Р Е П У Б Л И К А Б Ъ Л Г А Р И Я

ДЪРЖАВНА АГЕНЦИЯ „ЕЛЕКТРОННО УПРАВЛЕНИЕ“

Утвърден със заповед
№ДАЕУ-11218/ 12.05.2020 г.
на Председателя на Държавна агенция
„Електронно управление“

Стандарт за разработване на адаптери

Проект “Разширяване на обхвата на средата за Междурегистров обмен на справочна и удостоверителна информация, гарантиране работоспособността на същата, създаване и актуализиране на справки, адаптери, присъединяване на нови регистри и надграждане на средата (RegiX) с нови функционалности. Услугата се възлага по проект, който се изпълнява по процедура BG05SFOP001-1.004 чрез директно предоставяне на безвъзмездна финансова помощ с наименование „Надграждане на хоризонталните и централни системи на електронното управление“, финансирана по оперативна програма „Добро управление“ 2014-2020 г.

Съдържание

1 Въведение

Целта на този документ е да опише стандарт за разработка на адаптери за достъп до регистри на централната администрация. Документът е отчетен продукт от етап „Детайлизиране“ на Дейност 2 Надграждане на системата за междурегистров обмен (RegiX) от проекта „Разширяване на обхвата на средата за Междурегистров обмен на справочна и удостоверителна информация, гарантиране работоспособността на същата, създаване и актуализиране на справки, адаптери, присъединяване на нови регистри и надграждане на средата (RegiX) с нови функционалности", изпълняван от ТехноЛогика ЕАД по договор с възложител Държавната агенция „Електронно управление“ (ДАЕУ).

В стандарта са включени: конвенция за именуване, структура на адаптери, стандарт за описание на услугите, чрез които си взаимодействат ядрото на системата с адаптерите и адаптерите с бази данни и др.

RegiX адаптерите представляват софтуерни компоненти, унифициращи достъпа до регистри и системи на администрации. Адаптерите имат утвърдена структура и операции, които следват да реализират. Семантиката на достъпа до присъединените регистри (извличане на данни или подаване на данни) се определя от първичния администратор на данните (ПАД) и от консуматора на услугата. В ядрото на RegiX и адаптерите няма заложени ограничения, нито програмна логика, които да налагат такава семантика. Всички операции, които адаптерите изпълняват, се дефинират чрез дискретни набори от данни, които в посока консуматор -> регистър служат за параметри на операцията, а в посока регистър -> консуматор съдържат резултата от операцията. Основната ролята на адаптера е да приеме параметрите на операцията от ядрото, да ги подаде навътре по специфичния за съответния регистър интерфейс, а след като регистърът върне резултат (напр. данни, съдържащи извлечение от вписани в регистъра факти и обстоятелства или потвърждение за успешно приети данни операция/код за грешка, в зависимост от семантиката на операцията), адаптерът трансформира резултата до унифицирания XML формат, налага ограниченията за достъп до данни за конкретния консуматор, подготвя пакета с окончателния резултат и го връща към ядрото на RegiX.

Адаптерите извършват допълнителни действия върху данните, получени като резултат от операция за достъп до присъединен регистър: подписване на резултата и прилагане на ограничения, в случай че консуматорът няма права за всички полета от връщаната информация.

Адаптерите се доставят на две места - в ядрото на RegiX и в специфичен компонент - AdapterHost разположен в инфраструктурата на администрацията, географски разположен максимално близо до системата източник на данните.

2 Особености на съществуващата реализация

Ядрото на RegiX е отговорно за следните основни задачи:

- Автентикация/оторизация на консуматори на операции
- Рутиране на заявките за операции към съответния адаптер

Архитектурата на RegiX изисква ядрото да има информация за адаптерите. Тази информация е необходима за да могат да бъдат налагани ограничения върху данните връщани от операциите (различни консуматори достъпващи една и съща операция могат да получават достъп до различни полета от резултата). Също така позволява изграждането на по-сложни процеси - възможно е при извикване на една операция на ядрото да бъдат извършени обръщения към повече от един адаптер.

Всеки адаптер съдържа два основни компонентна:

- AdapterService - Логиката извършваща същинското обръщение към информационната система на администрацията. Изпълнява се в рамките на адаптер и е част от инфраструктурата на съответната адмиинстрация
- APIService - Извършва обръщение към един или повече адаптери. Изпълнява се в рамките на ядрото на RegiX. Част е от инфраструктурата на ДАЕУ. Предоставя възможност за осъществяването на комплексни извиквания към множество адаптери (в текущите реализации на адаптери всички APIService-и представляват 1:1 съответствие между операция на адаптер и операция на APIService).

Платформата използвана за реализация на RegiX е .NET Framework. Адаптерите, както и ядрото на RegiX използват WCF за реализиране на услуги и по-специфично - SOAP. Всички адаптери са изградени, чрез използването на описаните по-горе средства, съответно операционните системи, на които работят са Windows.

3 Промени в следващата версия на RegiX

За да се позволи инсталацията на адаптери на операционни системи различни от Windows се планира да бъдат направени основни промени в организацията на RegiX. Запазването на съществуващата архитектура на RegiX налага използването на .Net като платформа за разработка. Общата функционалност споделяна между ядрото на RegiX и адаптерите ще бъде отделена в отделни пакети таргетиращи .Net Standard - набор от общи API-та реализирани във всяка .NET имплементация. Нови адаптери ще могат да таргетират .NET Standard или .NET Framework. По този начин ще е възомжна регистрацията на адаптери в ядрото, като същите адаптери ще могат да работят върху операционна система различна от Windows.

4 Протоколи за комуникация между ядрото и адаптери

Протоколът за комуникация между ядрото и адаптерите в случая, когато се използва WCF може да бъде различен - двата компонента споделят общ contract, адресът на който

адаптера е достъпен може да бъде конфигуриран. Последната характеристика изгражда канал за комуникация между WCF базиран клиент и сървър е така наречения Binding. WCF поддържа различни Binding-и. Достатъчно е конфигурацията на Binding-ите от страна на ядрото и тези от страна на адаптера да съвпадат. За по-лесна поддръжка конфигурацията на адаптерите ще е част от така наречените host компоненти. Ще бъдат разработени два типа host компонента - един за WCF базирани адаптери и един за .NET standard адаптери.

.NET standard базираните адаптери ще бъдат host-вани от .NET Core базирано приложение. Протоколът, по който ще бъдат обменяни данни между ядрото и .NET Core базираният host е http (REST архитектура) чрез обмяна на XML като част от HTML request-ите и response-ите.

5 Структура на проекта на адаптер

Проектът за разработка на адаптер се състои най-малко от следните проекти:

- RegiX.**Name**Adapter
- RegiX.**Name**Adapter.Mock
- RegiX.**Name**Adapter.Test

5.1 RegiX.NameAdapter

Представлява проект за реализация на основната функционалност на адаптера, която включва, както следва:

- Логика за връзка със система или база данни на администрация
- Логика за връзка между ядрото и адаптера

5.1.1 AdapterService

В този пакет съдържа интерфейса на услугата за връзка със системата на администрацията, както и нейната реализация. Примерно файлово съдържание на такъв пакет TechnoLogica.RegiX.GraoNBDAdapter.AdapterService включва следните файлове:

- INBDAdapter.cs
- NBDAdapter.cs

5.1.1.1 INameAdapter

Интерфейсът I**Name**Adapter описва операциите, които даден адаптер предоставя. Необходимо е да бъде разширен интерфейсът IAdapterService и да се поставят следните атрибути:

- System.ServiceModel.ServiceContractAttribute
- System.ComponentModel.DescriptionAttribute

Пример за атрибутите и разширяван интерфейс:

```
[ServiceContract]
[Description("ОПИСАНИЕ_НА_АДАПТЕР")]
public interface INameAdapter : IAdapterService
{
  //...
}
```

Как трябва да изглежда дефиницията на една операция е описан в глава **Структура на операция**

Необходимите атрибути, които една операция трябва да има са следните:

- System.ServiceModel.OperationContractAttribute
- System.ComponentModel.DescriptionAttribute

Пример за атрибутите на операция:

```
[OperationContract]
[Description("ОПИСАНИЕ_НА_ОПЕРАЦИЯ")]
public CommonSignedResponse<RequestType, ResponseType> OperationName(
   RequestType argument,
   AccessMatrix accessMatrix,
   AdapterAdditionalParameters aditionalParameters)
```

5.1.1.2 NameAdapter

Класът **Name**Adapter представлява имплементация на интерфейсът I**Name**Adapter. За правилната работа на приложенията извършващи host-ването на адаптерите, както и за работата на ядрото на RegiX е необходимо да бъдат зададени следните атрибути на класа:

System.ComponentModel.Composition.ExportAttribute

TechnoLogica.RegiX.Adapters.Common.ExportExtension.ExportFullNameAttribute

TechnoLogica.RegiX.Adapters.Common.ExportExtension.ExportSimpleNameAttribute

Аргументите, необходими на изброените по-горе атрибути са видими от следния пример:

```
   [Export(typeof(IAdapterService))]
   [ExportFullName(typeof(NameAdapter), typeof(IAdapterService))]
   [ExportSimpleName(typeof(NameAdapter), typeof(IAdapterService))]
   public class NameAdapter :
BaseAdapterService, INameAdapter
```

5.1.2 APIService

В този пакет съдържа интерфейсът на услугата извършваща връзка меджу ядрото и адтера, както и нейната реализация. Примерно файлово съдържание на такъв пакет TechnoLogica.RegiX.GraoNBDAdapter.APIService включва следните файлове:

- INBDAPI.cs
- NBDAPI.cs

Info атрибут

Този атрибут се поставя на операции принадлежащи на интерфейс на адаптер в частта описваща изпълнението в ядрото (APIService операция).

```
[Info(
   requestXSD: "REQUEST_XSD_FILE_NAME.xsd",
   responseXSD: "RESPONSE_XSD_FILE_NAME.xsd",
   commonXSD: "COMMON_XSD_FILE_NAME.xsd",
   requestXSLT: "REQUEST_XSLT_FILE_NAME.xslt",
   responseXSLT: "RESPONSE_XSLT_FILE_NAME.xslt",
   sampleRequest: "SAMPLE_REQUEST.xml",
   sampleResponse: "SAMPLE_RESPONSE.xml",
   metaDataXML: "REQUEST_METADATA.xml"
   )]
```

Декоратори

5.1.3 XMLMetaData/RegiX.NameAdapter

Тази директория трябва да съдържа xml описващ начина за визуализиране на форма за създаване на заявление за операция. За всяка операция е необходимо да съществува файл със същото име и разширение xml.

Незадължителност

Данните, намиращи се в тази директория са незадължителни - те са необходими за работата на клиентските приложения на RegiX и не са приложими в случаите, когато дадена операция или всички операции на даден адаптер ще бъдат консумирани само и единствено от системи без участието на оператори.

5.1.4 XMLSamples/RegiX.NameAdapter

Данните намиращи се в тази директория представляват примери за заявка и отговор за всяка една от операциите. Конвенцита за именоване е посочена в таблицата **конвенции за именоване**. В случай, че имената на примерните данни не спазват конвенцията е възможно те да бъдат посочени чрез използването на Info атрибута описан в секцията **APIService**.

5.1.5 XMLSchemas/RegiX.NameAdapter

Данните намиращи се в тази директория представляват xsd описание на структурата на заявка и отговор за всяка една от операциите. Конвенцита за именоване е посочена в таблицата **конвенции за именоване**. В случай, че имената на xsd схемите не спазват конвенцията е възможно те да бъдат посочени чрез използването на Info атрибута описан в секцията **APIService**.

5.2 RegiX.NameAdapter.Mock

Представлява проект, съдържащ mock функционалност на адаптера. Тази фунцкионалност представлява връщането на предварително дефиниран резултат по очакван вход

5.3 RegiX.NameAdapter.Test

Представлява проект, съдържащ тестовата функционалност на адаптер. Тази фунцкионалност представлява unit тестове, проверяващи пълнотата (по отношение на предоставяните данни и услуги) и коректността на адаптера.

6 Конвенции за именоване

При разработката на адаптерите е добре да се следват установените конвенции за именоване, възприети при използване на C# и .NET платформата. На следния адрес могат да бъдат намерени общи насоки при разработката на библиотеки: Design Guidelines for Developing Class Libraries.

Специфични конвенции използвани при разработката на библиотека на адаптер са описани в таблицата:

Компонент	**Конвенция**	**Пример**
Namespace на адаптер	**DevelopmentOrganization**.RegiX.**Name**Adapter	**TechnoLogica**.RegiX.**GraoNBD**Adapter
Име на основен проект за адаптер	RegiX.**Name**Adapter	RegiX.**GraoNBD**Adapter
Име на mock проект за адаптер	RegiX.**Name**Adapter.Mock	RegiX.**GraoNBD**Adapter.Mock
Име на тестов проект за адаптер	RegiX.**Name**Adapter.Test	RegiX.**GraoNBD**Adapter.Tests
Име на интерфейс на адаптер	I**Name**Adapter	I**NBD**Adapter

Компонент	Конвенция	Пример
Име на реализация на адаптер	**Name**Adapter	**NBD**Adapter
Име на интерфейс на услуга на адаптер в ядрото	I**Name**API	I**NBD**API
Име на реализация на услуга на адаптер в ядрото	**Name**API	**NBD**API
Име на mock на адаптер	**Name**AdapterMock	**NBD**AdapterMock
Име на операция в рамките на адаптер	**OperationName**	ValidPersonSearch
Име на requestXSD файл	**OperationName**Request.xsd	**ValidPersonSearch**Request.xsd
Име на responseXSD файл	**OperationName**Response.xsd	**ValidPersonSearch**Response.xsd
Име на commonXSD файл	**OperationName**Common.xsd	**ValidPersonSearch**Common.xsd
Име на metaDataXML файл	**OperationName**.xml	**ValidPersonSearch**.xml

Компонент	Конвенция	Пример
Име на sampleRequest файл	**OperationName**Request.xml	**ValidPersonSearch**Request.xml
Име на sampleResponse файл	**OperationName**Response.xml	**ValidPersonSearch**Response.xml
Име на requestXSLT файл	**OperationName**Request.xslt	**ValidPersonSearch**Request.xslt
Име на responseXSLT файл	**OperationName**Response.xslt	**ValidPersonSearch**Response.xslt

7 Структура на операция

Операциите на адаптер трябва да следват следната обща структура:

```
public CommonSignedResponse<RequestType, ResponseType> OperationName(
   RequestType argument,
   AccessMatrix accessMatrix,
   AdapterAdditionalParameters aditionalParameters)
{
 //  Обръщение към система на администрация
 // ...
 // Прилагане на матрица за достъп
 // Резлутатът от обръщението е достъпен в променливата response
 return SigningUtils.CreateAndSign(
              argument, // входният параметър на операцията
              response,  // полученият резултат от изпълнение на операцията
              accessMatrix, // Матрица за достъпа, която да бъде приложена към резултата
              aditionalParameters, // Допълнителните параметри получени към заявката
              true // true в случай, че адаптерът поддържа подписване. false в потивен случай
              );
}
```

7.1 Прилагане на матрица за достъп

Всяка операция на адаптер получава като аргумент така наречената матрица за достъп. Това представлява обект описващ до кои полета на резултата даден консуматор има достъп. Преди да бъде върнат окончателния обект полетата, до които консуматор няма достъп трябва да бъдат премахнати. Това става като бъде приложена матрицата за

достъп. За целта се конфигурира специфичен обект наречен Mapper<T>. Разработени са няколко реализации на абстрактния клас Mapper<T>:

- ObjectMapper<S,T>
- DataSetMapper<T>
- XPathMapper<T>
- SelfMapper<S>

Всички реализации на Mapper<T> класа изискват като аргумент AccessMatrix обект. След като бъде конфигуран Mapper обекта се извиква методът:

public abstract void Map(object source, T destination);

Source представлява обектът източник а destination е резултатния обект. След изпълнение на Map методът в destination се получава крайния резултат почистен от стойности в полетата, за които консуматорът изпълняващ операцията няма достъп.

Самият процес по конфигурация на различните типове Mapper обекти е описано в следващите глави.

7.1.1 ObjectMapper

Този тип съответствие се прилага в случаите, когато резутатният тип на обекта е различен от типа, получен при комуникация със системата на администрация. Основните методи, използвани за конфигурация са:

- AddPropertyMap
- AddObjectMap
- AddCollectionMap
- AddFunctionMap

Повече информация за начина на употреба и дефиницията на избоените методи може да бъде открита в тяхната документация

7.1.2 DataSetMapper

Този тип съответствие се прилага в случаите, когато резултатният тип на обекта получен от системата на администрацията е DataSet. Основните методи използвани за конфигурация са:

- AddDataSetObjectInitializer
- AddDataSetMap
- AddDataRowMap
- AddFunctionMap
- AddDataColumnMap

Повече информация за начина на употреба и дефиницията на избоените методи може да бъде открита в тяхната документация

7.1.3 XPathMapper

Този тип съответствие се прилага в случаите, когато резлутатният тип на обекта получен от система на администрацията е XMLDocument. Основните методи използвани за конфигурация са:

- AddCollectionMap
- AddFunctionMap
- AddPropertyMap

Повече информация за начина на употреба и дефиницията на избоените методи може да бъде открита в тяхната документация

7.1.4 SelfMapper

Този тип съответствие се прилага в случаите, когато резултатният тип на обекта получен от системана на администрацията съвпада с типа на обекта, който адаптера връща. Този тип Mapper не се нуждае от конфигурация, тъй като представлява идентичност.

7.2 Полагане на печати

Адаптерите полагат два вида печати:

- Електронен печат;
- Електронен вревеми печат.

Управлението на това дали такива да бъдат поставяни се извършва по два начина.

- Консуматорът може да посочи дали желае такъв;
- Адаптерът може да посочи дали резултатът трябва да бъде подписан (в случай, че адаптерът е посочил, че резултата не трябва да се подписва - полученият аргумент част от контекста на изпълнение не се взима предвид).

8 Зависимости на адаптер

Адаптерите могат да зависят от библиотеки, като например такива за връзка с бази данни, уеб услуги и други външни системи. Реализация на базова функционалност за всички адаптери се съдържа в следните nuget пакети:

- RegiX.Common
- RegiX.Adapters.Common

- RegiX.Adapters.Mocks
- RegiX.Adapters.TestUtils

8.1 RegiX.Common

Съдържа обща функционалност споделяна от ядрото на RegiX и всички адаптери. В този пакет са дефинирани интерфейсите използвин от адаптерите, структурата на данните обменяна между ядро и адаптери и други общи функционалности. Основните интерфейси съдържани в този пакет са:

- IAPIService - Интерфейс на услуга предоставяна от ядрото
- IAdapterService - Интерфейс на услуга предоставяна от адаптер
- ISigner - Интерфейс съдържащ дефиниции на операции за подписване/полагане на електронен времеви печат
- IAdapterClient - Интерфейс описващ функционалност позволяваща извикването услуга имплементираща IAdapterService от услуга имплементираща IAPIService

8.2 RegiX.Adapters.Common

Съдържа обща функционалност споделяна от всички адаптери на RegiX. Дефинира интерфейси необходими за работата на адаптерите. Реализира базова функционалност за услугите на адаптера и в частта обслужваща ядрото. Част от интерфейсите предоставяни от компонента са:

- IParameterStore - Дефинира функционалност за запазване на данни за конфигурационни параметри на адаптер
- IRequestProcessor - Дефинира функционалност за обработка на получена заявка (може да бъде използван за получаване на заявки в различен от XML формат)
- IResponseProcessor - Дефинира функционалност за обработка на резултат на операция, който трябва да бъде предоставен в различен от XML формат
- IPersistanceProvider - Дефинира функционалност за запазване на заявки в случай на асинхронни операции
- IAsynchronousProcessor - Дефинира функционалност за асинхронна обработка на заявки.

8.3 RegiX.Adapters.Mocks

Съдържа функционалност за улесняване разработката на mock-ове на адаптери необходими за инсталация в тестовата среда на RegiX. Имплементира базов клас BaseAdapterServiceProxy<T> позвловящ еднотипното създаване на mock-ове за адаптери.

8.4 RegiX.Adapters.TestUtils

Съдържа базови класове улесняващи тестването на адаптери. Тестовете извършват еднотипни проверки валидиращи коректността и пълнотата на предоставяните от адаптера данни. Базовите класове са:

- AdapterTest<T, I> - Проверява фунцкионалността на адаптер услуга
- APITest<T, I> - Проверява фунцкионалността на адаптер услуга в частта ѝ работеща в адаптера
- MockTest<M, I> - Проверява функционалността на mock адаптер

8.5 Зависимост на проектите от базовите пакети

В следната таблица е описана връзката между всеки от проектите и базовите пакети. RegiX.Common е основният пакет и е dependency съдържано във всеки един от останалите пакети. Поради тази причина той не е част от таблицата.

	RegiX.Adapters. Common	**RegiX.Adapters. Mocks**	**RegiX.Adapters. TestUtils**
RegiX.NameAdapter	**X**		
RegiX.NameAdapter.Mock		**X**	
RegiX.NameAdapter.Test			**X**

В допълнение към горното RegiX.NameAdapter.Test проектът зависи от RegiX.NameAdapter.Mock и RegiX.NameAdapter

9 Избор на платформа

Адаптерите могат да бъдат инсталирани на две платформи - Windows и Linux. Добре е при разработка да се избере като целева платформа .net standard 2.0. В случай, че това не е възможно, платформата, която трябва да се посочи е .net framework **4.7.2**. Платформата, на която ще работи адаптерът е определяща за избора на базовия интерфейс на адаптер, който трябва да бъде имплементиран. Възможностите са:

- TechnoLogica.RegiX.Common.IAdapterServiceWCF
- TechnoLogica.RegiX.Common.IAdapterServiceNETCore

Съответно **IAdapterServiceWCF** се използва в случаите, когато адаптерът ще бъде внедрен на windows, а **IAdapterServiceNETCore**, когато ще се използва Linux. Възможна е и инсталацията върху windows за адаптери използващи **IAdapterServiceNETCore**.

10 Пакетиране на адаптер

След като бъде разработен адаптерът трябва да бъде пакетиран като nuget пакет. За всеки адаптер се създават два отделни nuget пакета - един за основната функционалност на адаптера и един за mock функционалността. Описание на необходимите части от адаптера се описват в .nuspec файлове. За всеки от проектите съдържащ основната функционалност и mock функционалността се съдържа по един такъв файл.

10.1 Основен проект

Съдържание на .nuspec файл описващ компонентите необходими да бъдат включени в nuget пакета.

```
<?xml version="1.0"?>
<package >
  <metadata>
    <id>$id$</id>
    <version>$version$</version>
    <title>$title$</title>
    <authors>$author$</authors>
    <owners>$author$</owners>
    <description>$description$</description>
    <contentFiles>
      <files include="any/any/XMLSchemas/$title$/*.xsd" buildAction="Content" copyToOutput="true"
flatten="false" />
      <files include="any/any/XMLSchemas/$title$/Transformations/*.xslt" buildAction="Content"
copyToOutput="true" flatten="false" />
      <files include="any/any/XMLSamples/$title$/*.xml" buildAction="Content" copyToOutput="true"
flatten="false" />
      <files include="any/any/XMLMetaData/$title$/*.xml" buildAction="Content" copyToOutput="true"
flatten="false" />
    </contentFiles>
  </metadata>
  <files>
    <file src="XMLSchemas\$title$\*.xsd" target="contentFiles\any\any\XMLSchemas\$title$" />
    <file src="XMLSchemas\$title$\Transformations\*.xslt"
target="contentFiles\any\any\XMLSchemas\$title$\Transformations" />
    <file src="XMLSamples\$title$\*.xml" target="contentFiles\any\any\XMLSamples\$title$" />
    <file src="XMLMetaData\$title$\*.xml" target="contentFiles\any\any\XMLMetaData\$title$" />
  </files>
</package>
```

10.2 Mock проект

Съдържание на .nuspec файл за mock функционалността

```
<?xml version="1.0"?>
<package >
  <metadata>
    <id>$id$</id>
    <version>$version$</version>
    <title>$title$</title>
    <authors>$author$</authors>
    <owners>$author$</owners>
    <description>$description$</description>
    <contentFiles>
      <files include="any/any/XMLData/*.xml" buildAction="Content" copyToOutput="true" flatten="false"
/>
    </contentFiles>
  </metadata>
  <files>
    <file src="XMLData\*.xml" target="contentFiles\any\any\XMLData" />
  </files>
</package>
```

11 Присъединяване на адаптер

След като адаптерът е разработен и пакетиран трябва да бъдат извършени следните стъпки за да може новата функционалност доставяна от него да бъде достъпна в средата на RegiX:

- Добяване на адаптера в подходящ host проект
- Добавяне на адаптера в ядрото на RegiX
- Добавяне на адаптера в информационното приложение на RegiX
- Регистриране на адаптера в административното приложение на RegiX

11.1 Добяване на адаптера в host проект

Добавянето на адаптер към host проект се извършва като на избрания тип проект се прави нав branch. Създаденият нов branch се променя като се добави референция към пакета на адаптера. Така конструираният проект е готов за публикуване и внедряване на получения резлутат върху инфраструктурата приготвена за deploy на адаптера. Същесвуват два типа host-ове за адаптери:

- NetCoreAdapterHost - използва се за адаптери, които се инсталират на Linux или Windows платформа
- WCFAdapterHost - използва се за адаптери, които се инсталират на Windows платформа.

11.2 Добавяне на адаптера в ядрото на RegiX

След като бъде разработен нов адаптер той трябва да бъде добавен като референция в ядрото на RegiX. След като това бъде направено се публикува нова версия на ядрото, която е готова за инсталиране в средата на RegiX

11.3 Добавяне на адаптера в информационното приложение на RegiX

След като бъде разработен нов адаптер той трябва да бъде добавен като референция в информационното приолжение на RegiX. След като това бъде направено се публикува нова версия на информационното приолжение, която е готова за инсталиране в средата на RegiX

11.4 Регистриране на адаптера в административното приложение на RegiX

След като е публикуван нов адаптер или нова версия на адаптер и ядрото на RegiX е обновено в административното приложение на RegiX ще има възможност за регистрация на настъпилите промени (добавяне на нов адаптер или актуализация на съществуващ такъв). При регистрацията на нов адаптер се попълват основни данни за адаптера като адрес, където се намира, използван тип binding и съответна негова конфигурация.

12 Описание на услугите за взаимодействие на ядрото на RegiX и адаптерите

RegiX позволява използването на различни протоколи за връзка между ядрото и адаптерите. В зависимост от избраната платформа и съответно използваният основен интерфейс за адаптерите (IAdapterServiceWCF/IAdapterServiceNETCore) протоколите за връзка между ядро и адаптери могат да се различават. Също така използването на различен Binding/BindingConfiguration могат да променят ефективния протокол. Достатъчно е конфигурацията в ядрото за даден адаптер и конфигурацията на самия адаптер да са съвместими (да е конфигурирана поддържка за желания binding) за да може да се осъществявав комуникация между тях.

13 Описание на услугите за взаимодействие на адаптер на RegiX и система на администрация

Архитектурата на RegiX не налага строги изисквания относно връзката между адаптер на RegiX и система на администрация. Съществува широко разнообразие от системи и начин за взаимодействие между тях и адаптерите (директна връзка към база данни,

обръщение към услуги и т.н.). При разработка на адаптери все пак е добре разработчиците да се придържат към следните насоки:

Използване на услуги като интеграционен слой между адаптер и системата на администрация

При невъзможност за създаване на услуги - използване на специално разработени view-та за операциите на RegiX

Целта на изброените по-горе точки е в максимална степен да се изолира вътрешната логика на системата в администрация и при евентуална промяна на тази вътрешна логика това да не се отрази на вече разработени адаптери.