DATEX II V2.0

SOFTWARE DEVELOPERS GUIDE

Document version: 1.2

30th of June 2011

European Commission

Directorate General for Mobility and Transport

Document control:

Prepared by :			
	Date	Comment	Version
ES5 Technical Group	30/06/2011	Updated by Jonas JäderBerg, Jean-Philippe MECHIN, Daniel ODDON and Stéphane VERGIER	1.2

Reviewed by :			
	Date	Comment	Version
ES5 Technical Group	30/06/2011		1.2

Approved by :			
	Date	Comment	Version
ES5 Technical Group	30/06/2011	Authorization for publication	1.2

TABLE OF CONTENTS

Introduction

1. Introduction

1.1. Objectives

This document is mainly targeted towards software developers who are in charge of DATEX II developments.

Software developers should first read the **[User Guide]** which is an introduction to DATEX II and will assist in the understanding of the other DATEX II documents.

1.2. Document structure

The document is structured into seven chapters with the following content:

- Chapter 1 “Introduction” gives a brief overview of the document’s objectives and structure,
- Chapter 2 “DATEX II exchange system” gives information on useful documents, architecture and operating modes,
- Chapter 3 “DATEX II data” is related to the exchange data content,
- Chapter 4, 5 and 6 give explanations on the different operating modes and profiles
    - Chapter 4 “Client Pull with http” describes the operating mode “Client Pull” with only http usage
    - Chapter 5 “Client Pull with Web Services” describes the operating mode “Client Pull” with usage of Web Services over http
    - Chapter 6 “Supplier Push with Web Services” describes Push systems with Web Services over http
- Chapter 7 provides answers to some frequently asked questions.

1.3. DATEX II reference documents

DATEX II documents can be downloaded from the DATEX II web site http://www.datex2.eu.

Reference in this document	DATEX II document	Document version	Date
[User Guide]	DATEX II v2.0 User Guide	1.2	30/06/2011
[Modelling methodology]	DATEX II v2.0 Modelling methodology	2.0	30/06/2011
[Data model]	DATEX II v2.0 Data model	2.0	30/06/2011
[Dictionary]	DATEX II v2.0 Data definitions	2.0	30/06/2011
[XML schema]	DATEX II v2.0 XML schema	2.0	25-05-2011
[Exchange PSM]	DATEX II v1.0 Exchange Platform Specific Model	2.0	30-06-2011
[Schema generation tool]	DATEX II v2.0 XML schema generation tool guide	2.0	25-05-2011

DATEX II exchange system

2. DATEX II exchange system

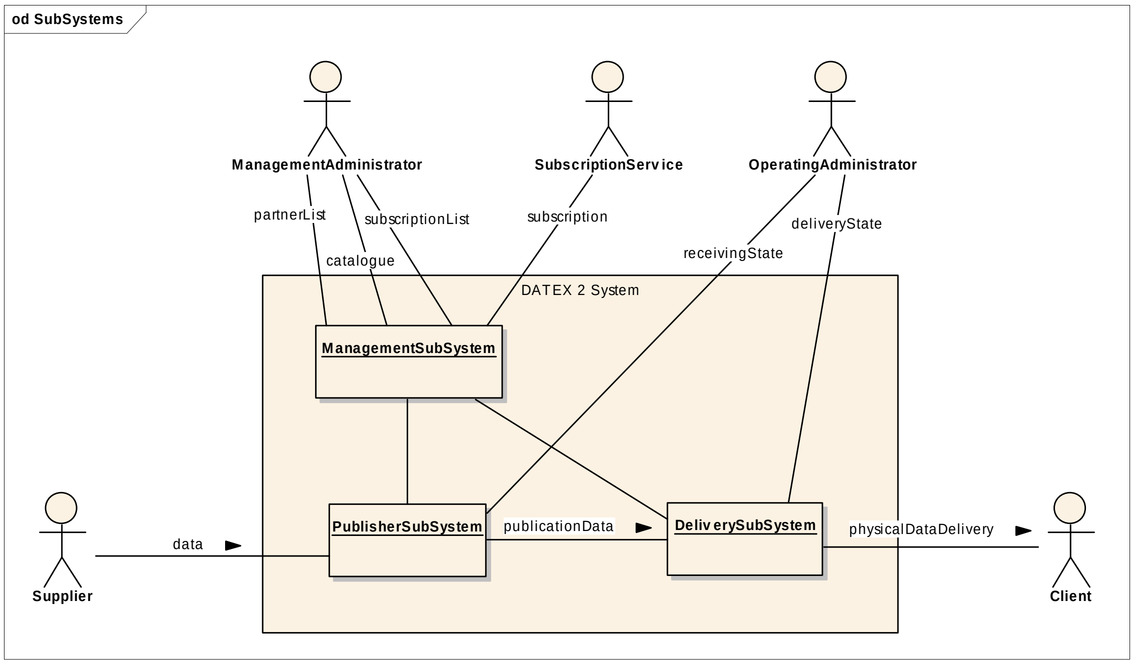
2.1. Useful documents

The [**User Guide**] has a chapter called “Exchange specifications” which introduces DATEX II exchanges.

The reference document [**Exchange PSM** deals with the exchange process. It presents, definitions, subsystems, use cases, metadata, and operating modes; it is specific for one platform, which has been chosen for DATEX II, called “Web Services over http”..

2.2. Architecture and main subsystems

For this architecture, an important diagram is the analysis diagram “Subsystems”:



Among the 3 subsystems, DATEX II only describes the delivery subsystem: that means that each developer is free to build:

- On the Supplier side
  - The Publisher subsystem which elaborates data to be delivered (the Payload),
  - The way to feed the Publisher subsystem (for instance, depending on the hardware and software architecture, a specific interface with a Traffic Centre, a local database, …),
  - The management subsystem:
    - For partner list management, if used (mainly, for authentication),
    - For catalogue management, if any (publication types available, locations available, …)
    - For subscriptions management, if any (for instance, locally by an administrator or with a Client browser with dynamic HTML pages, …),
  - The exchange administration, if any (survey of the delivery state, link with the Publisher system, …)
- And, on the Client side, the way to use DATEX II data (man machine interface, database …).

2.3. Operating modes

The choice of operating modes to be developed is fundamental.

There are 3 possible **operating modes** for data delivery:

1	Publisher Push on occurrence	Data delivery is initiated by the Publisher every time data is changed
2	Publisher Push periodic	Data delivery is initiated by the Publisher on a cycle time basis
3	Client Pull	Data delivery is initiated by the Client and data is returned as a response

The choice has to be made with users because:

- it can be related to data publication types,
- it has an impact on the transmission delay when data is available,
- it has an impact on complexity (for instance, “Publisher Push on occurrence” delivers data as soon as it is available, but is more complex to develop for situation publications, for instance).

2.4. Exchange profiles

For the "Client Pull" operating mode, two implementation profiles have been defined for implementing this operating mode over the Internet: by direct use of the HTTP/1.1 protocol or via Web Services over HTTP.

For the "Supplier Push" operating mode, one platform has been defined using Web Services over HTTP.

The common corresponding document, describing all operating modes and both profiles for Client Pull as well as their interoperability, is the [**Exchange PSM**].

Following a chapter concerning the data to be exchanged, the subsequent chapters detail the exchange process for each kind of system:

- Client Pull with http only,
- Client Pull with Web Services,
- Supplier Push with Web Services.

DATEX II data

3. DATEX II data

3.1. Data model

Data to be exchanged are described in the DATEX II data model. The [**User Guide**] helps understand this model.

3.2. Data dictionary

All DATEX II definitions are included in the data model and can be read by the Enterprise Architect tool but, to be more readable and normalized in a document, definitions have been automatically extracted from the data model with a special tool in order to constitute a dictionary.

At the present time, this "dictionary" is available on the Web Site with HTML pages and allows navigation between classes, attributes, enumerations …

The methodology and the semantics of the modelling structures which have been used for modelling is described in document [**Modelling methodology**].

3.3. XML schema generation tool

The data model is defined at a level that is independent from technologies, consistent with MDA (model driven architecture) principles. However for DATEX II a tool has been built which is able to generate an XML schema directly from the data model.

The tool is described in the document [**XML schema generation tool**]. It is available on the Web site.

3.4. XML Schema

The schema generation process is configurable and allows for data definitions to be included in the schema and is therefore self-descriptive. Questions concerning the schema can be raised via the DATEX II web site.

Client Pull with http

4. Client Pull with http

4.1. Content

This chapter describes the fundamental details of a service employing “Client Pull with http” which is the simplest implementation of DATEX II:

- There is no subscription,
- The Publisher periodically creates the payload,
- The payload is available on one URL for all authorized Clients.

On the Supplier side, different sets of data can be available via different addresses (URLs). In this case, the data is considered to be available via different services.

4.2. Supplier system

The Publisher subsystem elaborates data conforming to the DATEX II XML schema and the delivery subsystem makes it available on one URL.

The Supplier has to respect the clauses defined in document [**Exchange PSM**] at chapter “Client Pull simple http server profile”.

The Clients of this Supplier system can be:

- Client systems with simple http GET requests (see next paragraph),
- Client systems using Web Services (refer following chapter).

4.3. Client system

Periodically:

- The Client requests data (with http GET) to Suppliers using only http or to Suppliers using Web Services, at the specified URL.
- After reception, the Client manages data and errors from the Suppliers.

The Client has to respect the clauses defined in document [**Exchange PSM**] at chapter “Client Pull simple http server profile”.

4.4. Interoperability http server <-> Clients using Web services

In order to be interoperable with Clients using Web Services, the Supplier system must provide a (static) SOAP wrapper with the according SOAP namespace definitions.

The following sample illustrates the envelope of the d2LogicalModel XML node:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:env="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xdt="http://www.w3.org/2004/07/xpath-datatypes"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      env:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
                                    http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Body>
            <d2lm:d2LogicalModel xmlns:d2lm="http://datex2.eu/schema/2/2_0"
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                        xsi:schemaLocation="D2LogicalModel DATEXIISchema_2_2_0.xsd">
                        <xs:schema xmlns:D2LogicalModel="http://datex2.eu/schema/2/2_0"
                        xmlns:xs="http://www.w3.org/2001/XMLSchema"
                                    targetNamespace="http://datex2.eu/schema/2/2_0"
                                    elementFormDefault="qualified"
                                    attributeFormDefault="unqualified">
                        <d2lm:exchange>
                              ...
                      </d2lm:exchange>
                      <d2lm:payloadPublication>
                              ...
                              </d2lm:payloadPublication>
                  </d2lm:d2LogicalModel>
         </soapenv:Body>
</soapenv:Envelope>
```

1.1. Interoperability http port

For security reasons, more and more server centres prohibit the use of other TCP ports than 80.

To ensure a high level of interoperability and also to reduce the management on firewall, it would be wise to implement the standard http port 80.

Client Pull with Web Services

5. Client Pull with Web Services

5.1. Content

This chapter describes the fundamental details of a service employing “Client Pull with Web Services”:

- There is no online subscription,
- The Publisher periodically creates the payload,
- The payload is available on one URL for all authorized Clients.

On the Supplier side, different sets of data can be available via different addresses (URLs). In this case, the data is considered to be available via different services.

5.2. Supplier system

Recommendation: A Publisher can periodically create a payload and makes it available for the Supplier Push delivery subsystem. It may be more efficient than building a payload each time a Client requests data, although this depends on your system architecture.

The delivery subsystem is built with the DATEX II “Pull WSDL” which is available on the DATEX II Web site.

Two frameworks have already been used for tests, one based on Apache foundation Axis 1.4.1 and the other on Microsoft .Net 2.0.

The Axis2 has also been published and its use is slightly different than with Axis 1 as the philosophy is modified. It requires a good level of practice of this environment. Nevertheless the Axis 2 solution is recommended for sustainability as it is being extensively used for large projects worldwide.

A third solution, JAX WS, is now available and included in Java development platform JDK 1.6 . That platform is used directly with the standard notation. The implementation is quite direct and easier than with Axis.

This last framework can be completed with the CXF library from The Apache Software Foundation. As with the direct use of JDK, CXF will pose less constraint than Axis and will facilitate the integration with Spring framework and also for developing systems based on Enterprise Service Bus like ServiceMix.

The PHP programming language can also be used: since version 5 Web Services are natively supported. The SOAP extension has to be activated by adding the line extension=php_soap.dll in the php.ini configuration file.

Documentation to help implementation can be found on http://www.php.net/manual/en/ref.soap.php

5.2.1. Advice for building a Supplier with Apache foundation Axis 1.4.1[1]

- Use WSDL2Java tool with option - - server-side to create Web Service skeleton Java classes.

- Edit the impl.java class generated by WSDL2Java tool. For instance:

```
/**
 * ClientPullSoapBindingImpl.java
* This file was auto-generated from WSDL by the Apache Axis 1.4.1 WSDL2Java emitter.
 */
package org.datex2.wsdl.ClientPull;


import java.util.Calendar;
import org.apache.axis.MessageContext;
import org.apache.axis.encoding.DeserializationContext;
import org.apache.axis.encoding.Deserializer;
```

[1] This example is maintained is this document because already deployed solutions don’t move to Axis2 and in case of maintenance or extension of such systems, it could be useful for the developers to know the reason of the header of the classes.

```
import org.apache.axis.message.EnvelopeHandler;
import org.apache.axis.message.SOAPHandler;


public class ClientPullSoapBindingImpl implements org.datex2.wsdl.ClientPull.ClientPullInterface{


   public D2LogicalModel_pkg.D2LogicalModel getDatex2Data() throws java.rmi.RemoteException {
// body of Web Service method
   }
}
```

Deploy the Web Service by launching the following command on deploy.wsdd generated by WSDL2java: java -cp %AXISCLASSPATH% org.apache.axis.Client.AdminClient deploy.wsdd

Check the Web Service with the browser, for instance:

http://localhost:8080/axis/services/clientPullSoapEndPoint?method=getDatex2Data?WSDL

5.2.2. Advice for building a Supplier with Microsoft .Net

- Use wsdl.exe with option /serverInterface and XML schema name to create Web Service skeleton, for instance with the following command:

```
wsdl /serverInterface Pull.wsdl DATEXIISchema_2_2_0.xsd
```

- Edit the “impl” file (for instance clientPullServiceImpl.asmx) generated by wsdl.exe. For instance:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;
using System.Xml;
using System.IO;
using System.Xml.Serialization;


namespace clientPullServiceProjet
{
   /// <summary>
   /// clientPullService
   /// </summary>
   [WebService(Namespace="http://microsoft.com/webservices/")]
   public class clientPullServiceImpl : clientPullService
   {
           public clientPullServiceImpl()
           {
                   //CODEGEN : This call is required by the ASP.NET webservices designer.
                   InitializeComponent();
           }
           #region Code generated by Component designer


           //Required by the Web services Designer
```

```
		private IContainer components = null;

		/// <summary>
		/// Required method for the Designer hand over – Don't modify
		/// method content with the code editor
		/// </summary>
		private void InitializeComponent()
		{
		}
		/// <summary>
		/// Clean up of ressources.
		/// </summary>
		protected override void Dispose( bool disposing )
		{
			if(disposing && components != null)
			{
				components.Dispose();
			}
			base.Dispose(disposing);
		}

		#endregion

		[System.Web.Services.WebMethodAttribute()]
    [System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://datex2.org/wsdl/clientPul
l/getDatex2Data", Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Bare)]
		[return: System.Xml.Serialization.XmlElementAttribute("d2LogicalModel",
Namespace="D2LogicalModel")]
		public override D2LogicalModel getDatex2Data()
		{
// enter here body method
		}
    }
}
```

- Check the Web Service with the browser (for instance: http://localhost/service_path_and_name.asmx?WSDL)

5.2.3. Clients

The Clients of these Supplier systems can be:

- Client systems with simple http GET requests (see previous chapter),
- Client systems using Web Services (refer following paragraph).

5.2.4. Interoperability Web Services Suppliers <-> simple http Clients

If the simple http Client follows recommendations described in the previous chapter, there is nothing more to develop on the Web Services Supplier side, for interoperability.

5.3. Client system

The Client is built with the DATEX II “Pull WSDL” which is available on the DATEX II Web site.

Two frameworks have already been used for tests, based on Axis and the other on Microsoft .Net.

Other solutions are now available for new systems and are based directly on JAX WS the native Java API for XML Web Services included in Java JDK 1.6 combined or not with the CXF library which facilitates the integration with Spring framework and also for development based on Enterprise Service Bus like ServiceMix.

5.3.1. Advice for building a Client with Apache foundation Axis 1.4.11[2]

Generate the proxy with WSDL2java tool.

Create a Client by using proxy classes. For instance:

```
package org.datex2.wsdl.ClientPull;
import java.rmi.RemoteException;
import java.util.Calendar;
import java.util.Date;
import javax.xml.rpc.ServiceException;
import org.apache.log4j.Logger;
import D2LogicalModel_pkg.D2LogicalModel;
import D2LogicalModel_pkg.DateTime;
import D2LogicalModel_pkg.Exchange;
public class ClientPull{
    private static Logger logger = Logger.getLogger(ClientPull.class);
    public static void main(String[] args){
            ClientPullInterface ClientPullInterface = null;
            ClientPullService ClientPullService = new ClientPullServiceLocator();
            try {
                    ClientPullInterface = ClientPullService.getClientPullSoapEndPoint();
            }
            catch(ServiceException serviceexception){
                    System.out.println("Error Service Exception " + serviceexception.getMessage());
            }
            try {
                    D2LogicalModel logicalModel = null;
                    System.out.println("CLIENTPULLINTERFACE " + ClientPullInterface.toString());
                    logicalModel = ClientPullInterface.getDatex2Data();
                    System.out.println("LOGICALMODEL " + logicalModel.toString());
                    Exchange exchange = logicalModel.getExchange();
                    if (exchange != null) {
            System.out.println("EXCHANGE SUPPLID " + exchange.getSupplierIdentification());
                            Calendar calendar = exchange.getDeliveryTime().get_value();
            System.out.println("DELIVERY TIME " + new Date(calendar.getTimeInMillis()));
                    }
            }
```

[2] This example is maintained is this document because already deployed solutions don’t move to Axis2 and in case of maintenance or extension of such systems, it could be useful for the developers to know the reason of the header of the classes.

```
                catch(RemoteException rmiError){
                          rmiError.printStackTrace();
                }
    }
}
```

5.3.2. Advice for building a Client with Microsoft .Net :

With Visual Studio, for instance, there are two ways to build the client:

- First way : connection to the deployed Supplier Web Service through http:
    - On Visual Studio Working Project, add a “Web reference” and link it to the Web Service URL: proxy is automatically created,
    - Client class creation with Visual Studio, for instance:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Xml.Serialization;

namespace clientPullServiceConsumer
{
    /// <summary>
    /// Summary description of Form1.
    /// </summary>
    public class clientPullServiceConsumer : System.Windows.Forms.Form
    {
                private System.Windows.Forms.Button button1;
                private System.Windows.Forms.ListBox listBox1;
                /// <summary>
                /// Variable required by designer.
                /// </summary>
                private System.ComponentModel.Container components = null;

                public clientPullServiceConsumer()
                {
                          //
                          // Required for the Windows Forms Designer hand over
                          //
                          InitializeComponent();

                          //
                          // TODO : add builder code after having called InitializeComponent
                          //
```

```
		}

		/// <summary>
		/// Clean up of ressources.
		/// </summary>
		protected override void Dispose( bool disposing )
		{
			if( disposing )
			{
				if (components != null)
				{
					components.Dispose();
				}
			}
			base.Dispose( disposing );
		}

		#region Code générated by Windows Form Designer
		/// <summary>
		/// Required method for the Windows Forms Designer hand over – Don't mofify the method
		/// content with the code editor
		/// </summary>
		private void InitializeComponent()
		{
			this.button1 = new System.Windows.Forms.Button();
			this.listBox1 = new System.Windows.Forms.ListBox();
			this.SuspendLayout();
			//
			// button1
			//
			this.button1.Location = new System.Drawing.Point(16, 8);
			this.button1.Name = "button1";
			this.button1.TabIndex = 0;
			this.button1.Text = "appel WS";
			this.button1.Click += new System.EventHandler(this.button1_Click);
			//
			// listBox1
			//
			this.listBox1.Location = new System.Drawing.Point(16, 40);
			this.listBox1.Name = "listBox1";
			this.listBox1.Size = new System.Drawing.Size(256, 134);
			this.listBox1.TabIndex = 2;
			//
			// clientPullServiceConsumer
			//
			this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
```

```
            this.ClientSize = new System.Drawing.Size(280, 182);
            this.Controls.Add(this.listBox1);
            this.Controls.Add(this.button1);
            this.Name = "clientPullServiceConsumer";
            this.Text = "Form1";
            this.ResumeLayout(false);

        }
        #endregion

        /// <summary>
        /// Main application entry point.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.Run(new clientPullServiceConsumer());
        }

        private void button1_Click(object sender, System.EventArgs e)
        {
            localhost.clientPullServiceImpl myWS = new localhost.clientPullServiceImpl();

            //object recover
            localhost.D2LogicalModel d2LogicalModel = myWS.getDatex2Data();

            //object data display
            if (d2LogicalModel != null)
            {
                listBox1.Items.Add(d2LogicalModel.ToString());
                localhost.Exchange exc = d2LogicalModel.exchange;
                if (exc != null)
                {
                    listBox1.Items.Add("EXCHANGE NOT NULL");
                    listBox1.Items.Add(exc.supplierIdentification);
                }
            }
            else
            {
                listBox1.Items.Add("OBJET D2LOGICALMODEL NULL");
            }
            // serialization of recovered object
            XmlSerializer            myXmlSerializer            =            new
XmlSerializer(typeof(localhost.D2LogicalModel));
```

```
                        StreamWriter myWriter = new StreamWriter("c:/TEMP/D2LogicalModel Obstruction
with SpeedLimit.xml");
                        myXmlSerializer.Serialize(myWriter,d2LogicalModel);
              }
    }
}
```

- Second way: proxy generation thanks to wsdl.exe:
  - Use wsdl.exe with option XML schema name to create the proxy, for instance with the following command:
    - wsdl Pull.wsdl DATEXIISchema_2_2_0.xsd
  - Add generated classes to the Working Project
  - Create the client class (refer to previous example)

5.3.3. Advice for building a Client with Java and ServiceMix

After generating the proxy with the technical solution better known by the development team.

In the following example, the classes offer the capability to access a configuration file in order to manage the proxy link more dynamically:

```
package i2.application.tipi.smx.su.ws.consumer;


import i2.application.tipi.smx.components.ws.consumer.WsConsumerEndpoint;
import i2.application.tipi.smx.components.ws.consumer.interceptor.ArchiverInInterceptor;
import i2.application.tipi.smx.components.ws.consumer.interceptor.ContextInInterceptor;
import i2.application.tipi.smx.components.ws.consumer.interceptor.WSPolicy;
import i2.application.tipi.smx.shared.configuration.GlobalConfiguration;


import java.util.ArrayList;
import java.util.List;


import javax.xml.namespace.QName;


import org.apache.servicemix.soap.api.Policy;
import org.apache.servicemix.soap.core.AbstractInterceptor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import org.springframework.core.io.Resource;


@Configuration
@Import(GlobalConfiguration.class)
public class WsConsumerConfiguration {
   @Value("#{globalProperties['ws.datex2.objectName']}")
   private String objectName;


   @Value("#{globalProperties['ws.datex2.locationUri']}")
   private String locationURI;
```

```
    @Value("#{globalProperties['ws.datex2.publicLocationUri']}")
    private String publicLocationURI;

    @Value("classpath:Push_v2_0.wsdl")
    private Resource wsdl;

    /*
    <http:wsconsumer-endpoint                    service="supplierPush:supplierPushService"
targetService="smx:router"
        endpoint="supplierPushSoapEndPoint"
        locationURI="${ws.datex2.locationUri}"
        objectName="${ws.datex2.objectName}"
        targetServiceError="smx:error-in"
        publicLocationURI="${ws.datex2.publicLocationUri}"
        wsdl="classpath:Push_v2_0.wsdl"
        acknowledge="classpath:acknowledge.xml"
        useJbiWrapper="false"
        xsdBaseURI="${ws.datex2.xsdBaseURI}" >
            <property name="policies">
                <list>
                    <ref local="wsPolicy"/>
                </list>
            </property>
        </http:wsconsumer-endpoint>
    */

    @Bean
    public WsConsumerEndpoint wsConsumerEndpoint() {
        final WsConsumerEndpoint endpoint = new WsConsumerEndpoint();

        endpoint.setEndpoint("supplierPushSoapEndPoint"); //$NON-NLS-1$
        endpoint.setService(new                    QName("http://datex2.eu/wsdl/supplierPush/2_0",
"supplierPushService")); //$NON-NLS-1$ //$NON-NLS-2$
        endpoint.setTargetService(new QName("smx", "router")); //$NON-NLS-1$ //$NON-NLS-2$
        endpoint.setTargetServiceError(new QName("smx", "error-in")); //$NON-NLS-1$ //$NON-NLS-
2$
        endpoint.setObjectName(this.objectName);

        endpoint.setUseJbiWrapper(false);
        endpoint.setWsdl(this.wsdl);
        endpoint.setLocationURI(this.locationURI);
        endpoint.setPublicLocationURI(this.publicLocationURI);
        //endpoint.setXsdBaseURI(this.xsdBaseURI);

        endpoint.setPolicies(new Policy[]{wsPolicy()});
        return endpoint;
    }
….
}
```

The classes which describe the endpoint of the Push client have to contain some import. By instance:

```
package i2.application.tipi.smx.components.ws;

import i2.application.tipi.datex2.v2_0.modele.D2LogicalModel;[3]
import i2.application.tipi.datex2.v2_0.modele.SupplierPushInterface;
import i2.application.tipi.datex2.v2_0.modele.SupplierPushService;
import i2.application.tipi.smx.shared.Constants;
import i2.application.tipi.smx.shared.components.AbstractEndPoint;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import javax.jbi.management.DeploymentException;
import javax.jbi.messaging.MessageExchange;
import javax.jbi.messaging.MessagingException;
import javax.jbi.messaging.NormalizedMessage;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.Holder;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.util.StringUtils;
```

Example of a class which indicates the use of the D2LogicalModel to the ESB ServiceMix in a project named TIPI.

```
package i2.application.tipi.datex2.v2_0.modele;


import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlSchemaType;
import javax.xml.bind.annotation.XmlType;
/**
** Java class for D2LogicalModel complex type. * *
* The following schema fragment specifies the expected content contained within * this class. * *
```

[3] These imports are referring to one's own project. In this example they are referring for the first three ones to TIPI and the use of the Datex II model and the nest two ones refer to SeviceMix components.

```
* <complexType name="D2LogicalModel">
*   <complexContent>
*     <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
*       <sequence>
*         <element name="exchange" type="{http://datex2.eu/schema/2/2_0}Exchange"/>
*         <element name="payloadPublication
            type="{http://datex2.eu/schema/2/2_0}PayloadPublication" minOccurs="0"/>
*         <element name="d2LogicalModelExtension"
            type="{http://datex2.eu/schema/2/2_0}ExtensionType" minOccurs="0"/>
*       </sequence>
*       <attribute name="modelBaseVersion" use="required"
          type="{http://www.w3.org/2001/XMLSchema}anySimpleType" fixed="2.0" />
*     </restriction>
*   </complexContent>
* </complexType>
** **/
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "D2LogicalModel", propOrder = { "exchange",
    "payloadPublication", "d2LogicalModelExtension" })
public class D2LogicalModel {

  @XmlElement(required = true)
  protected Exchange exchange;
  protected PayloadPublication payloadPublication;
  protected ExtensionType d2LogicalModelExtension;
  @XmlAttribute(required = true)
  @XmlSchemaType(name = "anySimpleType")
  protected String modelBaseVersion;

  /**
   * Gets the value of the exchange property.
   *
   * @return possible object is {@link Exchange }
   *
   */
  public Exchange getExchange() {
    return exchange;
  }

  /**
   * Sets the value of the exchange property.
   *
   * @param value
   *           allowed object is {@link Exchange }
   *
   */
```

```
public void setExchange(Exchange value) {
    this.exchange = value;
}

/**
 * Gets the value of the payloadPublication property.
 *
 * @return possible object is {@link PayloadPublication }
 *
 */
public PayloadPublication getPayloadPublication() {
    return payloadPublication;
}

/**
 * Sets the value of the payloadPublication property.
 *
 * @param value
 *            allowed object is {@link PayloadPublication }
 *
 */
public void setPayloadPublication(PayloadPublication value) {
    this.payloadPublication = value;
}

/**
 * Gets the value of the d2LogicalModelExtension property.
 *
 * @return possible object is {@link ExtensionType }
 *
 */
public ExtensionType getD2LogicalModelExtension() {
    return d2LogicalModelExtension;
}

/**
 * Sets the value of the d2LogicalModelExtension property.
 *
 * @param value
 *            allowed object is {@link ExtensionType }
 *
 */
public void setD2LogicalModelExtension(ExtensionType value) {
    this.d2LogicalModelExtension = value;
}
```

```
    /**
     * Gets the value of the modelBaseVersion property.
     *
     * @return possible object is {@link String }
     *
     */
    public String getModelBaseVersion() {
        if (modelBaseVersion == null) {
            return "1.0";
        }
        return modelBaseVersion;
    }

    /**
     * Sets the value of the modelBaseVersion property.
     *
     * @param value
     *            allowed object is {@link String }
     *
     */
    public void setModelBaseVersion(String value) {
        this.modelBaseVersion = value;
    }


}
```

5.3.4. Suppliers

The Suppliers of these Clients systems can be:

- Supplier systems using only http (see previous chapter),
- Supplier systems using Web Services (refer previous paragraphs of this chapter).

5.3.5. Interoperability Web Services Clients <-> simple http Suppliers

If the simple http Supplier follows recommendations described in the previous chapter, there is nothing more to develop on the Web Services Client side, for interoperability.

Supplier Push with Web Services

6. Supplier Push with Web Services

6.1. Content

This chapter describes the fundamental details of a service employing "Supplier Push with Web Services". There is no subscription in this case.

6.2. Supplier system

The Publisher subsystem elaborates data conforming to the DATEX II XML schema.

The delivery subsystem is built with the DATEX II "Push WSDL" which is available on the DATEX II Web site.

The way to build the DATEX II Supplier of a Supplier Push system is the same as the DATEX II Client of a Client Pull system (refer previous chapter), because here the Supplier reaches the Web Service which is on the Client side and uses one operation of this Client Web Service to push data.

Warnings :

- Operating mode "Publisher Push on occurrence" :
  - in this mode, for situation publications (events, operator actions, …), situation and situation records lifecycles have to be managed (update, cancel, end). This is described in the document [**Exchange PSM**].

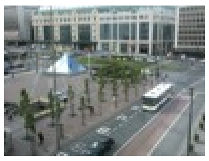
  Operating modes "Publisher Push on occurrence and Publisher Push periodic" :
  - after a link failure between the Supplier and its Client(s), recovery mechanisms have to be built (refer to [**Exchange PSM**].

6.3. Client system

The Client is built with the DATEX II "Push WSDL" which is available on the DATEX II Web site.

The way to build the DATEX II Client of a Supplier Push system is the same as the DATEX II Supplier of a Client Pull system (refer to the previous chapter), because here the Web Service is on the Client side and offers one operation to the DATEX II Supplier to push data.

General Issues



7. General issues

This chapter is a compilation of explanations about frequent asked questions or problems risen over recent years when using the previous version of DATEX II and that may facilitate new implementations.

7.1. Automatic Data base model generation

Several tools made possible the generation of the structure of the database (Table, link, etc…) by extracting the information either from the UML model or from the XML Schema (XSD).

Tools of this kind do not work with the Datex II UML model or with the Datex2 XSD, due to the abstract classes implemented in the model.

This matter is not a big problem because the number of classes used inside Datex II will provide a large number of tables and, with a good design, this number of tables can be reduced, thus easing comprehension of the database by developers and speeding up execution of the system.

7.2. A secure way to produce valid XML Files

The use of webservices can help to implement application.

7.3. Validity of the produced XML Files

An XML file is valid when it can be validate with the corresponding XSD schema.

A lot of tools (free or licensed) are available therefore.

7.4. How to distinguish Snapshot and on occurrence exchanges

The classes "Exchange" and "Subscription" must be implemented to access the attribute “updateMethod” which can have three values:

- "snapshot" is used for snapshots, which means for a photo containing all situations and situationRecords without any link with their previous status
- “singleElementUpdate" is used when only the updated SituationRecord is transmitted on occurrence
- "allElementUpdate" is used when all the SituationRecords of a situation are transmitted on occurrence, even if only one situation record is updated. To be transmitted, the situation must at least have one updated situationRecord.

It is very important for the system which receives the information contained in the XML file, to know the category of update method used.

If the updateMethode is allElementUpdate, that means that all previous SituationRecords received from the same supplier must be deleted and replaced by the new one. If a SituationRecord is no longer in the new XML File, that means that the event was ended during the time between the two publications and after the mapping, and all these events must be deleted from the database.

For singleElementUpdate, and also allElementUpdate, the supplier must implement the management class to transmit to the client the end of an event, or that one event is cancelled and so no longer alive.

7.5. Management Class

This class is an aggregation from SituationRecord class.

It must be implemented in order to manage the life cycle on a situationRecord. This information is needed for high-level links, between road operators for example.

The status “cancel” or “end” renders the situationRecord no longer active, which means that any other future version will not be transmitted.

The use of this class is mandatory for "singleElementUpdate" and "allElementUpdate" in order to inform the client that the situationRecord will no longer be active.

7.6. versiondIdentifiable

All classes which are version-identifiable contain in the header of the XML Marker an Id and a version number.

The Id will be unique and the couple (Id, VersionNumber) will also be unique.

The version number is a non-negative integer. The version number is upgraded each time a change appears. A number of systems increment this number from 1 and the first version is very often 1.