ZEISS PMS Web Service Integration Guide

Interface Definition
Version 1.1.7

Contents

1 General

1.1 Purpose

This document guides the integration of the SOAP interface for patient management systems (PMS) for the VISUCONNECT 500 / VISUSCREEN 500 ZEISS Essential Line devices. Normally this definition file is provided by the according device when opening a web service connection to it. If you do not have a device the wsdl file is provided separately.

1.2 Scope

This documentation describes the general web service interface on Essential Line devices from the Carl Zeiss Vision GmbH and the Carl Zeiss Meditec AG for PMS, EMR systems, respectively.

PMS is used synonymously for PMS and EMR systems in this document.

1.3 Relevant Documents

Title	Document No.	Version
Interface definition device connectivity PMS	003/15	1.1
Interface definition device connectivity PMS XML Schema 1.1.7	005/15	1.0

Table 1 Relevant documents

1.4 Definitions and Acronyms

EMR	Electronic Medical Record
CLR	Common Language Runtime
GUID	Global Unique Identifier
HTTP	Hypertext Transfer Protocol
ID	Identifier
PMS	Patient Management System
SOAP	Protocol specification for exchanging structured information in the implementation of Web Services in computer networks
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	XML Schema (W3C)

1.5 Tables

1.6 Figures

2 Web Service Interface Definition

2.1 Technology

The Web service is implemented as SOAP Web service over HTTP.

2.2 Connection Parameters (URL)

URL:	http://<DeviceName>:<Port>/<DeviceSpecificPath>
WSDL:	http://<DeviceName>:<Port>/<DeviceSpecificPath>?wsdl

2.2.1 VISUCONNECT 500

The PMS interface is active as soon as the user activates this interface via network settings.

	Device	Remark
DeviceName	VISUCONNECT 500	Network name or the IP address of the device.
Port		User defined in network settings.
DeviceSpecificPath		“pms”
Example connection string		http://192.168.99.100:50000/pms http://192.168.99.100:50000/pms?wsdl

2.2.2 VISUSCREEN 500

The SOAP interface is only active, when the application is running. The application starts as soon as a client has entered the VISUSCREEN 500 web interface otherwise the connection string throws a timeout exception.

	Device	Remark
DeviceName	VISUSCREEN 500	Network name or the IP address of the device.
Port		50000 (static)
DeviceSpecificPath		“pms”
Example connection string		http://192.168.99.98:50000/pms http://192.168.99.98:50000/pms?wsdl

3 Supported Features

3.1 VISUCONNECT 500

VISUCONNECT 500 enables serial devices from the ZEISS Essential Line to transfer measurement data via network to a connected PMS. Following functions are supported by VISUCONNECT 500 SOAP PMS interface:

3.1.1 Supported Functions

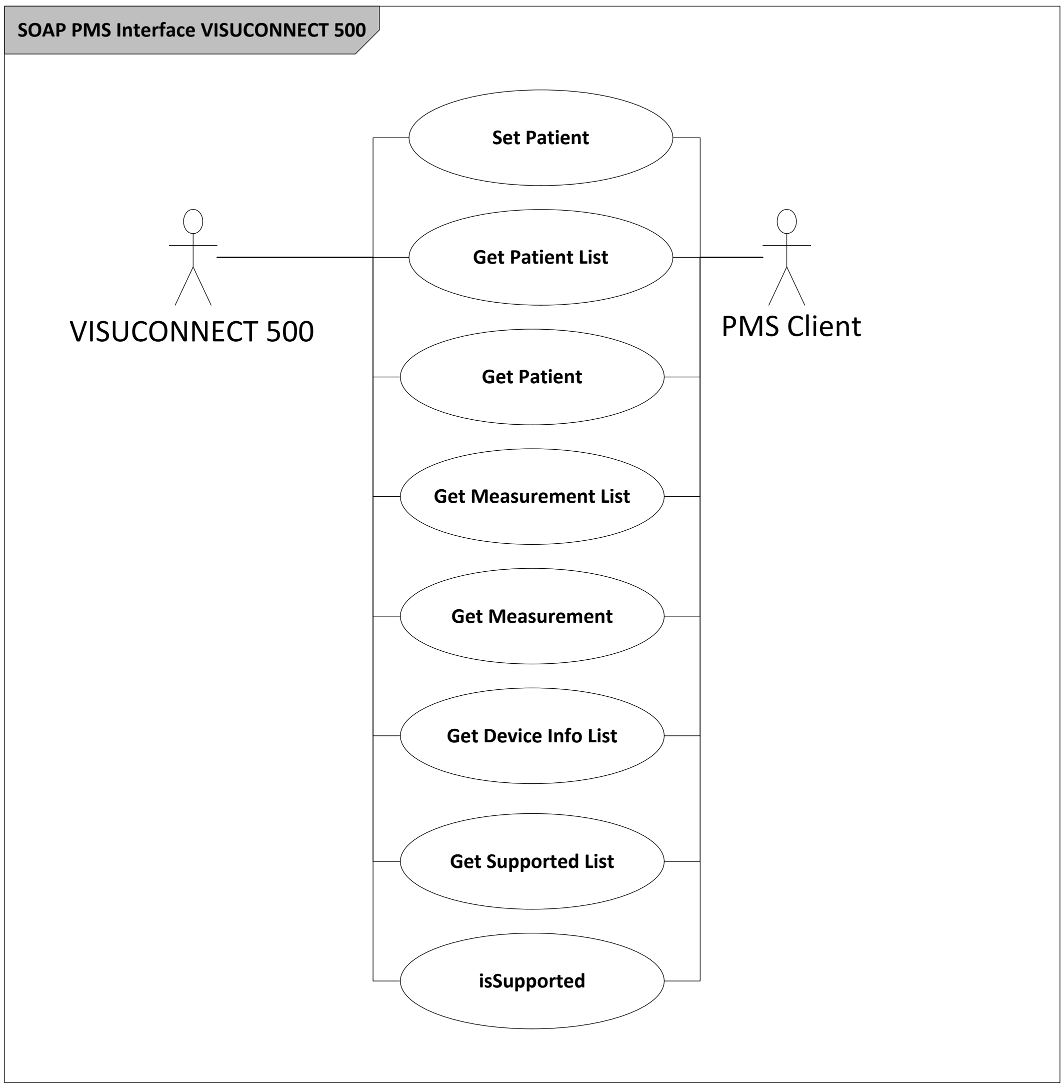

Figure 1 Supported functions VISUCONNECT 500

3.1.2 Use Flow

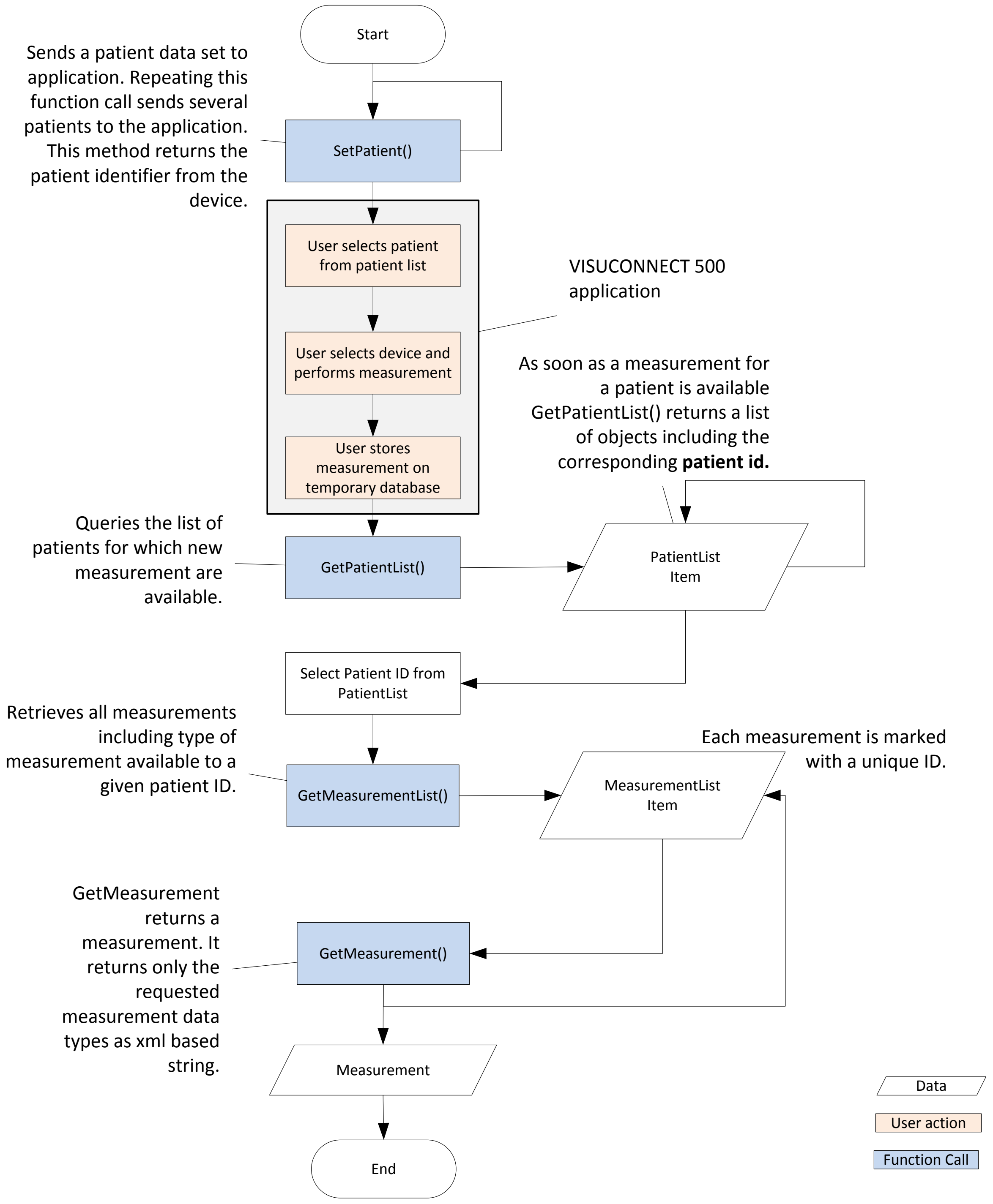
Start
SetPatient()
Sends a patient data set to application. Repeating this function call sends several patients to the application. This method returns the patient identifier from the device.
User selects patient from patient list
User selects device and performs measurement
User stores measurement on temporary database
VISUCONNECT 500 application
As soon as a measurement for a patient is available GetPatientList() returns a list of objects including the corresponding **patient id.**
Queries the list of patients for which new measurement are available.
GetPatientList()
PatientList Item
Select Patient ID from PatientList
Retrieves all measurements including type of measurement available to a given patient ID.
GetMeasurementList()
MeasurementList Item
Each measurement is marked with a unique ID.
GetMeasurement returns a measurement. It returns only the requested measurement data types as xml based string.
GetMeasurement()
Measurement
End
Data
User action
Function Call

Figure 2 Use flow VISUCONNECT 500

3.2 VISUSCREEN 500

The VISUSCREEN 500 with its accessory the VISUPHOR 500 digital phoropter operates as an ophthalmic device and therefore produces measurement results and also consumes previously made measurements from autorefractors, lensmeters and other subjective refraction units.

3.2.1 Supported Functions

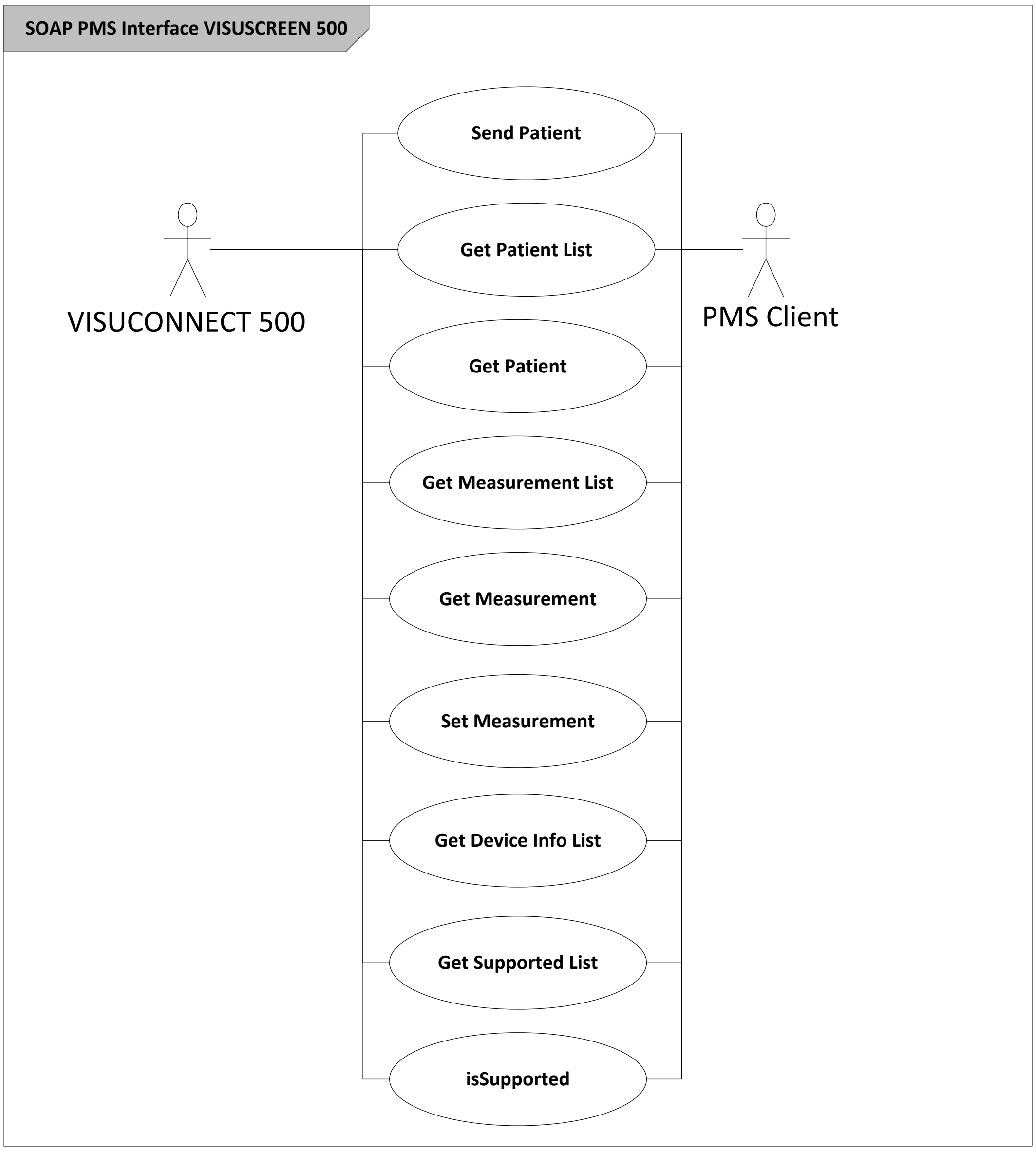

Figure 3 Supported functions VISUSCREEN 500

3.2.2 Use Flow VISUSCREEN 500

3.2.2.1 Anonymous Patient

The anonymous patient feature is used to perform a measurement without previously sending a patient to the VISUSCREEN 500. Hence, in this scenario the unscheduled case is exemplified.

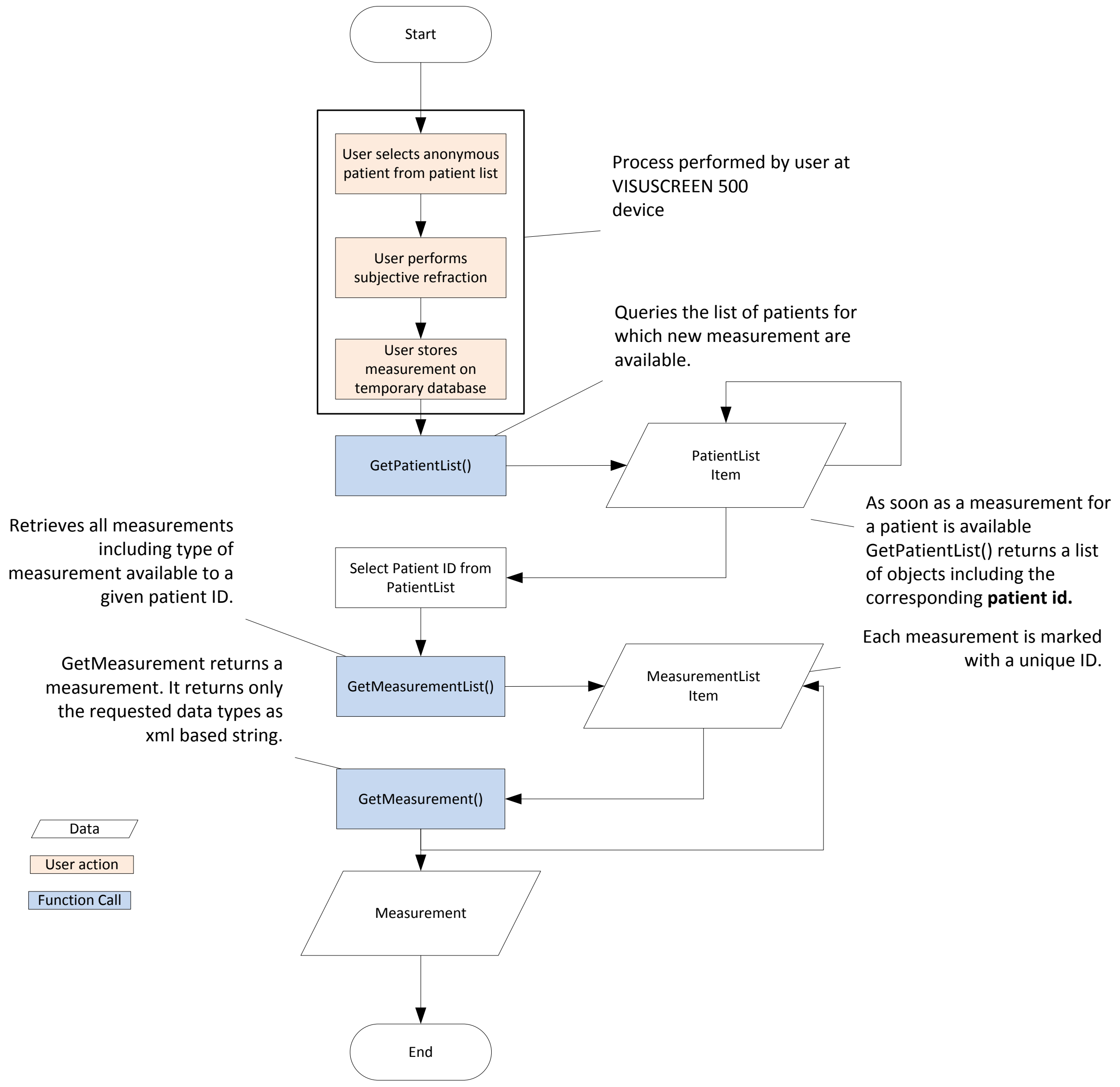
Start
User selects anonymous patient from patient list
User performs subjective refraction
User stores measurement on temporary database
Process performed by user at VISUSCREEN 500 device
Queries the list of patients for which new measurement are available.
GetPatientList()
PatientList Item
As soon as a measurement for a patient is available GetPatientList() returns a list of objects including the corresponding patient id.
Retrieves all measurements including type of measurement available to a given patient ID.
Select Patient ID from PatientList
Each measurement is marked with a unique ID.
GetMeasurement returns a measurement. It returns only the requested data types as xml based string.
GetMeasurementList()
MeasurementList Item
GetMeasurement()
Data
User action
Function Call
Measurement
End

Figure 4 Anonymous use flow VISUSCREEN 500

Measurements received from anonymous patients are matched to patient family name “Anonym”. The patient ID is given by a device generated GUID. Matching of this data to a real patient needs to be done by the connected PMS.

3.2.2.2 Sending a Patient to VISUSCREEN 500

The VISUSCREEN 500 supports direct patient management. Multiple patients can be send from the connected Patient Management System (PMS) to the VISUSCREEN.

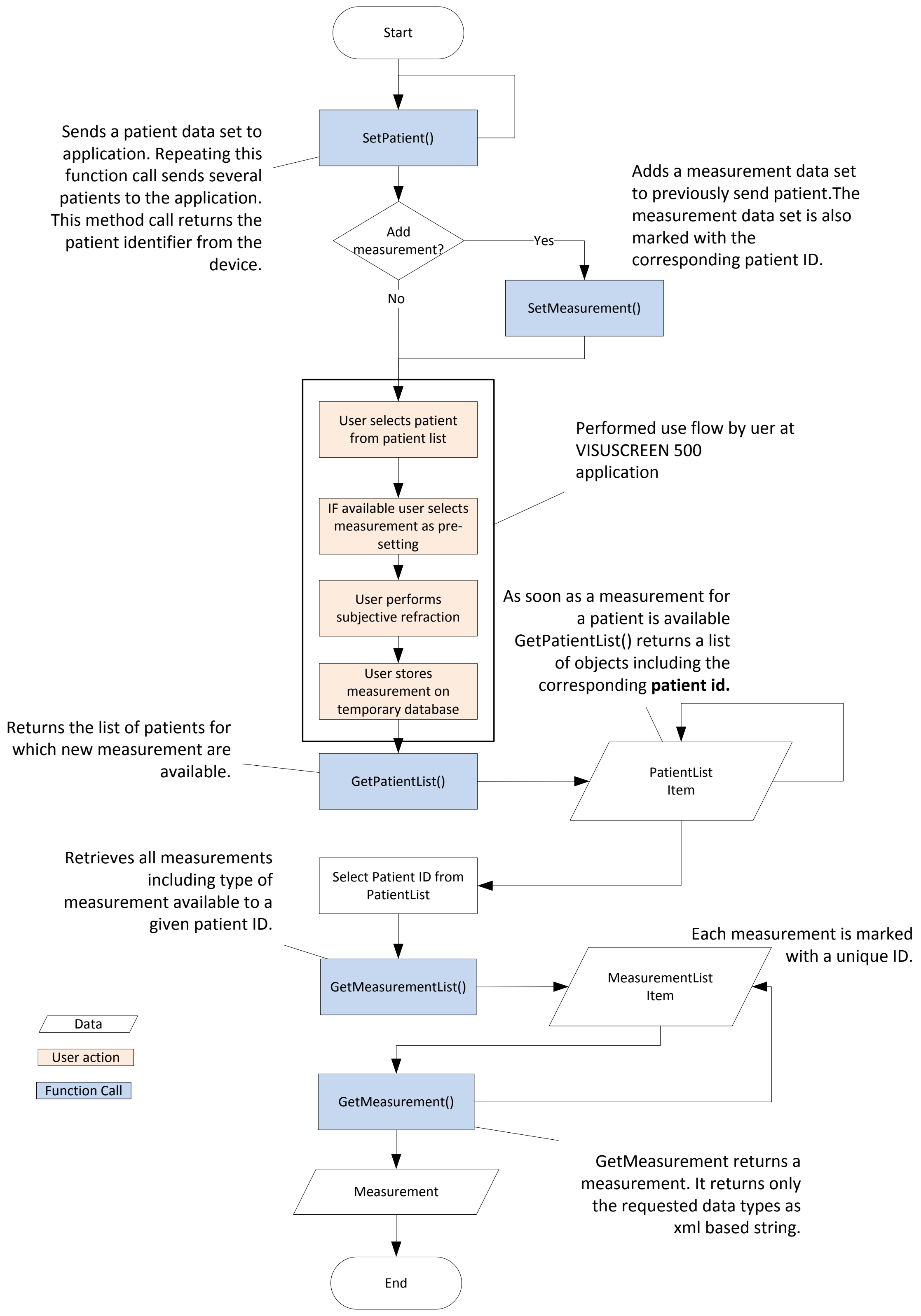
Start
SetPatient()
Sends a patient data set to application. Repeating this function call sends several patients to the application. This method call returns the patient identifier from the device.
Add measurement?
Yes
No
SetMeasurement()
Adds a measurement data set to previously send patient.The measurement data set is also marked with the corresponding patient ID.
User selects patient from patient list
IF available user selects measurement as pre-setting
User performs subjective refraction
User stores measurement on temporary database
Performed use flow by uer at VISUSCREEN 500 application
As soon as a measurement for a patient is available GetPatientList() returns a list of objects including the corresponding **patient id.**
Returns the list of patients for which new measurement are available.
GetPatientList()
PatientList Item
Select Patient ID from PatientList
Retrieves all measurements including type of measurement available to a given patient ID.
GetMeasurementList()
MeasurementList Item
Each measurement is marked with a unique ID.
Data
User action
Function Call
GetMeasurement()
GetMeasurement returns a measurement. It returns only the requested data types as xml based string.
Measurement
End

Figure 5 Set patient use flow VISUSCREEN 500

With respect to the patient ID, multiple measurement data sets can also be added to the system. As soon as the a measurement set is added the VISUSCREEN 500 displays the received measurement after a refresh of the patient list (pull down to refresh) is performed. Measurement sets send to the device are not mandatory. These sets are used as pre-setting for the digital phoropter.

3.2.2.3 Prerequisites

The PMS SOAP interface is available as soon as a user has logged into the system for the first time via PanelPC or iPad web browser. Otherwise the PMS SOAP interface will throw a timeout exception.

4 Integrating the PMS SOAP Service in C#

4.1 Overview

The Web service is implemented as SOAP Web service over HTTP. Access to the interface methods and data structures requires two preparation steps:

Create a C# class file with all the interface methods from the wsdl file.

Create a C# class file with all objects required for data exchange (RD.Connectivity).

After describing these two steps the chapter provides detailed information for each of the interface methods following the order of the flow charts presented above.

4.2 Creating a C# Class for Access to Service Methods and required Objects

The PMS SOAP interface is defined in within the following 3 files:

pms.wsdl
pms_1.wsdl and
pms.xsd

These files can by compiled to a C# class using the Microsoft tool wsdl.exe as demonstrated in the following example. The command line

"C:\Program Files (x86)\Microsoft SDKs\Windows\v8.1A\bin\NETFX 4.5.1 Tools\x64\wsdl.exe" /out:"SOAPService.cs" "pms_1.wsdl" "pms.wsdl" "pms.xsd" /namespace:PMSSOAPService

generates the class SOAPService.cs within the namespace PMSSOAPService. The parameters class name and namespace can be adapted to the requirements of your application.

The service objects for data exchange are defined within the file *RD.Connectivity.xsd*. This file can be compiled to a C# class using the Microsoft tool xsd.exe as demonstrated in the following example. The command line

"C:\Program Files (x86)\Microsoft SDKs\Windows\v8.1A\bin\NETFX 4.5.1 Tools\x64\xsd.exe" "Zeiss.Vision.DTI.Connectivity.xsd" /c /n:RD.Connectivity /o:..\.

generates the CLR class *Zeiss_Vision_DTI_Connectivity.cs.*

4.3 Using the PMS SOAP Service

After adding the generated class files to your application, the web service can be used. This section describes the available methods in the order of a typical use flow.

4.3.1 Initializing the PMS SOAP Service

In a first step the web service needs to be initialized. This can be accomplished as follows:

```
// Init SOAP Client
PmsSoapService SOAPclient = new PmsSoapService();
SOAPclient.Url = SOAPconnectionAddress;
```

with *SOAPconnectionAddress* defining the URL property. URL definitions need to follow the scheme http://**DeviceName**:**Port**/pms where *DeviceName* is the network name or the IP address and *Port* defines the used port of the device to be connected to. For VISUCONNECT the used port is user defined within the network settings menu. Default value is 50000. For VISUSCREEN the port is predefined to 50000 and cannot be changed by the user.

4.3.2 SetPatient() - Providing Patient Data

Before measurement data can be transferred to the device at least one patient needs to be allocated on it. The following example demonstrates how the SetPatient method is used to provide a minimum set of patient data to the device.

Create a SetPatient request object:

```
1           // init patient request
2           SetPatient patReq = new SetPatient();
```

The interface supports several writing systems to define a patient name. Create a list for the corresponding writing system required by the PersonName object. In accordance to ref [**Fehler! Verweisquelle konnte nicht gefunden werden.**], at least one writing system must be given.

```
3           List<PMSSOAPService.PersonName> lst_persName = new
4           List<PMSSOAPService.PersonName>
5           PMSSOAPService.PersonName persName = new PMSSOAPService.PersonName
6           persName.type = PMSSOAPService.PatientNameType.Alphabetic
7           persName.family = "Musterfrau_1"
8           persName.given = "Petra"
9           lst_persName.Add(persName
10          patReq.patient.patient.name = lst_persName.ToArray
```

The interface supports following writing systems:
Alphabetic
Ideographic and
Phonetic.

In addition to the patient name an identifier including its issuer of identifier and patient´s date of birth is required. Each patient requires at least one identifier.

```
11          List<PMSSOAPService.Identifier> lst_patientIdent = new
12          List<PMSSOAPService.Identifier>
13          PMSSOAPServi   .Identifier     = new PMSSOAPService.Identifier
14          id.issuer = "TestPMS"  // required
15          id.Value = "TestID789"  // required and unique for patient
16          lst_patientIdent.Add
17          patReq.patient.patient.id = lst_patientIdent.ToArray
18          patReq.patient.patient.dateOfBirth = "1930-05-01"   // required
```

Line 18 defines the patients date of birth. VISUCONNECT 500 and VISUSCREEN 100/500 support a date of birth value with reduced accuracy, used for peoples which are don’t know her exact date of birth. Examples:

1950	The patient is born in the year 1950.
1950-02	The patient is born in the February of the year 1950.
1950-02-12	The patient is born on the twelfth February 1950.

The generated patient request can now be send to the application via following method:

```
19          Identifier sendPatID = soapClient.SetPatient(patReq);
```

The SetPatient method returns the ID of the patient received to provide status of successful transmission.

4.3.3 SetMeasurement() – Add a Measurement Data to a Patient

A new SetMeasurement request needs to be created as starting point for presetting the digital phoropter (VISUPHOR 500).

```
1             PMSSOAPService.SetMeasurement measReq = new
PMSSOAPService.SetMeasurement();
```

The patient ID of the particular patient the measurement is mapped to is also required. If the device supports anonymous measurements (check IsSupported with "SetMeasurement" and "Anonymous"), this parameter can be inexistent. And otherwise it must not be empty.

```
2           if (PatientID != null)
            {
3               measReq.patientId = PatientID;
            }
4           PMSSOAPService.Measurement measurement = new PMSSOAPService.Measurement();
```

The SetMeasurement request object itself requires an identifier. This property must not be null.

```
1            List<PMSSOAPService.Identifier> id = new List<PMSSOAPService.Identifier>();
2            PMSSOAPService.Identifier idItem = new PMSSOAPService.Identifier();
```

Line 3 defines the issuer of the unique identifier of measurement. This property must not be set to "PMS" or "EMR". Otherwise the error message 220105 will be thrown by the service. Please use a meaningful name for the issuer of patient id.

```
3            idItem.issuer = "AnyPMS";
```

Line 8 creates a new GUID for the particular measurement. If a measurement ID is not managed by the PMS this property should be set a unique value, in this example a GUID is used instead.

```
4            // measurement/id must contain one measurement identifier with the PMS
5            // identifier value. This identifier must not be already used for another
6            // measurement.
7            // Any value in measurement/source will be replaced by PMS.
8            idItem.Value = Guid.NewGuid().ToString();
9            id.Add(idItem);
10           measurement.id = id.ToArray();
```

Line 11 sets the category of the send measurement. The category must be one of the following types:

Type of measurement	Description
`Prescription`	Lensmeter readings
`ObjectiveRefraction`	Autrefractor readings
`SubjectiveRefraction`	Phoropter readings

```
11            measurement.category = typeOfMeasurement;
```

Line 12 defines the source of measurement. In this use case this property must be set to PMS since the data set is managed by and send from a Patient Management System.

```
12            measurement.source = PMSSOAPService.MeasurementSource.PMS;
13            measurement.timestamp = DateTime.UtcNow;
14            measurement.remark = "TEST MESSUNG";
```

4.3.3.1 Prescription

```
1                           List<PMSSOAPService.MeasurementData>  prescMeasData  =  new
2                   List<PMSSOAPService.MeasurementData>();
3                              PMSSOAPService.MeasurementData  prescMeasDataItem  =  new
4                   PMSSOAPService.MeasurementData();
5                                         prescMeasDataItem.type   =   "Prescription";
6                   prescMeasDataItem.version = "1.1.7";
```

Line 5 and 6 define the type of measurement (`"Prescription"`) and used RD.Connectivity schema version (`"1.1.7"`), respectively. Both properties are mandatory.

4.3.3.1.1 Sphere (Mandatory)

```
7                   RD.Connectivity.Prescription presc = new Prescription();
```

Line 8 defines the measurements with respect to patient´s eye side. Hence, the array can be initialized according to the measurements available with 1 (OS or OS) item or 2 items (OS and OD). There are no restrictions which side to begin with.

```
8                   presc.eye = new PrescriptionSide[2];
9                   presc.eye[0] = new PrescriptionSide();
```

Line 10 defines the side of the measurements defined in the following properties and objects. Prescription object requires only the property sphere being defined. All following other properties are optional.

```
10                  presc.eye[0].side = Side.Left;
11                  presc.eye[0].sphere = 3f;
```

4.3.3.1.2 Prism (Optional)

Prism values are defined by prism base and prism power as float values.

```
12                  presc.eye[0].prism = new Prism();
13                  presc.eye[0].prism.@base = 45f;
14                  presc.eye[0].prism.power = 1f;
```

4.3.3.1.3 Cyliner (Optional)

```
15                  presc.eye[0].cylinder = new Cylinder();
16                                           presc.eye[0].cylinder.axis   =   10f;
17                  presc.eye[0].cylinder.power = 1f;
```

4.3.3.1.4 Addition (Optional)

```
18                  presc.eye[0].addition = new Addition[3];
19                  presc.eye[0].addition[0] = new Addition();
20                  presc.eye[0].addition[0].type = AdditionType.Near;
21                  presc.eye[0].addition[0].power = 2.00f;
22                  presc.eye[0].addition[0].viewingDistance = 25f;
23                  presc.eye[0].addition[1] = new Addition();
24                  presc.eye[0].addition[1].type = AdditionType.Intermediate;
25                  presc.eye[0].addition[1].power = 2.00f;
26                  presc.eye[0].addition[1].viewingDistance = 50;
27                  presc.eye[0].addition[2] = new Addition();
28                  presc.eye[0].addition[2].type = AdditionType.Other;
29                  presc.eye[0].addition[2].power = 3.00f;
30                  presc.eye[0].addition[2].viewingDistance = 75;
```

4.3.3.1.5 Back Vertex Distance (Optional)

```
31              presc.eye[0].backVertexDistance = 12.0f;
32              presc.eye[0].backVertexDistanceSpecified = true;
```

4.3.3.2 Objective Refraction

```
1                      List<PMSSOAPService.MeasurementData> objRefracMeasData = new
2               List<PMSSOAPService.MeasurementData>();
3                        PMSSOAPService.MeasurementData objRefracMeasDataItem = new
4               PMSSOAPService.MeasurementData();
5               objRefracMeasDataItem.type = "ObjectiveRefraction";
6               objRefracMeasDataItem.version = "1.1.7";
```

Line 6 and 7 define the type of measurement (`"ObjectiveRefraction"`) and used RD.Connectivity schema version (`"1.1.7"`), respectively. Both properties are mandatory.

```
7               ObjectiveRefractions objRefractions = new ObjectiveRefractions();
8                                    objRefractions.refraction   =     new
9               RD.Connectivity.ObjectiveRefraction[1]; // max 4 allowed
10              objRefractions.refraction[0] = new ObjectiveRefraction();
```

The object ObjectiveRefraction is wrapped by an object ObjectiveRefraction<u>s</u>. Hence, line 7 and 8 initialize the object ObjetiveRefractions and within that object the object ObjectiveRefraction is given as an array of ObjectiveRefraction objects.
The refraction type of measurement is optional but can be set in case a different distance is assumed.

```
11              objRefractions.refraction[0].type = RefractionType.Far;
```

4.3.3.2.1 Sphere (Mandatory)

Line 11 defines the measurements with respect to patients side. Hence, the array can be initializes according to the measurements available with 1 (OS or OS) item or 2 items (OS/OD). There are no restrictions with which side to begin with.

```
12              objRefractions.refraction[0].eye = new ObjectiveRefractionSide[2];
13                                 objRefractions.refraction[0].eye[0]  =  new
ObjectiveRefractionSide();
14              objRefractions.refraction[0].eye[0].side = Side.Left;
15              objRefractions.refraction[0].eye[0].sphere = 1.0f;
```

4.3.3.2.2 Cylinder (Optional)

```
16              objRefractions.refraction[0].eye[0].cylinder = new Cylinder();
17              objRefractions.refraction[0].eye[0].cylinder.power = 1.0f;
18              objRefractions.refraction[0].eye[0].cylinder.axis = 123f;
```

4.3.3.2.3 Back Vertex Distance (Optional)

```
19             objRefractions.refraction[0].eye[0].backVertexDistance = 11f;
20                objRefractions.refraction[0].eye[0].backVertexDistanceSpecified  =
true;
```

4.3.3.2.4 Pupil Size (Optional)

```
21              objRefractions.refraction[0].eye[0].pupilSize = 12.0f;
22              objRefractions.refraction[0].eye[0].pupilSizeSpecified = true;
```

4.3.3.2.5 Corneal Size (Optional)

```
23              objRefractions.refraction[0].eye[0].cornealSize = 10.0f;
24              objRefractions.refraction[0].eye[0].cornealSizeSpecified = true;
```

4.3.3.3 Subjective Refraction

Line 5 and 6 define the type of measurement (`"SubjectiveRefraction"`) and used RD.Connectivity schema version (`"1.1.7"`), respectively. Both properties are mandatory.

```
1              List<PMSSOAPService.MeasurementData> subRefracMeasData = new
2       List<PMSSOAPService.MeasurementData>();
3                PMSSOAPService.MeasurementData subRefracMeasDataItem = new
4       PMSSOAPService.MeasurementData();
5       subRefracMeasDataItem.type = "SubjectiveRefraction";
6       subRefracMeasDataItem.version = "1.1.7";
7       SubjectiveRefractions subRefractions = new SubjectiveRefractions();
8                                 subRefractions.refraction   =    new
9       RD.Connectivity.SubjectiveRefraction[2]; // max 4 allowed
10      subRefractions.refraction[0] = new SubjectiveRefraction();
```

Line 9 indicates the number of refractive measurements; in this example two measurements are created. With respect to the measurement type 4 refractive measurements are supported:
Far
Intermediate
Near and
Other.

In case more than one refractive measurement is defined every additional refractive measurement must be specified by this property.

4.3.3.3.1 Pupillary Distance (Optional)

The pupillary distance is defined within the property monocularPupilDistance for each eye side.

```
subRefractions.refraction[0].eye[0].monocularPupilDistance = 65f;
subRefractions.refraction[0].eye[0].monocularPupilDistanceSpecified = true;
```

It is recommended to set the property monocularPupilDistanceSpecified to true.

4.3.3.3.2 Subjective Refraction Combined

The Object Subjective Refraction Combined holds all information required to set a refraction.

4.3.3.3.2.1 Cylinder (Mandatory)

```
subRefractionCombinedLeft.cylinder = new Cylinder();
subRefractionCombinedLeft.cylinder.axis = 130.0f;
subRefractionCombinedLeft.cylinder.power = -2.0f;
```

4.3.3.3.2.2 Prism (Optional)

```
subRefractionCombinedLeft.prism = new Prism();
subRefractionCombinedLeft.prism.@base = 45f;
subRefractionCombinedLeft.prism.power = 1.0f;
```

4.3.3.3.2.3 Addition (Optional)

To set an addition it is mandatory to define the refraction type:

- Near
- Intermediate
- Far and
- Other

```
subRefractions.refraction[1] = new SubjectiveRefraction();
subRefractions.refraction[1].type = RefractionType.Near;
subRefractions.refraction[1].eye = new SubjectiveRefractionSide[2];
subRefractions.refraction[1].eye[0] = new SubjectiveRefractionSide();
SubjectiveRefractionRelative subRefRelLeft = new SubjectiveRefractionRelative();
subRefRelLeft.addition = 1.25f;
```

4.3.3.4 Serialize Measurements

Measurements created in chapters 4.3.3.1, 4.3.3.2, and 4.3.3.3 need to be serialized to xml before being send to the device. Following method allows the serialization to an xml string, which is then added to the data property of the measurement object.

```
public string Serialize<T>(T value)
{
    if (value == null)
    {
        return null;
    }
    XmlSerializer serializer = new XmlSerializer(typeof(T));
    XmlWriterSettings settings = new XmlWriterSettings();
    settings.Encoding = new UTF8Encoding(false); ;
    settings.Indent = false;
    settings.OmitXmlDeclaration = false;
    settings.CloseOutput = false;
    using (StringWriter textWriter = new StringWriterWithEncoding(Encoding.UTF8))
    {
        using (XmlWriter xmlWriter = XmlWriter.Create(textWriter, settings))
        {
            serializer.Serialize(xmlWriter, value);
        }
        return textWriter.ToString();
    }
}
```

Following example states the serialization call and mapping of the created prescription measurement (ref. to. 4.3.3.1).

```
prescMeasDataItem.data = soapHelper.Serialize<Prescription>(presc);
```

```
prescMeasData.Add(prescMeasDataItem);
measurement.data = prescMeasData.ToArray();
 measReq.measurement = measurement;
```

To ease the process of array initialization and management lists are used, which are then transformed to arrays via .NET .ToArray() method.

As soon as the request is complete the service method SetMeasurement() can be called as follows:

```
PMSSOAPService.Identifier measID = SOAPclient.SetMeasurement(measurement);
```

4.4 Retrieve a Measurement Result

The method GetPatientList returns a list of patients for which measurements are available. Using a do-while loop allows to wait for the measurements available.

```
1                                    PMSSOAPService.GetPatientList   patListreq    =    new
            PMSSOAPService.GetPatientList();
2           PMSSOAPService.PatientList patList = null;
            do
            {
3               patList = soapClient.GetPatientList(patListreq);
4               Thread.Sleep(1000);
5            } while (patList.items.Count() == 0);
```

Line 1 creates an empty GetPatientListRequest. Any filter options are not supported. Therefore, all patients available with a valid measurements are requested.

The returned list contains a PatientInfo object item, which also holds the patient ID required to retrieve the measurement. The request for obtaining the corresponding measurements is created as follows.

```
1                               PMSSOAPService.GetMeasurementList   measListReq   =   new
2            PMSSOAPService.GetMeasurementList();
3            measListReq.patientId = new PMSSOAPService.Identifier();
4            measListReq.patientId.Value = patientInfo.patient.id[0].Value;
5            measListReq.patientId.issuer = patientInfo.patient.id[0].issuer;
6            measListReq.startIndex = 0;
7            measListReq.maximumNumber = 100;
```

Where line 4 and 5 indicate the patient for whom the measurements are queried.

```
PMSSOAPService.MeasurementList measList = soapClient.GetMeasurementList(measListReq);
```

In case several measurements are available the obtained MeasurementList object contains these measurements in an array. Therefore, a cycle through all measurements is required.

```
1                getMeasReq = new GetMeasurementRequest();

2                List<string> measurementSentences = new List<string>();
3                foreach (PMSSOAPService.MeasurementInfo measInfoItem in measList.items)
                 {
4                                               PMSSOAPService.GetMeasurement   measReq   =
5                           getMeasReq.createMeasurementRequest(measInfoItem.id[0].Value,
                    measInfoItem.id[0].issuer);
6                                              PMSSOAPService.Measurement   measurement   =
                    soapClient.GetMeasurement(measReq)
                 }
```

Received measurements can be de-serialized into the corresponding measurement object type by the following method: First the received measurement is mapped to its object type that is known from the measurement type property. Secondly, the received xml string (buffer) is send to the BytesToObject method, which de-serializes the xml stream into the corresponding measurement object.

```
1           public  object  deserializeMeasurement2Object(string  _measurmentType,  string
        _measurementData)
        {
2           AppSupport appSupport = new AppSupport();
3           byte[] Buffer = appSupport.StringToBytesUTF8(_measurementData);
4           if (_measurmentType.Equals("ObjectiveRefraction"))
            {
                // de-serialize measurement result
5                                                   ObjectiveRefractions    measurement    =
                appSupport.BytesToObject<ObjectiveRefractions>(Buffer);
6               return measurement as ObjectiveRefractions;
            }
7           if (_measurmentType.Equals("SubjectiveRefraction"))
            {
              // de-serialize measurement result
8                                                  SubjectiveRefractions    measurement    =
              appSupport.BytesToObject<SubjectiveRefractions>(Buffer);
9             return measurement as SubjectiveRefractions;
            }
10          if (_measurmentType.Equals("Prescription"))
            {
                // de-serialize measurement result
11                                                       Prescription     measurement     =
                appSupport.BytesToObject<Prescription>(Buffer);
12              return measurement as Prescription;
            }
            return null;
        }
```

Following method maps the received xml stream to the corresponding object properties.

```
13        public T BytesToObject<T>(byte[] value) where T : class
          {
14            if (value == null)
                return null;
15            XmlReaderSettings Settings = new XmlReaderSettings();
16            Settings.IgnoreComments = true;
17            Settings.IgnoreProcessingInstructions = true;
18            Settings.IgnoreWhitespace = true;
19            Settings.CloseInput = false;
20            T Result;
21            using (MemoryStream MemStream = new MemoryStream(value))
              {
22                using (XmlReader Reader = XmlReader.Create(MemStream, Settings))
                  {
23                    XmlSerializer Serializer = new XmlSerializer(typeof(T));
24                    Result = Serializer.Deserialize(Reader) as T;
                  }
              }
25            return Result;
          }
```

The return value (line 25) contains the object with all properties being de-serialized from the xml stream.

Appendix

Appendix 1: Feature / Subfeature Support List VISUSCREEN 500

Following example code describes how feature and corresponding subfeature support can be queried from the device. It is recommended to use this function checking, whether this feature is supported, prior to calling the method and communicating with the device.

```
PMSSOAPService.IsSupported isSupportedReq = new
PMSSOAPService.IsSupported();
isSupportedReq.feature = "SetPatient";
isSupportedReq.subFeature = "ReducedDateOfBirth";
bool isSupported = SOAPclient.IsSupported(isSupportedReq);
```

Device / Application	Feature	Subfeature	Supported
VISUSCREEN 500	GetPatientList		X
		PatientFilter	
		ActivePatients	
		MarkedPatients	
		IssuerFilter	
		MeasurementFilter	
		ConsultationFilter	
		Sort	
	GetPatient		X
	SetPatient		X
		ReducedDateOfBirth	X
		AppointedTime	X
	DeletePatient		
	AssociatePatient		
	GetMeasurementList		X
		MeasurementFilter	
	GetMeasurement		X
	SetMeasurement		X
		Anonymous	
	GetConsultationList		
	GetConsultation		
	GetDeviceInfoList		X
	GetSupportedList		X
	IsSupported		X

Appendix 2: Feature / Subfeature Support List VISUCONNECT 500

Device / Application	Feature	Subfeature	Supported
VISUCONNECT 500	GetPatientList		X
		PatientFilter	
		ActivePatients	
		MarkedPatients	
		IssuerFilter	
		MeasurementFilter	
		ConsultationFilter	
		Sort	
	GetPatient		X
	SetPatient		X
		ReducedDateOfBirth	X
		AppointedTime	X
	DeletePatient		
	AssociatePatient		
	GetMeasurementList		X
		MeasurementFilter	
	GetMeasurement		X
	SetMeasurement		(X)[1]
		Anonymous	
	GetConsultationList		
	GetConsultation		
	GetDeviceInfoList		X
	GetSupportedList		X
	IsSupported		X

[1] VISUCONNECT 500 PMS interface supports the feature set measurement. However, the product is not designed to manage any external data sets.

Carl Zeiss Meditec AG
Goeschwitzer Str. 51-52
07745 Jena
Germany

Phone:	+49 (0) 3641 220 333
Fax:	+49 (0) 3641 220 112
Email:	info.meditec@zeiss.com
Internet:	www.zeiss.com/med

ZEISS PMS Web service Integration Guide
Version 1.1.7
Specifications subject to changes