations

Abbreviation	Description
DMS	Dealer management system
EPC	Electronic Parts Catalog: An application used to browse for replacement parts and to create shopping lists.
EWANAPI	Java based middleware to control WIS/ASRA/EPC
WIS/ASRA	Workshop Information System: application providing workshop literature and service information.
PAI	Proactive Infrastructure: A framework provided by Daimler to provide a uniformed implementation basis.
Web Start	A framework that allows users to start application software for the Java Platform using a Web browser.
YOY	Year over year
JO	Joborder.txt file that contains information exported from WIS/ASRA.
XFR	Shopping list file that contains spare part data exported from EPC. (Short for TransFeR).
SL	Shopping list
WAS	Websphere application server
GSS	Generic storage entry. Data construct used by WIS/ASRA/EPC to store data in EWAs central database.

2. The Data Exchange Service

2.1 Core Concept

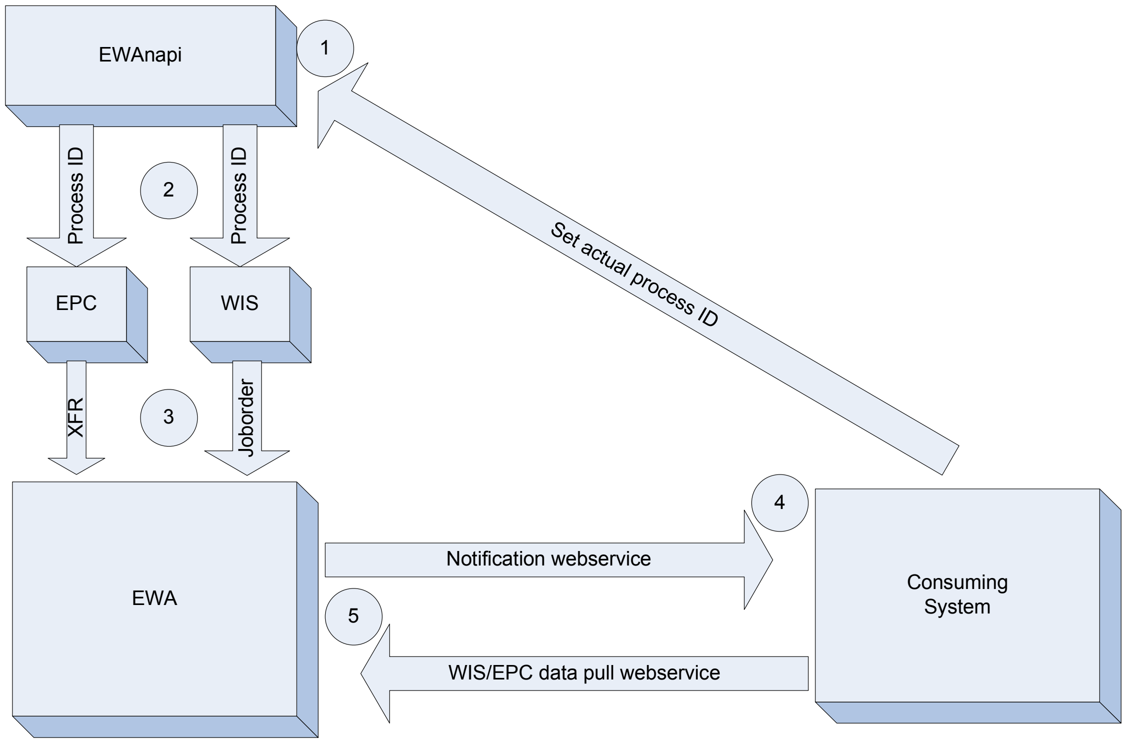
EWA provides the Job Order (JO) / Shopping List (SL) data via a central stateless web service. This web service is able to transfer the JO/SL data from the EWA server (used to run the client applications WIS/ASRA and EPC) to the consuming system that processes the data.
To use this Data Exchange Service, the client applications need to be initiated with the information that the web service should be used. This is done by starting the applications with EWA's commandline tool EWANAPI. This tool provides several arguments that are passed to WIS/ASRA or EPC (for further information about EWANAPI and its parameters, please refer to the EWANAPI documentation).
In order to use the Data Exchange Service, it is required to pass a process ID to the applications. Process identification data is needed for each JO/SL dataset, because every single user will have multiple JO/SL data sets stored in EWA's central core database.

Only if EPC or WIS/ASRA is started with a process ID, the Data Exchange Service is used by default to store JO/SL data. The client applications also pass this process ID to each other, if necessary.Example:

EPC starts WIS/ASRA and thereby passes its current process ID to WIS/ASRA. It also works the other way, thus when WIS/ASRA starts EPC.

After using the client applications and creating JO/SL data sets, the data is stored within the database of the EWA server. The Data Exchange Web Service is used to retrieve this data.

A web service notification mechanism is additionally implemented to inform the following systems, if a new JO/SL was stored in EWA's core database. In response to a notification, the consuming system queries EWA's web service to retrieve the data. The implementation of the Subscriber Web Service is optional for usage of the Data Exchange Service. If the implementation is not done, the consuming system has to pull the exchange data manually.



Step Number	Description
1	The consuming system starts EWANAPI and submits the actual process ID.
2	The process ID gets submitted to WIS/ASRA or EPC.
3	JO/SL data is saved to the EWA backend.
4	All subscribers (multiple consuming systems can be registered as subscribers) get a new data available notification for a particular process ID.
5	The subscriber created the process ID, recognizes the notification and gets the new dataset via a Web service offered by EWA.

2.2 Process ID generation

A Globally Unique Identifier (GUID) should be used as process IDs for initiating the Data Exchange mechanism, because different non related systems are going to create process IDs. The primary purpose of the GUID is to have a unique number.

Java supports the generation GUIDs with the standard java.util.UUID class. This mechanism has to be used by any system utilizing the exchange Web service. The UUID will be passed as a string to ensure operability of the Web service provider and client. The format of the String is defined by the java.util.UUID.toString() method, e.g. f47ac10b-58cc-4372-a567-0e02b2c3d479 (for more information see http://download.oracle.com/javase/6/docs/api/java/util/UUID.html#toString())

2.3 EWANAPI

EWANAPI is a middleware to trigger and start the EWA applications WIS/ASRA and EPC. It is written in Java and can be used via command line or Java Web Start.

WIS/ASRA/EPC needs to be informed about the actual process ID to store the exchange data under the right context. This is done via EWANAPI.

Therefore a new EWANAPI parameter is provided. This parameter ("processid") is optional to the existing parameters in order to guarantee backwards compatibility.

Example for an EWANAPI-call:

ewanapi.exe -application EPC-net -userid myuser -userpwd mypassword –processid f47ac10b-58cc-4372-a567-0e02b2c3d479

When executed, this command starts the EPC application (that is not yet running) and passes by the process-id "f47ac10b-58cc-4372-a567-0e02b2c3d479".

A new process-id can **not** be given to an EPC application that is already started. EPC has to be started by EWANAPI.

2.4 Web Service Descriptions

The Data Exchange Service consists of two separate Web Services:

- Subscriber Web Service
- Data Exchange Web Service

The ***Subscriber Web Service*** needs to be implemented by each consuming system that needs to be notified about any changes of the data. The Web service needs to be registered within the EWA server in order to be notified. The implementation of the Subscriber Web Service is optional for usage of the Data Exchange Service. If the implementation is not done, the consuming system has to contact the Data Exchange Web Service and try to get the data whenever the consumer suspects data to be available.

The ***Data Exchange Web Service*** is the service that needs to be called by the consuming system to get or delete the data.

2.4.1 Subscriber Web Service

EWA will notify subscriber applications when new JO/SL data is saved or deleted by EPC or WIS/ASRA. Subscribers must implement an own Web Service offering the void notify(String application, String processID) method.

The EWA server needs to be configured to inform the subscriber about changes. To do this, the EWA server will call the subscriber's web service. JAX-WS is used to implement this call from EWA server to subscriber web services. Please note that implementation of the Subscriber Web Service with JAX-WS is not mandatory, but recommended.

The WSDL of the service that subscribers must implement is as follows.

```
<?xml version='1.0' encoding='UTF-8'?>
<wsdl:definitions name="DataExchangeSubscriberService" targetNamespace="http://dataexchange.ewa.mercedes-
benz.com/" xmlns:ns1="http://cxf.apache.org/bindings/xformat" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://dataexchange.ewa.mercedes-benz.com/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
  <xs:schema targetNamespace="http://dataexchange.ewa.mercedes-benz.com/" version="1.0"
xmlns:tns="http://dataexchange.ewa.mercedes-benz.com/" xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="notify" type="tns:notify" />
   <xs:element name="notifyResponse" type="tns:notifyResponse" />
   <xs:complexType name="notify">
    <xs:sequence>
     <xs:element name="application" type="xs:string" />
     <xs:element name="processID" type="xs:string" />
    </xs:sequence>
   </xs:complexType>
   <xs:complexType name="notifyResponse">
    <xs:sequence />
   </xs:complexType>
  </xs:schema>
 </wsdl:types>
 <wsdl:message name="notify">
  <wsdl:part element="tns:notify" name="parameters">
  </wsdl:part>
 </wsdl:message>
 <wsdl:message name="notifyResponse">
  <wsdl:part element="tns:notifyResponse" name="parameters">
  </wsdl:part>
 </wsdl:message>
 <wsdl:portType name="DataExchangeSubscriberInterface">
  <wsdl:operation name="notify">
   <wsdl:input message="tns:notify" name="notify">
   </wsdl:input>
   <wsdl:output message="tns:notifyResponse" name="notifyResponse">
   </wsdl:output>
  </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="DataExchangeSubscriberServiceSoapBinding" type="tns:DataExchangeSubscriberInterface">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="notify">
   <soap:operation soapAction="urn:Notify" style="document" />
   <wsdl:input name="notify">
    <soap:body use="literal" />
   </wsdl:input>
   <wsdl:output name="notifyResponse">
    <soap:body use="literal" />
   </wsdl:output>
  </wsdl:operation>
 </wsdl:binding>
```

```
  <wsdl:service name="DataExchangeSubscriberService">
    <wsdl:port binding="tns:DataExchangeSubscriberServiceSoapBinding" name="DataExchangeSubscriberPort">
      <soap:address
location="https://webparts.example.com/DataExchangeSubscriber/services/DataExchangeSubscriberPort" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

2.4.1.1 Configuration of EWA instance

The administrator of an EWA instance creates a dataExchangeSubscribers_cfg.xml in the EWA_HOME/config/ directory and configures it with information about all subscribers.

- The endpoint address will be provided by a subscriber.
- Optionally supply a Java key stores containing the certificate to validate the HTTPS connection
- Optionally supply a second Java key store containing the user certificate (for certificate based authentication).
- Foreach store a password must be configured.

```
<?xml version="1.0" encoding="UTF-8"?>
<xml>
 <SECTION name="DataExchangeSubscribers">

  <!-- The name of this section can freely chosen and is used in logging statements -->
  <SECTION name="Consumer1">
   <!-- The Web Service endpoint address (URL) to the Web Service -->
   <PARAMETER name="endpointAddress">https://consumer1.example.com:8443/someURL</PARAMETER>

   <!-- If certificate based authentication is active on the Web Service Server then -->
   <!-- supply the path to a Java keystore file containing the client certificate.   -->
   <!-- This store must contain the private key. Leave empty if certificate          -->
   <!-- based authentication is not required.                                        -->
   <PARAMETER name="pathToKeystoreFile">C:/certificates/consumer1_user.jks</PARAMETER>
   <!-- Password for the pathToKeystoreFile. -->
   <!-- The password for the Certificate in the keystore must also have exactly the same password.-->
   <PARAMETER name="keystorePassword">password</PARAMETER>

   <!-- If the Web Service uses HTTPS and you want to supply your own trust CA or Certificate -->
   <!-- store then you need to supply the path to a Java keystore file containing it.         -->
   <!-- Leave empty if you are not using HTTPS or if you are using a web server               -->
   <!-- certificate signed by one of the renowned                                             -->
   <!-- Certificate Authorities (in the JRE cacerts file).                                    -->
   <PARAMETER name="pathToTruststoreFile">C:/certificates/consumer1_server.jks</PARAMETER>
   <!-- Password for the pathToTrustFile. -->
   <PARAMETER name="truststorePassword">password</PARAMETER>
  </SECTION>

  <!-- A second subscribe that will be notified of changes -->
  <SECTION name="Consumer2">
   <PARAMETER name="endpointAddress">https://consumer2.example.com:8443/someURL</PARAMETER>
   <PARAMETER name="pathToKeystoreFile">C:/certificates/consumer2_user.jks</PARAMETER>
   <PARAMETER name="keystorePassword">password</PARAMETER>
   <PARAMETER name="pathToTruststoreFile">C:/certificates/consumer2_server.jks</PARAMETER>
   <PARAMETER name="truststorePassword">password</PARAMETER>
  </SECTION>
 </SECTION>
</xml>
```

The Data Exchange Service must be activated in the EWA configuration (core_cfg.xml) by setting the start parameter to true.

```
<!– Settings for Data Exchange Service. You also need to supply a dataExchangeSubscribers_cfg.xml in the config directory
–>
<SECTION name="DataExchangeService">
   <!– Activate the notification service? true or false –>
   <PARAMETER name="start">false</PARAMETER>
   <!– How many ms will be slept if there were no outstanding notifications –>
   <PARAMETER name="sleepTimeOnIdleMillis">250</PARAMETER>
   <!– How many milliseconds will be slept if there was an error when sending a notification –>
   <PARAMETER name="sleepTimeOnErrorMillis">5000</PARAMETER>
   <!– The maximum number of outstanding notifications queued for a single subscriber –>
   <PARAMETER name="maxQueueSize">1000</PARAMETER>
   <!– The minimum age in days that an entry must be before it will be automatically deleted –>
   <PARAMETER name="minimumAgeDays">7</PARAMETER>
</SECTION>
```

2.4.2 Data Exchange Web Service

JAX-WS is used to implement the Web Services provided by EWA.

The web services must not be accessed by unauthorized third party users or systems. PAI defines certificate based authorization for security issues, therefore EWA will follow this example and use certificate based authentication over HTTPS (SSL/TLS).

The Data Exchange Web Service provides the following functions:

String get(String application, String processID) throws DataNotFoundException
The get method returns the exchange data String containing the data in Job Order or XFR format depending on the exporting application.

Void delete(String application, String processID) throws DataNotFoundException
The delete method explicit deletes the exchange data for an application and process ID. If no data exist, then a DataNotFoundException is thrown.

The following is the WSDL description of the Data Exchange Web Service provided by EWA

```
<?xml version="1.0" ?>
<wsdl:definitions name="DataExchangeService" targetNamespace="http://dataexchange.ewa.mercedes-benz.com/"
xmlns:ns1="http://schemas.xmlsoap.org/wsdl/soap/http" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://dataexchange.ewa.mercedes-benz.com/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
  <xs:schema attributeFormDefault="unqualified" elementFormDefault="unqualified"
targetNamespace="http://dataexchange.ewa.mercedes-benz.com/" xmlns:tns="http://dataexchange.ewa.mercedes-
benz.com/" xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="delete" type="tns:delete"></xs:element>
   <xs:element name="deleteResponse" type="tns:deleteResponse"></xs:element>
   <xs:element name="get" type="tns:get"></xs:element>
   <xs:element name="getResponse" type="tns:getResponse"></xs:element>
   <xs:complexType name="delete">
    <xs:sequence>
     <xs:element name="application" type="xs:string"></xs:element>
     <xs:element name="processID" type="xs:string"></xs:element>
    </xs:sequence>
   </xs:complexType>
   <xs:complexType name="deleteResponse">
    <xs:sequence></xs:sequence>
   </xs:complexType>
   <xs:complexType name="get">
    <xs:sequence>
     <xs:element name="application" type="xs:string"></xs:element>
     <xs:element name="processID" type="xs:string"></xs:element>
    </xs:sequence>
   </xs:complexType>
   <xs:complexType name="getResponse">
    <xs:sequence>
     <xs:element name="exchangeData" type="xs:string"></xs:element>
    </xs:sequence>
   </xs:complexType>
   <xs:element name="DataNotFoundException" type="tns:DataNotFoundException"></xs:element>
   <xs:complexType name="DataNotFoundException">
    <xs:sequence></xs:sequence>
   </xs:complexType>
  </xs:schema>
 </wsdl:types>
 <wsdl:message name="deleteResponse">
  <wsdl:part element="tns:deleteResponse" name="parameters">
  </wsdl:part>
 </wsdl:message>
 <wsdl:message name="get">
  <wsdl:part element="tns:get" name="parameters">
  </wsdl:part>
 </wsdl:message>
 <wsdl:message name="getResponse">
  <wsdl:part element="tns:getResponse" name="parameters">
  </wsdl:part>
 </wsdl:message>
 <wsdl:message name="delete">
  <wsdl:part element="tns:delete" name="parameters">
  </wsdl:part>
 </wsdl:message>
 <wsdl:message name="DataNotFoundException">
  <wsdl:part element="tns:DataNotFoundException" name="DataNotFoundException">
  </wsdl:part>
 </wsdl:message>
```

```
  <wsdl:portType name="DataExchangeWebServiceInterface">
    <wsdl:operation name="delete">
      <wsdl:input message="tns:delete" name="delete">
      </wsdl:input>
      <wsdl:output message="tns:deleteResponse" name="deleteResponse">
      </wsdl:output>
      <wsdl:fault message="tns:DataNotFoundException" name="DataNotFoundException">
      </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="get">
      <wsdl:input message="tns:get" name="get">
      </wsdl:input>
      <wsdl:output message="tns:getResponse" name="getResponse">
      </wsdl:output>
      <wsdl:fault message="tns:DataNotFoundException" name="DataNotFoundException">
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="DataExchangeServiceSoapBinding" type="tns:DataExchangeWebServiceInterface">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"></soap:binding>
    <wsdl:operation name="delete">
      <soap:operation soapAction="urn:Delete" style="document"></soap:operation>
      <wsdl:input name="delete">
        <soap:body use="literal"></soap:body>
      </wsdl:input>
      <wsdl:output name="deleteResponse">
        <soap:body use="literal"></soap:body>
      </wsdl:output>
      <wsdl:fault name="DataNotFoundException">
        <soap:fault name="DataNotFoundException" use="literal"></soap:fault>
      </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="get">
      <soap:operation soapAction="urn:Get" style="document"></soap:operation>
      <wsdl:input name="get">
        <soap:body use="literal"></soap:body>
      </wsdl:input>
      <wsdl:output name="getResponse">
        <soap:body use="literal"></soap:body>
      </wsdl:output>
      <wsdl:fault name="DataNotFoundException">
        <soap:fault name="DataNotFoundException" use="literal"></soap:fault>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="DataExchangeService">
    <wsdl:port binding="tns:DataExchangeServiceSoapBinding" name="DataExchangePort">
      <soap:address location="https://ewafarm.example.com/EWA-net/services/DataExchangePort"></soap:address>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

2.5 Joborder / Shoppinglist data

The data returned by the web service equals the data contained within the offline files. A description of the files and their content can be found in the ASRA joborder description and the EPC shopping list documentation.

2.6 EWA Database

EWA automatically deletes exchange data that exceeds a defined amount of time. The default storage time is 7 days. This timeframe is freely configurable in EWA to maintain the flexibility to adjust this timeframe.

```
<SECTION name="DataExchangeService">
  ...
  <!-- The minimum age in days that an entry must be before it will be automatically deleted -->
  <PARAMETER name="minimumAgeDays">7</PARAMETER>
</SECTION>
```

3. Implementation Advices

3.1 Retail Factory Integration

The TSS Retail Factory (https://retailfactory.mercedes-benz.com) requires that every HTTP request includes the EWA username.

To pass the username an HTTP Cookie must be set in the Web Service request. The following code snippet shows the JAX-WS code required to do this. The implementer must implement the getEWAUserForProcessID (...) method to get the EWA username associated with the ProcessID being processed.

Note:

- The NEW CODE, below, is necessary for both calls to the get method and to the delete method of the Data Exchange Web Service
- The EWA username must be passed to the cookie in lowercase letters
- The Cookie name is EWAnetUsername
- You must implement this extension, if your client is connecting to the Retail Factory

```
// NEW CODE

// Add the EWAnetUsername cookie for connecting to RetailFactory
String ewaUsername = getEWAUserForProcessID(processID);
ewaUsername = ewaUsername.toLowerCase(); // Username must be in lower case
List<String> cookies = Collections.singletonList("EWAnetUsername=" + ewaUsername);
Map<String, List<String>> requestHeaders = Collections.singletonMap("Cookie", cookies);
BindingProvider bindingProvider = (BindingProvider)port;
bindingProvider.getRequestContext().put(MessageContext.HTTP_REQUEST_HEADERS, requestHeaders);

// END OF NEW CODE

// Sample call to the get method.
String exchangeData = port.get(application, processID);
```

3.2 JAX-WS

If possible the implementer of the subscriber Web Service and the client to access the EWA Data Exchange Web Service should use Java, JAX-WS and Apache CXF (Version 2.2.9 was used for the EWA implementation) to ensure compatibility. Compatibility with other Web Service implementation is not guaranteed, but might be possible.

3.3 Subscribers

When a subscriber is notified of a change, it should return the response as soon as possible. It should not do any long running actions before responding as this could cause a timeout on the EWA side. It is suggested that long running actions be handed off to worker threads allowing the thread being notified to return the response almost immediately.

4. Operational scenarios

The JO/SL data exchange Web Services are designed to be used for backend to backend data exchange.
The Web Service implementation can be operated with client certificates in combination with HTTPS or plain HTTP.

Client based certificates need to be issued by a Daimler authorized certificate authority, such as VeriSign or Thawte. The certificates must be deployed on both back-ends (EWA server and consuming system). A certificate normally expires after one to three years and must be replaced on both back-ends after the expiration date. If one of the certificates is expired and is not replaced on time, the Web Services are going to be inoperable.
The HTTPS / certificate based solution can be used over the internet. The proxy/firewall constellations of the back-ends must be taken into account in this scenario.

If HTTP is used, the cost of the certificates as well as the error source of expired certificates can be avoided. Avoiding HTTPS will also provide a slight performance advantage. However a secured network like DCN has to be used to guarantee a secured data exchange.

Daimler AG
GSP/ORM
D-70546 Stuttgart
http://aftersales.mercedes-benz.com