

[euromuse.net](http://euromuse.net)



# ***euromuse.net* SPECIFICATIN OF HARMONISE 2.0 INTERFACES**

version 1.0

ENGLISH

## Table of contents

<b>Acronyms and Abbreviations</b> .....	<b>3</b>
<b>Web Services specifications</b> .....	<b>5</b>
<i>WSDL specifications</i> .....	5
<i>Sample client</i> .....	9
<b>Send and Query API specifications</b> .....	<b>12</b>
<i>Send API specifications</i> .....	12
<i>Query API specifications</i> .....	13
<i>Example of XML query file: euromuse.net query service</i> .....	14
<b>Integrating Harmonise query service into HTML documents</b> .....	<b>16</b>

## Acronyms and Abbreviations

API	Application Programming Interface
CMS	Content Management System
DTD	Document Type Definition
HTML	HyperText Markup Language
HTTP	Hyper Text Transfer Protocol
IFRAME	Inline Frame
JAR	Java ARchive
JAXB	Java Architecture for XML Binding
JAX-RPC	Java API for XML-based RPC
JAX-WS	Java API for XML Web Services
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WSDL	Web Services Description Language
XML	eXtensible Markup Language

## Introduction

The aim of Harmonise is to provide a technological solution which allows data exchange amongst many actors, each one using its own data structure.

A User Interface has been developed providing functionalities to send data to other participants and to query data from other participants. For more details on the usage of the Web front-end see the corresponding manual.

The goal of this document is to explain how Harmonise can be used to automatically exchange data with other users without using the Web Interface.

The first chapter describes how to take advantage of Harmonise Web Services for:

- a. sending data to other users;
- b. running queries on databases of other users to search for specific data.

The second chapter specifies the APIs to be implemented by a participant for:

- a. receiving data from other participants through Harmonise, automatically writing them in his own CMS;
- b. supporting queries which can be posed by external actors through Harmonise, automatically retrieving and sending data from his own CMS.

Finally the last chapter explains how to integrate Harmonise query service within existing web pages (e.g. web sites, portals, ...), thanks to the usage of the IFRAME technology.

## Web Services specifications

### WSDL specifications

Here below a description of Harmonise Web Services is presented, in terms of WSDL graphic schemas and XML specifications.

Harmonise technological solution exposes a `DataExchangeManager` Web Service Interface that can be used by any registered user to exchange data with all the other registered users.

This Web Service offers two operations:

- `send` operation: this can be called to upload data to a specific participant
- `query` operation: this allows to request specific data from a specific participant

The ***send*** operation takes as input parameters:

- `sender`: name of the sender, as registered in Harmonise
- `receiver`: name of the receiver, as registered in Harmonise
- `xml_file`: the XML file to be sent (using the data model of the sender)

It uploads and sends the input XML file to the specified recipient, either putting it in his inbox or pushing it directly in his system depending on the recipient's configuration. The operation returns true if the sending succeeds, false otherwise and can raise one of the following exceptions:

- `ACLEException`: if the participant is not authorised to perform the requested send operation or a problem occurs during authorisation checking
- `ConnectorException`: if a problem occurs connecting either to the sender or to the receiver connector
- `ReconciliationException`: if a problem occurs during the transformation of the data from the sender format to the receiver format
- `IOException`: if a problem occurs reading the input XML file

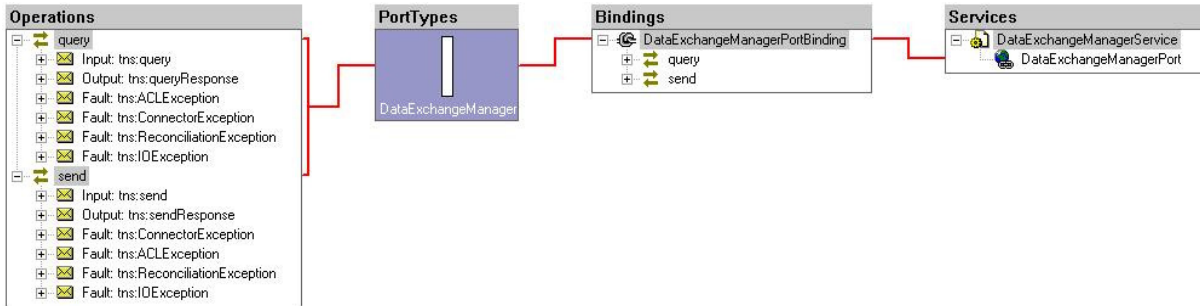
The ***query*** operation takes as input parameters:

- `sender`: name of the sender of the query, as registered in Harmonise
- `receiver`: name of the queried participant, as registered in Harmonise
- `maxItems`: max number of results returned (a negative integer means no restrictions)
- `xml_query`: the XML file containing the query to be forwarded (see paragraph 2.3 for an example of XML query file)

It forwards the XML file to the specified recipient, running the corresponding query and returning back to the sender the results, either putting them in his inbox or pushing them directly in his system depending on the sender's configuration. The operation returns true if the query succeeds, false otherwise and can raise the same exceptions of the send operation:

- `ACLEException`: if the participant is not authorised to perform the requested query operation or a problem occurs during authorisation checking
- `ConnectorException`: if a problem occurs connecting either to the sender or to the receiver connector

- `ReconciliationException`: if a problem occurs during the transformation of the query results from the receiver format to the sender format
- `IOException`: if a problem occurs reading the input XML query file



The schema depicted above shows the graphic representation of the Web Service WSDL elements, which are specified in the WSDL file below.

The WSDL file is structured in two main parts. The first part describes the abstract service interface definition, i.e., `<Types>`, `<Message>`, `<PortType>` and `<Binding>` elements. The second part describes the concrete end points of service implementation definition, i.e., `<Service>` element.

## WSDL specifications

```
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://service.webservices.services.harmonise.org/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://service.webservices.services.harmonise.org/"
name="DataExchangeManagerService">
  <types>
    <xs:schema xmlns:tns="http://service.webservices.services.harmonise.org/"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ns1="http://www.w3.org/2005/05/xmlmime"
xmlns:ns2="http://www.w3.org/2005/05/xmlmime"
targetNamespace="http://service.webservices.services.harmonise.org/" version="1.0">
      <xs:element name="ACLEException" type="tns:ACLEException"/>
      <xs:element name="ConnectorException" type="tns:ConnectorException"/>
      <xs:element name="IOException" type="tns:IOException"/>
      <xs:element name="ReconciliationException" type="tns:ReconciliationException"/>
      <xs:element name="query" type="tns:query"/>
      <xs:element name="queryResponse" type="tns:queryResponse"/>
      <xs:element name="send" type="tns:send"/>
      <xs:element name="sendResponse" type="tns:sendResponse"/>
      <xs:complexType name="send">
        <xs:sequence>
          <xs:element name="arg0" type="xs:string" minOccurs="0"/>
          <xs:element name="arg1" type="xs:string" minOccurs="0"/>
          <xs:element name="arg2" type="xs:base64Binary" minOccurs="0"
ns1:expectedContentTypes="application/octet-stream"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="sendResponse">
        <xs:sequence>
          <xs:element name="return" type="xs:boolean"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </types>

```

```

        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ConnectorException">
        <xs:sequence>
            <xs:element name="errorCode" type="xs:int"/>
            <xs:element name="message" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ACLException">
        <xs:sequence>
            <xs:element name="errorCode" type="xs:int"/>
            <xs:element name="message" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ReconciliationException">
        <xs:sequence>
            <xs:element name="errorCode" type="xs:int"/>
            <xs:element name="message" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="IOException">
        <xs:sequence>
            <xs:element name="message" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="query">
        <xs:sequence>
            <xs:element name="arg0" type="xs:string" minOccurs="0"/>
            <xs:element name="arg1" type="xs:string" minOccurs="0"/>
            <xs:element name="arg2" type="xs:int"/>
            <xs:element name="arg3" type="xs:base64Binary" minOccurs="0"
ns2:expectedContentTypes="application/octet-stream"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="queryResponse">
        <xs:sequence>
            <xs:element name="return" type="xs:boolean"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
</types>
<message name="query">
    <part name="parameters" element="tns:query"/>
</message>
<message name="queryResponse">
    <part name="parameters" element="tns:queryResponse"/>
</message>
<message name="ACLException">
    <part name="fault" element="tns:ACLException"/>
</message>
<message name="ConnectorException">
    <part name="fault" element="tns:ConnectorException"/>
</message>
<message name="ReconciliationException">
    <part name="fault" element="tns:ReconciliationException"/>
</message>
<message name="IOException">
    <part name="fault" element="tns:IOException"/>
</message>
<message name="send">
    <part name="parameters" element="tns:send"/>
</message>
<message name="sendResponse">
    <part name="parameters" element="tns:sendResponse"/>
</message>
<portType name="DataExchangeManager">
    <operation name="query">

```

```

    <input message="tns:query"/>
    <output message="tns:queryResponse"/>
    <fault name="ACLEException" message="tns:ACLEException"/>
    <fault name="ConnectorException" message="tns:ConnectorException"/>
    <fault name="ReconciliationException" message="tns:ReconciliationException"/>
    <fault name="IOException" message="tns:IOException"/>
  </operation>
  <operation name="send">
    <input message="tns:send"/>
    <output message="tns:sendResponse"/>
    <fault name="ConnectorException" message="tns:ConnectorException"/>
    <fault name="ACLEException" message="tns:ACLEException"/>
    <fault name="ReconciliationException" message="tns:ReconciliationException"/>
    <fault name="IOException" message="tns:IOException"/>
  </operation>
</portType>
<binding name="DataExchangeManagerPortBinding" type="tns:DataExchangeManager">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="query">
    <soap:operation/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
    <fault name="ACLEException">
      <soap:fault name="ACLEException" use="literal"/>
    </fault>
    <fault name="ConnectorException">
      <soap:fault name="ConnectorException" use="literal"/>
    </fault>
    <fault name="ReconciliationException">
      <soap:fault name="ReconciliationException" use="literal"/>
    </fault>
    <fault name="IOException">
      <soap:fault name="IOException" use="literal"/>
    </fault>
  </operation>
  <operation name="send">
    <soap:operation/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
    <fault name="ConnectorException">
      <soap:fault name="ConnectorException" use="literal"/>
    </fault>
    <fault name="ACLEException">
      <soap:fault name="ACLEException" use="literal"/>
    </fault>
    <fault name="ReconciliationException">
      <soap:fault name="ReconciliationException" use="literal"/>
    </fault>
    <fault name="IOException">
      <soap:fault name="IOException" use="literal"/>
    </fault>
  </operation>
</binding>
<service name="DataExchangeManagerService">
  <port name="DataExchangeManagerPort" binding="tns:DataExchangeManagerPortBinding">
    <soap:address
location="http://euromuse.ectrldev.com:80/webAccessPortal/DataExchangeManager"/>
  </port>
</service>

```

## Sample client

Harmonise Web Services have been developed using JAX-WS 2.0, an important part of the Java EE 5 platform. A follow-on release of JAX-RPC 1.1, JAX-WS simplifies the task of developing web services using Java technology. It addresses some of the issues in JAX-RPC 1.1 by providing support for multiple protocols such as SOAP 1.1, SOAP 1.2, XML, and by providing a facility for supporting additional protocols along with HTTP. JAX-WS uses JAXB 2.0 for data binding and supports customizations to control generated service endpoint interfaces. With its support for annotations, JAX-WS simplifies web service development and reduces the size of runtime Jar files.

In order to access Harmonise the `DataExchangeManager` Web Service described in the previous paragraph, a client program is needed. Here are the steps to follow to build the client:

1. Generate portable artifacts required to compile the client
2. Write the client
3. Compile and run the client

It is possible to use any kind of technology to develop such a client (e.g. Axis, Spring WS, etc...), here below is presented how to build it using JAX-WS technology.

### 1. Generate portable artifacts required to compile the client

The client stub classes which can be used to invoke the Web Services operations can be generated automatically with the `wsimport` command tool. Such tool processes an existing WSDL file and generates the required portable artifacts for developing JAX-WS Web Service applications.

The `wsimport` tool can be launched using the command line script `wsimport.sh` (Unix) or `wsimport.bat` (Windows). An Ant task for the `wsimport` tool is provided along with the tool. The attributes and elements supported by the Ant task are listed below:

```
<wsimport
  wsdl="..."
  destdir="directory for generated class files"
  sourcedestdir="directory for generated source files"
  keep="true|false"
  extension="true|false"
  verbose="true|false"
  version="true|false"
  wsdlLocation="..."
  catalog="catalog file"
  package="package name"
  <binding dir="..." includes="..." />
</wsimport>
```

Attribute	Description	Command line
-----------	-------------	--------------

Attribute	Description	Command line
wsdl	WSDL file	WSDL
destdir	Specify where to place output generated classes	-d
sourcedestdir	Specify where to place generated source files, keep is turned on with this option	-s
keep	Keep generated files, turned on with sourcedestdir option	-keep
verbose	Output messages about what the compiler is doing	-verbose
binding	Specify external JAX-WS or JAXB binding files	-b
extension	allow vendor extensions (functionality not specified by the specification). Use of extensions may result in applications that are not portable or may not interoperate with other implementations	-extension
wsdllocation	The wsdl URI passed thru this option will be used to set the value of <code>@WebService.wsdlLocation</code> and <code>@WebServiceClient.wsdlLocation</code> annotation elements on the generated SEI and Service interface	-wsdllocation
catalog	Specify catalog file to resolve external entity references, it supports TR9401, XCatalog, and OASIS XML Catalog format. Additionally, ant <code>xmlcatalog</code> type can be used to resolve entities, see <code>wsimport_catalog</code> sample.	-catalog
package	Specifies the target package	-p

The artifacts needed for developing `Harmonise DataExchangeManager` client have already been packaged in a JAR file (`euromuse_webServices-stub.jar`) which is included in the Harmonise release.

## 2. Write the client

The following class, `DataExchangeManagerClient`, provides sample code that can be used to develop the client. It invokes both the send and the query operations on the deployed service.

### Sample Client

```
import javax.activation.DataHandler;
```

```

import javax.activation.DataSource;
import javax.activation.FileDataSource;
import org.harmonise.services.webservices.client.stub.DataExchangeManager;
import org.harmonise.services.webservices.client.stub.DataExchangeManagerService;

/**
 * Sample client for calling Harmonise push and query web services.
 * Before running the client ("run-client" ant task) it is necessary
 * to generate the stub ("dist-client" ant task).
 */
public class DataExchangeManagerClient {

    private static final String PUSH = "push";
    private static final String QUERY = "query";

    public static void main(String[] args) {
        if(args.length != 4) {
            System.out.println("Error parsing args: != 4 arguments");
            return;
        }
        String sender = args[1];
        String receiver = args[2];
        String xml_file = args[3];
        DataExchangeManagerClient client = new DataExchangeManagerClient();
        if(args[0].equals(DataExchangeManagerClient.PUSH))
            client.doTestSend(sender, receiver, xml_file);
        else if(args[0].equals(DataExchangeManagerClient.QUERY))
            client.doTestQuery(sender, receiver, xml_file);
        else
            System.out.println("Error parsing first arg: 'push' or 'query'");
    }

    public void doTestSend(String sender, String receiver, String xml_file) {
        try {
            DataExchangeManagerService service = new DataExchangeManagerService();
            System.out.println("Retrieving port from service: " + service);
            DataExchangeManager port = service.getDataExchangeManagerPort();
            System.out.println("Invoking the send operation on the port.");
            DataSource source = new FileDataSource(xml_file);
            boolean r = port.send(sender, receiver, new DataHandler(source));
            System.out.println("response: " + r);
        } catch(Exception e) {
            System.err.println("Error in sending the file: " + e.getMessage());
        }
    }

    public void doTestQuery(String sender, String receiver, String xml_query) {
        try {
            DataExchangeManagerService service = new DataExchangeManagerService();
            System.out.println("Retrieving port from service: " + service);
            DataExchangeManager port = service.getDataExchangeManagerPort();
            System.out.println("Invoking the query operation on the port.");
            DataSource source = new FileDataSource(xml_query);
            boolean r = port.query(sender, receiver, -1, new DataHandler(source));
            System.out.println("response: " + r);
        } catch(Exception e) {
            System.err.println("Error in sending the query: " + e.getMessage());
        }
    }
}

```

### 3. Compile and run the client

After the client has been compiled using the *javac* command tool (either using Ant or command line) it is possible to run it.

The sample client described above takes as input parameters:

- *operation*: either “push” or “query”
- *sender*: sender of the operation
- *receiver*: receiver of the operation
- *xml\_file*: file to be sent, either XML file or XML query

## Send and Query API specifications

In order to activate the automatic writing of data into the CMS and the automatic retrieval and sending of data from the CMS, a participant has to implement two APIs:

- Send API
- Query API

These are two HTTP post invocations which are called by Harmonise to interact with the participant’s proprietary system.

The **SendAPI** is called by Harmonise to send content to a participant. By default the content is stored in the participant’s inbox. Users who are interested in receiving and storing it directly in their CMS should implement this HTTP call handler and configure it in their directory service configuration page.

The **QueryAPI** is called by Harmonise to query a participant’s content. Users who are interested in supporting queries which can be posed by external actors through Harmonise, automatically retrieving and sending data from his own CMS, should implement this HTTP call handler and configure it in their directory service configuration page. This call should return the contents matching the query passed as XML.

## Send API specifications

The file is sent from Harmonise to the participant using a standard HTTP multipart post request.

The parameters which are sent are:

- *senderid*: Harmonise id of the sender
- *sender*: name of the sender
- *semail*: email of the sender
- *filename*: name of the file to be uploaded
- *notes*: some annotations (optional)
- *xml\_file*: the XML file to be uploaded (using the data model of the receiver)

Harmonise calls the URL implemented by the receiver and configured in the directory service administration page, by passing the above mentioned parameters.

The invocation of Harmonise to the participant for sending content, from the participant’s point of view, looks like a submit of the following form:

```
<form method="post"
  action="url implemented by the participant"
  enctype="multipart/form-data">
```

```

Sender id:      <input type="text" name="senderid" value="123">
Sender name:    <input type="text" name="sender" value="Museum A">
Sender email:   <input type="text" name="semail" value="museum@a.com">
File name:     <input type="text" name="filename"
value="sample.xml">
Notes:         <input type="text" name="notes" value="My notes">
XML File:      <input type="file" name="xml_file">

<input type="submit" name="import" value="submit">
</form>

```

## Query API specifications

The query is sent from Harmonise to the participant using a standard HTTP multipart post request.

An XML file with an example of the “product” is sent along with other parameters following the so called Query By Example approach.

The parameters which are sent are:

- `senderid`: Harmonise id of the sender of the query
- `sender`: name of the sender
- `semail`: email of the sender
- `language`: language of the query
- `limit`: max number of results returned (an empty string means no restrictions)
- `xml_query`: the XML file containing a sample item used as “query by example” (see paragraph 2.3 for an example of XML query file)

Harmonise calls the URL implemented by the receiver and configured in the directory service administration page, by passing the above mentioned parameters.

The queried participant should return an XML file (using its own data model) containing all items matching the query by example passed as parameter.

The invocation of Harmonise to the participant for querying content, from the participant’s point of view, looks like a submit of the following form:

```

<form method="post"
  action="url implemented by the participant"
  enctype="multipart/form-data">

  Sender id:      <input type="text" name="senderid" value="123">
  Sender name:    <input type="text" name="sender"
value="visiteurope.com">
  Sender email:   <input type="text" name="semail"
                  value="info@visiteurope.com ">
  Language:      <input type="text" name="language" value="en">
  Limit:         <input type="text" name="limit" value="10">
  XML File:      <input type="file" name="xml_query">

```

```
<input type="submit" name="export" value="submit">
</form>
```

For an example of XML file containing the query see the paragraph 2.3.

### Example of XML query file: *euromuse.net* query service

The format of the XML file describing the query should be specified by the participant who receives the query, in order to easily map the XML query file with a real query to its CMS.

Here below is presented an example of such XML query file, i.e. the one provided by *euromuse.net* portal. The structure of the XML file containing *euromuse.net* specific query is specified by the following DTD.

#### xml\_query.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT categories (category_id)>
<!ELEMENT category_id (#PCDATA)>
<!ELEMENT date_end (#PCDATA)>
<!ELEMENT date_start (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT exhibition (name, description, date_start, date_end, location,
categories)>
<!ELEMENT exhibitions (exhibition)>
<!ELEMENT location (location_country, location_town)>
<!ELEMENT location_country (#PCDATA)>
<!ELEMENT location_town (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT webservice (exhibitions)>
```

In this example the possible elements which can be queried are:

- name of the exhibition
- description
- date range (from-to)
- location (country, city)
- category

From the participant's point of view, which in this case is *euromuse.net*, the `xml_query` file parameter of the Query API call looks therefore as follows.

#### xml\_query.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<webservice>
```

```
<exhibitions>
  <exhibition>
    <name></name>
    <description></description>
    <date_start>20090101</date_start>
    <date_end>20091231</date_end>
    <location>
      <location_country>51</location_country>
      <location_town>65</location_town>
    </location>
    <categories>
      <category_id>10</category_id>
    </categories>
  </exhibition>
</exhibitions>
</webservice>
```

Where:

1. the search values contained in the descriptive fields (`name`, `description`) should be interpreted as keywords (i.e. a query should return all exhibitions whose name *contains* the value specified in the `name/description` field and not only the ones whose name *is exactly* that one) and should be considered together with the `language` parameter
2. `date_start` and `date_end` should be interpreted as a date range (i.e. respectively as date from and date to)
3. `location_country`, `location_town` and `category_id` are identifiers taken from Euromuse reference lists (e.g. country "51" means "Germany", town "65" means "Berlin" and category "10" means "photography").

## Integrating Harmonise query service into HTML documents

Harmonise technical solution offers also the possibility to integrate the query service functionality into existing web pages, such as web sites or portals. This can be done very simply thanks to the usage of the IFRAME technology.

The IFRAME element defines an inline frame. An inline frame ("floating frame") is a construct which embeds a document into an HTML document so that embedded data is displayed inside a subwindow of the browser's window. This does not mean full inclusion: the two documents are independent, and both them are treated as complete documents, instead of treating one as part of the other.

Basically, an iframe element is of the form:

```
<iframe src="URL" more attributes>  
alternative content for browsers which do not support iframe  
</iframe>
```

Browsers which support iframe display the document referred to by the URL in a subwindow, typically with vertical and/or horizontal scroll bars; such browsers ignore the content of the iframe element (i.e. everything between the start tag `<iframe . . . >` and the end tag `</iframe>`). Browsers which do not support iframe (or have such support disabled) do the opposite: process the content as if the `<iframe . . . >` and `</iframe>` tags were not there.

When inline frames are used, the browser (if it supports them) sends a request to the server referred to by the URL in the iframe element, and after getting the requested document displays it inside an inline frame. In this sense inline frames are a joint browser-server issue, but only the browser needs to be specifically iframe-aware; from the server's point of view, there's just a normal HTTP request for a document, and it sends the document without having (or needing) any idea on what the browser is going to do with it.

Therefore, in order to integrate Harmonise query service into an existing web page it is enough to put the IFRAME element inside the desired HTML document, specifying the URL of the Harmonise HTML page which contains the query service as the value of the IFRAME `src` attribute. This allows to perform queries using Harmonise solution and to visualise the results directly in the web page where the IFRAME is placed.

Here is a brief description of the IFRAME main attributes:

- IFRAME's `src` attribute provides the location of the frame content--typically an HTML document. E.g. in the case of the euromuse.net online query it is:  
`http://euromuse.harmonet.org/webAccessPortal/wap/queryForm.htm`
- The optional `name` attribute specifies the name of the inline frame, allowing links to target the frame.
- The content of the IFRAME element is used as an alternative for browsers that are not configured to show or do not support inline frames. The content may consist of inline or block-level elements, though any block-level elements must be allowed inside the containing element of IFRAME. For example, an IFRAME within an H1 cannot contain an H2, but an IFRAME within a DIV can contain any block-level elements.
- The `longdesc` attribute gives the URI of a long description of the frame's contents. This is particularly useful for full descriptions of embedded objects. Note that `longdesc` describes the frame content while the content of the IFRAME element acts as a replacement when the external resource cannot be inlined.

- The `width` and `height` attributes specify the dimensions of the inline frame in pixels or as a percentage of the available space.
- The `frameborder` attribute specifies whether or not a border should be drawn. The default value of 1 results in a border while a value of 0 suppresses the border. Style sheets allow greater flexibility in suggesting the border presentation.
- The `align` attribute specifies the alignment of the inline frame. The values `top`, `middle`, and `bottom` specify the frame's position with respect to surrounding content on its left and right. `align=middle` aligns the vertical center of the frame with the current baseline. To center the frame horizontally on the page, place the frame in a centered block. The other `align` values, `left` and `right`, specify a floating frame; the frame is placed at the left or right margin and content flows around it. To place content below the frame, use `<br clear=left|right|all>` as appropriate. The `vertical-align` and `float` properties of Cascading Style Sheets provide more flexible methods of aligning inline frames.
- The `marginwidth` and `marginheight` attributes define the number of pixels to use as the left/right margins and top/bottom margins, respectively, within the inline frame. The value must be a non-negative integer.
- The `scrolling` attribute specifies whether scrollbars are provided for the inline frame. The default value, `auto`, generates scrollbars only when necessary. The value `yes` gives scrollbars at all times, and the value `no` suppresses scrollbars--even when they are needed to see all the content.