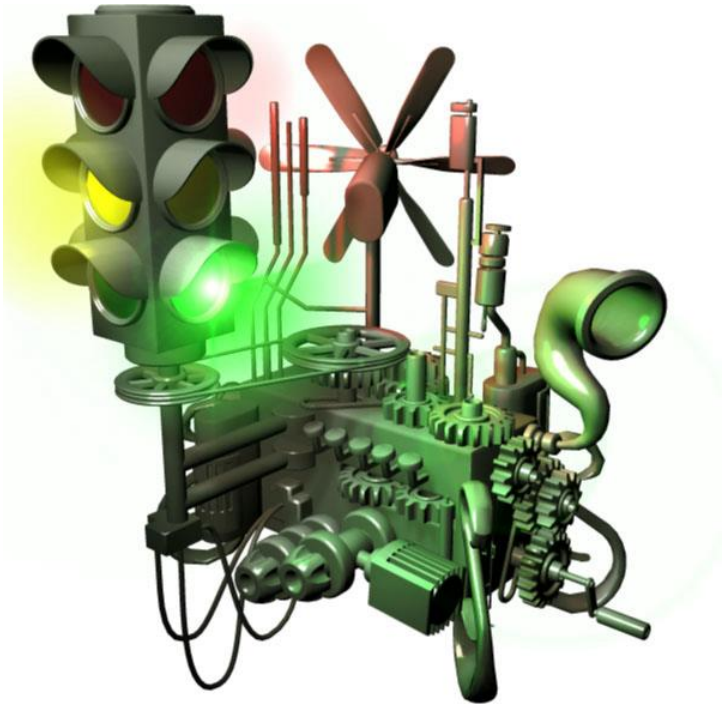
# DecisionMaker

## Integration Guide revision 8.0

(valid from version 8.x and up)






Bisnode Norge AS, Langkaia 1, 0150 Oslo, Postboks 1419 Vika, N-0115 Oslo, Norway
Phone: (+47) 22 45 93 00 | www.bisnode.no

dun & bradstreet

# CONTENTS






Bisnode Norge AS, Langkaia 1, 0150 Oslo, Postboks 1419 Vika, N-0115 Oslo, Norway
Phone: (+47) 22 45 93 00 | www.bisnode.no

Bisnode Norge AS, Langkaia 1, 0150 Oslo, Postboks 1419 Vika, N-0115 Oslo, Norway
Phone: (+47) 22 45 93 00 | www.bisnode.no

# 1 Target

This document is positioned towards software developers and integrators who are to integrate DecisionMaker (DM) in their own systems.

# 2 Documentation

In addition to this document there is a document describing the credit model in question if such exists. The *credit model* documentation contains the necessary details for delivering the correct arguments to DecisionMaker and to interpret the answer returning from DecisionMaker.

# 3 About this document

In this document we have deliberately NOT listed or specified any XML Schema Definition (XSD) or Document Type Definition (DTD). The reason for this is that some customers happened to use that definition document in their code to verify the answer from us, thus making their system fail when we made some additions or changes that was backward compatible. It is essential NOT to make your application use such definitions, as we will reserve the right to make changes (i.e. make additions but keeping backward compatibility) without warning.

## 3.1 What to read

If you already know you are going to use DecisionMaker WebServices, have a look at chapter 10.

But if you want to use plain XML Services, you are looking for chapter 11.

# 4 To be aware of

Be aware that the person asking for information or credit rating for another person, must be uniquely identified according to Datatilsynet. This means that if you (a Bisnode customer) use one and only one username and password for your integrated solutions, no matter who uses the solution, you are obliged to log who did the search, and whom he/she searched for. The person in question may come back and ask what kind of information Bisnode delivered, and the company responsible for the search (you, the Bisnode customer) must be able to answer who did the search, when and why.

*Due to new credit legislation (active from July 1, 2022) a letter of notification must be triggered when searching on sole proprietorships registered in the Register of Business Enterprises.*

Making fake requests at very frequent times towards the system to check its availability will be categorized as a try at jamming the system and the user may be blocked without warning. The customer responsible for such requests will be notified immediately after it is discovered.

A better solution for testing system availability is to let the client system work in two *modes*. The *normal mode* is where everything is working fine. *No requests* are sent to DecisionMaker to test availability. When a request fails (e.g. communication error) the system switches to a *testing mode*. In this mode the client system may give its users a warning or message about what kind of problem has occurred. A specific test request is sent at regular/periodic times to DecisionMaker. When a positive reply is again returned from DecisionMaker, the client system returns to normal mode.

**NOTE!**
***Where information is delivered as an XML, there is no guarantee that XML tags at the same level will always be returned in the same order, it may even vary between two equal requests. This is also true for WebServices; we do not guarantee that objects in an array will follow the same order between two equal requests. Neither is there any guarantee that the number of objects is the same for two different companies or persons. This is not anything especially about our services. On the contrary, it is more due to the nature of XML which has no such requirements, and neither does WebServices objects as such in any programming language.***

***Please be aware of this and do not rely on a specific order of tags, elements, or objects, but always test on the type of information therein.***

## 4.1 Batch solutions

Be aware that DecisionMaker is made and optimized for handling *single* requests. Meaning request for one *applicant* (i.e. person or company) at a time. If you need a batch solution, i.e. need to run many requests (like several thousand) in one go, you should get in touch with the sales representative in Bisnode and talk about this.

## 4.2 About testing for up-time

DecisionMaker is running 24 hours, 7 days all year.
Some may think it is a good idea to do recurring fake requests to check if the system is available.

This is not a good idea. Doing this, DecisionMaker will have to handle these requests and thus will have less time to handle the real requests. The result is decreased total performance. For that reason, such recurring requests will be detected on an irregular basis, and *the userid* used to execute these requests *will be suspended without warning*.

DecisionMaker (DM) receives many requests all the time at any hour. This means that we will usually discover any problems before any of our customers do.

A better “test for availability” pattern will be like this:

- If DM replies satisfactory the client system runs in *normal mode* and will do no “test requests” towards DM.
- If DM fails to reply or replies in an erroneous way on one or more ordinary requests, the client system should be set in a *test mode*, running recurring test requests (let’s say once a minute or once every second minute) until the client system gets proper responses again. When that happens, the client system can be set back to *normal mode*.

# 5 Terminology

By ***Applicant*** we mean the person or company the *customer* is requesting information for.
By ***Customer*** we mean the customer or client software that is using DecisionMaker. Every customer must be set up with the correct subscriptions and users with passwords.
***Decision cutoffs*** are cutoff values telling which zones should end up in what decision. Zone 0 is always *decision* 'o', and usually (i.e. by default) zone 1 is 'n', zone 2 is 'c' and higher zones are 'y'. The information about this is returned as part of the result. The configuration on which zones is to go into what decision is to be found in each separate credit model if different than default.
***Information handler***. Customers not wanting a score request, but just obtaining information for some information provider, may use an *information handler* instead. An information handler is more or less customized depending on the information the customer wants. No matter how the information provider delivers the information to DecisionMaker, DecisionMaker delivers it as XML. The lay-out of the XML may vary from provider to provider, but the root tags are always the same (i.e. `<dmResponse>` and `<response>` tags).
***Information Provider*** is used as a general name for the company, bureau, database, or any kind of source that DM gets information from.
***Policyrules*** may override the zone value that comes out of the score points and score cutoffs. Policyrules may be positive, negative, or neutral. Neutral policyrules do not affect the zone value and serves only as information. Positive policyrules may affect the zone value in positive direction, e.g. if the zone value is 3 and a positive policyrules occurs that wants to increase the zone up to 4, the final zone becomes 4. If the zone was 5 in advance, the zone remains 5, as the policyrule is positive and hence cannot decrease the zone value. Negative policyrules however can affect the zone value in negative direction. If the score zone is 3 and the negative policyrule says 4, the final zone remains 3. If the negative policyrule says 1, the final zone becomes 1. The policyrules that occur are always returned as part of the result, and in the XML the tag name is *POLICY_ELEMENT*.
***Scoring*** is the term used for the decision process (DM executing the customized credit model resulting in a *zone* and a *decision)*.
***Score card***. The score card contains the rules for how to calculate a rate/scoring value. The score card consists of one or more score elements each having its own rate value. All these values are summed and added to a base point value which is a constant specific to each scorecard. This sum again ends up in a value within the range of the Credit model in question. Usually details about the scorecard are NOT returned as a response to a score request, only the final value. But in some occasions it may be returned on request if the customer subscribes for that information. In the XML the tag name is SCORE_ELEMENT.
***Score cutoffs****.* When the score points are calculated, the *score zone* is decided. The score zone is decided based on the zone cutoff values. I.e. if the score points calculated are 37 and the score cutoffs are 0, 30, 60, 80, it means there are 4 zones in this *Credit model* and with a value of 37, this applicant is rated to zone 2. The cutoffs in this example denotes that zone 1 is from 0 (included) and up to 30 (not included), zone 2 is from 30 (included) and up to 60 (not included), zone 3 is from 60 (included) and up to 80 (not included) and zone 4 is from 80 to 100 (both included). The score point range is decided by the score card in question. As an example, the *Decision Score* scorecard for persons ranges from 0.0 to 1.0 and denotes probability. The range for the *Decision Score* scorecard for companies ranges from 0 to 100.
***Score decision***. When a scoring is performed it results in a score *zone* and a *decision*. The *decision* from DecisionMaker is usually one out of three, mirroring the green, yellow and red lights from a traffic light. The decision is returned as a single letter, respectively 'y' (as in "yes"), 'c' (as in "check") and 'n' (as in "no"). If the scoring fails for some reason, the result might be an 'f' ("failed") unless an explicit error code is returned (it usually is). The decision may also be the letter 'o' (as in "other") if the decision could not be set (may be the applicant could not be rated for some reason). For a *customized credit model*, it is essential that the integrating client system do not implement a mapping from zone to the decision. This is the responsibility of the credit model. The response will include both the zone value and the *recommended* decision. Use this decision value in the client system and use the zone value only for informational purposes if wanted.
***Generic Credit model***. The information provider may deliver one or more *Credit models*. These are numbered with an integer value. Each model may have its own set of *zones* and its own set of *score cutoffs*. The information about number of zones and score cutoffs is returned as part of the result from a score request. A model may be *linear* or *non-linear*. A *non-linear* model has score points from 0 to 1 and a *linear* model uses the range 0 through 100.

*NOTE that the Norwegian generic rating models from Bisnode with model number 1 and 2 is old models and is to be considered as obsolete and must therefore be ignored (although they are still returned).*
***Score points*** or ***Rating value***. The calculation done in processing the *score card* results in a score point or rating value, usually a value (may very well be a floating point value) between 0 and 100 or between 0 and 1 (depends on the type of *Credit model*).
By ***Customized Credit model*** (also referred to as *product*) we mean the credit policyrules the customer has decided to use. *The credit model* contains a data collection module, score card, policyrules and a decision module:

- Data collection
- Score points/rating value calculation. This is also referred to as the ***Score card***.
- Policyrules
- Decision deduction and response definition

***NOTE!*** *The response from a customized credit model contains, in addition to the zone value, a recommended decision. It is a good idea to use this decision value in any external client application, not the zone value. The credit model contains a mapping from zone to decision. If the client also should have such a mapping, one must keep the client and the credit model in synch at all times. Ignoring the zone value (use it as an informational element only) and considering only the recommended decision makes the client non-dependent on any changes in the credit model.* ***Bisnode cannot take any responsibility for mapping tables in any client system that maps to a different decision than what is returned from the credit model.***
***Score zone*** is an integer value starting at value 1. The number of score zones may vary depending on the *Credit model* used. *Usually* applicants in zone 1 are not worthy of credit, zone 2 is questionable, and zone 3 and up is ok. Zone 0 may also be returned but is used to denote those applicants that are not scored or rated for some reason. If this is the case, the reason is to be found in one or more *policyrules*. The *policyrules* that occur may influence the final zone value.
***WebServices*.** A customer may communicate with DecisionMaker using *WebServices*.
***XML Services***. A customer may communicate with DecisionMaker using *XML Services*. This is plain XML strings both on requests and response.

# 6 What is DecisionMaker?

DecisionMaker is a server application for performing …

- Scorings/credit ratings and evaluations of persons and companies based on customer specific *credit models*
- Data gathering from miscellaneous sources and converting these to XML format or a WebServices object
- User identification, authentication, and password verification
- Delivers information from different sources to end-user

DecisionMaker is configured to collect information from the sources that are needed. Information from external sources (”Information Providers”) is converted to an internal XML representation, ahead of DM utilizing the information.
Then DM can …

- Deliver information directly to the client by means of *Information Handlers*
- Deliver information to DM’s own web client.
- Perform a scoring based on the information provided

*DM Information Handlers* gives a client access to the information wanted without performing a scoring/credit rating on a specific credit model. May be:

- Information directly from an external source using the proper *information handler*

## 6.1 What can be done using DecisionMaker?

Via WebServices/SOAP or XML Services the following can be done (examples):

- Get address information from a specified source
- Execute a scoring based on a customized *credit model*
- Get credit information
- Get other information

### 6.1.1 Summary of services

As a short summary we can group the services DM provides as follows:

- *Scoring* (includes performing scorings for customized credit models)
- *Information* (fetching specific information using customized *information handlers*) from virtually any source.

## 6.2 What happens? (Score process)

Figure 1 shows what happens when DM receives a request.

By default, DM stores the complete social security number (ssn) in its database.

- If the request contains a complete ssn it is checked for validity and DM looks for it in its own database. If no ssn exists in the database, a new *applicant* record is saved to the database. Then the specified credit model is started, and depending on that, the wanted data providers are called for information, which in its turn calls any external information providers if needed.

- If the request does NOT contain a complete ssn, all the information that the request contains about the applicant is sent to the main data provider/information provider for unique identification. If not found, an error is returned to the client. If found, DM looks for it in its own database and goes on as described above.

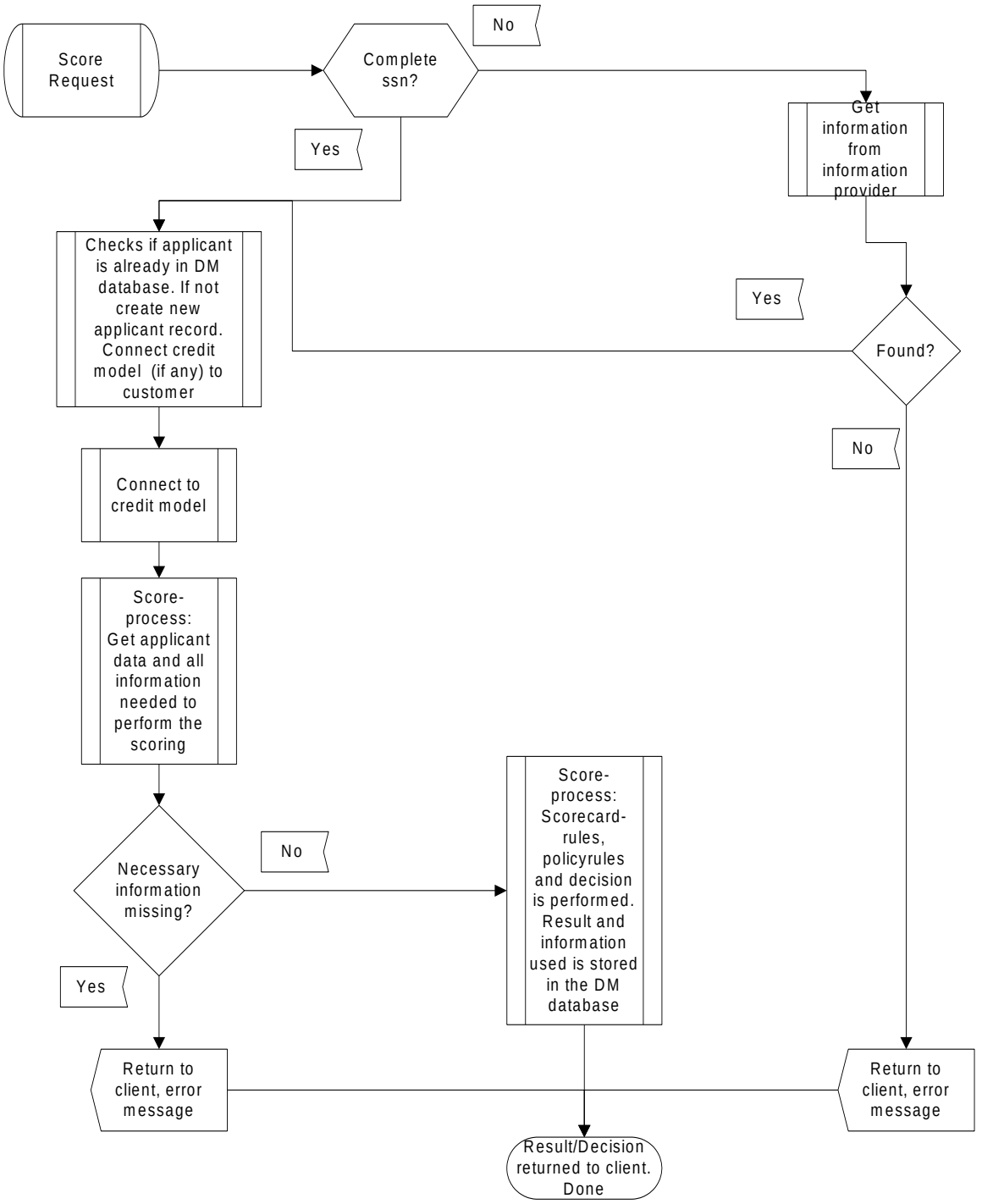



Figure 1

## 6.3 Authentication

Every request to DecisionMaker must be authenticated. Figure 2 shows schematically how a user of a given client system is authenticated all the way through DM and to the information provider.

The end user of some client logs in and is verified/authenticated by that system. The client system may then use the same user ident (username) or another ident towards DM. The ident used towards DM must be registered in the DM database together with the customer identification it is connected to. The customer is connected to some *subscriptions* (like information only, scoring, WebServices, person information, company information, etc.) depending on what kind of information and services the user wants from DM. These subscriptions are limited in time and in number of requests. The customer and/or the ident may be connected to none, 1 or more credit models. The customer and/or ident is also connected to some *rights* (like allowed to score, allowed to update previous scorings, allowed to perform scorings, etc.). The information provider may use the user identification information in its turn to send information to the applicant about who has asked for information about him/her and what kind of information was asked for.

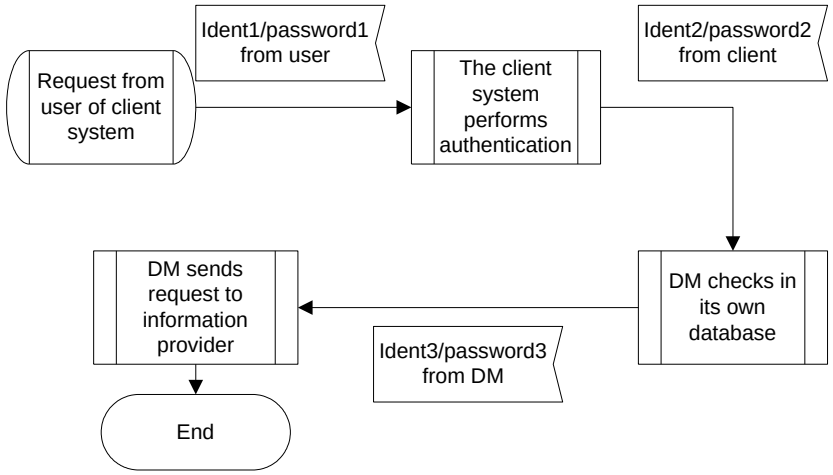



Figure 2

# 7 About searching for persons

When a request is sent to DecisionMaker, the person in question is looked up in the DM database. If not found the person is looked up at the information provider. If person is still not found, an error code is issued. If request data is not accurate (complete ssn) or a one-to-one match is not found, there may be several persons matching the data, another error message is issued. [1])
For Swedish information, an exact ssn or organization/company number must be used.
For Norwegian information, an exact ssn is not necessary, and the rules below apply. But note that using a complete ssn is far more secure and accurate as this will identify the person uniquely.

If the person is found and it is a score request, the applicant is connected to the specified credit model, and the credit models fetched from the database.

The score process may fail in several ways. For some reason, the data from the information provider may not be sufficient in order to perform a scoring. In this case another error is issued (e.g. code 46).
There may also be a coding bug inside the credit model itself, and an error message with code 40 may be issued.

## 7.1 Social security numbers (ssn) and organization numbers to be used for test

### 7.1.1 When the information provider is Bisnode

#### *7.1.1.1 Persons*

Be aware that when asking for actual persons a letter of notification is sent to the person. Fictitious persons may be delivered on request. Contact ksb.no@bisnode.com.

Due to the new credit legislation (active from July 1, 2022) a sole proprietorship is now also considered a physical person.

Bisnode is obliged to issue a letter of notification (lon) when a person credit rating is performed. Standard delivery is a printed letter sent to the official address for the person in question.

There is however an electronic alternative, that is using mobile phone number. This requires a special permission, so you will need to contact your sales representative.

Implementation details may be found in the following chapters:

- WebServices – DmWSAppInfo
- XML Services – applicant

#### *7.1.1.2 Companies*

In cases where the company type is an ENK or an ANS where a person behind it is considered, a letter of notification will be sent to this person. No such letter is sent when checking other company types.

Due to the new credit legislation (active from July 1, 2022) a sole proprietorship is now also considered a physical person. Searches on the specific company type sole prop (ENK) - both ENK reg. in FR and ER - will now trigger a LoN.

[1] This is just an example. The complete list of error codes may be requested separately from Bisnode.

# 8 Sources for information

DecisionMaker connects to several sources to get information. Thus, information from different sources may be gathered into the same credit model as the customer wants.

The main source for identifying Norwegian persons and companies is Bisnode's own database.
Requests for Swedish persons and companies are also accepted.

As Bisnode is a credit information company, we are not allowed to do pure address verification on persons unless credit information is also asked for. To give customers a good solution to this problem, our WebServices may also be used to address verification towards another company, namely *Bisnode Match:it*.

The details on information from the different sources are to be found in separate documents on request.
A separate customer contract must be set up for each source of information before it will be made available.

Some information on sources may be found at www.soliditetd.no.

## 8.1 Bisnode, Norwegian information

Bisnode has their own Norwegian database for all Norwegian taxpayers and all registered Norwegian companies and organizations. This is the *default* source for Norwegian information.
Note that complete Social Security Numbers are only available (i.e. returned in a response) for customers who are authorized to deal with such numbers.

## 8.2 Swedish information

Swedish person and company information is also available.

## 8.3 Match:it, Norwegian information

Match:it (http://www.matchit.no, which is another Bisnode company) has their information from postal registers, phone companies etc.

For persons, the following applies:
Because Bisnode is basing their information on the Norwegian tax register, two situations may occur:

1. Name and address for a person returned from Match:it may not be the same as registered at Bisnode
2. The person returned from Match:it may *not* be registered at Bisnode. This means the person in question has never paid Norwegian taxes. Usually because the person is too young.

Because of 1) above, using the name and address from Match:it in a new request towards Bisnode (for credit information or scoring) may fail. A special flag may be added to the last request to help maximizing the hit chance.

See separate document, available for download at www.soliditetd.no.

# 9 Methods

There are several methods to use to “talk to” DecisionMaker (DM).

- WebServices/SOAP
- “XML Services”: Simple strings containing a specific XML structure via HTTP POST

The methods are described in detail later in separate chapters.

## 9.1 What method to choose?

You may send requests using the HTTP/HTTPS POST method. The GET method is also supported but is only useful for simple XML Services test requests *during development*. For production purposes the POST method must be used.

It is the system integrator that must decide which method (XML Services or WebServices) they think is the easiest to use for implementation. For performing scorings, the WebServices is the easiest and most flexible to use if you have a development system that support this. By a simple request to the DM server application you will get a WSDL file which your development system may use to generate the WebServices client classes no matter what programming language you want to use.
You will find a link to the WSDL file at www.soliditetd.no. This is the production system. The system to be used for test is to be found at test.soliditetd.no.

Using XML Services is very simple, but to generate a request you will have to generate a specific XML described in this document, and to parse the answer which is also an XML document. Hence it may be more difficult sending and receiving (i.e. interpreting) structured data. By using WebServices much of this job is done for you. However, if you only are going to transform the result using XSLT to generate an HTML file, the XML Services way may be best.

When using WebServices for *Information* services (as contrast to *Scoring* services) you will get the result as an XML string, and you will not be able to get the parts of that XML using methods in the classes generated. This is because the *Information* services are customized for the use wanted (see *Information Handlers*), and hence the *methods* would change all the time as well as the fact that the number of *Information Handlers* are quite high, so the services provided this way would be difficult to both maintain and handle well if they were to be customized for each handler. On some occasions though, when asking via an information handler about an applicant’s name and address, or when using an information handler to ask for the results of an earlier scoring, you will get the result as objects *as well as plain XML*.

The communication handler is HTTP (for systems where the DM server application and the clients are behind the same firewall) or HTTPS (for systems where the DM server application is behind a different firewall than the client(s)).

Note that all XML tags are case sensitive.

## 9.2 WebServices

Our WebServices are built using Apache Axis 1 version 1.1. (http://ws.apache.org/axis) .
This does *not* limit you to not use other versions of Axis 1 (the latest is 1.4). There is also an Axis2 project out there that someone might want to try at http://ws.apache.org/axis2/.

Standards supported are currently as described at
http://wiki.apache.org/ws/FrontPage/Axis/StandardsSupported.

To get a WSDL file, access the address as follows: http://**<HOST NAME >**/webservices.
For production, use www.soliditetd.no. For test, use test.soliditetd.no. You will then get a menu that brings you further on to a WSDL file for our *DecisionMakerWebservices.* The WSDL contains all classes and methods needed for both scoring and use of information handlers.

Up to and including version 2, *RPC/Encoded* WSDL style is used. From version 3 and up, *Document/Literal Wrapped* is used. The latter is *WS-I Compliant*, the first is not (http://www.ws-i.org/). See more about styles at http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/.

For examples on how to create a client and how to do a request, see chapter 10 DecisionMaker WebServices.

More information on SOAP may be found at http://www.w3schools.com/soap/.

For information on WebServices in general, see http://www.oracle.com/technetwork/java/index-jsp-137004.html, http://www.xml.com, http://www.webservices.org and
http://wiki.apache.org/ws/FrontPage/Axis/AxisGeneral.
http://www.w3schools.com is also a very good resource for everything that has to do with the web (in this case, especially http://www.w3schools.com/wsdl/).

If you are looking for free WebServices tools, have a look at http://cxf.apache.org/ and http://ws.apache.org. Apache CXF, Apache Axis 1 and 2 are free and make you able to automatically create client stub code for Java and/or C/C++. Many IDE's can generate client stubs for you as well. NetBeans (http://www.netbeans.org), Eclipse (http://www.eclipse.org/), IntelliJ IDEA (http://www.jetbrains.com/idea/) all have good support for WebServices as well as many other tools out there.

## 9.3 XML Services (*not* WebServices)

This method involves using plain XML strings and sends them to DecisionMaker using HTTPS (POST).
You may need an XML parser and/or a style sheet processor. Have a look at http://xml.apache.org for free stuff of both;
There you will find parsers (Xerces) for Java and C++. They also have a Perl (wrapper around C++) version.
A COM wrapper (around the C/C++ version, see http://xml.apache.org/#xerces) for compatibility with the Microsoft MSXML parser (http://www.microsoft.com/downloads/details.aspx?FamilyID=3144b72b-b4f2-46da-b4b6-c5d7485f2b42&DisplayLang=en) is also available.
XOM (http://www.xom.nu) is an efficient alternative to JDOM (http://www.jdom.org) and DOM4J (http://dom4j.sourceforge.net). There is not much activity on the two latter ones…..
Apache Xalan is a style sheet processor available for both Java and C++.

## 9.4 HTTPS using Java? Need certificate?

In the following it is assumed the client is written using Java.

It is *not* necessary to install any certificate on the client side to communicate with us. However, some communication libraries require that this is done. Using Java this should not be required.
The Java client must use “Java Secure Socket Extension (JSSE)” which is a part of 1.4.x and upwards, and can be downloaded from Sun at http://java.sun.com/products/jsse as an add-on if you use an earlier version.
In test mode, you may want to write some code that bypasses the certificate validation check. See chapter 10.10.2 Apache Axis 1 for Java 38 on how to do this using Apache Axis 1.

### 9.4.1 Getting the certificate

If you use the Internet Explorer and put www.soliditetd.no in the address field you get the start page on the production server. In the lower right corner, you can see a padlock. Click twice on that, enter the “Details” tab page and press the “Copy to File” button. Pressing “Next” two times brings you to a dialog asking you for the name of the file. Find a location you can remember and name the file “crediteasy.cer” and press “Next” again, then “Finish”. You now have a copy of the certificate to put into the Java keystore.
You may do the same for the test server. The address is test.soliditetd.no and gives the certificate file the name “testcrediteasy.cer”.

### 9.4.2 Saving certificate to Java keystore

We will start storing the server certificate in the *keystore*.
The keystore password is most likely *changeit* (if you haven’t changed it).

The following *path*s are of course dependent of the current installation.

Example (the symbol ⇳ denotes the lines should be on one single line):

```
$ ./keytool.exe -import -alias test.soliditetd.no -storetype jks -file ⇳ testcrediteasy.cer
-keystore ⇳
c:/programs/java/jdk1.6.0_17/jre/lib/security/cacerts
```

The above command should be executed from the `$JDK_HOME` bin directory (`c:/programs/java/jdk1.6.0_17/`). It reads the certificate file "`testcrediteasy.cer`" and puts it in the *keystore* "`c:/programs/java/jdk1.6.0_17/jre/lib/security/cacerts`".

```
$ ./keytool.exe -list -keystore ⇳
c:/programs/java/jdk1.6.0_17/jre/lib/security/cacerts
```

The above command is used to verify that the certificate is stored properly.

When this is done, the Java code (the client code) must be modified to use SSL:

```
// import JSSE
import java.security.*;
// set keystoretype
System.setProperty("javax.net.ssl.keyStoreType", "jks");
// create new protocol handler
System.setProperty( "java.protocol.handler.pkgs",
                    "com.sun.net.ssl.internal.www.protocol");
// add protocol handler
Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
```

Compile and test the code.

# 10 DecisionMaker WebServices

The WSDL available, contains methods for information handlers and for accessing the scoring service. If you go to www.soliditetd.no you will see a page with a link to the DecisionMaker WebServices and the WSDL file. Accessing the link will generate or download a WSDL file for you.

There are several tools out there that help you create clients for WebServices. Here are some:

- ***Visual Studio .NET*** offers to help you generate classes and code automatically in the language of your choice if it is a supported one of their packages.
- ***Microsofts Visual Express Tools***. Have a look at http://www.microsoft.com/express/. They are free!!
- ***Borland/CodeGear*** has several versions of their ***Builder*** (Java, C#, Delphi, C++ etc) which all support WebServices
- ***Glassfish/Metro*** may be a good starting point. Glassfish/Metro is the reference implementation of JAX-WS
- ***Apache CXF*** from http://cxf.apache.org/. CXF supports the latest Java JAX-WS.
- ***Apache Axis 1 and 2*** are free solutions and makes it possible to generate the client code

….and there are many others out there:

- NetBeans (http://www.netbeans.org/)
- XMLSpy (www.xmlspy.com)
- Stylus Studio (www.stylusstudio.com)
- Web Service Studio from Microsoft for testing (search the web for latest version)
- Eclipse (http://www.eclipse.org/)
- Oracle JDeveloper (www.oracle.com)
- …etc…

http://ws.apache.org/ may also be a good page for some free tools.

All DecisionMaker WebServices classes are prefixed with **`DmWS`**.

***NOTE!*** One of the WebServices methods is named *doNotUseThis*. The method has no actual meaning and does nothing at all. It appears that some WebServices client generating tools do not generate classes that are not directly referenced in a method call. The method is therefore just present to help these tools to generate all classes as they should. The method should never be called.

## 10.1 The WebServices methods and objects

The DecisionMaker WebServices methods are:

- getVersion
- getStatus
- score/scoreX
- getInformation/getInformationX

### 10.1.1 Request objects

The main request object holds an array of what we call *single request objects* (`DmWSSingleRequest`). This class holds an array of the applicants asked for and a request id. The request id is optional, but if set it is returned as response id.

The only classes inherited from `DmWSSingleRequest` are `DmWSScoreRequest` and `DmWSInformationRequest`. `DmWSSingleRequest` must be treated as an abstract (pure virtual in C++ terminology) class and *never be instantiated directly*.

#### *10.1.1.1 DmWSRequest*

This class corresponds to the `<dmRequest>` tag. The class has the following data members:

| name | type | description |
|---|---|---|
| `user` | `string` | The user's name must be registered at Bisnode prior to being able to perform a request. *Required.* **Case sensitive.** |
| `passWord` | `string` | The password as specified by Bisnode. *Required*. **Case sensitive.** |
| `userRef` | `string` | Future use. Could be used as an additional reference in external system to be connected to this request. *Optional*. |
| `customerId` | `string` | Future use. *Optional.* |
| `requests` | `DmWSSingleRequest[]` | A list of requests to be executed. I.e. a list of `DmWSScoreRequest` and/or `DmWSInformationRequest` objects. Note that the `DmWSSingleRequest` class is to be considered as an abstract class never to be directly instantiated. |

#### *10.1.1.2 DmWSSingleRequest*

This class corresponds to the `<request>` tag. The class is to be considered as an abstract (pure virtual) and should never be directly instantiated. Use the `DmWSScoreRequest` and/or `DmWSInformationRequest`. It has the following data members:

| name | type | description |
|---|---|---|
| `requestId` | `string` | An id string that may be used to identify the response. The same string is returned in the response class (`DmWSSingleResponse`). *Optional.* |
| `applicants` | `DmWSApplicant[]` | A list of applicants to be requested for. *Optional for some information handlers*. At least one applicant is required for a scoring. Whether more than one should be used depends on the credit model in question |
| `addInfo` | `DmWSElement[]` | Future use. May be used for some information handlers. Will be described when needed. *Optional.* |

#### 10.1.1.3 DmWSScoreRequest

This class has no direct XML tag equivalent. It IS equivalent to an XML request where the credit model name is specified. The class has the following data members:

| name | type | description |
|---|---|---|
| productName | string | Name of the credit model to be executed. Case sensitive. *Required.* |
| comment | string | This is the comment added to the scoring entered in the request (see DmWSScoreRequest). *Optional.* This field could be used for adding stuff like information that identifies the person performing this search. This information will be stored together with the result of this scoring. |
| scoreGroupName | string | Score group name that this scoring should belong to. *Optional.* |
| extended | boolean | Requests that response should contain details about the scorecard. Requires special rights. Ignored if rights not set. *Optional.* |

#### 10.1.1.4 DmWSInformationRequest

This class corresponds to the <informationHandler> tag. The class encapsulates the request holding the name of the informationhandler. The class has the following data members:

| name | type | description |
|---|---|---|
| informationHandler | string | Name of information handler to call. *Required.* |

### 10.1.2 Response objects

The main response object DmWSResponse holds an array of DmWSSingleResponse objects. Only three classes inherit from the DmWSSingleResponse class, and DmWSSingleResponse itself should be treated as an abstract class (pure virtual).
The three classes are:

- DmWSError
- DmWSScoreResponse
- DmWSInformationResponse

#### 10.1.2.1 DmWSResponse

This class corresponds to the <dmResponse> tag.
The class has the following data members:

| name | type | description |
|---|---|---|
| responses | DmWSSingleResponse[] | Holds an array of responses |
| userRef | String | User reference (as in the request). *WebServices version 3.x only. For prior versions, see the* DmWSSingleResponse *class* |

#### *10.1.2.2 DmWSSingleResponse*

This class corresponds to the `<response>` XML tag. The class is to be considered as an abstract (pure virtual) class. Implementing classes are: `DmWSError`, `DmWSScoreResponse` and `DmWSInformationResponse`. This class has the following data members:

| name | type | description |
|---|---|---|
| `responseId` | `string` | An id string that may be used to identify the response. This is the same string as specified in the request (see `DmWSSingleRequest`). |
| `applicants` | `DmWSApplicant[]` | A list of applicants to be requested for. *Optional for some information handlers*. At least one applicant is required for a scoring. Whether more than one should be used depends on the credit model in question. |
| `handler` | `string` | Name of user. *Versions prior to 3.x only.* |
| `user` | `string` | Name of user. *Version 3.x only.* |
| `userRef` | `String` | User reference (as in the request). *WebServices versions prior to 3.x only. For version 3.x see the* `DmWSResponse` *class.* |

#### *10.1.2.3 DmWSScoreResponse*

This class has no direct XML tag equivalent but IS equivalent to a request where the credit model name was specified. An instance of this class holds one or more scoreresults (`DmWSScoring`). The class has the following data members:

| name | type | description |
|---|---|---|
| `scoreId` | `string` | Every scoring is given a unique score id |
| `created` | `string` | Date scored. Format is yyyy-MM-dd |
| `finalResult` | `string` | The final decision. Not used, see `result` |
| `result` | `string` | The final recommended decision: 'y': Yes, 'n': No, 'c': Check, 'o': Other. The latter occurs if no scoring has been performed. Usually explained why in a policy element. Use this value to find out what the recommended decision is. Do not map the zone value to a decision of your own in your client system. |
| `zone` | `string` | Final score zone. This is the result from the score card *and* the policyrules. Do not use this value and map it to a specific decision in your client system. The recommended decision based on the credit policy is to be found in the *result* member. |
| `points` | `string` | The result in points (from the *scorecard*). *Should never be used in a client system to get the decision, besides for plain information.* |
| `overruled` | `string` | A decision from a score may be overruled using `updateScoreDecisionById`. If done, this will contain the updated decision value ('y': Yes, 'n': No, 'c': Check) |
| `overruleComment` | `string` | This is the same value as entered in `decisionCause` argument to the `updateScoreDecisionById` method. |
| `overruledBy` | `string` | This is the username entered as argument to the `updateScoreDecisionById` method. |
| `overruledDate` | `string` | This is the date the `updateScoreDecisionById` method was called. |
| `comment` | `string` | This is the comment added to the scoring entered in the request (see `DmWSScoreRequest`) |
| `scorings` | `DmWSScoring[]` | An array of score results, usually only one. |
| `productName` | `string` | Credit model name. |
| `scoreGroupName` | `string` | Score group name that this scoring belongs to. |
| `xml` | `string` | The score response as an xml string (*not used*). |

#### *10.1.2.4 DmWSScoring*

This class holds one score result. The corresponding XML tag is a combination of `<SCORING>` and `<scoreresult>`. The class has the following data members:

| name | type | description |
|---|---|---|
| `name` | `string` | Name of the scoring is usually the same as the credit model name |
| `result` | `string` | The final decision: 'y': Yes, 'n': No, 'c': Check, 'o': Other. The latter occurs if no scoring has been performed. Usually explained why in a policy element. Use this to get the recommended decision, not the value of *zone* or *points* |
| `zone` | `string` | Final score zone. This is the resulting zone from the score card *and* the policyrules |
| `scoreDate` | `string` | Date when scoring was performed |
| `basePoints` | `string` | The scorecard usually has a constant number added to the sum. This number is called `basePoints` |
| `zoneModelId` | `string` | A scoring may be based on one of several models from the provider. This is the model number |
| `linear` | `boolean` | The credit model may be of a linear type or non-linear. The first means the sum of points is a number between 0 and 100, the latter means the points are actually not points, but represents a probability between 0 and 1 |
| `zones` | `DmWSZone[]` | Each model may have a different no. of zones. Each zone is listed and described here |
| `dataElements` | `DmWSDataElement[]` | Described later on |
| `scoreElements` | `DmWSScoreElement[]` | A score element for each element in the *score* card. Described further later on |
| `policyElements` | `DmWSPolicyElement[]` | A score result may include none, one or more policy elements. Described further later on |
| `totalScoreElements` | `DmWSTotalScoreElement[]` | The total score element (usually only one) concludes the results and output of the scoring. Described further later on |

##### `10.1.2.4.1 DmWSZone`

This class corresponds to the XML `<zone>` tag. An array of instances of this class corresponds to the `<ZONE_INFO>` tag. A zone is an integer number from 1 and up. A number of 0 usually denotes the applicant is NOT scored. This class shows the mapping from calculated points to a credit zone value, and from a credit zone value to a recommended decision. What each zone means is described for each credit model.
The class has the following data members:

| name | type | description |
|---|---|---|
| `zone` | `string` | The zone number. |
| `result` | `string` | The decision for this zone: 'y': Yes, 'n': No, 'c': Check |
| `borderValue` | `float` | Related to the sum of points or probability (if non-linear). This zone starts from and including this value and upwards to where the next zone starts. |
| `zoneNo` | `int` | Usually the same as `zone`, unless otherwise specified for the Credit model in question |

#### *10.1.2.5 DmWSInformationResponse*

This class has no direct XML tag equivalent, but IS equivalent to a response where the credit model name is NOT specified (but where the informationhandler name IS). The class has the following data members:

| name | type | description |
|---|---|---|
| informationResult | string | The result as an XML |

#### *10.1.2.6 DmWSError*

This class corresponds to the <error> tag. If an error occurs within DecisionMaker, a DmWSError object is returned. The class has the following data members:

| name | type | description |
|---|---|---|
| errorCode | string | An error number |
| errorMessage | string | A descriptive error message |
| errorMeasure | string | Information on what to do when the error message occurs. It may be a message about who to contact, or how to avoid the error or on how to correct the error situation. |
| detailedMessage | string | A detailed error message (only when DM is in test mode, usually the callstack). |
| productName | string | The name of the credit model if the error was a response to a score request |
| Time | string | Timestamp (when this error occurred) |

### 10.1.3 Classes used in both request and response

#### *10.1.3.1 DmWSApplicant*

The DmWSApplicant holds information about the applicant requested for, both in request and in the response. The class corresponds to the <APPLICANT> tag. One or more applicants may be specified. In XML the array of applicants corresponds to the <applicant> tag.

This class has the following data members:

| name | type | description |
|---|---|---|
| applicantNo | integer | The main applicant is numbered 0 (zero). *Optional* if only one applicant. *Required* if more than one. |
| type | string | Type of applicant. 'P' for person, 'F' for company or organization. *Required.* |
| ssn | string | Social Security Number. The number that uniquely identifies a person within a country. If no complete ssn is available, ssn may be used to specify a person's birth date on the form "ddMMyy". Also used to specify company or organization numbers. *Optional* if name and address is enough to identify the person or company. |
| extClientNr | string | Bisnode gives all registered applicants a unique number, called an *object number*. If you have this number, you may use that instead of the ssn |
| firstName | string | The first name(s) of the person to ask for. *Optional* |
| lastName | string | The last name of the person to ask for. *Optional* |
| name | string | Name of company or organization. *Optional* |
| address | DmWSAddress | The address for the company or person. *Optional* |

| | | |
|---|---|---|
| nationality | string | Specifies which nationality the ssn or organization is. If implemented in DM, the proper ssn verification routine is run. The values to be used are two-letter codes as specified in *Values must correspond with the two-letter character code as specified on* http://www.iso.org/iso/english_country_names_and_code_elements . This value must not be confused with country in the address class, which specifies the country the person lives in (registered address). *Optional.* |
| email | string | E-mail address. *Optional.* |
| eiendomsInfo | DmWSEiendomsInfo | Some requests may require information about a property. Depends on the credit model or the information handler. *Optional.* |
| appInfo | DmWSAppInfo | Some credit models require extra information delivered with the request. These are specified in the appInfo data member. *Optional.* |

### *10.1.3.2 DmWSAddress*

This class corresponds to the <address> tag within the <APPLICANT> tag.
This class has the following data members:

| name | type | description |
|---|---|---|
| street | string | Street name. E.g. “Testgaten 44 C” |
| zipCode | string | The zip code. For Norway this is a 4 digit number |
| city | string | City or place. |
| country | string | Country name. If applied with an applicant (DmWSApplicant) the default value is the country for the specified nationality. *Optional*. |
| houseLetter | string | The letter for the house in the street address. E.g. for “Testgaten 44 C” the letter is ’C’. However this is used only for products or information handlers that specifically state it must be specified like this. Normally use the street data member. |
| houseNo | string | The number in the street address. E.g. for “Testgaten 44 C” the number is ’44’. However this is used only for products or information handlers that specifically state it must be specified like this. Normally use the street data member. |
| houseSubNo | string | Some street addresses may contain an extra number, like an entrance number, floor number etc. However this is used only for products or information handlers that specifically state it must be specified like this. |

### *10.1.3.3 DmWSEiendomsInfo*

This class is used to specify a property and corresponds to the <eiendom> tag within the <APPLICANT> tag. This class has the following data members:

| name | type | description |
|---|---|---|
| kommunenr | string | County number |
| bruksnr | string | Identifies the property |
| gardsnr | string | Identifies the property |
| seksjonsnr | string | Section number |
| festenr | string | Identifies the property |
| areal | string | Area in square meters |
| address | DmWSAddress | Address of the property |

### *10.1.3.4 DmWSAppInfo*

This class corresponds to the `<appInfo>` tag. The class is used to specify a set of custom designed input data for a specific credit model. If more than one applicant is used as input to a credit model, add this information to the main applicant object only.

This class has the following data members:

| name | type | description |
|---|---|---|
| `name` | `string` | Name of this set |
| `memberValues` | `DmWSMemberValue[]` | array of data |

#### `10.1.3.4.1 DmWSMemberValue`

This class corresponds to the `<MEMBER_META_DATA>` tag and has the following data members:

| name | type | description |
|---|---|---|
| `set` | `boolean` | Currently not used |
| `name` | `string` | Name of member |
| `val` | `string` | Value of member |

#### 10.1.3.4.2 Electronic letter of notice

Special permission is required to use this alternative delivery method for letter of notification (lon). You will need to contact your sales representative. Object `DmWSAppInfo` and `DmWSMemberValue` are used to implement the use of electronic lon.

Implementation is pretty straight forward. Example:

```
// create your ws applicant object
DmWSApplicant applicant = new DmWSApplicant();
applicant.setType("P");
applicant.setFirstName("demo");
applicant.setLastName("person");
applicant.setSsn("201062"); // ddMMyy

// create your ws member value object for mobile (mandatory)
DmWSMemberValue memberValueMobile = new DmWSMemberValue();
memberValueMobile.setName("MOBILE"); // fixed upper-case constant
memberValueMobile.setVal("12345678");

// create your ws member value object for email
// (optional, but we recommend to use it if possible, ref better user experience)
DmWSMemberValue memberValueEmail = new DmWSMemberValue();
memberValueEmail.setName("EMAIL"); // fixed upper-case constant
memberValueEmail.setVal("test@test.no");

// create your ws member value array
ArrayOfDmWSMemberValue memberValues = new ArrayOfDmWSMemberValue();
memberValues.getItem().add(memberValueMobile);
memberValues.getItem().add(memberValueEmail);

// create your ws applicant info object
DmWSAppInfo appInfo = new DmWSAppInfo();
appInfo.setName("ELEKTRONISKGJENPART_STANDARD");
appInfo.setMemberValues(memberValues);

applicant.setAppInfo(appInfo);
```

### 10.1.4 getVersion

The method has no arguments. Hence you do not need a username and password to access it.
It returns an instance of the `DmWSVersion` class which has two string data members;

- `dmVersion` tells you the version of DecisionMaker e.g. “4.1.0”
- `wsVersion` tells you the version of the WebServices, e.g. “2.1”. The first number denotes the major version number. A change in this may mean there has been a new method added. The latter number increases when smaller changes has occurred, like bug fixes etc

### 10.1.5 getStatus

The method has two arguments; username and password. It returns an array of `DmWSStatus` which holds the status of miscellaneous sources within DecisionMaker. You may get one status object for the connection between DecisionMaker (DM) and its own database, one for the connection between DM and its external providers etc.
The `DmWSStatus` object has two data string members in addition to those inherited from the `DmWSSingleResponse` class;

- name which is the name of this “source”, e.g. “alpha”
- status which is the current status of the source. If nothing is wrong the string says “OK”

### 10.1.6 score / scoreX

***To be used for customized credit models only!!***
***NOTE!*** *The response from a customized credit model contains, in addition to the zone value, a recommended decision. It is a good idea to use this decision value in any external client application, not the zone value. The credit model contains a mapping from zone to decision. If the client also should have such a mapping, one must keep the client and the credit model in synch at all times. Ignoring the zone value (use it as an informational element only) and considering only the recommended decision makes the client non-dependent on any changes in the credit model.*
*Also the actual score value in points is returned. Be aware that this value is only part of the evaluation. Any policyrule is added on top of this. Thus you should never use the score points directly in any logic other than for pure informational purposes. Stick to the decision value and/or the zone value only.*

This method is to be used for those customers that have agreed (with Bisnode, BC for short) on a specific *credit model* according to their company’s credit rules. This credit model will then be implemented by BC and given a specific name. This is the name of the credit model, sometimes also named “productName”.

***Note!*** If you are looking for getting one of the generic rating models (like *Decision Score* from BC) this is available as raw information, and the *getInformation* method must be used. Product codes for these are CR (Decision Score Rating, using rating zone values), RA (Standard Credit Rating, using Triple-A coding) or a SC for using scorecard from another score engine (than DecisionMaker). See separate source documentation for more on these product codes.

There are two score methods available.

```
1. DmWSResponse scoreX(DmWSRequest request)
2. DmWSSingleResponse score(String user, String passWord, String userRef, String
   requestId, DmWSApplicant applicant, String productName)
```

These are methods that process score requests based on a named credit model.

The `scoreX` method is the most generic one and has only one argument, `DmWSRequest`, and a response object, `DmWSResponse`. The request object holds the username and password and an array of actual

request, each encapsulated in a `DmWSScoreRequest` object. It is not wise to add many score objects in one request, as it will result in a rather long response time, and it may be difficult to know if it is a problem or just a very long response time. We recommend keeping this number very low, preferably only one at a time. If a batch-system is what you need, you must contact your Bisnode sales representative. *Frequent dummy/fake requests done to check DecisionMaker's availability will be categorized as a try to jam the system, and the use will be blocked without warning.* The score request object holds the credit model name, a score group name, a comment to the score, a request id, an array of applicants, etc. Only the credit model name and some information about the applicant(s) (`DmWSApplicant`) are absolutely required.

The response, `DmWSResponse`, holds an array of `DmWSScoreResponse` objects which again holds the results for each of the `DmWSScoreRequest`'s. The `DmWSScoreResponse` object holds the result of the score process. To make it even more complicated, a score process may itself consist of several scorings. The `DmWSScoreResponse` object therefore holds an array of `DmWSScoring` objects. How many `DmWSScoring` objects are held within a score response, depends on the actual credit model. The usual is only one `DmWSScoring` object.

The `DmWSScoring` objects again holds arrays of all policy elements (`DmWSPolicyElement`), all data elements (`DmWSDataElement`), all score elements (`DmWSScoreElement`) and all total score elements (`DmWSTotalScoreElement`). There are *always* a set of `DmWSTotalScoreElement`'s in the response. If some policyrules have been applied, there are one `DmWSPolicyElement` for each policyrule that has occurred. Score elements are only returned if the *extended* data member is set in `DmWSScoreRequest`. However, this requires that the user has special rights within DecisionMaker and that the score card is included in the credit model. The data elements are meant for informational use, but are rarely or never used. All the `DmWS*Element` classes inherits from `DmWSElement` (The XML tag equivalent for this class is `<ELEMENT>`).This means they all have a *name* and a *value* (the data members are called *name* and *val* respectively. The member *val* is named like this because some development tools have used the word *value* as a special keyword). All objects also holds an array of other `DmWSElement`'s. The usual level of depth here is one. An array of attributes (`DmWSAttribute`) is also held by `DmWSElement`. Attribute objects have two data members only; *name* and *text*. As an example, a `DmWSPolicyElement` usually has an attribute named "action" containing a descriptive text telling what it is all about, like "No credit, contact someone@somewhere.com". In this case the policy object will hold a `DmWSAttribute` object with *name* equal to "action" and *text* as just mentioned.

The `DmWSScoreResponse` object also holds a `DmWSApplicant` object for each applicant. The information here is not necessarily the same as in the object in the `DmWSScoreRequest` object, as the person (or company) may have a different address and/or name than what was requested for. The applicant object in the response object holds information that is registered at the data provider (e.g. Bisnode).

#### *10.1.6.1 DmWSDataElement*

The XML tag equivalent is `<DATA_ELEMENT>`. A data element may be added when a special situation occurs in the credit model. It has an extra data member *result* telling the final decision based on this situation. Whether this will occur or not depends on the actual credit model but is rarely or never used anymore. It may be ignored unless otherwise specified in the credit model description.

#### *10.1.6.2 DmWSScoreElement*

The XML tag equivalent is `<SCORE_ELEMENT>`.The score element has no attribute objects but usually one or more sub-elements (`DmWSElement` objects) holding the values for the data involved in this score element. E.g. if the score element has no. of addresses last 3 years, there *may* be an element for each year containing the actual address.
It also has an extra data member named *result*. The *result* data member contains a number (floating point or integer) representing the element's point value for the scoring.

What kind of score element and which sub-elements are parts of a score element is decided by the design of the actual *score card*.

#### *10.1.6.3 DmWSPolicyElement*

The XML tag equivalent is `<POLICY_ELEMENT>`. The `DmWSPolicyElement` object has the following special attributes:

- key Name of this policy element
- name A more descriptive name of this policy element (may be the same as key in some cases)
- level The policy level. If negative, it is a policyrule influencing on the decision in negative direction. If positive, it is a policyrule influencing on the decision in positive direction. If the level is zero, it is a neutral policyrule, i.e. it has no influence on the decision and serves only as information.
- action A descriptive text explaining what the policyrule is about.
- zone The policyrule will increase or decrease the final zone towards this value
- result The decision this policyrule will lead to (values are “y” (yes), “c” (check) or “n” (no)).

A policy element object usually has one or more `DmWSElement` objects. These objects hold the values for the information elements that participate in this policyrule. E.g. a policyrule that has a combined check on wealth and income may have `DmWSElement` objects; one for wealth and another for income. The first could then be named *wealth* and have the actual amount in the *val* data member, and the second could be named *income* and have the actual income amount in the *val* data member.

#### *10.1.6.4 DmWSTotalScoreElement*

The XML tag equivalent is `<TOTALSCORE_ELEMENT>`. The total score elements hold information specific to each credit modeland which is specified to be returned from the scoring from the credit modelin question.

There is usually one `DmWSTotalScoreElement` object for each `DmWSScoring` object with a `DmWSScoreResponse` response. Usually it has more than one `DmWSElement` sub-elements.
The `DmWSTotalScoreElement` object itself has the following additional data members:

- *result* is the final decision for this scoring
- *points* is the total points for this scoring
- *zone* is the final zone for this scoring

***NOTE!*** *It is a good idea to use the* result *value in any external client application to get the recommended decision, not the zone value. The credit model contains a mapping from zone to decision. If the client also should have such a mapping, one must keep the client and the credit model in synch at all times. Ignoring the zone value (use it as an informational element only) and considering only the recommended decision makes the client non-dependent on any changes in the credit model.*

As there usually is only one `DmWSScoring` object for each `DmWSScoreResponse`, there is also only one `DmWSTotalScoreElement` object for each `DmWSScoreResponse`. Therefore these three data members will hold the same values as `DmWSScoreResponse` holds.

The following sub-elements are returned by default:

| | | attribute | | |
|---|---|---|---|---|
| **name** | **val** (example) | **name** | **text** | **description** |
| TotalScoring | n, y, c | zone | *Zone number* | The value and zone number are the results from the *score card* before any policyrules applies. |
| TotalPolicy | n, y, c | zone | *Zone number* | The value and zone number are the results from all the applied policyrules (if any) before the result from the *score card* has been applied. |
| Product Revision | Revision: XXX | | | The revision number for the credit model. |
| Fødselsdato [2] | 19730204 | dateFormat | yyyyMMdd | Birth date for the person in question. |
| Alder [3] | 31 | | | Age of person. |
| Fornavn [4] | | | | First name of person. |
| Etternavn [5] | | | | Last name of person. |
| Navn [6] | | | | Name of company. |
| Adresse | | | | Street address of person or company. |
| Postnr | | | | Zip code. |
| Sted | | | | City or place. |
| Land | | | | Name of country. |

Some elements may be added or removed depending on the specific credit model.

[2] For persons only
[3] For persons only
[4] For persons only
[5] For persons only
[6] For companies only

### 10.1.7 getInformation / getInformationX

There are two versions of this method:

1. `DmWSResponse getInformation(String user, String passWord, String userRef, String requestId, DmWSApplicant applicant, String informationHandler)`
2. `DmWSResponse getInformationX(DmWSRequest request)`

These methods make it possible to get information directly from an external (to DecisionMaker) or internal source.
The first method is best if you want to get information on a single applicant, which is the case most of the time. The latter method is for more generic uses.

Usually you will get an XML string back that must be parsed. The layout of this XML depends on the source, and that is the reason it cannot be converted into a WebServices object. However, DecisionMaker may extract information from some XML's if special support for those XML layouts has been implemented.
Among such XML's are:

- Score results from DecisionMaker itself (internal source)
- Score results from Bisnode directly (external source)
- Applicant information from DecisionMaker itself (internal source)
- Applicant information from Bisnode directly (external source)

Applicants will be held by one or more `DmWSApplicant` objects, and score results by one or more `DmWSScoreResponse` objects.

The second `getInformation` method has a `DmWSRequest` object as input and a `DmWSResponse` object as output, just like the `score` method. The request object must contain one or more `DmWSInformationRequest` objects. The information request object has one added data member: *informationHandler*. This is the name of a handler customized for what the customer requires and needs.

Here are examples of some handlers (there are many more):

| ***Description*** | ***Information handler*** |
|---|---|
| Own default (only when agreed with Lindorff and Bisnode) | `alpha:mislighold_filter:xml:server` |
| Information about name, address and generic rating | `alpha:products(ho,cr):xml:server` |
| Whatever you are subscribing for | `alpha:get_subscribed:xml:server` |
| Get specific products | `edrws:products(ED11116010, ED11316010):xml:server` |

These are just examples.

The `DmWSResponse` object holds an array of response objects that may be a score response (`DmWSScoreResponse`) or an information response (`DmWSInformationResponse`). In case of error it will be a `DmWSError` object. The information response objects holds one added data member, *informationResult*, which is a string containing the returned XML.
If DecisionMaker has support for finding applicants and score results in this XML, the `DmWSInformationResponse` object also holds the applicant(s). The score result(s) are added to the `DmWSResponse` object as separate `DmWSScoreResponse` objects.

For executing scorecards which is processed outside DecisionMaker, have a look at *10.3 Adding additional scorecard information and arguments for product code SC*

## 10.2 WebServices objects and XML tags mapping table

| WebServices Class name | XML tag name | Comment |
|---|---|---|
| DmWSRequest | <dmRequest> | |
| DmWSResponse | <dmResponse> | |
| DmWSSingleRequest | <request> | |
| DmWSSingleResponse | <response> | |
| DmWSAnyResponse | | Used by administration only |
| DmWSStatus | | No XML tag equivalent |
| DmWSScoringRequest | | Using the <response> tag and speciying the <productName> |
| DmWSScoreResponse | <scoreresult> | Holds final result (recommended decision) and an array of DmWSScoring objects/<SCORING> tags |
| DmWSInformationRequest | | Using the <response> tag and speciying the <informationHandler> |
| DmWSInformationResponse | | Holds the xml as delivered from external provider |
| DmWSError | <error> | |
| | <applicant> | Array of <APPLICANT> |
| DmWSApplicant | <APPLICANT> | |
| DmWSElement | <ELEMENT> | |
| DmWSMemberValue | <MEMBER_META_DATA> | Used in a request to a scoring. Owned by DmWsAppInfo/<appInfo> |
| DmWsAppInfo DmWSAppInfo | <appInfo> | In a scorerequest, holds instances of arguments to a credit model encapsulated in DmWSmemberValue/<MEMBER_META_DATA> |
| DmWSTotalScoreElement | <TOTALSCORE_ELEMENT> | |
| DmWSPolicyElement | <POLICY_ELEMENT> | |
| DmWSScoreElement | <SCORE_ELEMENT> | |
| DmWSDataElement | <DATA_ELEMENT> | |
| DmWSScoring | <SCORING> | |
| DmWSAddress | <address> | |
| DmWSEiendomsInfo DmWSProperty | <eiendom> | |
| DmWSAttribute | | An array of these are held by DmWSElement. Holds the same information as XML-tag attributes for the <ELEMENT> and the <*_ELEMENT> tags. |
| DmWSVersion | | No XML tag equivalent |

## 10.3 Adding additional scorecard information and arguments for product code SC

Using the scoring code SC one may request for a scorecard not run directly by DecisionMaker. A model number must be specified using the key “arg_scoreModel”.

### 10.3.1 Example using WebServices

*Example (your code may be a bit different depending on the generator used to create client code):*

```
DmWSInformationRequest theRequest = new DmWSInformationRequest();
DmWSElement addInfoModelName = new DmWSElement();
addInfoModelName.setName("arg_scoreModel");
addInfoModelName.setVal(3);     //The model no for the scorecard in this example is 3
DmWSElement[] addInfoArr = new DmWSElement[]{addInfoModelName};
theRequest.setAddInfo(addInfoArr);
theRequest.setInformationHandler("alpha:products(HO,SC):xml:server");
```

*If the scorecard needs arguments these can be supplied using the key “arg_scoreModelInput”:*

```
DmWSApplicant applicant = new DmWSApplicant();
DmWSAppInfo appInfo = new DmWSAppInfo();
appInfo.setName("arg_scoreModelInput");

DmWSMemberValue yearsEmployed = new DmWSMemberValue();
yearsEmployed.setName("YEARSEMPLOYED");
yearsEmployed.setVal("3”);
DmWSMemberValue grossIncome = new DmWSMemberValue();
grossIncome.setName("GROSSINCOME");
grossIncome.setVal("475000”);

//Two arguments in this example:
DmWSMemberValue[] argsArray = new DmWSMemberValue[]{
yearsEmployed, grossIncome };

appInfo.setMemberValues(argsArray);
applicant.setAppInfo(appInfo);
therequest.setApplicants(new DmWSApplicant[]{applicant});
```

### 10.3.2 Example using XML Services

```
<dmRequest version="2.0">
	<user>${user}</user>
	<password>${password}</password>
	<request>
		<informationHandler>alpha:products(HO,SC):xml:server</informationHandler>
		<addInfo>
			<hashtableXML>
				<element key="arg_scoreModel">1</element>
			</hashtableXML>
		</addInfo>
		<applicant>
			<APPLICANT>
				<type>P</type>
				<ssn>12345678901</ssn>
				<appInfo><![CDATA[
					<APP_INFO name="arg_scoreModelInput">
						<MEMBER_VALUE name="YEARSEMPLOYED">
							<VALUE>3</VALUE>
						</MEMBER_VALUE>
						<MEMBER_VALUE name="GROSSINCOME">
							<VALUE>475000</VALUE>
						</MEMBER_VALUE>
					</APP_INFO>
				]]></appInfo>
			</APPLICANT>
		</applicant>
	</request>
</dmRequest>
```

## 10.4 Differences between version 3.x of our WebServices and prior versions

Version 3.x is using the wrapped/literal style, while prior versions use RPC/encoded. Tools based on JAX-WS, MS Visual Studio tools and other newer tools used for client code generation does not support RPC/Encoded style.

The class DmWSResponse has a new data member in version 3.x named “userRef”. This will have the same contents as entered in the request in the DmWSRequest class.
The classes DmWSSingleResponse, in version 3.x, no longer has a data member named *userRef*.
The data member named “handler” in the same class is renamed to “user” in version 3.x.

## 10.5 How to get through a proxy (using Java)

While developing code you may be behind a firewall forcing you to go through a proxy to get out on the internet. In order to get through you may set the following flags using the `-D` option on the Java command line: `proxy.httpHost` and `proxy.httpPort`.

However, it seems to me that this does not work for all frameworks/libraries. Some use `http.proxyHost` and `http.proxyPort`. Some also require `http.proxySet` to be set to `true`. Or is it `proxy.httpSet`?

Because I am tired of finding out which is the absolutely correct one to use, I use to set them all: `proxyXXXX`, `http.proxyXXXX`, `https.proxyXXXX`, `proxy.httpXXXX`, `proxy.httpsXXXX`.
Just replace the `XXXX` with the words `Set`, `Host` and `Port`.

If you want to change them dynamically while your application is running, use
`System.setProperty("http.proxyXXXX","whatever it might be")`

*or*

```
System.getProperties().put("http.proxyXXXX","whatever it might be")
```

If you're using WebServices and you have a stub implementing the javax.xml.rpc.Stub interface, you could set it on the stub directly;

```
stub._setProperty("http.proxyXXXX","whatever it might be")
```

## 10.6 Getting through to an HTTPS server (using Java)

You may need to do something in order to be able to communicate to an HTTPS server (like https://www.soliditetd.no).

One of the following may do the trick for you:

```
System.setProperty("java.protocol.handler.pkgs",
                   "com.sun.net.ssl.internal.www.protocol");
```

or

```
Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
```

## 10.7 SOAP message logging

Occasionally, especially during development of a client, you would like to see the actual soap request and response. You can do that using a proxy application like *SoapMonitor* or *TcpMon*. The problem with that is that it is a hassle to set up and route the request through these application (my personal opinion).
If you're using Axis 1 v.1.1 this is fairly simple; if you configure your log4j category `org.apache.axis` to `debug`, you will get all the logging details you want and more. Set it back to `error` to stop all the gory logging details.

If you are using a JAX-WS implementation you may set one of the following:

```
System.setProperty("com.sun.xml.ws.transport.http.client.HttpTransportPipe.dump", "true");
System.setProperty("com.sun.xml.ws.util.pipe.StandaloneTubeAssembler.dump", "true");
System.setProperty("com.sun.xml.ws.transport.local.LocalTransportPipe.dump", "true");
```

Which one to use depends on which transport library you are using. To be sure you can set all three…

## 10.8 Getting access to the SOAP messages using JAX-WS

JAX-WS converts the XML received using JAXB. If you want to keep that functionality, but still get your hands on the Soap XML (not just for logging as in previous chapter, but for use in your application), you will have to do some tricks.
When generating the JAX-WS client code you will have some code generated for you. Typically a port class like DecisionMakerWebServices3 and a client class like DecisionMakerWebServices3Service.
The code may look something like:

```
/** Calling the service code **/
Qname portName = new QName("urn:DecisionMakerWebServices3",
                           "DecisionMakerWebServices3");
DecisionMakerWebServices3Service service =
   new DecisionMakerWebServices3Service(
      "https://www.soliditetd.no/webservices/services/DecisionMakerWebServices3?wsdl",
      portName);

DecisionMakerWebServices3 port = service. getDecisionMakerWebServices3Port();
```

```
// In order to pick up the soap messages we add a SOAPHandler
SOAPMessageStringHandler soapHandler =
SOAPMessageStringHandler.addSoapHandler((BindingProvider)port);

DmWSResponse response = port.scoreX(/* put request args here  */);

// Now, the soap request and response xml may be fetched from the soapHandler:
System.out.println("Request: "+soapHandler.getOutBoundString());
System.out.println("Response: "+soapHandler.getInBoundString());
/** end of calling the service code **/
```

***The `SOAPMessageStringHandler` class may be implemented like this:***

```
package no.bisnode.creditwsclients;

import javax.xml.namespace.QName;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.Binding;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.handler.Handler;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;
import java.io.ByteArrayOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;

/*
 * This SOAPHandler will fetch the contents of incoming and outgoing messages.
 */
public class SOAPMessageStringHandler implements SOAPHandler<SOAPMessageContext> {
   // change this to redirect output if desired
   private String inMsg;
   private String outMsg;

   public Set<QName> getHeaders() {
      return null;
   }

   public boolean handleMessage(SOAPMessageContext smc) {
      saveMessage(smc);
      return true;
   }

   public boolean handleFault(SOAPMessageContext smc) {
      saveMessage(smc);
      return true;
   }

   // nothing to clean up
   public void close(MessageContext messageContext) {
   }

   private void saveMessage(SOAPMessageContext smc) {
      SOAPMessage message = smc.getMessage();
      if (isOutBound(smc))
         outMsg = getMsgAsString(message);
      else
         inMsg = getMsgAsString(message);
   }

   private String getMsgAsString(SOAPMessage message) {
```

```
      String msg = null;
      try {
         message.setProperty(SOAPMessage.CHARACTER_SET_ENCODING, "iso-8859-1");
         ByteArrayOutputStream baos = new ByteArrayOutputStream();
         message.writeTo(baos);
         msg = baos.toString("iso-8859-1");
      }
      catch (Exception e) {
         System.out.println("WS Soap Message convertion failure: ", e);
      }
      return msg;
   }

   private Boolean isOutBound(SOAPMessageContext smc) {
      return (Boolean)smc.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
   }

   public String getInBoundString() {
      return inMsg;
   }

   public String getOutBoundString() {
      return outMsg;
   }

   public static SOAPMessageStringHandler addSoapHandler(
                                        BindingProvider bindingProvider) {
      SOAPMessageStringHandler soapHandler;
      Binding binding = bindingProvider.getBinding();
      List<Handler> handlerList = binding.getHandlerChain();
      if (handlerList == null)
         handlerList = new ArrayList<Handler>();

      soapHandler = new SOAPMessageStringHandler();

      handlerList.add(soapHandler);
      binding.setHandlerChain(handlerList);
      return soapHandler;
   }
}
```

## 10.9 Visual Studio .NET

The following descriptions relate to Microsoft Visual Studio .NET

Visual Studio has the ability to let you point at a WSDL file and generate the code for you. The language may be your own choice between those in the package; C++, C#, J# and Visual Basic.

You may also be interested in having a look at Microsoft's newly released free tools at http://www.microsoft.com/express/ available for all languages from Microsoft.

In this chapter we will point out for you how to integrate with DecisionMaker WebServices.
You may get some help on the subject if you inside Visual Studio enter the text "WebServices" in the help system and pressing the search button. Sort the hit list by name and see if you can find "Programming the Web with XML WebServices". Look for "Accessing XML WebServices".

When you have created your new project, enter the Project menu and select "Add Web Reference". Paste the string "https://test.soliditetd.no/webservices/services/DecisionMakerWebServices3?wsdl" for TEST and

“https://www.soliditetd.no/webservices/services/DecisionMakerWebServices3?wsdl” for production into the URL edit box and press the “Go” button. Set a name on the reference and press the “Add Reference” button. The client stub code will now be generated. On the right hand side, in the Solution Explorer, you will find the DecisionMaker WebServices under “WebServices”. Double-click on the reference, and you will see the generated classes in the Object Browser on the left hand side. It is named “DecisionMakerWebServices3Service”.

**NOTE!** If you choose C++ as your language, Visual Studio uses another application named “sproxy.exe” to generate the code. See http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore/html/vcconSProxyexe.asp. “sproxy.exe” can also be found in the “\vc*\bin” directory of your Visual C++ installation. Unfortunately this application does NOT support inheritance (as of 24. May 2006, you get “…: error SDL1030 : sproxy.exe does not support extension of complexType”), and since `DmWS*Element` classes all inherit from `DmWSElement`, you will get an error. It seems that “sproxy.exe” is a bit out of date and not followed up by Microsoft. BUT, if you choose the .NET version of C++, Visual Studio will use the “managed” version of C++ instead of the one that generates native code. The managed version generates Common Language Runtime code (CLR) instead of native code. But, even when using that, Visual Studio generates a bug in the code. The class `DmWSElement`, which is the base class for several other classes, is generated AFTER one or more of its sub-classes, thus creating a compile error claiming `DmWSElement` is not defined (in our case: “`no.lindorffd.ws.h(370): error C2504: 'no::lindorffd::ws::DmWSElement' : base class undefined`”). If you get this, click on the error message. This opens up the editor pointing at the sub-class where it claims the base class is undefined. Locate the `DmWSElement` class and cut and paste it before the point where you got the error. Save and recompile/build. If you get more of the same error, repeat the operation until the error messages are gone.

We presume that Microsoft will prefer that you use C#, Visual Basic, or managed C++ rather than native code. Hence *you cannot generate C++ native code for DecisionMaker WebServices using Visual Studio*. A better choice could be another tool, like one of the Borland/CodeGear tools or some other tool out there.

Be also aware that Visual Studio *does NOT generate* getter and setter methods for the data members as would be natural for object oriented code (“data hiding” is one of the main principles in object orientation that Microsoft seems NOT to follow).

If you, for some reason, while developing your code, you want to bypass the certificate check, the following shows a C# example on how to avoid this:

```
public class MyCertificateTestPolicy : System.Net.ICertificatePolicy
{
    public MyCertificateTestPolicy ()
    {
    }

    public bool CheckValidationResult(ServicePoint sp,
System.Security.Cryptography.X509Certificates.X509Certificate cert, WebRequest
request, int problem)
    {
        return true;
    }
}
```

Then:

```
System.Net.ServicePointManager.CertificatePolicy = new MyCertificateTestPolicy
();
```

## 10.10 Apache Axis

Latest versions may be downloaded from http://ws.apache.org/axis. A version 2 of Axis is available at http://ws.apache.org/axis2. Axis 2 supports JAX-WS. Axis 2 is supposed to support generation of both Java and C/C++.
The code presented here serves as an *example* on how things can be done, not necessarily how it *should* be done… The examples are based on Axis 1.

### 10.10.1 Axis 1 C/C++

Get it at http://ws.apache.org/axis/cpp.

Documentation is to be found at http://ws.apache.org/axis/cpp/documentation.html or http://wiki.apache.org/ws/FrontPage/Axis.

At http://ws.apache.org/axis/cpp/winuser-guide.html#ssl you can read about how to communicate via SSL using your C/C++ client.

When generating the C or C++ code you need to run a Java application.

```
java org.apache.axis.wsdl.wsdl2ws.WSDL2Ws <wsdl file> -o<output directory>
-l<c|c++> -s<(server|client)>.
```

Also note that there cannot be any spaces after a switch (i.e. -o, -l).
For some reason it seems that the –lc switch (generate C code) does not work. C++ code is generated anyway.

So the steps are as follow:
Open the page www.soliditetd.no and then use the right mouse button on the link for the service you want to generate WSDL file for. Choose “Save Target As” and save it wherever you want it. We suggest you name the WSDL file “`DecisionMakerWebServices3.wsdl`”.

To generate client code for scoring:

```
java org.apache.axis.wsdl.wsdl2ws.WSDL2Ws DecisionMakerWebServices3.wsdl –osrc/cpp
–lc++ -sclient
```

This command generates the C++ code in the “src/cpp” directory. If you get an error during generation, try putting the `wsdl2ws.jar` as the very first jar in the class path and run it again.

Unfortunately, the Apache Axis 1 generated C++ does *not* define setter and getter methods to the data members as the Java generated code does. This means there are no methods like `setPassWord( “password” );` as you can do in Java (in the `DmWSRequest` class).
Instead you will have to do `passWord = “password”;` All strings are of type `xsd__string`, which is of type `AxisChar*` where `AxisChar` again is the same as **`char`** (a `#define` in `GDefine.h`).

*Be aware though* that the C++ code generation does NOT generate code with inheritance, meaning `DmWSPolicyElement` does not inherit from `DmWSElement` as it should. Another example is `DmWSScoreRequest` which should inherit from `DmWSSingleRequest`. Instead the generated code for let’s say `DmWSScoreRequest` includes all data members that is also in `DmWSSingleRequest`. A bit strange behavior and it force you to cast the object from to `DmWSScoreRequest` on `DmWSSingleRequest` request and from DmWSSingleResponse to DmWSScoreResponse on response. We haven’t tried out this way of doing it, but it should probably work even though it has nothing to do with C++ inheritance.

The reason for this weird “inheritance scheme” is that Axis 1 C++ (v. 1.6) does not support WSDL extensions which are necessary for this to happen.

At http://ws.apache.org/axis/cpp/winuser-guide.html you will find a description on how to call the server using the generated code, and there are also some hints on using Axis with Microsoft Visual C++.

Apache also has a good XML parser for C/C++ at http://xml.apache.org/xerces-c.

If you want a good free C/C++ development tool, take a look at Dev-C++ at http://bloodshed.net/dev/devcpp.html.

Other free C/C++ compilers:

- http://www.mingw.org/ (a descent port of the GNU CC compiler)
- Borland/CodeGear's BCC32 compiler can be downloaded from http://www.codegear.com/downloads/free/cppbuilder. They also have several more or less free tools under the "Turbo" brand. These can be downloaded from http://cc.codegear.com/free/turbo.
- List of free compilers: http://www.bloodshed.net/compilers

### 10.10.2 Apache Axis 1 for Java

Documentation is to be found at http://ws.apache.org/axis/java.
Axis 2 can be found at http://ws.apache.org/axis2/. However, the Axis 2 is a completely new and rewritten version of Axis and has nothing to do with Axis 1. We haven't tested the Axis 2. The examples herein are based on Axis 1 version 1.1.

If you in the test version of your Axis Java client wants to bypass checking the certificate, add this to your code:

```
System.setProperty("axis.socketSecureFactory",
"org.apache.axis.components.net.SunFakeTrustSocketFactory");
```

Add this line before the code that instantiates the service object.
NOTE: this code should not be used in a production environment.

Generate the Java client code (example):

```
java org.apache.axis.wsdl.WSDL2Java -p no.bisnode.creditwsclients.ws3.jaxrpc.generated -o
src/java/ https://www.soliditetd.no/webservices/services/DecisionMakerWebServices3?wsdl
```

The generated code from this example is in the `no.bisnode.creditwsclients.ws3.jaxrpc.generated` package in the "src/java" directory.

For more examples have a look at 12 Java examples.

### *10.10.2.1 How to get through a proxy using Axis 1 v.1.1*

Unfortunately going through a proxy using Axis 1 v.1.1 seems not to work as described in chapter 10.5 How to get through a proxy.
In order to do that you must replace the http transport library used by Axis 1. You may do that by using the Apache HttpClient (http://hc.apache.org/) instead of the built-in one. What you do is open up the `axis.jar`, go to `org.apache.axis.client` and open up the file named `client-config.wsdd`. In this file you change the line:

```
<transport name="http" pivot="java:org.apache.axis.transport.http.HTTPSender"/>
```

into

```
<transport name="http" pivot="java:org.apache.axis.transport.http.CommonsHTTPSender"/>
```

Then repackage the jar.
Remember to download the Apache HttpClient jar file. http://hc.apache.org/httpclient-3.x.
A version 4.x is available from http://hc.apache.org/.

However, still I was unable to make this work smoothly, so I took the Axis 1 v.1.1 source, opened up the `org.apache.axis.transport.http.CommonsHTTPSender.java` file, in the `invoke` method, after the line that says:

```
HostConfiguration hostConfiguration = getHostConfiguration(httpClient, targetURL);
```

I entered the following code:

```
/*** THE FOLLOWING CODE IS ADDED TO HANDLE PROXY SETTINGS ***/
javax.xml.rpc.Call call = (javax.xml.rpc.Call)msgContext.getProperty("call_object");
boolean proxyIsSet = false;
if (call != null) {
	String proxyHost = (String)call.getProperty("http.proxyHost");
	if ((proxyHost != null) && (proxyHost.trim().length() > 0)) {
		String proxyPort = (String)call.getProperty("http.proxyPort");
		if (proxyPort != null) {
			try {
				int iPort = Integer.parseInt(proxyPort);
				if (iPort > 0) {
					hostConfiguration.setProxy(proxyHost, iPort);
					proxyIsSet = true;
				}
			}
			catch (Throwable e) {
				// ignore, could occur if 'proxyPort' is not a number
			}
		}
	}
}
/*** THE ABOVE CODE IS ADDED TO HANDLE PROXY SETTINGS ***/
```

Then recompile the file using Java 1.4.x (it seems Axis 1 doesn't like newer versions) and put it back in the axis.jar.
Probably there is a nicer way to accomplish the same. But this solution does work.

## 10.11 Apache CXF

The CXF library (http://cxf.apache.org) seems to be very modern and update. It supports JAX-WS. It requires at least JDK 1.5. It seems to be well documented, although we haven't tried it out. From the documentation the CXF seems to be easier to use than Axis2 but this doesn't mean it is a not so advanced library.

*Note* that because CXF does not support the *RPC/Encoded* WSDL style you *must* use version 3 or higher of our WebServices WSDL files.

Another CXF based library is *FUSE* (http://open.iona.com/products/enterprise-cxf/).

Read about the difference between the two at
http://cxf.apache.org/faq.html#FAQ-What%2527sthedifferencebetweenCXFandFUSE%253F.

## 10.12 JAX-WS (Glassfish/Metro)

You can find the Glassfish /Metro reference implementation at https://jax-ws.dev.java.net/. Read more about JAX-WS in general at http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/.

*Note* that because JAX-WS does not support the *RPC/Encoded* WSDL style you *must* use version 3 or higher of our WebServices WSDL files.

In order to generate Java client stub:

```
java com.sun.tools.ws.WsImport -verbose -p
no.bisnode.creditwsclients.ws3.jaxws.generated -Xnocompile -s src/java
https://www.soliditetd.no/webservices/services/DecisionMakerWebServices3?wsdl
```

The code generated will be put in the src/java directory in this case.

You may find a simple example on how to do a call in chapter 10.8 Getting access to the SOAP messages using JAX-WS. Besides that the example code in 12 Java examples are valid.

# 11 Plain XML Services (NOT WebServices)

XML Services are just pure strings representing an XML as request and response. This will make it easy to transform to other formats, it be HTML, PDF, etc, using some kind of conversion tool (like Apache Xalan (http://projects.apache.org/projects/xalan_for_java_xslt_processor.html).

## 11.1 Request

A request may look like this:

Table 1 Example on Request XML

```
<dmRequest version="1.0">
        <user> USER </user>
        <password> PASSWORD </password>
        Specific request format
</dmRequest>
```

The <user> and <password> must ALWAYS be present. The servlet authenticates the user before the job starts. If the user is not authenticated an error XML structure is returned. The version attribute to the `dmRequest` tag is optional and denotes the XML Services Structure version. Default is version 1.x.

The "specific request format" mentioned here is illustrated in the following example:

Table 2 Example on Request XML

```
<request id="whatever">
        Applicant specification
        Request contents
</request>
```

The "id" attribute is optional and could be set to anything you like. It is returned as an attribute with the response (see below). It may therefore be used to identify the response if needed.

The "Request contents" depends on what you want to do;

- Score request (on *customized* credit model)
- Information handler request (includes *generic* rating/score)
- Status

For the two first, an applicant is almost always required.
See chapters about "DmWSApplicant" and "applicant".

## 11.2 Response

Table 3

Example on XML output.

```
<dmResponse version="2.0">
  <response id="1">
    info
  </response>
</dmResponse>

<dmResponse>
    <response id="1">
       info
    </response>
    <error id="2">
       info
    </error>
</dmResponse>

<dmResponse>
    <response id="1">
       <error>
              info
       </error>
    </response>
</dmResponse>
```

The *id*-argument to the *response*-tag has the same value as for the *request* tag. In the example above there is an ordinary reply (uppermost) and the two last responses shows that one of the requests resulted in an error message. The id-value is in this case delivered back as an attribute to the error-tag. Note that the error-tag may appear both inside and outside of the response tag. If it is at the outside, it is a clear indication on that the main request has failed, like if the user tag was missing or wrong, or if the `dmResponse` tag was not ended, etc.
The version attribute is set to the same as the corresponding attribute in the `dmRequest` tag. It is also only returned if the version number is 2.0 or higher.

## 11.3 applicant

An “applicant” is the person or company/organization you want information on. There are two versions of the applicant specifications.

Table 4

Applicant XML structure for XML Services version 1.0 in a request

```
<applicant>
        <APPLICANT
                type=”P”
                ssn=”social security number”
                extClientNr=objectno”
                firstName=”first name”
                lastName=”last name”
                street=”street”
                zipCode=”zip code”
                city=”city”
                country=”country”
                email=”some@somewhere.com”
                lonMobile=”mobile phone number”
                lonEmail=”some@somewhere.com”
                nationality=”no”
                no=”1”
                areal=”nnn”
                bruksnr=”nn”
                festenr=”nn”
                gardsnr=”nn”
                kommunenr=”nn”
                seksjonsnr=”nn”/>
</applicant>
```

Table 5

Applicant XML structure for XML Services version 2.0 in a request

```
<applicant>
        <APPLICANT
           type=”P” nationality=”no” no=”0”>
                <ssn>social security number</ssn>
                <extClientNr>objectno</extClientNr>
                <firstName>first Name</firstName>
                <lastName>last Name</lastName>
                <name>company name</name>
                <email>some@somewhere.com</email>
                <lonMobile>mobile phone number</lonMobile>
                <lonEmail>some@somewhere.com</lonEmail>
                <address>
                        <street>street</street>
                        <zipCode>zip code</zipCode>
                        <city>city</city>
                        <country>country</country>
                </address>
                <eiendom>
                        <areal></areal>
                        <bruksnr></bruksnr>
                        <festenr></festenr>
                        <gardsnr></gardsnr>
                        <kommunenr></kommunenr>
                        <seksjonsnr></seksjonsnr>
                        <address>
                                <street>street</street>
                                <houseNo>43</houseSubNo>
                                <houseSubNo>3</houseSubNo>
                                <houseLetter>B</houseLetter>
                                <zipCode>zip code</zipCode>
                                <city>city</city>
                                <country>country</country>
                        </address>
                </eiendom>
        </APPLICANT>
</applicant>
```

| | | Version 1.x | | Version 2.x | |
|---|---|---|---|---|---|
| **Name** | **Description** | **Attribute or tag** | **Parent** | **Attribute or tag** | **Parent** |
| applicant | Contains description of one or more applicants. *Required* | Tag | request | Tag | request |
| APPLICANT | Holder describing one applicant. *Required* | Tag | applicant | Tag | applicant |
| type | "P" for person or "F" for company. *Required* | Attribute | APPLICANT | Attribute | APPLICANT |
| nationality | Specifies nationality of the applicant. Norwegian is default. *Optional* *Values must correspond with the two-letter character code as specified on* http://www.iso.org/iso/english_country_names_and_code_elements | Attribute | APPLICANT | Attribute | APPLICANT |
| no | Applicant number. 0 is the main applicant and is default. 1 denotes the first co applicant. *Optional.* | Attribute | APPLICANT | Attribute | APPLICANT |
| ssn | The social security number (for persons) or the organization number (for companies) must be specified here. If a complete ssn is not available, only birth date may be set on the form ddMMyy. Using the Social Security Number, i.e. **ssn**, to find a person or company is the most secure way to find a unique match. For persons, if complete ssn is specified, only that is used to find the person, and the rest of the data is ignored. Hence using another person's ssn in combination with the name will result in a request for data on the person that matches the ssn only and not the name. Be aware of this and check that the name and address DM returns are correct. If complete ssn is not given, all data is used in finding the person. *Optional.* | Attribute | APPLICANT | Tag | APPLICANT |
| extClientNr | Bisnode gives all applicants stored in their database a unique number, called an *object number*. It is possible to use the object number in the request in place of the ssn. Use extClientNr attribute or tag *instead* of ssn. If both are specified, the object number is ignored. *Optional.* | Attribute | APPLICANT | Tag | APPLICANT |
| firstName | First names of person. *Optional.* | Attribute | APPLICANT | Tag | APPLICANT |
| lastName | Last name of person. *Optional.* | Attribute | APPLICANT | Tag | APPLICANT |
| name | Name of company. *Optional.* | Attribute | APPLICANT | Tag | APPLICANT |
| email | E-mail address. Not in use. *Optional.* | Attribute | APPLICANT | Tag | APPLICANT |

| | | | | | |
|---|---|---|---|---|---|
| lonMobile | Mobile phone number. Used for electronic letter of notice, requires special permission. Mandatory if to use electronic lon. | Attribute | APPLICANT | Tag | APPLICANT |
| lonEmail | E-mail address. Used for electronic letter of notice, requires special permission. Optional if to use electronic lon. | Attribute | APPLICANT | Tag | APPLICANT |
| address | Specifies the address for the applicant. *Optional.* | - | - | Tag | APPLICANT |
| street | Street address. *Optional.* | Attribute | APPLICANT | Tag | address |
| zipCode | Zip code. *Optional.* | Attribute | APPLICANT | Tag | address |
| city | City or place. *Optional.* | Attribute | APPLICANT | Tag | address |
| country | *Optional*. Default is taken from the *nationality* tag/attribute. | Attribute | APPLICANT | Tag | address |
| eiendom | Specifies a property. | - | - | Tag | APPLICANT |
| areal | Area in square meters. | Attribute | APPLICANT | Tag | eiendom |
| bruksnr | Identifies the property. *Optional.* | Attribute | APPLICANT | Tag | eiendom |
| gardsnr | Identifies the property. *Optional.* | Attribute | APPLICANT | Tag | eiendom |
| festenr | Identifies the property. *Optional.* | Attribute | APPLICANT | Tag | eiendom |
| seksjonsnr | Identifies the property. *Optional.* | Attribute | APPLICANT | Tag | eiendom |
| kommunenr | County number. *Optional.* | Attribute | APPLICANT | Tag | eiendom |
| address | Specifies the address for the property. *Optional.* | - | - | Tag | eiendom |
| street | Street address of property. *Optional.* | - | - | Tag | address |
| zipCode | Zip code for property. *Optional.* | - | - | Tag | address |
| city | City or place of property. *Optional.* | - | - | Tag | address |
| country | Country of property. *Optional*. Default is taken from the *nationality* tag/attribute. | - | - | Tag | address |
| houseLetter | The letter for the house in the street address. E.g. for “Testgaten 44 C” the letter is ’C’. However this is used only for products or information handlers that specifically state it must be specified like this. Normally use the `street` data member. | - | - | Tag | address |
| houseNo | The number in the street address. E.g. for “Testgaten 44 C” the number is ’44’. However this is used only for products or information handlers that specifically state it must be specified like this. Normally use the `street` data member. | - | - | Tag | address |
| houseSubNo | Some street addresses may contain an extra number, like an entrance number, floor number etc. However this is used only for products or information handlers that specifically state it must be specified like this. | - | - | Tag | address |

If using XML Services version (see 11.1 Request) and version is 2.0 or above, more than one APPLICANT tag may be supplied. This is only needed when performing a score request with co applicants. In such a case

the `no` attribute will denote which is the main applicant and which is the co applicant. The main applicant is always numbered as zero and the first co applicant as number 1.

### 11.3.1 Adding a co Applicant

Co applicants may be added to a score request if the credit model supports it. Otherwise it will be ignored. Only XML Services version 2.0 and upwards supports co applicants. The co applicant(s) are added as APPLICANT tags and numbered from 1 and up.

## 11.4 addInfo (adding arguments to an information handler)

Table 6

AddInfo XML structure Hashtable XML.

```
<addInfo>
   <hashtableXML>
      <element key="KEY1" type="CLASSNAME1">VALUE1</element>
      <element key="KEY2" type="CLASSNAME2">VALUE2</element>
   </hashtableXML>
</addInfo>
```

`CLASSNAME` is a Java-class name, for example `java.lang.Integer`. The class name may be omitted (assumed to be a String).

`<addInfo>` is used to send additional information to be used in the request. This is used in customized requests using information handlers and is not used in score requests. See an example in chapter 11.8.1. The actual use will be specified for the information handler in question when needed.

## 11.5 Scoring

When it comes to scoring there are several servlets; *ScoreServlet* which performs the actual scoring and the *UpdateServlet* where one may update the DecisionMaker database with a different decision for an already performed scoring.
Of these, the first is of most interest as most customers logs/saves the decisions returned from DecisionMaker themselves, and therefore updating it in the DM database is of no interest or use.

### 11.5.1 ScoreServlet (scoring/score)

***To be used for customized credit models only!***
***NOTE:*** *The response from a customized credit model contains, in addition to the zone value, a recommended decision. It is a good idea to use this decision value in any external client application, not the zone value. The credit model contains a mapping from zone to decision. If the client also should have such a mapping, one must keep the client and the credit model in synch at all times. Ignoring the zone value (use it as an informational element only) and considering only the recommended decision makes the client non-dependent on any changes in the credit model.*

Accessing this servlet is the same as calling the DecisionMaker WebServices *score* method. See chapter 10.1.6 score on page 25.

This servlet is to be used for those customers that have agreed (with Bisnode, BC for short) on a specific (customized) *credit model* according to their company's credit rules. This credit model will then be implemented by BC and given a specific name. The name of the credit model is sometimes also named "productName".

If you are looking for getting one of the generic rating models (like *Decision Score* from BC) this is available as raw information, and the information handler servlet must be used. See 11.6.3 GetInformationServlet.

***XML Services:***
The ScoreServlet is named *scoring/score* and a request using the HTTP GET method would be:
`https://<HOST NAME>:<PORT>/scoring/score?request=<request XML>`

*Note* that both the `<scoreGroupName>` tag and the `<comment>` tag may be omitted from the request. A suggested use for the *comment* tag is to add some information to uniquely identify the person that is responsible for the request.

### *11.5.1.1 Request*

Table 7

Request XML format.

```
<request id="1">
        APPLICANT
        APPINFO
        <productName>CREDIT MODEL NAME</productName>
        <scoreGroupName>SCOREGROUPNAME</scoreGroupName>
        <comment>user identification?</comment>
</request>
```

The `productName` tag is required and is case sensitive. It specifies the exact name of the credit model.

The `scoreGroupName` is optional. The content is a name chosen by the user to identify a group of scorings. Let us say it is some sort of advertising campaign going on. All scorings during the campaign may then be grouped in a score group named "Spring 2010". The score group may be used later on to perform analysis on how the campaign went. If `<scoreGroupName>` is missing "default" is used, and if the scoring is performed by using our web client, "webscoring" is used.

`comment` tag is also optional. It is a free text field making it possible to attach a comment to the specific scoring. One type of use is to add information that identifies the person performing the request.

The contents of `comment` and `scoreGroupName` are both returned with the result. The name of the credit model is also returned.

APPLICANT structure is shown on page 43.
APPINFO structure is explained on page 48.

#### 11.5.1.1.1 A score request example

Table 8

Example on a call to the score servlet

```
<dmRequest version="2.0">
        <user>username</user>
        <password>password</password>
        <request id="1">
                <applicant>
                   <APPLICANT type="P">
                      <ssn>12345678901</ssn>
                   </APPLICANT>
                </applicant>
                <productName>Standard Person</productName>
                <scoreGroupName>Scoregroupname</scoreGroupName>
                <comment>user id info</comment>
        </request>
</dmRequest>
```

The “id” attribute to the <request> tag is optional. It will be returned with the response as “id” attribute to the <response> tag.

#### 11.5.1.1.2 appInfo (adding arguments to a score request)

`<appInfo>` is to be used for extra parameters/arguments for the specific credit model. The name of the elements to be used is credit model specific and will be documented in the credit model documentation.

Table 9 Applicant Info XML structure in a request

```
<appInfo>
    <![CDATA[
        <APP_INFO>
            <MEMBER_VALUE isSet=”true” name=”isCustomer”>
                <VALUE>true</VALUE>
            </MEMBER_VALUE>
        </APP_INFO>
    ]]>
</appInfo>
```

The extra arguments are to be put in a special XML within a CDATA section in `appInfo`. This XML starts with the APP_INFO tag. Each argument is then specified in a MEMBER_VALUE tag. The attribute “`isSet`” should always be set to true. The name of the argument must be specified in the “name” attribute. The value is to be put in a special VALUE tag inside the MEMBER_VALUE.

##### *11.5.1.1.2.1 Adding co applicant arguments*

A score request may be done on the main applicant, but a co-applicant may also be used. The co applicant arguments must be added using the `appInfo` tag.

As for the `MEMBER_VALUE` tag it is to be put in the `CDATA` section, but the tag name is `CO_APP_INFO`. The “name” attribute must be set to “`coApp1`”. If you need more than one co applicant, add them and name the “`coApp2`”, “`coApp3`” and so on. However, this is only supported in version 2.x

In XML Services version 1.x, specify another `APPLICANT` tag inside this tag.
In XML Services version 2.x, add the second `APPLICANT` tag to the main applicant tag.

The co applicant may be set up with arguments; specify the needed `MEMBER_VALUE` tags inside the `CO_APP_INFO` tag as in the example below.

Table 10

Applicant Info XML structure for co applicant (using XML Services version 1.x)

```
<appInfo>
    <![CDATA[
        <APP_INFO>
            <MEMBER_VALUE isSet="true" name="isCustomer">
                <VALUE>true</VALUE>
            </MEMBER_VALUE>
            <CO_APP_INFO name="coApp1">
               <APPLICANT
                       type="P"
                       ssn="SSN"
                       firstName="First Name"
                       lastName="Last Name"
                       street="Street"
                       zipCode="Zip Code"
                       city="City"
                       country="Country"
               />
               <MEMBER_VALUE isSet="true" name="isCustomer">
                 <VALUE>true</VALUE>
               </MEMBER_VALUE>
            </CO_APP_INFO>
        </APP_INFO>
    ]]>
</appInfo>
```

It has no meaning specifying co applicants if the credit model is not done in such a way that this is taken into consideration.
*Note!* In XML Services version 2.x the `APPLICANT` tag is to be added to the main `<applicant>` tag! See chapter 11.3.1 Adding a co Applicant.

### *11.5.1.2 Response*

See Table 11 for an example on how a response from a score request may look.

- <id> is an integer, a reference number unique for this scoring. The number is automatically generated by DecisionMaker. It is possible to use this number in order to get details on the scoring at a later stage. It might therefore be an idea to keep this number in the client system as a reference if the client is supposed to keep information about the score results.
- <created> is the date the scoring is performed. The date is on the format yyyy-MM-dd, unless otherwise specified in a "format" attribute.
- <result> is the final *recommended* decision:
    - Y : "Yes", green light
    - C : "Check", yellow light)
    - N : "No", red light
    - F : "Failed". When failed, it is usually thrown an exception resulting in an error message. You should therefore never see this decision returned.
    - O: "Other". For some credit models the values of Yes, Check and No has no meaning. In such cases, and in cases where no decision could be made, this decision value is used. In the first case, the credit model will return a value (in another tag) that is agreed with the customer.
- <points> is the rating value or the sum of points calculated for the applicant. Ignore this and use the value from the zone tag instead (if needed).
- <decision> contains more detailed information about the scoring.
- <zone> is the final zone value.
- <productName> shows the name of the credit model.

***NOTE!*** *It is a good idea to use the* result *value in any external client application to get the recommended decision, not the* zone *value or the* points*. The credit model contains a mapping from points to zone and*

*from zone to decision. If the client should have such a mapping in their system as well, one must keep the client and the credit model in synch at all times. Ignoring the zone value (use it as an informational element only) and considering only the recommended decision makes the client more non-dependent on changes in the credit model.*

Example of a response from a score request for the credit model “My Product” in Table 11.

Table 11

```
<?xml version="1.0" encoding="iso-8859-1"?>
<dmResponse>
 <response id="1">
  <info><internal>
    <dm>
     <scoreresult>
      <id>2184852</id>
      <created>2004-02-06</created>
      <handler>username</handler>
      <result>c</result>
      <points>0</points>
      <zone>2</zone>
      <score>
        <SCORING name="My Product">
           <SCORE_ELEMENT name="Rate3" result="0.19">
              <ELEMENT name="Rate3" value="0.19"></ELEMENT>
           </SCORE_ELEMENT>
         </SCORING>
      </score>
      <decision>
       <SCORING name="My Product" result="c" scoredate="2004-02-06"
zone="2" HLimit="71-100" MLimit="60-&gt;71" LLimit="0-&gt;60">
        <ZONE_INFO>
         <zone no="1" result="n">0.0</zone>
         <zone no="2" result="c">60.0</zone>
         <zone no="3" result="y">71.0</zone>
         <zone no="4" result="y">81.0</zone>
         <zone no="5" result="y">91.0</zone>
        </ZONE_INFO>
        <POLICY_ELEMENT name="PF14" level="-1" result="c" zone="2"
action="Foretaksform ikke scoret"></POLICY_ELEMENT>
        <TOTALSCORE_ELEMENT name="TotalScoring" result="c" points="0"
zone="2">
         <ELEMENT name="TotalScoring" value="o" zone="0"/>
         <ELEMENT name="TotalPolicy" value="c" zone="2"/>
         <ELEMENT name="Product Revision" value="$Revision: 3.6 $"/>
         <ELEMENT name="Orgnr" value="877273512"/>
         <ELEMENT name="Navn" value="company name"/>
         <ELEMENT name="Adresse" value="street 1"/>
         <ELEMENT name="Postnr" value="zip code"/>
         <ELEMENT name="Sted" value="a place"/>
        </TOTALSCORE_ELEMENT>
       </SCORING>
      </decision>
      <productName>My Product</productName>
     </scoreresult>
    </dm>
   </internal>
  </info></response>
</dmResponse>
```

<SCORING> shows details on the scoring. Details will vary from between credit models. Almost anything can be returned by specifying it in the credit model. <scoring><SCORING> contains information about the *score card* inside this *credit model*. <decision><SCORING> contains information about any applied policyrules, and details about the final decision. In the case of this example, the score card obviously contains only one element which is the precalculated rating value for the person in question from Bisnode. For those using a credit model with a customized scorecard the number of SCORE_ELEMENT tags will be more than one. The details of the score card are only returned if the *extended* attribute was set in the request.

The answer to the scoring is to be found inside the <SCORING> tag.

| ***Tag*** | ***Attribute*** | ***Description*** |
|---|---|---|
| <SCORING> | name | The name on this scoring, usually the same as the credit model name. |
| | result | The decision result for this scoring<br>• Y : "Yes", green light)<br>• C : "Check", yellow light<br>• N : "No", red light<br>• F : "Failed"<br>• O : "Other" |
| | scoredate | The date the scoring has been done. |
| <DATA_ELEMENT> | | If the information to be collected is missing, a <DATA_ELEMENT> indicates what kind of information is missing. Whether this will result in an error message or not, depends on how the credit model has been implemented / written. |
| <POLICY_ELEMENT> | | There may be zero, 1 or more of these describing what have happened and how it affects the final zone. The "action" attribute contains a descriptive text about what has happened. |
| <SCORE_ELEMENT> | | These are not parts of the regular score response. Previous score results may be picked up if subscribed for.<br>Example:<br><SCORE_ELEMENT name="Age" result="-5"><br><ELEMENT name="Age" value="21" type="java.lang.Integer"/><br></SCORE_ELEMENT><br>The value of the element is to be found in the value attribute to the <ELEMENT> tag. The score points for this element alone is in the result attribute. The name of the element is set in the *name* attribute. |
| <TOTALSCORE_ELEMENT> | | This element contains the total result of the score elements in the <ELEMENT> tag named *TotalScoring*. The absolute final zone and result is set in the attributes for the TOTALSCORE_ELEMENT itself. The zone and result of the policyrules alone are in the ELEMENT tag with the name attribute set to *TotalPolicy*. There are also other sub <ELEMENT> tags here. The name of these and the number of such tags depends on the credit model. |

Date values are on the form yyyy-MM-dd unless otherwise specified in the XML response.

## 11.6 Information

If doing (customized) scoring is not the task wanted, it is possible to fetch information from different sources/information providers through information handlers. However, doing this requires specific rights, permissions, or subscriptions.
An information handler is especially designed for the purpose of use. If you tell us what kind of information you want to be returned, we can tell you what kind of information handler to use. The name of the handler you need will then be given you at request.

The following servlets represents the “information interface”;

- *InformationHandlerServlet*, lists available *informationhandlers*
- *GetInformationServlet*, fetches information from a source
- *StatusServlet*, checks DecisionMaker status

**NOTE!**
***We will NOT change the overall structure, remove existing tags or attributes without prior warning. It is not guaranteed that all tags and attributes are present for every company or person. Do not rely on any specific order of tags in the xml.***

### 11.6.1 Information handlers

The information handler name is given to the *GetInformationServlet* and/or *UpdateInformationServlet* in order to specify what kind of information is wanted. E.g. if you want to fetch *a previous* ***customized*** *scoring*, you may use `dm:score_result_simple:xml:server or dm:score_result_extended:xml:server`) with the *GetInformationServlet*.

Here are examples of some handlers (there are many more):

| ***Description*** | ***Information handler*** |
|---|---|
| Own default (only if agreed with Lindorff and Bisnode) | `alpha:mislighold_filter:xml:server` |
| Specified products: address and generic rating | `alpha:products(ho,cr):xml:server` |
| Whatever you are subscribing for | `alpha:get_subscribed:xml:server` |

If you know the name of the information handler and give this to the InformationHandlerServlet, the response will be details about how to use the handler. These are just examples.

Table 12

Example on request for info on a specific information handler

```
<request>
        <informationHandler>alpha:get_subscribed:xml:server
        </informationHandler>
</request>
```

### 11.6.2 InformationHandlerServlet

The InformationHandlerServlet lists information about the information handlers in the DecisionMaker server. In order to have access to this, certain rights have to be connected to the user.

There is no WebServices method for this.

***XML Services:***

The InformationHandlerServlet is named *information/informationhandlers* and an HTTP GET request would look like:

```
https://<HOST NAME>:<PORT>/information/informationhandlers?request=<request XML>
```

### 11.6.3 GetInformationServlet

The *GetInformationServlet* gets information from DecisionMaker server by calling one of the information handlers.

The WebServices equivalent is one of the *getInformation* methods. See chapter about "getInformation".

***XML Services:***

The *GetInformationServlet* is named *information/get* and an HTTP GET request would look like:

```
https://<HOST NAME>:<PORT>/information/get?request=<request XML>
```

See 11.8.1 How to get a previous scoring result from a customized credit model (example) and 11.8.2 Example: requesting address information.

#### *11.6.3.1 Request*

Table 13

Request XML format.

```
<request>
        APPLICANT
        <informationHandler>HANDLER</informationHandler>
       ADDINFO
</request>
```

See chapter APPLICANT for detailed description about the applicant.

ADDINFO is described in chapter 11.4 addInfo and must be adjusted according to the specification of the actual information handler.
HANDLER is the name of the information handler.

### 11.6.4 The variable getInformation handler

There is one specific informationhandler that is somewhat variable, in contrast to any other informationhandler that has a specific and absolute name.

The information returned from any informationhandler comes on the XML form:
"…/info/exti/*provider*/*node*/…."
The *provider* is an external (to DecisionMaker) information provider. The *node* is a main node or a *product node*.

By using the variable segment getInformation handler one may specify which *product nodes* one wants to get back.
The general form of this informationhandler is like this:

```
provider:products(s1,s2,s3,s4,s5,……):xml:server
```

Example (address and tax information on persons delivered by Bisnode):
`alpha:products(ho,sk):xml:server`

**`ho`** is the name of the product node for person and address information and **`sk`** is the name of the product node for tax information. The use of the product nodes requires that the corresponding subscriptions for these data are in place.

## 11.7 Error messages

If a scoring fails, the request is incorrect or something else wrong happened, an error message is returned. If DecisionMaker is set in debug mode, a call stack is also returned.

Errors on the client side, like socket communication failures, is impossible for DecisionMaker to know about. Using our WebServices solution will give you a better control of what is going on in that area.

This chapter deals with error messages generated by DecisionMaker only.

Unpredicted errors will throw an exception and generate a generic error message.
Other messages will, hopefully, give a good explanation of the problem.

By default the system returns error messages in the Norwegian language. By adding the attribute “locale” to the dmRequest tag you may specify English using the two letter codes as specified in http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt. For English this means that the letters “en” must be used:
`<dmRequest version=”2.0” locale=”en”><user……etc……`
Unfortunately this is currently not possible to specify using the WebServices.

The error message text is described in an XML document named *exception-messages_XX.xml,* where XX is the locale code. The document may be delivered on request.

Table 14 Error message

```
<error>
       <Time format="yyyyMMdd HH:mm:ss">20040206 14:27:52</Time>
        <errorCode>70</errorCode>
         <errorMessage>An error has occurred </errorMessage>
         <errorMeasure>
                Contact systemadministrator </errorMeasure>
        <detailedErrorMessage>
                <![CDATA[ stacktrace ]]>
         </detailedErrorMessage>
         <productName>name of credit model </productName>
</error>
```

Here is:

- `errorCode`
  a unique error code for this error. The code is specified in the *exception-messagesXX.xml* file.
- `errorMessage`
  is the error message as specified in the exception message file
- `errorMeasure`
  is a descriptive text guiding the user how to deal with the problem
- `detailedErrorMessage`
  shows the stack trace when DecisionMaker is in debug mode
- `productName`
  is the name of the credit model if the error response was the result from a score request
- `Time`
  is a timestamp set when the error occurred

## 11.8 EXAMPLES

### 11.8.1 How to get a previous scoring result from a customized credit model (example)

***To be used for customized credit models only!***

Two information handlers;

1. Simple (`dm:score_result_simple:xml:server`) (decision only)
2. Advanced (`dm:score_result_extended:xml:server`) (all details)

#### *11.8.1.1 Simple Request*

The `dm:score_result_simple:xml:server` is used to get the scoring:

Table 15

**Simple request:**
Get scorings for a period
(Aug. 10. 2004 to Aug. 30. 2004)

```
<dmRequest>
       <user>user</user>
       <password>password</password>
       <request>
              <informationHandler>
                     dm:score_result_simple:xml:server
              </informationHandler>
              <addInfo>
                     <hashtableXML>
                     <element key="created" type="java.lang.String">
                      -GE 2004-08-10 -AND -LE 2004-08-30
                     </element>
                     </hashtableXML>
              </addInfo>
       </request>
</dmRequest>
```

Table 16

**Extended request:**
Get scoring for specified person

```
<dmRequest>
       <user>username</user>
       <password>password</password>
       <request>
         <informationHandler>
              dm:score_result_extended:xml:server
         </informationHandler>
         <addInfo>
          <hashtableXML>
              <element key="applicantFirstName"
              type="java.lang.String">Eynar</element>
              <element key="applicantLastName"
              type="java.lang.String">Jensen</element>
              <element key="created" type="java.lang.String">
                      -GE 2004-08-10 -AND -LE 2004-08-30
              </element>
           </hashtableXML>
          </addInfo>
       </request>
</dmRequest>
```

### *11.8.1.2 Response*

Table 17

Example: Response on request in Table 15 using `dm:score_result_simple:xml:server`

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<dmResponse>
  <response id="">
     <info>
       <internal>
          <dm>
            <scoreresult>
               <id>14</id>
               <created>22-10-01 00:00:00.0</created>
               <handler>username</handler>
               <finalresult>null</finalresult>
               <result>f</result>
               <points>-1</points>
               <zone></zone>
               <overruled>null</overruled>
               <overrulecomment>null</overrulecomment>
               <overruledby>null</overruledby>
               <overruleddate>null</overruleddate>
            </scoreresult>
            <scoreresult>
               <id>15</id>
               <created>22-10-01 00:00:00.0</created>
               <handler>system</handler>
               <finalresult>null</finalresult>
               <result>y</result>
               <points>65</points>
               <zone>4</zone>
               <overruled>null</overruled>
               <overrulecomment>null</overrulecomment>
               <overruledby>null</overruledby>
               <overruleddate>null</overruleddate>
            </scoreresult>
            <scoreresult>
               <id>45</id>
               <created>02-11-01 00:00:00.0</created>
               <handler>guest</handler>
               <finalresult>null</finalresult>
               <result>y</result>
               <points>86</points>
               <zone>5</zone>
               <overruled>null</overruled>
               <overrulecomment>null</overrulecomment>
               <overruledby>null</overruledby>
               <overruleddate>null</overruleddate>
            </scoreresult>
          </dm>
       </internal>
     </info>
  </response>
</dmResponse>
```

Table 18:

Example: response on request in Table 16, using `dm:score_result_extended :xml:server:`

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<dmResponse><response id="">
<info><internal><dm>
<decision>
  <id>45</id>
  <created>2-11-01 00:00:00.0</created>
  <handler>username</handler>
  <finalresult>null</finalresult>
  <result>o</result>
  <points>86</points>
 <zone></zone>
  <overruled>null</overruled>
  <overrulecomment>null</overrulecomment>
  <overruledby>null</overruledby>
  <overruleddate>null</overruleddate>
    <score>
    <SCORING name="produktnavn" result="o" scoredate="2001-11-02" HLimit="70- 100"
    BasePoints="89" MLimit="31-69" LLimit="0-30">
    <SCORE_ELEMENT name="hovedstol" result="-3">
        <ELEMENT name="hovedstol" value="0.0" type="java.lang.Float" />
    </SCORE_ELEMENT>
    <SCORE_ELEMENT name="priorCase" result="0">
        <ELEMENT name="priorCase" value="Nei" type="java.lang.String" />
    </SCORE_ELEMENT>
    <SCORE_ELEMENT name="noOfPaymentRemarks" result="0">
        <ELEMENT name="noOfPaymentRemarks" value="0" type="java.lang.String" />
    </SCORE_ELEMENT>
    <SCORE_ELEMENT name="netIncome" result="0">
        <ELEMENT name="netIncome" value="476900" type="java.lang.Integer" />
    </SCORE_ELEMENT>
    <SCORE_ELEMENT name="netWealth" result="0">
        <ELEMENT name="netWealth" value="0" type="java.lang.Integer" />
    </SCORE_ELEMENT>
    <SCORE_ELEMENT name="taxClass" result="0">
        <ELEMENT name="taxClass" value="1" type="java.lang.String" />
    </SCORE_ELEMENT>
    </SCORING>
    </score>
    <decision>
    <SCORING name="produktnavn" result="o" scoredate="2001-11-02" HLimit="70-100"
    BasePoints="89" MLimit="31-69" LLimit="0-30">
    <TOTALSCORE_ELEMENT name="TotalScoring" result="o" points="86">
        <ELEMENT name="Hendelseskode" value="HP801" type="java.lang.String" />
        <ELEMENT name="Antall betalingsanmerkninger LD" value="0"
        type="java.lang.Integer" />
        <ELEMENT name="Saksnummer" value="0" type="java.lang.Integer" />
        <ELEMENT name="Fødselsnummer" value="01234567890" type="java.lang.String" />
        <ELEMENT name="Navn" value="FRANSEN PETTER" type="java.lang.String" />
        <ELEMENT name="Adresse" value="NYGÅRDSGATEN 16" type="java.lang.String" />
        <ELEMENT name="Postnr" value="9999" type="java.lang.String" />
        <ELEMENT name="Poststed" value="OSLO" type="java.lang.String" />
        <ELEMENT name="TotalScoring" value="o" type="java.lang.String" />
        <ELEMENT name="TotalPolicy" value="o" type="java.lang.String" />
    </TOTALSCORE_ELEMENT>
    </SCORING>
    </decision>
</decision>
</dm></internal></info></response>
</dmResponse>
```

The number value in the <id> tag is the unique score id in which can be used when updating the score result.

#### *11.8.1.3 Request*

Table 19

Example: Updates a scoring with score id 45. The decision is set to 'n' (i.e. "No"). A comment (reason why) is set on the update

```
<dmRequest>
        <user>username</user>
        <password>password</password>
        <request>
                <informationHandler>
                        dm:update_score_result:xml:server
                </informationHandler>
                <addInfo>
                        <hashtableXML>
                                <element key="id" type="java.lang.Long">45
                                </element>
                                <element key="finalResult"
                                type="java.lang.String">n
                                </element>
                                <element key="comment" type="java.lang.String">
                                ptp test comment
                                </element>
                        </hashtableXML>
                </addInfo>
        </request>
</dmRequest>
```

### 11.8.2 Example: requesting address information

#### *11.8.2.1 Request*

Table 20

Example: Two address information request in one single request.

```
<dmRequest version="2.0">
  <user>username</user>
  <password>username</password>
    <request>
        <applicant>
          <APPLICANT type="P">
            <ssn>01234567890</ssn>
           </APPLICANT>
        </applicant>
        <informationHandler>
          alpha:address_filter:xml:server
        </informationHandler>
    </request>
    <request>
       <applicant>
          <APPLICANT type="P">
            <ssn>12345678901</ssn>
          </APPLICANT>
       </applicant>
       <informationHandler>
           alpha:address_filter:xml:server
       </informationHandler>
    </request>
</dmRequest>
```

#### *11.8.2.2 Response*

Table 21

Example: address information response

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<dmResponse version="2.0">
<response>
<info><external>
<alpha>
  <address>
    <name>name 1</name>
    <birthdate>060857</birthdate>
    <street>MIDTÅSEN 3</street>
    <zipcode>1166</zipcode>
    <city>OSLO</city>
    <age>44</age>
    <sex>M</sex>
  </address>
</alpha>
</external></info>
</response>
<response>
<info><external>
<alpha>
  <address>
    <name>Hansen Ole</name>
    <birthdate>111155</birthdate>
    <street />
    <zipcode>8064</zipcode>
    <city>RØST</city>
    <age>45</age>
    <sex>M</sex>
  </address>
</alpha>
</external></info>
</response>
</dmResponse>
```

## 11.9 HTTP GET Request strings to try

Here are some HTTP GET requests to try. Copy and paste into the address line in a browser.
Remember to set correct IP-address if different than www.soliditetd.no, as well as username, password, applicant's ssn, name of the credit model (i.e. "productName") etc. Remember if the request concerns an organization or company, use an 'F' instead of 'P' in the APPLICANT tags type attribute.
*NOTE* that the GET method is supported for simple tests only. The POST method is to be used in real code.

**Check status:**

```
https://www.soliditetd.no/information/get?request=
<dmRequest><user>USERNAME</user><password>PASSWORD</password><request>
<informationHandler>dm:status:text:server</informationHandler></request></dmRequest>
```

**Getting paymentremarks:**

```
https://www.soliditetd.no/information/get?request=
<dmRequest version="2.0"><user>USERNAME</user><password>PASSWORD</password><request>
<applicant><APPLICANT type="P"><ssn>02030146134</ssn> </APPLICANT></applicant>
<informationHandler>alpha:payment_remarks:xml:server</informationHandler></request>
</dmRequest>
```

dun & bradstreet

**Getting tax information:**

https://www.soliditetd.no/information/get?request=
<dmRequest version="2.0"><user>USERNAME</user><password>PASSWORD</password>
<request><applicant><APPLICANT type="P"><ssn>02030146134</ssn></APPLICANT></applicant>
<informationHandler>alpha:tax_filter:xml:server</informationHandler></request>
</dmRequest>

**Getting "fail to pay" information:**

https://www.soliditetd.no/information/get?request=
<dmRequest version="2.0"><user>USERNAME</user><password>PASSWORD</password><request>
<applicant><APPLICANT type="P"><ssn>02030146134</ssn></APPLICANT></applicant>
<informationHandler>alpha:mislighold_filter:xml:server</informationHandler></request>
</dmRequest>

**Getting address information:**

https://www.soliditetd.no/information/get?request=
<dmRequest version="2.0"><user>USERNAME</user><password>PASSWORD</password><request>
<applicant><APPLICANT type="P"><ssn>02030146134</ssn></APPLICANT></applicant>
<informationHandler>alpha:address_filter:xml:server</informationHandler></request>
</dmRequest>

**Run a scoring 1:**

https://www.soliditetd.no/scoring/score?request=
<dmRequest version="2.0"><user>USERNAME</user><password>PASSWORD</password><request>
<applicant><APPLICANT type="P"><ssn>02030146134</ssn></APPLICANT></applicant>
<productName>creditModeln**ame**</productName>
<scoreGroupName>scoregroupname</scoreGroupName></request></dmRequest>

**Run a scoring 2:**

https://www.soliditetd.no/scoring/score?request=
<dmRequest version="2.0"><user>USERNAME</user><password>PASSWORD</password><request>
<applicant><APPLICANT type="P"><ssn>02030146134</ssn></APPLICANT></applicant>
<productName>**creditModelName**</productName>
<scoreGroupName>scoregroupname</scoreGroupName><appInfo><![CDATA[<APP_INFO>
<MEMBER_VALUE isSet="true" name="**feltnavn1**"><VALUE>**value**</VALUE></MEMBER_VALUE>
<MEMBER_VALUE isSet="true" name="**feltnavn2**"><VALUE>value</VALUE></MEMBER_VALUE>
</APP_INFO>]]></appInfo></request></dmRequest>

# 12 Java examples

Sample code may be downloaded from www.soliditetd.no.

## 12.1 Accessing the service using Axis 1

The examples in this chapter is based on an Axis 1 generated Java client. See chapter 10.10.2 Apache Axis 1 for Java.

Generate the client:

```
java org.apache.axis.wsdl.WSDL2Java -v -a --timeout 30 -T 1.1 -p
no.bisnode.creditwsclients.ws3.jaxrpc.generated -o src/java/
https://www.soliditetd.no/webservices/services/DecisionMakerWebServices3?wsdl
```

In order to access the generated Java code, we can create some helper classes like this:
Classname `DecisionMakerWebServicesImplSuper.java`:

```
import no.bisnode.creditwsclients.ws3.jaxrpc.generated.DmWSVersion;

import javax.xml.rpc.ServiceException;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;

public abstract class DecisionMakerWebServicesImplSuper {
   private static final String DEFAULT_HOST = "www.soliditetd.no";
   private URL url;
   private org.apache.axis.client.Stub stub;

   public DecisionMakerWebServicesImplSuper(String hostName) throws MalformedURLException, ServiceException {
      if ((hostName == null) || (hostName.trim().length() == 0)) {
         hostName = DEFAULT_HOST;
      }

      String urlName;
      if (!hostName.startsWith("http")) {
         urlName = "https://" + hostName;
      }
      else {
         urlName = hostName;
      }
      url = new URL(urlName + getEndPointUrl());

      stub = getStubInstance();
      stub.setTimeout(60000 * 5);
   }

   protected abstract org.apache.axis.client.Stub getStubInstance() throws ServiceException;

   protected abstract String getEndPointUrl();

   /**
    * Get version of DecisionMaker and version of the DecisionMaker WebServices
    */
   abstract public DmWSVersion getVersion() throws RemoteException;

   public org.apache.axis.client.Stub getStub() {
      return stub;
   }

   public URL getUrl() {
      return url;
   }
```

```
   public static String getDefaultHost() {
      return DEFAULT_HOST;
   }
}
```

Classname `DecisionMakerWebServicesImpl.java`:

```
import no.bisnode.creditwsclients.ws3.jaxrpc.generated.*;

import javax.xml.rpc.ServiceException;
import java.net.MalformedURLException;
import java.rmi.RemoteException;

public class DecisionMakerWebServicesImpl extends DecisionMakerWebServicesImplSuper implements
DecisionMakerWebServices_PortType { //named just DecisionMakerWebServices up to 1.3. _PortType was added in 1.3
   private static final String END_POINT_URL = "/webservices/services/DecisionMakerWebServices?wsdl";

   public DecisionMakerWebServicesImpl() throws MalformedURLException, ServiceException {
      this(null);
   }

   public DecisionMakerWebServicesImpl(String hostName) throws MalformedURLException, ServiceException {
      super(hostName);
   }

   protected org.apache.axis.client.Stub getStubInstance() throws ServiceException {
      DecisionMakerWebServicesService3Locator serviceLocator = new DecisionMakerWebServicesServiceLocator();
      return (DecisionMakerWebServicesSoapBindingStub)serviceLocator.getDecisionMakerWebServices(getUrl());
   }

   protected String getEndPointUrl() {
      return END_POINT_URL;
   }


   /**
    * Get version of DecisionMaker and version of the DecisionMaker WebServices
    */
   public DmWSVersion getVersion() throws RemoteException {
      return ((DecisionMakerWebServicesSoapBindingStub)getStub()).getVersion();
   }

   /**
    * Get status of server, its connection to its databases and its connection status to external
    * dataproviders
    */
   public DmWSStatus[] getStatus(String userName, String password) throws RemoteException {
      return ((DecisionMakerWebServicesSoapBindingStub)getStub()).getStatus(userName, password);
   }

   /**
    * Perform specified scoring
    */
   public DmWSResponse score(DmWSRequest request) throws RemoteException {
      return ((DecisionMakerWebServicesSoapBindingStub)getStub()).score(request);
   }

   /**
    * Get information on a previously performed scoring
    * @param userName
    * @param passWord
    * @param requestId
    * @param scoreId   The scoreid to request for
    * @param extended  requesting for details of the scorecard if such. Needs special permissions
    * @return
    * @throws RemoteException
    */
   public DmWSResponse getScoreInformationById(String userName, String password, String requestId, String scoreId,
                                               boolean extended) throws RemoteException {
```

```
      return ((DecisionMakerWebServicesSoapBindingStub)getStub()).getScoreInformationById(userName, passWord,
requestId, scoreId, extended);
   }

   /**
    * Update the score decision for a specified scoring. Needs special permission. Seldom used/needed.
    */
   public DmWSResponse updateScoreDecisionById(String userName, String passWord,
                                               String requestId, String scoreId,
                                               String decision, String decisionCause)
         throws RemoteException {
      return ((DecisionMakerWebServicesSoapBindingStub)getStub()).updateScoreDecisionById(userName, passWord,
requestId, scoreId, decision, decisionCause);
   }

   /**
    * Get information, calls an information handler
    */
   public DmWSResponse getInformation(DmWSRequest request) throws RemoteException {
      return ((DecisionMakerWebServicesSoapBindingStub)getStub()).getInformation(request);
   }

   /**
    * Get information, calls an information handler
    */
   public DmWSResponse getInformation(String userName, String passWord, String userRef, String requestId,
DmWSApplicant applicant, String handlerName) throws RemoteException {
      return ((DecisionMakerWebServicesSoapBindingStub)getStub()).getInformation(userName, passWord, userRef,
requestId, applicant, handlerName);
   }

   /**
    * Performs a scoring on specified applicant using the specified credit model. Takes only one single
    * applicant as request. Assumes a single response is returned (should always be).
    * @param userName
    * @param passWord
    * @param applicant
    * @param productName
    * @return The DmWSScoreResponse or DmWSError object
    * @throws RemoteException
    */
   public DmWSSingleResponse score(String userName, String passWord, String userRef,
                                   String requestId,
                                   DmWSApplicant applicant, String productName)
         throws RemoteException {
      // we need an applicant object:
      if (applicant.getNationality() == null) {
         applicant.setNationality("no"); // it's a Norwegian by default
      }
      if (applicant.getType() == null) {
         applicant.setType("P"); // it's a person (not a company) by default
      }
      // run the request:
      return ((DecisionMakerWebServicesSoapBindingStub)getStub()).score(userName, passWord, userRef, requestId,
applicant, productName);
   }

   /**
    * Finds the text message for the attribute named "action". If none found, null is returned
    * @param element
    * @return the text string
    */
   public String getActionMessage(DmWSElement element) {
      return getAttributeValue(element, "action");
   }

   /**
    * Looks through all attributes for a specific named attribute key
    * @param element
    * @param key
```

```
    * @return the text/value associated with the attribute. If none found, null is returned
    */
   public String getAttributeValue(DmWSElement element, String key) {
      if ((element == null) || (key == null)) return null;
      DmWSAttribute[] attrs = element.getAttributes();
      if ((attrs != null) && (attrs.length > 0)) {
         for (int i = 0; i < attrs.length; i++) {
              DmWSAttribute attr = attrs[i];
              if (attr != null) {
                 if (attr.getName().equals(key)) {
                    return attr.getText();   //got it, return text
                 }
              }
         }
      }
      return null;
   }
}
```

By creating this class, we have hidden the fact that we use the ServiceLocator and the stub to access the WebServices methods. It will appear as if it has been an ordinary local object. In order to perform a scoring on a person with social security number “12345678900” using the credit model named “Standard Person” we can do the following:

```
// build the request
DmWSRequest mainRequest = new DmWSRequest();
DmWSScoreRequest[] requests = new DmWSScoreRequest[]{new DmWSScoreRequest()}; // list of one
mainRequest.setUser("username");                    // user identification/login name
mainRequest.setPassWord("password");                // password
mainRequest.setRequests(requests);                  // set list of requests
requests[0].setProductName("Standard Person");      // credit model
requests[0].setRequestId("test score no. 1");       // will be returned with response (optional)
requests[0].setComment("user identification");      // optional: example: identifies the actual user

// we need an applicant object:
DmWSApplicant[] applicants = new DmWSApplicant[]{new DmWSApplicant()}; // list of one
applicants[0].setSsn("12345678900");         // who to search for
applicants[0].setNationality("no");          // it’s a Norwegian
applicants[0].setType("P");                  // it’s a person (not a company)
requests[0].setApplicants(applicants);       // give the applicants list to the request
// run the request:

try {
        DmWSResponse response = new DecisionMakerWebServicesImpl("www.soliditetd.no").score(mainRequest);
        /* Find the result in the response class */
}
catch (RemoteException e) {
        e.printStackTrace();
}
catch (MalformedURLException e) {
        e.printStackTrace();
}
catch (ServiceException e) {
        e.printStackTrace();
}
```

However, you might find this a bit heavy if the only thing you want to do is to perform a score on a single person or company, nothing more, nothing less. To put all the hassle away, you might consider the second form of the score method, which will make the above example a lot simpler:

```
// we need an applicant object:
DmWSApplicant applicant = new DmWSApplicant();
applicant.setSsn("1234567890"); // who to search for
applicant.setType("P");
// run the request:
```

```
DecisionMakerWebServicesImpl dmws = null;
DmWSSingleResponse response = null;
try {
   dmws = new DecisionMakerWebServicesImpl("www.soliditetd.no");
   response = dmws.score("username", "password", null, null, applicant, "Test Person");
}
catch (MalformedURLException e) {
   e.printStackTrace();
}
catch (ServiceException e) {
   e.printStackTrace();
}
catch (RemoteException e) {
   e.printStackTrace();
}
```

Now we must add code to handle the response:

```
// handling the result:
if (response == null) {
   // should never happen, but should be handled anyway
} else if (response instanceof DmWSError) {
   // So an error occurred:
   DmWSError error = (DmWSError)response;
   System.out.println("Error code: " + error.getErrorCode());
   System.out.println("Error message: " + error.getErrorMessage());
   System.out.println("What to do: " + error.getErrorMeasure());
} else if (response instanceof DmWSScoreResponse) {
   // we got a score response:
   DmWSScoreResponse sr = (DmWSScoreResponse)response;
   System.out.println("Decision zone:" + sr.getZone());
   String decision = sr.getResult();
   if (decision.equals("y"))
      decision = "Approved";
   else if (decision.equals("n"))
      decision = "Not approved";
   else if (decision.equals("c"))
      decision = "Check";

   System.out.println("Decision: " + decision);

   // A policyrule might have occurred, gotta check:
   DmWSScoring[] scorings = sr.getScorings();
   // usually only one scoring object
   if ((scorings != null) && (scorings.length > 0)) {
      DmWSPolicyElement[] pes = scorings[0].getPolicyElements();
      if ((pes != null) && (pes.length > 0)) {
         for (int i = 0; i < pes.length; i++) {
            DmWSPolicyElement pe = pes[i];
            if (pe != null) {
               System.out.println("Policy name: " + pe.getName());
               System.out.println("Policy message: " + dmws.getActionMessage(pe));
            }
         }
      }
      // It may also be interesting to get the TotalScore elements from the scoring:
      DmWSTotalScoreElement[] tes = scorings[0].getTotalScoreElements();
      if ((tes != null) && (tes.length > 0)) {
         // A totalscore element is usually only one ….
         if (tes[0] != null) {
            // … and with several sub-elements
            DmWSElement[] elements = tes[0].getElements();
            if ((elements != null) && (elements.length > 0)) {
               for (int i = 0; i < elements.length; i++) {
                  // Display name and value:
                  System.out.println(elements[i].getName() + ": " + elements[i].getVal());
               }
            }
         }
```

```
      }
   }
} else {
   // Unknown result
   // should never happen, but should be handled anyway
   System.out.println("Unknown result: " + response.toString());
}
```

## 12.2 Accessing the service using JAX-WS (Glassfish)

Generating client code (make sure the subdirectory “src/java” exists):

```
java com.sun.tools.ws.WsImport -verbose -p
no.bisnode.creditwsclients.ws3.jaxws.generated -keep -Xnocompile -s src/java
https://www.soliditetd.no/webservices/services/DecisionMakerWebServices3?wsdl
```

The sample code will look very similar:

```
package no.bisnode.creditwsclients.ws3.jaxws;

import no.lindorff.info.edr.ws.generated.jaxws.SendFault;
import no.lindorff.info.util.WSClientHelper;
import no.lindorff.util.xml.dom.XmlDOMDocument;
import no.lindorff.util.xml.dom.XmlDOMElement;
import no.bisnode.creditwsclients.SOAPMessageStringHandler;
import no.bisnode.creditwsclients.ws3.jaxws.generated.*;
import org.xml.sax.SAXException;

import javax.xml.namespace.QName;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.ws.BindingProvider;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.List;

public class MyJAXWSv3Test {
   public static final String REVISION = "$Id$";
   private static transient final org.apache.commons.logging.Log log =
           org.apache.commons.logging.LogFactory.getLog(MyJAXWSv3Test.class);
   private static SOAPMessageStringHandler soapHandler;

   private static URL getUrl(String serverUrl) {
      URL url = null;
      try {
         URL baseUrl;
         baseUrl = DecisionMakerWebServices3Service.class.getResource(".");
         if (!serverUrl.toLowerCase().endsWith("wsdl"))
            serverUrl = serverUrl + "/DecisionMakerWebServices3?wsdl";
         url = new URL(baseUrl, serverUrl);
      }
      catch (MalformedURLException e) {
         log.warn("Failed to create URL for the wsdl Location: " + serverUrl, e);
      }
      return url;
   }

   public static DecisionMakerWebServices3 getPort(String serverUrl)
           throws SendFault, MalformedURLException {

      WSClientHelper.getInstance().setupProxyIfNeeded(null, true);
      if (log.isDebugEnabled())
         WSClientHelper.enableJAXWSClientLog(true);

      QName portName = new QName("urn:DecisionMakerWebServices3", "DecisionMakerWebServices3");
      DecisionMakerWebServices3Service service = new
                  DecisionMakerWebServices3Service(getUrl(serverUrl), portName);
```

```
      DecisionMakerWebServices3 port = service.getDecisionMakerWebServices3();

      // in order to pick up the real request and response XML's, add a customized SOAO handler
      soapHandler = SOAPMessageStringHandler.addSoapHandler((BindingProvider)port);
      return port;
   }

   public static void main(String[] args) {
      try {
         // Constuct webservice object using default host name (www.soliditetd.no)
         // or take the name from input argument:
         String hostName = null;
         if ((args != null) && (args.length > 0) && (args[0].trim().length() > 0)) {
            hostName = args[0].trim();
         }

         DecisionMakerWebServices3 soliditetdWS = getPort(hostName);
         no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSApplicant applicant =
                 new no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSApplicant();
         // COMPANY SEARCH:
         //applicant.setType("F"); // The applicant is a Company, not a person ("P")
         //applicant.setName("Bisnode Norge");       // search by name.....
         //applicant.setSsn("975374939");          // ...or by org. no.
         //applicant.setNationality("no");       // norwegian is default

         // PERSON SEARCH:
         //applicant.setType("P"); // The applicant is a Person, not a company ("F")
         //applicant.setSsn("10106011430");
         //applicant.setFirstName("Eynar");
         //applicant.setLastName("Jensen");

         // set username and password
         String userName = "system";
         String passWord = "system";

         // Scoring on specific company policies (if any)
         String productName = "123 Person";
         no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSSingleResponse srspScore =
                 soliditetdWS.score(userName, passWord, null, null, applicant, productName);
         handleOneResponse(0, srspScore);


         // Uncomment the following if to get raw information
         // (including Decision Score) from provider:
/*
         no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSResponse rsp =
                 soliditetdWS.getInformation(userName, passWord, null, null, applicant,
                                             "alpha:hocrba:xml:server"); // this is a sample
         List<DmWSSingleResponse> srsps = rsp.getResponses();
         int applicantsWritten = 0;
         for (DmWSSingleResponse srspInfo : srsps) {
            if (srspInfo != null) {
               applicantsWritten = handleOneResponse(applicantsWritten, srspInfo);
            }
         }
*/
         log.debug("Request:" + soapHandler.getOutBoundString());
         log.debug("Response:" + soapHandler.getInBoundString());
      }
      catch (MalformedURLException e) {
         e.printStackTrace();
      }
      catch (SendFault sendFault) {
         sendFault.printStackTrace();
      }
   }

   private static int handleOneResponse(int applicantsWritten, DmWSSingleResponse srspInfo) {
      if (applicantsWritten == 0) {
         // output info about the applicant (but only once)
```

```
         applicantsWritten = outputApplicantInfo(srspInfo);
      }
      handleOneresponse(srspInfo);
      return applicantsWritten;
   }

   /**
    * Outputs info about the error, scoring or information result
    *
    * @param rsp
    */
   private static void handleOneresponse(
         no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSSingleResponse rsp) {
      if (rsp instanceof no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSError) {
         handleErrorResponse((no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSError)rsp);
         // we got an error
      }
      else if (rsp instanceof
no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSInformationResponse) {

   handleInformationResponse((no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSInformationResponse
)rsp);
      }
      else if (rsp instanceof no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSScoreResponse) {
         handleScoreResponse((no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSScoreResponse)rsp);
      }
   }

   private static void handleScoreResponse(
         no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSScoreResponse srsp) {
      System.out.println("\n\nFrom ScoreResponse object:");
      // If this response only contains old obsolete norwegian scoremodels like 1 or
      // 2, they are of no interest,
      // so we have to check for some we want first. These are models 3, 4,
      // 5 or 6. 4, 5 and 6 never occurs at
      // the same time. model 3 (Payment Index) is delivered simultaneously if subscribed for
      no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSScoring scoring = null;
      List<DmWSScoring> scorings = srsp.getScorings();
      if (scorings != null) {
         for (no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSScoring dmWSScoring : scorings) {
            if (dmWSScoring != null) {
               String mId = dmWSScoring.getZoneModelId();
               int modelNoL = Integer.parseInt(mId);
               if ((modelNoL == 3) || (modelNoL == 4) || (modelNoL == 5) || (modelNoL == 6)) {
                  scoring = dmWSScoring;
                  break;   //got it, so break;
               }
            }
         }
      }
      if (scoring == null) {
         // none of interest, ignore it
         return;
      }

      // get zone value:
      int izone = stringToInteger(scoring.getZone());

      // get the decision:
      String result = srsp.getResult();
      String decision = "not set";
      if (result != null) {
         decision = getDecision(result);
      }
      else {
         // if the result is from the raw data from provider (Bisnode)
         // the result is not set.
         // Bisnode delivers the result only as a zone value
         // This means we will have to convert the zone into a decision.
         // We have an array of no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSZone objects.
```

```
   // There is one object per zone, telling what decision that
   // zone results in. In other words, we must loop thru the array, find the correct
   // zone object and take
   // the decision value from there.
   List<DmWSZone> zones = scoring.getZones();
   if (zones != null) {
      for (no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSZone zone : zones) {
         if (stringToInteger(zone.getZone()) == izone) {
            decision = getDecision(zone.getResult());
            break;  // we got it, no need to look for more
         }
      }
   }
}
System.out.println("The decision is \"" + decision + "\"");

// Policyrules may affect the total decision.
// If a client/applicant is denied, one may want to know why.
// This is due to either a very low rating zone, or to one or more policy rule(s)
// that occurred. The applicant may owe a lot of money (payment remarks)
// and the store (like Tele 2)
// may not want such customers.

// The score/rating zone (0 and 1 is deny, 2 is check, 3,4 and 5 is approved)
//System.out.println("Rating zone is " + srsp.getZone());

int modelNo = stringToInteger(scoring.getZoneModelId());
String zoneText = getZoneText(modelNo, izone);
System.out.println("Kredittsone: " + izone + ": " + zoneText);

// The following policyrules occured:
//int policiesFound = 0;
List<DmWSPolicyElement> pes = scoring.getPolicyElements();

if ((pes != null) && (pes.size() > 0)) {
   // Ok, some policyelements occurred, loop thru:
   System.out.println("\n\nThe following policy rules occurred:");
   for (no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSPolicyElement pe : pes) {
      // print policyname and message
      String pename = pe.getName();
      System.out.print(pename + ": " + getActionMessage(pe));
      // in some cases the customer (like Tele2) wants even more details
      // A policyelement may have an array of
      // no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSElement
      // objects containing even more info:
      List<DmWSElement> elements = pe.getElements();
      if ((elements != null) && (elements.size() > 0)) {
         for (no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSElement element : elements) {
            //policiesFound++;
            // The element has a name and a value,
            // e.g: "Income: 345000", "Paymentremarks: 4" etc
            // if only one sub element, don't bother
            // displaying the name once again if the same as parent:
            String name = element.getName();
            if (pename.equalsIgnoreCase(name) && (elements.size() == 1))
               System.out.println(", " + element.getVal());
            else
               System.out.println(name + ": " + element.getVal());
         }
      }

   }
}

// Want to get hold of the totalscore elements?
// Only returned for actual score, not for raw data:
// Actually same as for policyelements:
List<DmWSTotalScoreElement> tes = scoring.getTotalScoreElements();
if ((tes != null) && (tes.size() > 0)) {
   System.out.println("\n\nThe following total score elements are present:");
```

```
         for (no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSTotalScoreElement te : tes) {
            // print element name and message
            System.out.println("\n" + te.getName() + ": ");
            List<DmWSAttribute> attributes = te.getAttributes();
            if (attributes != null) {
               for (no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSAttribute attribute :
attributes) {
                  System.out.println("\nAttribute name: " + attribute.getName()
                                     + " value is \"" + attribute.getText() + "\"");
               }
            }
            // in some cases the customer (like Tele2) wants even more details
            // A total score element may have an array of
            // no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSElement
            // objects containing even more info:
            List<DmWSElement> elements = te.getElements();
            if (elements != null) {
               if ((elements != null) && (elements.size() > 0)) {
                  for (no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSElement element : elements)
{
                     // The element has a name and a value,
                     System.out.println(element.getName() + ": " + element.getVal());
                  }
               }
            }
         }
      }

      /**
       * Finds the text message for the attribute named "action". If none found, null is returned
       *
       * @param element
       * @return the text string
       */
      static public String getActionMessage(
         no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSElement element) {
         return getAttributeValue(element, "action");
      }

      /**
       * Looks through all attributes for a specific named attribute key
       *
       * @param element
       * @param key
       * @return the text/value associated with the attribute. If none found, null is returned
       */
      static public String getAttributeValue(
         no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSElement element, String key) {
         if ((element == null) || (key == null)) return null;
         List<DmWSAttribute> attrs = element.getAttributes();
         if ((attrs != null) && (attrs.size() > 0)) {
            for (no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSAttribute attr : attrs) {
               if (attr != null) {
                  if (attr.getName().equals(key)) {
                     return attr.getText(); // got it, return text
                  }
               }
            }
         }
         return null;
      }

      private static String getZoneText(int modelNo, int izone) {
         String zoneText;
         String modelText;
         if (modelNo == 3) { // Decision Payment Index
            // Payment Index er en scoringløsning som rangerer foretakene
            // etter sannsynlighet for mislighold.
            modelText = "Decision Payment Index";
```

```
            switch (izone) {
               case 1:
                  zoneText = "Rød sone";
                  break;
               case 2:
                  zoneText = "Gul sone";
                  break;
               case 3:
                  zoneText = "Grønn sone";
                  break;
               default:
                  zoneText = "Payment Index ikke beregnet";
            }
         }
         else if (modelNo == 5) { // Decision Score Company
            modelText = "Decision Score Company";
            switch (izone) {
               case 1:
                  zoneText = "Kreditt kun mot sikkerhet";
                  break;
               case 2:
                  zoneText = "Kreditt mot sikkerhet anbefales";
                  break;
               case 3:
                  zoneText = "Kredittverdig";
                  break;
               case 4:
                  zoneText = "Høy kredittverdighet";
                  break;
               case 5:
                  zoneText = "Høyeste kredittverdighet";
                  break;
               default:
                  zoneText = "Foretaket er ikke ratet";
                  break;
            }
         }
         else { // presumeably model 4 (actually ENK companies) or 6, i.e. Decision Score Person
            modelText = "Decision Score " + (modelNo == 6 ? "Person" : "Company");
            switch (izone) {
               case 1:
                  zoneText = "Høy risiko";
                  break;
               case 2:
                  zoneText = "Moderat risiko";
                  break;
               case 3:
                  zoneText = "Lav risiko";
                  break;
               case 4:
                  zoneText = "Lav risiko";
                  break;
               case 5:
                  zoneText = "Lav risiko";
                  break;
               default:
                  zoneText = "Personen er ikke ratet";
                  break;
            }
         }

         System.out.println("\n\nCredit model: " + modelText);
         return zoneText;
      }

      private static void handleInformationResponse(
         no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSInformationResponse irsp) {
         System.out.println("\n\nFrom InformationResponse object:");
         String informationResult = irsp.getInformationResult();
         System.out.println("Raw Xml:\n" + informationResult);
```

```
   // It could be a scoring in there
   // We could extract some stuff from that using XPath.
   // In this example we've used Dom4J and Xerces

   try {
      XmlDOMDocument doc = new XmlDOMDocument(informationResult);
      String szone = doc.getXPathValue("/info/exti/alpha/cr/Rating3/CreditZone/@value");
      String smodelNo = doc.getXPathValue("/info/exti/alpha/cr/Rating3/Model/@value");

      String zoneText = getZoneText(stringToInteger(smodelNo), stringToInteger(szone));
      System.out.println("Kredittsone fra xml: " + szone + ": " + zoneText);

      // Any policycodes?
      List nodeList = doc.getXPathList("/info/exti/alpha/cr/Rating3/Comment/Comment");
      if ((nodeList != null) && (nodeList.size() > 0)) { //any?
         // avoid foreach loop on DOM4J, won't work
         //noinspection ForLoopReplaceableByForEach
         for (int i = 0; i < nodeList.size(); i++) {
            XmlDOMElement element = XmlDOMElement.getAsElement(nodeList.get(i));
            if (element != null) {
               System.out.println("Policyname from xml: " + element.getAttributeValue("value"));
            }
         }
      }
   }
   catch (IOException e) {
      e.printStackTrace();
   }
   catch (ParserConfigurationException e) {
      e.printStackTrace();
   }
   catch (SAXException e) {
      e.printStackTrace();
   }
}


private static void handleErrorResponse(
   no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSError error) {
   System.out.println("Error: " + error.getErrorCode() + ": " + error.getErrorMessage());
   // One may also want to get hold of the error measure
   System.out.println("What to do: " + error.getErrorMeasure());
}

private static String getDecision(String result) {
   String decision = "not set";
   if (result.equals("y"))
      decision = "approved";
   else if (result.equals("n"))
      decision = "denied";
   else if (result.equals("c"))
      decision = "to be checked";
   return decision;
}

private static int outputApplicantInfo(
   no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSSingleResponse srsp) {
   int didSome = 0;
   List<DmWSApplicant> apps = srsp.getApplicants();
   // Unless the score was done with a coapplicant, there is only one applicant, but lets
   // pass thru all:
   if ((apps != null) && (apps.size() > 0)) {
      for (int i = 0; i < apps.size(); i++) {
         DmWSApplicant app = apps.get(i);
         if (app != null) {
            didSome++;
            if (apps.size() > 1)  // more than one applicant returned?
               System.out.println("Applicant no.: " + (i + 1));
            if (app.getType().equals("P"))
```

```
                System.out.println("Name: " + app.getFirstName() + " " + app.getLastName());
             else
                System.out.println("Name: " + app.getName());

             no.bisnode.creditwsclients.ws3.jaxws.generated.DmWSAddress address = app.getAddress();

             if (address != null) {
                System.out.println("Address: " + address.getStreet());
                // ...etc...
             }
             // ...etc...
          }
       }
    }
    else {
       // Shouldn't happen, but the applicant info is also stored as totalscore elements
    }
    return didSome;
 }

 private static int stringToInteger(String str) {
    try {
       return Integer.parseInt(str);
    }
    catch (Exception e) {
       return 0; // when unable to convert
    }
 }
```

## 12.3 Score with arguments

```
DmWSResponse response = null;
DmWSRequest mainRequest = new DmWSRequest();
DmWSScoreRequest request = new DmWSScoreRequest();
mainRequest.setUser("username");                // user identification/login name
mainRequest.setPassWord("password");            // password
mainRequest.setRequests(new DmWSScoreRequest[]{request});   // set list of requests
request.setProductName("Standard Person");      // set name of credit model
// we need an applicant object:
DmWSApplicant applicant = new DmWSApplicant();
applicant.setSsn("01016012345");               // who to search for
applicant.setNationality("no");                // it's a norwegian
applicant.setType("P");                        // it's a person (not a company)
request.setApplicants(new DmWSApplicant[]{applicant}); // applicants list to the request

// add arguments to the credit model:
DmWSAppInfo appInfo = new DmWSAppInfo();
applicant.setAppInfo(appInfo);
List appInfos = new ArrayList();
addArgument(appInfos, "Student", "yes");
addArgument(appInfos, "arg2", new Float(0.86505F));
addArgument(appInfos, "arg3", "something");
addArgument(appInfos, "Category", "C");

appInfo. setMemberValues((DmWSMemberValue[])
       appInfos.toArray(new DmWSMemberValue [appInfos.size()]));

// run the request:
try {
       response = new DecisionMakerWebServicesImpl("www.soliditetd.no").score(mainRequest);
       // add response handling
}
catch (MalformedURLException e) {
       e.printStackTrace();
}
catch (ServiceException e) {
       e.printStackTrace();
}
catch (RemoteException e) {
       e.printStackTrace();
```

```
}


private static void addArgument(List appInfos, String name, Object value) {
        DmWSMemberValue wse = new DmWSMemberValue ();
        wse.setSet(true);
        wse.setName(name);
        wse.setVal(value.toString());
        appInfos.add(wse);
}
```