

DATEX II V2.0

SOFTWARE DEVELOPERS GUIDE

Document version: 1.0

1st of July 2009

European Commission

Directorate General for Energy and Transport

Copyright © 2009

Document control:

Prepared by :			
	Date	Comment	Version
ES5 Technical Group	01/07/2009		1.0

Reviewed by :			
	Date	Comment	Version
ES5 Technical Group	01/07/2009		1.0

Approved by :			
	Date	Comment	Version
ES5 Technical Group	01/07/2009	Authorization for publication	1.0

TABLE OF CONTENTS

1.	Introduction	5
1.1.	Objectives	5
1.2.	Document structure	5
1.3.	DATEX II reference documents.....	5
2.	DATEX II exchange system	7
2.1.	Useful documents	7
2.2.	Architecture and main subsystems	7
2.3.	Operating modes.....	8
2.4.	Exchange profiles.....	8
3.	DATEX II data.....	10
3.1.	Data model.....	10
3.2.	Data dictionary	10
3.3.	XML schema generation tool.....	10
3.4.	XML Schema.....	10
4.	Client Pull with http.....	12
4.1.	Content.....	12
4.2.	Supplier system.....	12
4.3.	Client system.....	12
4.4.	Interoperability http server <-> Clients using Web services.....	12
5.	Client Pull with Web Services	14
5.1.	Content.....	14
5.2.	Supplier system.....	14
5.2.1.	Advice for building a Supplier with Apache foundation Axis 1.4.1.....	14
5.2.2.	Advice for building a Supplier with Microsoft .Net.....	15
5.2.3.	Clients.....	16
5.2.4.	Interoperability Web Services Suppliers <-> simple http Clients.....	16
5.3.	Client system.....	17
5.3.1.	Advice for building a Client with Apache foundation Axis 1.4.1.....	17
5.3.2.	Advice for building a Client with Microsoft .Net :	18
5.3.3.	Suppliers.....	21
5.3.4.	Interoperability Web Services Clients <-> simple http Suppliers.....	21
6.	Supplier Push with Web Services.....	23
6.1.	Content.....	23
6.2.	Supplier system.....	23
6.3.	Client system.....	23
7.	General issues.....	25
7.1.	A secure way to produce valid XML Files.....	25
7.2.	Validity of the produced XML Files.....	25
7.3.	How to distinguish Snapshot and on occurrence exchanges	25

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	1.0	01-07-2009
[Modelling methodology]	DATEX II v2.0 Modelling methodology	1.0	01-07-2009
[Data model]	DATEX II v2.0 Data model		01-07-2009
[Dictionary]	DATEX II v2.0 Data definitions	1.0	01-07-2009
[XML schema]	DATEX II v2.0 XML schema		01-07-2009
[Exchange PSM]	DATEX II v1.0 Exchange Platform Specific Model	1.0	01-07-2009
[XML schema generation tool]	DATEX II v2.0 XML schema generation tool guide	1.0	01-07-2009

DATEX II exchange system



2. DATEX II exchange system

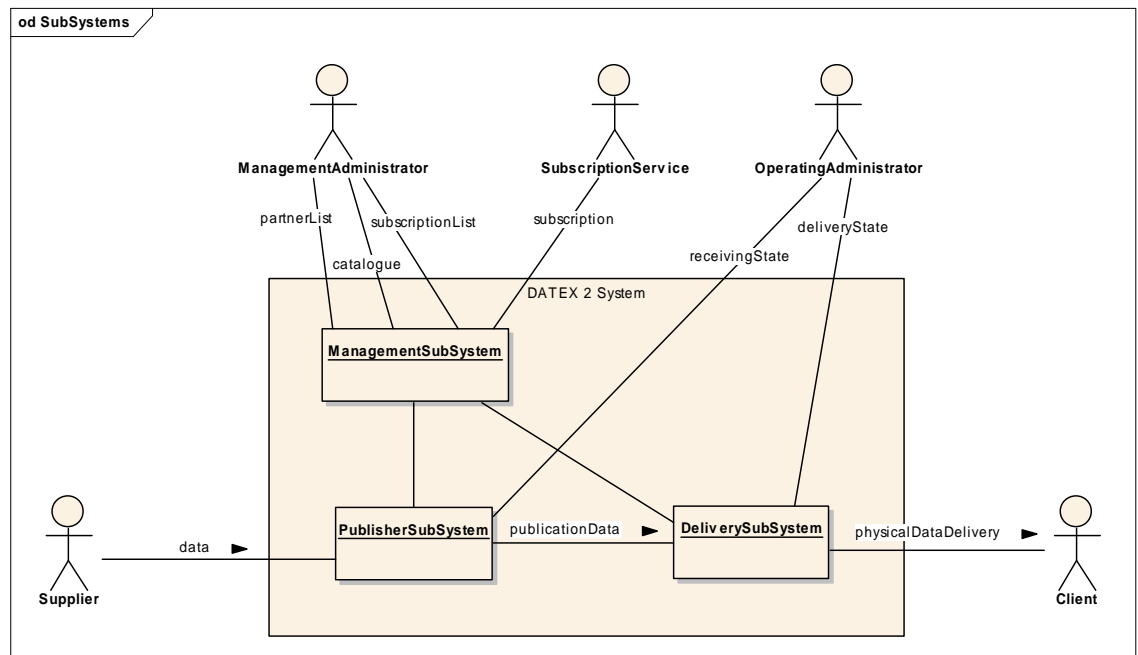
2.1. Useful documents

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_0RC1/2_0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="D2LogicalModel DATEXIIISchema_2_0RC1_2_0.xsd">
      <d2lm:exchange>
        ...
      </d2lm:exchange>
      <d2lm:payloadPublication>
        ...
      </d2lm:payloadPublication>
    </d2lm:d2LogicalModel>
  </soapenv:Body>
</soapenv:Envelope>
```

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 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.

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

- 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;
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 DATEXIIISchema_2_0RC1_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.

5.3.1. Advice for building a Client with Apache foundation Axis 1.4.1

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()));
            }
        }
        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 généré par Windows Form Designer
/// <summary>
/// Required method for the Windows Forms Designer hand over – Don't modify 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_ORC1_2_0.xsd`
 - Add generated classes to the Working Project
 - Create the client class (refer to previous example)

5.3.3. 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).

5.3.4. 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 raised last years when using the previous version of DATEX II and that may facilitate new implementations.

7.1. A secure way to produce valid XML Files

The use of webservices can help to implement application.

7.2. 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.

7.3. 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:

- "allElementUpdate" used for snapshots
- "singleElementUpdate" in case the only updated SituationRecord is transmitted on occurrence
- "allElementUpdate" in case all the SituationRecords of a situation are transmitted on occurrence even if only one situation record is updated.