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

Reference name	Url
Admin	http://[CprBrokerUrl]/Services/Admin.asmx
Part	http://[CprBrokerUrl]/Services/Part.asmx
Subscriptions	http://[EventBrokerUrl]/Services/Subscriptions.asmx

2.2.2 Request application token

```
string newAppName = "[Application name]";
var newApplicationResult = AdminService.RequestAppRegistration(newAppName);
var newApplication = newApplicationResult.Item;
```

2.2.3 Approve app 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;
Create
PartService.ApplicationHeaderValue = new Part.ApplicationHeader() { ApplicationToken =
```

```
newApplication.Token, UserToken = "[Any string]" };

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

from now on, you should use a the new Token

2.3 Reading person data

2.3.1 GetUuid & Read

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

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

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);
```

2.3.2 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.

2.3.3 Calling List

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

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

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

var listResult =TestRunner.PartService.List(input);

var persons = listResult .LaesResultat;
```

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()
 {
    SoegObjekt = new Part.SoegObjektType()
    {
```

```
        SoegAttributListe = new Part.SoegAttributListeType()
        {
            SoegEgenskab = new Part.SoegEgenskabType[]
            {
                new Part.SoegEgenskabType()
                {
                    NavnStruktur=new Part.NavnStrukturType()
                    {
                        PersonNameStructure=new Part.PersonNameStructureType()
                        {
                            PersonGivenName= "[First name]",
                            PersonMiddleName= "[Middle name]",
                            PersonSurnameName= "[Last Name]",
                        }
                    }
                }
            }
        }
    };
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 subscriptions

```
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;

int birthdatePriordays = 10;

int? birthdateAgeYears = null;

var res = SubscriptionsService.SubscribeOnBirthdate( fileShareChannel, birthdateAgeYears,
birthdatePriorDays, uuids);
```

2.4.2 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]" };
```

2.4.3 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 following steps

3.1 Data provider class

You need to createa 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 repective 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

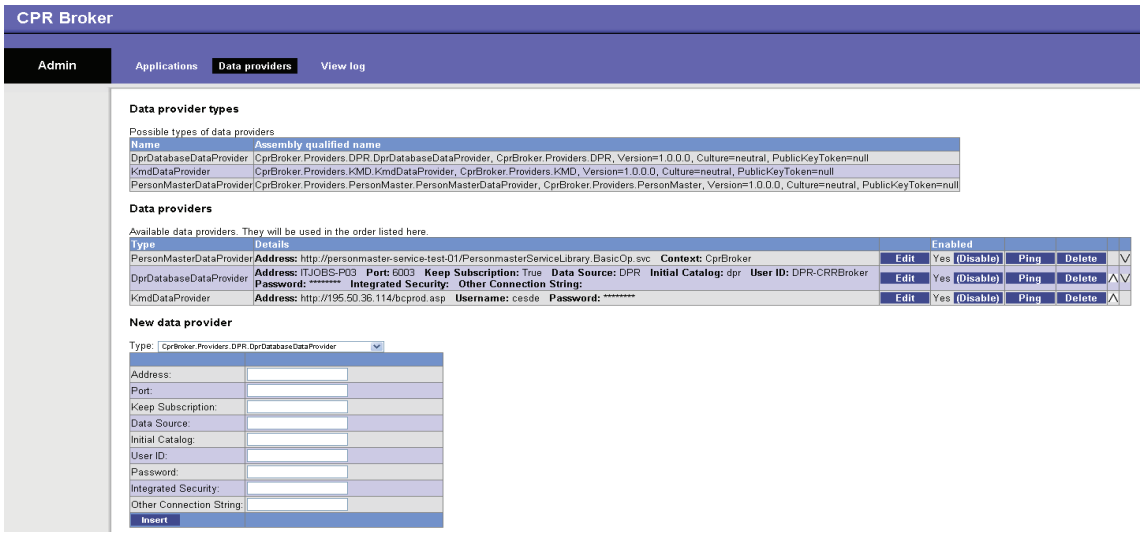
```
<add type="CprBroker.Providers.KMD.KmdDataProvider, CprBroker.Providers.KMD"/>
```

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