CPR BROKER

Developer manual

MAGENTA aps

TABLE OF CONTENTS

1 INTRODUCTION

This document will describe how to build applications based on CPR Broker. This will include client applications and also how to extend the broker to include other data sources.

This document is organized as a how to guide. The code will assume using Visual Studio .NET, but the concepts can be generalized to use other tools or platforms for building client applications.

2 BUILDING CLIENT APPLICATIONS

2.1 Concepts and facts

- Communication with the broker is done through SOAP 1.2 web services
- To be able to use the system, you need a valid application token
- All responses contain an object of StandardReturType that includes a status code and text.

2.2 First steps

2.2.1 Add references

You need to add web references / service references that point to the broker web services.

The following table describes the needed references

2.2.2 Request and approve application token

In order to call CPR Broker web services, you must use an approved application token. An application token is simply a string that identifies the client application that is calling CPR broker. You can do that using the user interface or through web service SOAP calls. This section describes the latter method. For details on how to do it through the user interface, please refer to CPR Broker installation guide.

For all web service calls to CPR broker, you need to fill the application token and the user token.

Example:

```
PartService.ApplicationHeaderValue = new Admin.ApplicationHeader() { ApplicationToken = "[token of approved application]", UserToken = "[Any string]" };
```

2.2.2.1 Request application token

```
string newAppName = "[Application name]";

var newApplicationResult = AdminService.RequestAppRegistration(newAppName);

var newApplication = newApplicationResult.Item;
```

2.2.2.2 Approve application token

```
AdminService.ApplicationHeaderValue = new Admin.ApplicationHeader() { ApplicationToken = "07059250-E448-4040-B695-9C03F9E59E38", UserToken = "[Any string]" };

var result = AdminService.ApproveAppRegistration(TestData.AppToken);
```

```
AdminService.ApplicationHeaderValue.ApplicationToken = newApplication.Token;
```

Now you should store the application token somewhere and use it from now on.

```
PartService.ApplicationHeaderValue = new Part.ApplicationHeader() { ApplicationToken =
myApplicationToken, UserToken = "[Any string]" };

SubscriptionsService.ApplicationHeaderValue = new Subscriptions.ApplicationHeader()
{ ApplicationToken = myApplicationToken, UserToken = "[Any string]" };
```

2.2.3 Passing credentials

If the CPR broker administrator has decided to use Windows authentication instead of Anonymous authentication, you would probably also need to pass your current credentials to the service. You will need something like this:

```
PartService.Credentials = System.Net.CredentialCache.DefaultCredentials;
```

2.3 Reading person data

2.3.1 General notes on methods related to historical data

The methods ReadSnapshot, ReadPeriod, ListSnapshot and ListPeriod all rely on the broker containing historical data.

To use these methods it is a requirement for the records of the broker be subscribed to historical data extracts from the CPR office.

The methods will work on previously build up historical data in the broker, but inconsistency may appear. Corrections (a record field being corrected, caused by incorrect data) will, for instance, not be included and thereby the produced results may potentially be misleading.

2.3.2 Data source selection

Some of the methods allow the calling client to select how to (not)use local data in CPR Broker. This is achieved through a SOAP header called 'sourceUsageOrderHeader'. This header contains a single element (SourceUsageOrder) that can take values as the following table:

Value	Behaviour
LocalThenExternal	CPR broker first looks for data in its local database. If data is not found, it will start looking in external data providers. This is the default.
LocalOnly	Data is looked up only in local database. No attempts are made in external providers even if data is not found locally.
ExternalOnly	Local database is ignored and the broker goes directly to external providers.

2.3.2.1 Notes

- Usage of this parameter is limited to Read() and List() operations.
- If an external provider is used to get the result (in case of ExternalOnly or LocalThenExternal), the local database is updated with new data (if needed).

2.3.3 GetUuid & Read

```
var uuidResult = PartService.GetUuid(cprNumber);

Part.LaesInputType input = new Part.LaesInputType() { UUID = uuidResult.UUID };

// Optional
PartService.SourceUsageOrderHeaderValue = new SourceUsageOrderHeader() { SourceUsageOrder = SourceUsageOrder.LocalThenExternal };

LaesOutputType readResult = PartService.Read(input);

var person = readResult.LaesResultat.Item as RegistreringType1

var personName = reg.AttributListe.Egenskab[0].NavnStruktur.PersonNameStructure;

Console.WriteLine(string.Format("{0} {1} {2}", personName.PersonGivenName, personName.PersonMiddleName,  personName.PersonSurnameName);
```

2.3.4 RefreshRead

This method has exactly the same signature as Read, except that it will only get data from external data providers (DPR or KMD). It will not use the local database, but will update it if necessary.

2.3.5 ReadSnapshot

```
var uuid = PartService.GetUuid([cpr-nummer]);

Part.LaesOejebliksbilledeInputType input = new Part.LaesOejebliksbilledeInputType() {
        UUID = uuid.UUID,
        VirkningDato = [dato],
};

LaesOutputType person = PartService.ReadSnapshot(input);
```

2.3.6 ReadPeriod

```
var uuid = PartService.GetUuid([cpr-nummer]);

Part.LaesPeriodInputType test = new Part.LaesPeriodInputType() {
        UUID = uuid.UUID,
        VirkningFraDato = [fradato],
        VirkningTilDato = [tildato],
```

```
};

LaesOutputType person = PartService.ReadSnapshot(input);
```

2.3.7 Calling List

List method can be used to get many persons in one request

```
Part.ListInputType input = new Part.ListInputType()

{ UUID = new string[]{ “[uuid 1”, “uuid 2”, .....} };

// Optional
PartService.SourceUsageOrderHeaderValue = new SourceUsageOrderHeader() { SourceUsageOrder
= SourceUsageOrder.LocalThenExternal };

var listResult =PartService.List(input);

var persons = listResult .LaesResultat;
```

2.3.8 ListSnapshot

```
Part.ListOejebliksbilledeInputType listInput = new Part.ListOejebliksbilledeInputType() {
        UUID = [cpr1, cpr2, cpr3, cpr4, cpr5, cpr6],
        VirkningDato = [dato],
};

Part.ListOutputType1 listResult = partService.ListSnapshot(listInput);
```

2.3.9 ListPeriod

```
Part.ListPeriodInputType periodInput = new ListPeriodInputType() {
        UUID = [cpr1, cpr2, cpr3, cpr4, cpr5, cpr6],
        VirkningFraDato = [fradato],
        VirkningTilDato = [tildato],
};

Part.ListOutputType1 listResult = partService.ListPeriod(periodInput);
```

2.3.10 Searching for people

The broker implements limited search capabilities. A call to Search will search the broker's local database. Search can be made for person name and CPR number.

```
var searchCriteria = new Part.SoegInputType1()
{
   Subscriptions.SoegObjektType SoegObjekt = new Subscriptions.SoegObjektType()
   {
      SoegAttributListe = new Subscriptions.SoegAttributListeType()
```

```
    {
        SoegRegisterOplysning = new Subscriptions.RegisterOplysningType[]
        {
            new Subscriptions.RegisterOplysningType()
            {
                Item = new Subscriptions.CprBorgerType()
                {
                    FolkeregisterAdresse = new Subscriptions.AdresseType()
                    {
                        Item = new Subscriptions.DanskAdresseType()
                        {
                            AddressComplete = new Subscriptions.AddressCompleteType()
                            {
                                AddressAccess = new Subscriptions.AddressAccessType()
                                {
                                    MunicipalityCode = "615"
                                }
                            }
                        }
                    }
                }
            }
        }
    }
};
var result = PartService.Search(searchCriteria);
var personUuids = result.Idliste;
// Call list now to get the actual persons' data
```

2.4 Subscribing to events

2.4.1 Creating general subscriptions

Subscriptions to any changes on persons can be attached via the Subscribe method.

The method takes two parameters (as shown below): a channel and an array of UUIDs.

```
Var uuids = new Guid[]{}; // set to null for all persons
var fileShareChannel = new Subscriptions.FileShareChannelType(){ Path="[Channel folder path]" };
var subscriptionResult = SubscriptionsService.Subscribe(fileShareChannel, uuids);
var subscription = subscriptionResult.Item;
var subscriptionId = subscription.SubscriptionId;
```

2.4.2 Creating specialized subscriptions

There are two specialized subscription methods: SubscribeOnBirthdate and SubscribeOnCriteria.

2.4.2.1 SubscribeOnBirthdate

This method attaches a subscription on persons set to a given age. A notification will be handed over to the client application (via the specified channel).

The method takes four parameters: a channel, the target age, the amount of days in advance the notification should be sent and an array of UUIDs.

```
var fileShareChannel = new Subscriptions.FileShareChannelType(){ Path="[Channel folder path]" };
int birthdatePriordays = 10;
int? birthdateAgeYears = null;
var res = SubscriptionsService.SubscribeOnBirthdate( fileShareChannel, birthdateAgeYears,
birthdatePriorDays, uuids);
var subscriptionId = subscription.SubscriptionId;
```

2.4.2.2 SubscribeOnCriteria

This method attaches a subscription to persons matching a given criterion (f.ex. Municipality code). Subscriptions will then be put on each person meeting the criterion.

The method takes two parameters: a channel and a SoegObjectType describing the criterion.

Note: The action of this method is taking place in the brokers backend service and will (depending on the size of the dataset and machine power) take some minutes to finish (2 min. based on a dataset of 40000 records).

```
var fileShareChannel = new Subscriptions.FileShareChannelType(){ Path="[Channel folder path]" };
var SoegObjekt = new Subscriptions.SoegObjektType()
{
  UUID = "12345678",
  BrugervendtNoegleTekst = "test",
  SoegAttributListe = new Subscriptions.SoegAttributListeType()
  {
    SoegEgenskab = new Subscriptions.SoegEgenskabType[]
    {
      new Subscriptions.SoegEgenskabType()
      {
        AndreAdresser = new Subscriptions.AdresseType(
        {
          Item = new Subscriptions.DanskAdresseType()
          {
            AddressComplete = new Subscriptions.AddressCompleteType()
            {
              AddressAccess = new Subscriptions.AddressAccessType()
              {
                MunicipalityCode = "104"
              }
            }
          }
        }
      }
    }
  }
};
var subscription = SubscriptionsService.SubscribeOnCriteria(fileShareChannel, SoegObjekt);
var subscriptionId = subscription.SubscriptionId;
```

2.4.3 Web service channel

Create a web service that matches the definition at

http://[EventBrokerUrl]/Templates/Notification.wsdl

```
var webServiceChannel = new Subscriptions.WebServiceChannelType() { WebServiceUrl = "http://[web service url]" };
```

2.4.4 Removing subscriptions

```
var res1 = SubscriptionsService.Unsubscribe( new Guid("[SubscriptionId]"));

var res2 = TestRunner.SubscriptionsService.RemoveBirthDateSubscription( new Guid("[SubscriptionId]"));
```

3 IMPLEMENTING NEW DATA SOURCES

The broker does not own data itself. It relies on getting data from other sources and then stores this data into its database for usage in the future.

To implement a new data source, you need to do the steps in the following sections.

3.1 Data provider class

You need to create a class that gets the data provider. To do this, you need it to implement at least 2 interfaces. First is CprBroker.Engine.IExternalDataProvider. The other is the respective interface for the business need. For Example, the KMD data provider is defined as:

```
using CprBroker.Engine;

public partial class KmdDataProvider : IDataProvider, IExternalDataProvider,
IPartReadDataProvider

{

    public bool IsAlive()    { // implementation    }

    public DataProviderConfigPropertyInfo[] ConfigurationKeys

    get    {

        return new DataProviderConfigPropertyInfo[]

        {

            new DataProviderConfigPropertyInfo(){Name="Address",Required=true,
Confidential=false},

            new DataProviderConfigPropertyInfo()
{Name="Username",Required=true,Confidential=false},

            new DataProviderConfigPropertyInfo(){Name="Password"
,Required=true,Confidential=true}

        };

    }

    public RegistreringType1 Read(CprBroker.Schemas.PersonIdentifier uuid, LaesInputType input,
Func<string, Guid> cpr2uuidFunc, out CprBroker.Schemas.QualityLevel? ql)

    {

        .......

    }

}
```

3.2 Register the provider type

Copy the DLL that contains the data provider to the /bin folder in the broker website.

Now in the Web.config file of CPR broker website, open the node configuration/dataProvidersGroup/dataProviders/knownTypes

Add a new 'add' node for the new type

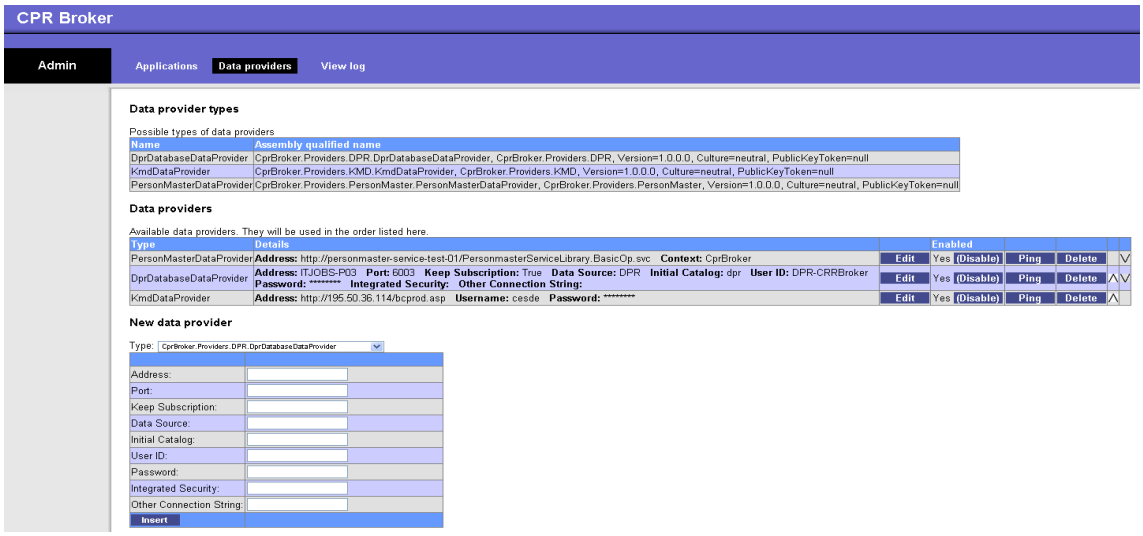
```
<add type="CprBroker.Providers.KMD.KmdDataProvider, CprBroker.Providers.KMD"/>
```

3.3 Add data provider instance

Open http://[Cpr Broker Url]/Pages/DataProviders.aspx

You should see the new type in the table on top.

Now select the type from the drop down on the bottom of the page (Under 'New Data Provider'), fill the parameters and click 'Insert'



This is a basic interface but it does get the job done.

Normally when external applications register themselves with *CPR Broker Service* they get an application token, but they are not allowed to do anything with the service before the application has been *Approved*.

To approve an application, simply click *Edit* for the application in question and check the *Approved* check box. Then click *Update* (only shown after *Edit*).

You can also enter a new application manually. Simply give it a *Name*, a *Token* and whether it should be initially approved (it probably should). Then click *Insert*. The application is now listed under *Applications*.

4 SETTING UP LOGGING

CPR Broker can log to file, Windows Event Log, to the Database and to email.

There place to setup logging: In the *loggingConfiguration.config* file for CPR Broker web service. The default position for this is *C:\Program Files\ITST\CPR Broker(Event Broker)\Web\Config*

Additional location for Event Broker: in *the CprBroker.EventBroker.Backend.exe.config* file for the Backend service. The default position for this is *C:\Program Files\ITST\Event Broker\Web\bin*.

The procedure is the same for both files. Locate the `< loggingConfiguration>` tag in the specific config file. Under the `<listeners>` tag you will find four `<add` tags. The `"CprDatabase"` as well as the `"EventLog"` should be left untouched in all cases.

In `"FlatFile"` you should look for the `fileName` attribute. This should be set to the full path and name of the where to put the log file.

In `name="Email"` there are more settings. The ones most likely to be adjusted are: `toAddress`, `fromAddress`, `smtpServer` and perhaps `smtpPort`.

Please note: In the last 3 cases, you need to make sure that the 'NT AUTHORITY\NETWORK SERVICE' account has sufficient access rights to the destination.

You have now adjusted the settings for each type of logging, but you have yet to set what types of logging are *active*. You now look for the `<specialSources>/ <allEvents>` tag. In this you will another `<listeners>` tag. Per default `"CprDatabase"` is active, which can be seen from the fact that it is not commented out like e.g. `<!--add name="EventLog" /-->` is.

To enable a specific listener simply remove the `<!--` and `-->` characters from the line. And to disable a listener simply put them back in.

MAGENTA aps

adresse
Studiestræde 14, 1.
1455 København K

email
info@magenta-aps.dk

telefon
(+45) 33 36 96 96