Министерство образования и науки Российской Федерации
(МИНОБРНАУКИ РОССИИ)
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (ТГУ)
Факультет информатики
Кафедра прикладной информатики

УДК 681.03

ДОПУСТИТЬ К ЗАЩИТЕ В ГАК
Зав. кафедрой, проф., д.т.н.
_____ С.П. Сущенко
«___» _____ 2011г.

Бакланов Михаил Александрович

РАЗРАБОТКА АЛГОРИТМА ОПРЕДЕЛЕНИЯ МЕСТОПОЛОЖЕНИЯ ОБЪЕКТОВ В ПРОСТРАНСТВЕ С ПОМОЩЬЮ ТЕХНОЛОГИИ WI-FI

Выпускная квалификационная работа

Научный руководитель,
Ассистент кафедры ПИ П. В. Приступа

Исполнитель,
студ. гр. 1472 М. А. Бакланов

Электронная версия дипломной работы помещена
в электронную библиотеку. Файл
Администратор

Томск – 2011

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ 3
ВВЕДЕНИЕ 5
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6
1.1. Существующие технологии определения местоположения. 6
1.1.1. Специзиалированные системы позиционирования 6
1.1.2 GSM-системы позиционирования 7
1.1.3. Wi-Fi-системы позиционирования 8
2. РАЗРАБОТКА ТЕХНОЛОГИИ ПОЗИЦИОНИРОВАНИЯ 10
2.1. Характеристики разрабатываемой технологии. 10
2.2. Программный интерфейс для работы с Wi-Fi. 10
2.3. Механизм позиционирования. 11
2.3.1. Определение прямоугольной области позиционирования. 12
2.3.2. Позиционирование в заданной области. 12
3. ИССЛЕДОВАНИЕ ПРИЧИН ВОЗНИКНОВЕНИЯ ОШИБОК В ОПРЕДЕЛЕНИИ МЕСТОПОЛОЖЕНИЯ 14
3.1. Предполагаемые причины возникновения ошибок 14
3.2. Предмет и методика исследования. 14
3.3. Показания, полученные в ходе исследования. 15
ЗАКЛЮЧЕНИЕ 16
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 17
ПРИЛОЖЕНИЕ А. 18
ПРИЛОЖЕНИЕ Б 21

РЕФЕРАТ

Дипломная работа 36 страниц, 5 рисунков, 8 источников, 2 приложения.

МЕСТОПОЛОЖЕНИЕ, LBS, WI-FI, СИСТЕМЫ ПОЗИЦИОНИРОВАНИЯ, ГЛОНАСС, GPS, GSM, СИЛА СИГНАЛА, ТРИЛАТЕРАЦИЯ

Объект исследования – алгоритмы определения местоположения.

Цель работы – изучение работы алгоритмов определения местоположения, разработка алгоритма определения местоположения повышенной точности.

Метод исследования – теоретическое исследование, вычислительный эксперимент.

Основные результаты – исследованы алгоритмы определения местоположения, их эффективность и быстродействие, а также разработан алгоритм позволяющий осуществлять позиционирование с точностью не ниже 0,5 метра.

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

SNAP (Subnetwork Access Protocol) - протокол доступа к подсети

GPS (Global Positioning System) – глобальная система навигации и определения положения

GSM (Global System for Mobile Communications) - глобальная система связи с подвижными объектами LBS

CC – сила сигнала

LBS – сервисы основанные на местоположении

PNM (Pedestrian Navigation Module) – система определения местоположения использующая компас и систему гироскопов

ГЛОНАСС – глобальная навигационная спутниковая система

Wi-Fi (англ. *Wireless Fidelity* — «беспроводная точность») — торговая марка Wi-Fi Alliance для беспроводных сетей на базе стандарта IEEE 802.11.

Введение

В настоящее время одним из самых быстрорастущих рынков является рынок LBS-сервисов. Существует множество технологий позволяющих определять местоположение объекта и все они имеют свои характеристики, наиболее важными из которых являются: распространённость технологии, точность, энергопотребление, стоимость и работа внутри помещений. Как правило, технология выбирается, исходя из того, какие характеристики требуются для решения задач, которые ставятся перед системой.

Наиболее распространенными являются системы спутниковой навигации типа GPS и ГЛОНАСС. Они обеспечивают точность позиционирования до 3х метров и работают в любом месте прямой видимости спутников. Недостатками данной технологии являются высокое энергопотребление, невозможность работать внутри помещений и ограниченность точности позиционирования.

Поэтому цель работы это создание программного продукта, позволяющего определять местоположение объектов с точностью до 1 метра с помощью технологии Wi-Fi.

Идея использовать точки доступа Wi-Fi для определения местоположения объектов не нова и широко применяется для самых разных задач, наиболее передовыми в этой области являются сервисы skyhook, wi2geo, Google maps и Яндекс карты. Эти сервисы предоставляют информацию о местоположении объекта в любой точке города с помощью зарегистрированных Wi-Fi сетей, используя метод триангуляции. Однако, на практике максимальная точность, которую может обеспечить данная методика, не превышает 15 метров, при условии крайне высокой концентрации Wi-Fi сетей.

Для решения этой проблемы была разработана альтернативная технология позиционирования, которая основана на определении помещения, в котором находится объект и затем уже определении его позиции в этом помещении и организации соответствующих сервисов для этого помещения.

Возможности применения указанной технологии, за счёт высокой точности позиционирования и мобильности, довольно широки. Эта технология может обеспечить предоставление LBS-услуг внутри помещений, определение местоположения подвижных объектов, навигацию внутри музеев, крупных торговых центров, выставок и т.п.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Существующие технологии определения местоположения.

В то время как с определением местоположения на открытых пространствах успешно справляются спутниковые системы, внутри помещений эта задача до сих пор трудноразрешима. С одной стороны мы попытаемся проанализировать основные технические средства, используемые для определения местоположения, а с другой разберем существующие системы позиционирования, основанные в основе которых лежит Wi-Fi технология.

На настоящий момент различают два вида позиционирования: дискретное и непрерывное. Дискретное определяет местоположение мобильных устройств в конечном множестве позиций. Такие позиции определяются перед использованием системы позиционирования. Например, система RADAR, которая определяет множество позиций в виде множества точек, из которых замерена сила сигнала от передатчиков. Непрерывные системы позиционирования определяет местоположение объекта в несчётном множестве позиций, например в 3х-мерной системе координат. Такое позиционирование осуществляется при помощи математических моделей таких, как SNAP-WPS. Следует отметить, что системы, основанные на наборах данных, не могут быть непрерывными ввиду того, что неограниченное число возможных позиций имеют бесконечный вес, в плане объема занимаемой памяти.

1.1.1.Специзиалированные системы позиционирования

GPS – система позволяющая определять местоположение устройства на поверхности земли или в атмосфере используя информацию со спутников местоположение которых известно. Расстояние вычисляется по времени задержки распространения сигнала от посылки его спутником до приёма антенной GPS-приёмника. То есть, для определения трёхмерных координат GPS-приёмнику нужно знать расстояние до трёх спутников и время GPS системы. Таким образом, для определения координат и высоты приёмника, используются сигналы как минимум с четырёх спутников. Для этого используется метод трилатерации. Точность определения местоположения для гражданских приёмников не превышает 5 метров. Такой точности вполне достаточно для определения местоположения автомобилей движущихся со значительно более существенной скоростью.

PNM (Pedestrian Navigation Module) был разработан компаниями EPFL и Leica Vectronix. Его работа основана на показаниях магнитного компаса, трёхосевого акселерометра и гироскопа. Обычно использование магнитного компаса приводит к накопленной ошибке в определении местоположения. Поэтому в PNM, показания компаса корректируются гироскопом и наоборот, когда у гироскопа сбивается азимут, компас может вернуть его. Акселерометр используется для подсчета шагов человека, использующего устройство. PNM способен определять местоположение с погрешностью не более 5% от пройденной дистанции как снаружи, так и внутри зданий. Погрешность становится весьма существенной на больших дистанциях, но может быть скорректирована системой GPS. Эта система является уменьшенным вариантом навигационных систем применявшихся в авиации.

1.1.2 GSM-системы позиционирования

GSM, на сегодняшний день, это наиболее распространенный стандарт мобильной связи. Этот стандарт дал рождение 3 методам позиционирования: позиционирование по базовой станции, вычисление расстояния по силе сигнала, вычисление расстояния по разнице во времени.

Позиционирование по базовой станции.

Опрос базовых станций позволяет определить, к какой из них в тот или иной момент подключен мобильный телефон. Таким образом, местоположение телефона определяется радиусом охвата базовой станции. Этот метод может давать различную точность от 100м до нескольких километров (в зависимости от радиуса охвата). В некоторых случаях подобной точности достаточно, но её не может быть достаточно для предоставления сервисов основанных на местоположении (LBS).

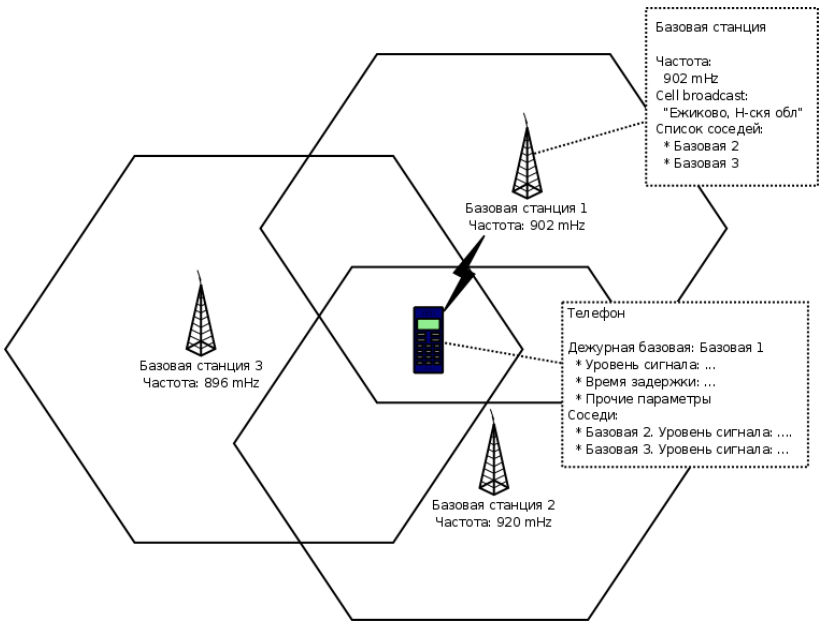


Рисунок 1 – Позиционирование мобильного телефона по силе сигнала

Позиционирование по силе сигнала.

Данный вид позиционирования требует знания, как ослабевает сигнал в зависимости от расстояния между приёмником и передатчиком. Таким образом, расстояние до базовой станции может быть экстраполировано измерением силы сигнала. Далее, зная расстояние до базовых станций, местоположение телефона получается методом трилатерации. Точность составляет от 50 до 500м с очень грубыми ошибками при наличии каких-либо препятствий, например, зданий.

Позиционирование по разнице времени (EOTD).

Данный метод оценивает время, за которое сигнал доходит от мобильного телефона до базовой станции. Эта технология обычно использует пересылку таймингов мобильных телефонов позволяя планировать размещение пакетов во временных слотах. Зная время, которое требуется сигналу, чтобы дойти от мобильного телефона до базовой станции, и скорости радиоволны мы можем вычислить расстояние. А далее, метод трилатерации получить позицию мобильного телефона.

1.1.3. Wi-Fi-системы позиционирования

Методики позиционирования с помощью Wi-Fi могут быть разделены на две основных группы, одна из них основана на СС-картографии, а другая на моделировании распространения радиоволн, которая определяет отношение между силой сигнала и расстоянием. Определение расстояний между известными точками и мобильным устройством позволяет использовать трилатерацию.

СС-картография

С помощью технологии RADAR, мобильное устройство использует СС-карту необходимого пространства. Исходная СС-карта формируется из координат, СС-измерений и расположения мобильного устройства. СС-карта может строиться как на основе вычислений, так и на основе физических измерений. Сила сигнала от каждой точки доступа сравнивается с соответствующими показателями в базе данных, на основании чего предполагается соответствующее местоположение. Transparent location fingerprinting использует карту базисных точек. Базисная точка представляет собой последовательность пар (ss_i, c_i). Где ss_i это множество измерений силы сигнала, а c_i – соответствующие физические координаты. Согласование нового множества измерений силы сигнала ss осуществляется выбором k базисных точек ближайших к полученным измерениям, средний вес c_f которых рассчитывается по формуле:

$$c_f = \frac{\sum_{j=1}^{k} \frac{1}{d(ss_j, ss) + \varepsilon} . c_j}{\sum_{j=1}^{k} \frac{1}{d(ss_j, ss) + \varepsilon}},$$

Где $d(ss_j$, ss) это Евклидово расстояние между двумя тройками точек доступа, а ε –некая константа. Средняя погрешность данного метода составляет 1.78 метра, но максимальная ошибка может достигать 10м.

Вероятностные технологии используют распределение вероятностей силы сигнала для каждой базисной точки с показателями выше среднего. К примеру, система позиционирования Экахау, которая использует две функции оценки для согласования измерений с базой данных. Первая функция вычисляется по методу ядра, а вторая по методу гистограмм.

$$C_{xy}(Si)=\begin{cases}0, & \text{if } r(\theta)+re(\theta)\le d(Si,P)\\ f(\theta)e^{-\lambda q\beta}, & \text{if } r(\theta)-re(\theta)<d(Si,P)<r(\theta)+re(\theta), \text{and } \theta\in[0,2\pi]\\ f(\theta), & \text{if } r(\theta)-re(\theta)\ge d(Si,P), \text{ and } \theta\in[0,2\pi]\end{cases} \quad (3)$$

Система HORUS основана на Байесовской логике. Главной её особенностью является группировка данных, обеспечивающая снижение вычислительной стоимость и времени требуемого для определения местоположения мобильного устройства. Этот подход также является вероятностным. Каждая базисная точка содержит выборку из 240 измерений. Выборки хранятся в виде гистограмм, каждая из которых объединяет все точки доступа в совместное распределение.

Моделирование распространения радиоволн.

Цель подобного моделирования в выражении математического отношения между расстоянием от приёмника до передатчика и силой сигнала. Математическое выражение получается из полиномиальной регрессии третьего порядка. Главное преимущество этой технологии в скорости позиционирования. Однако, для регрессии требуется большое количество точной информации о силе сигнала в течение довольно долгого периода времени. Эта техника обеспечивает точность позиционирования от 1 до 3м. Для определения отношения между дистанцией и силой сигнала требуется несколько десятков измерений. Отсюда следует, что данная модель не является полностью динамической.

2. РАЗРАБОТКА ТЕХНОЛОГИИ ПОЗИЦИОНИРОВАНИЯ

2.1. Характеристики разрабатываемой технологии.

Из анализа, проведенного в предыдущей главе, можно сделать вывод о том, что идея позиционирования с помощью Wi-Fi не нова, и существует множество решений, реализующих данную идею с разной степенью успешности. Основными характеристиками, позволяющими определить эффективность работы технологии, являются средняя точность определения местоположения, максимально возможная ошибка в определении, время определения/обновления координат. Все вышеуказанные технологии и методы работают с Wi-Fi точками доступа, которые установлены случайным образом и основной целью установки которых, является обеспечения доступа в сеть, а не позиционирование объектов. В тоже время, в связи с невысокой стоимостью точек доступа, их преднамеренная установка в помещениях, где необходимо обеспечить точное и быстрое позиционирование, вполне рентабельна. Стоимость оборудования для зала площадью 500м^2 не превысит 3500 рублей. При этом средняя точность позиционирования в этом зале не будет превышать 1м, а скорость позиционирования будет в пределах двух секунд.

2.2. Программный интерфейс для работы с Wi-Fi.

Данный программный интерфейс обеспечивает идентификацию поступающих Wi-Fi сигналов и позволяет получить к службам, предоставляемым соединениями. Содержит в себе классы, позволяющие обрабатывать различного рода события такие, как подключение к новой сети, утрата сети, информация о доступных сетях, получение и отправка данных и т.п.

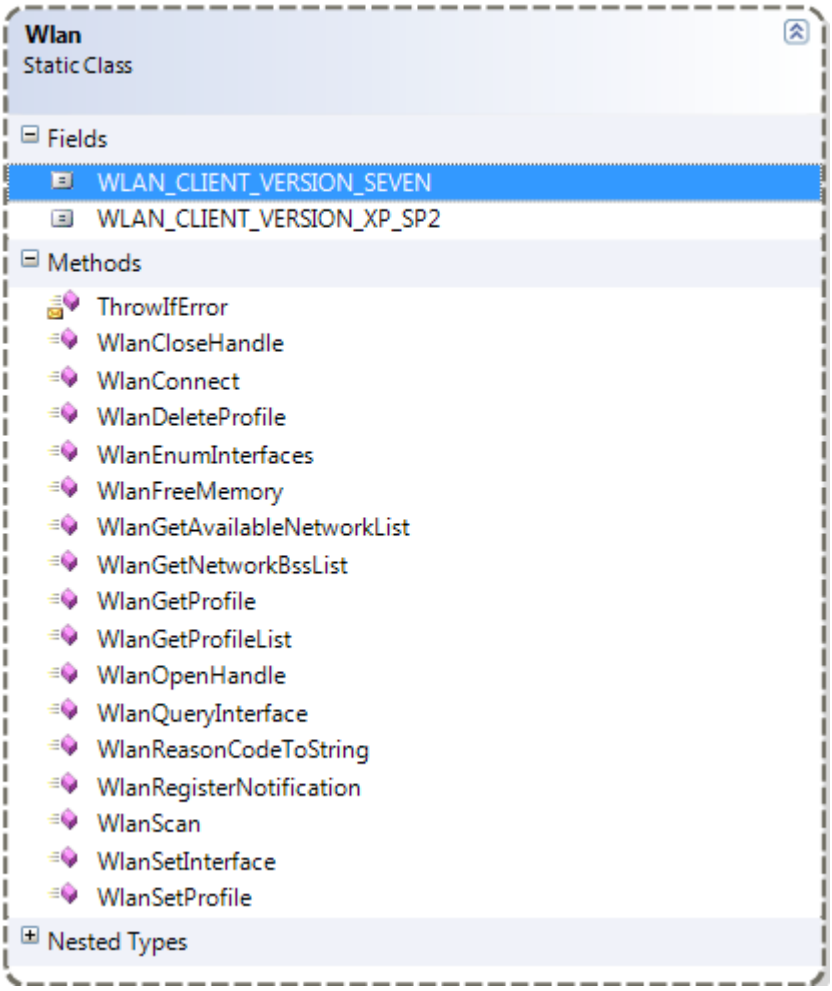


Рисунок 2 – Методы класса Wlan (из API для работы с Wi-Fi)

2.3. Механизм позиционирования.

В связи с тем, что существующие на настоящий момент технологии позиционирования не адаптированы для работы в искусственно создаваемых условиях с высокой точностью, для обеспечения позиционирования с точностью до 1 метра необходима разработка адаптированной методики позиционирования.

Идея этой методики заключается в учёте погрешности, которая получается в результате установления отношения между силой сигнала и расстоянием между приёмником и передатчиком в результате чего, точность определяемого расстояния колеблется в пределах от 1,5 до 0,2 метра в зависимости от расстояния до точки доступа и препятствий. Таким образом, возможное местоположение приёмника представляется в виде кольца для каждой точки доступа. Пересечение всех колец и даёт возможное местоположение. Предполагается, что в каждом помещении будут стоять по 4 точки доступа образующие прямоугольники. Для реализации данного механизма необходимо решить 2 задачи:

1) Определить в каком из прямоугольников находится приёмник

2) Определить местоположение приёмника в заданной области

2.3.1. Определение прямоугольной области позиционирования.

В отличие от метода триангуляции, в котором определяется принадлежность точки треугольнику в данной системе удобней проверять на принадлежность прямоугольнику, ввиду того, что во всех помещениях устанавливается 4 точки доступа.
Область позиционирования определяется динамически путём построения треугольников из точки и стороны прямоугольника, а затем вычисляются углы, и производится проверка каждого угла. Если у всех треугольников прилежащие к сторонам прямоугольника углы не тупые, то точка находится в этом прямоугольнике.

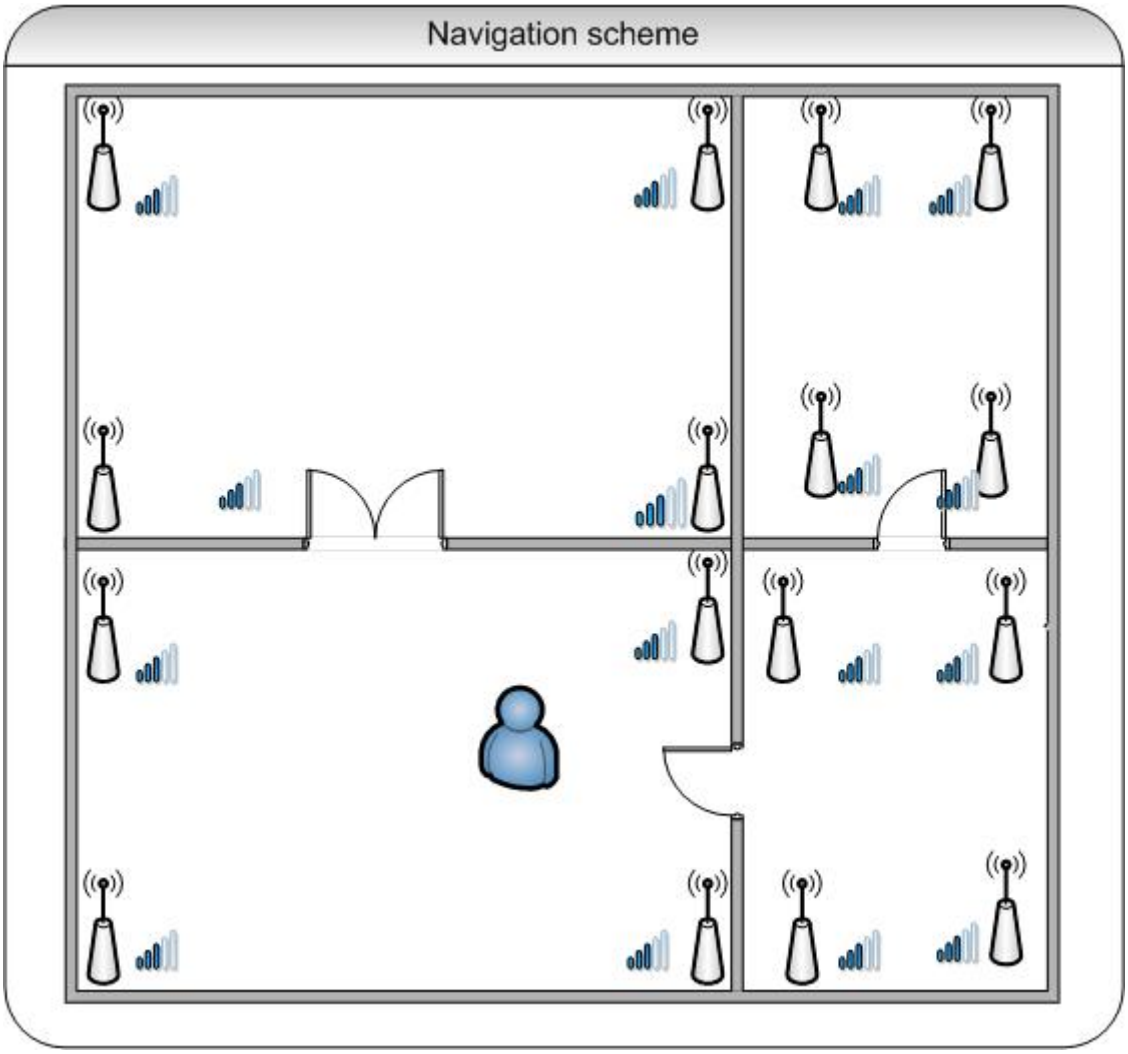


Рисунок 3 – Схема расположения точек доступа в помещении

2.3.2. Позиционирование в заданной области.

После определения области позиционирования необходимо определить конкретное местоположение объекта.

1) Нахождение в заданной области ближайшей к приёмнику точки доступа;
2) Построение участка кольца принадлежащего области позиционирования с шириной обратно пропорциональной расстоянию до точки;
3) Разбиение кольца на множество точек в зависимости от требуемой точности позиционирования;
4) Проверка каждой точки полученного множества на принадлежность трём другим кольцам

3. ИССЛЕДОВАНИЕ ПРИЧИН ВОЗНИКНОВЕНИЯ ОШИБОК В ОПРЕДЕЛЕНИИ МЕСТОПОЛОЖЕНИЯ

3.1. Предполагаемые причины возникновения ошибок

Главной причиной возникновения погрешностей при определении местоположения с помощью Wi-Fi предположительно неоднородность прохождения радиоволн через различные препятствия, что периодически приводит к серьезным ошибкам в определении местоположения.

Цель исследования – проверить, действительно ли препятствия в виде стен и перегородок могут привести к ошибке более чем в 7 метров.

3.2. Предмет и методика исследования.

Предметом исследования выступило изменение силы сигнала от Wi-Fi передатчика в зависимости от расстояния и наличия препятствий.

В качестве методики исследования был избран экспериментальный замер силы сигнала в разных точках помещения с помощью условно-бесплатной программы WirelessMon.

В ходе исследования были произведены замеры в 4 точках без препятствий (расстояние 0 м, 1 м, 5 м и 7 м) и в 2 с наличием препятствий в виде двери из ДСП и бетонной стены (расстояние 7 м и 5 м).

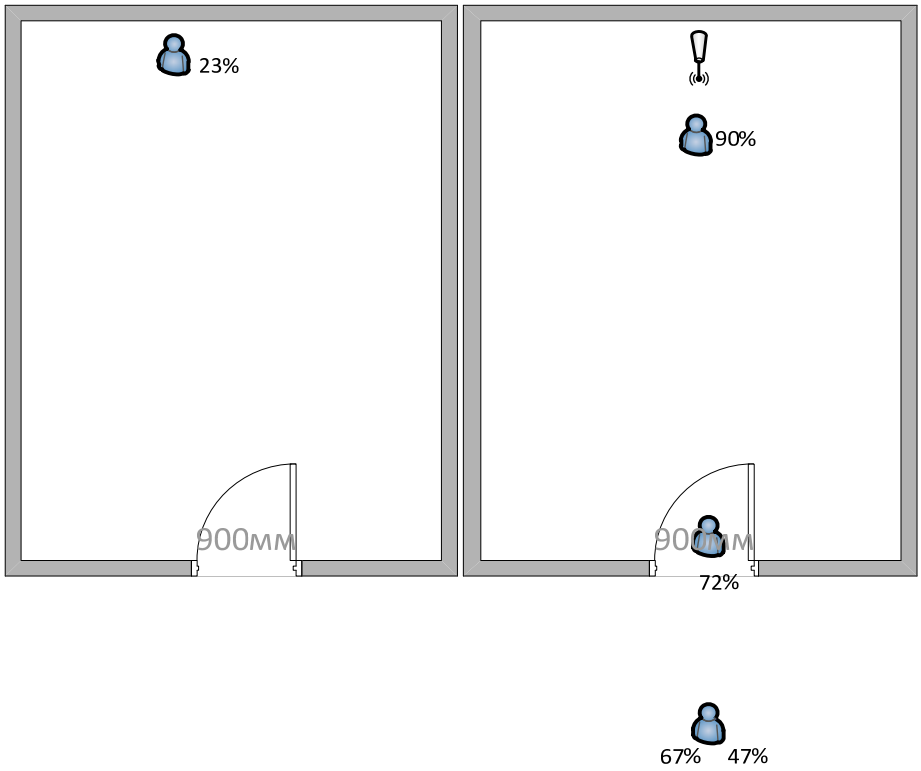


Рисунок 4 – Схема помещений для исследования

3.3. Показания, полученные в ходе исследования.

В ходе замеров силы сигнала были обнаружены следующие особенности:

1) Препятствия оказывают существенное влияние на силу сигнала, влияние зависит от толщины препятствия и его структуры;
2) В разных погодных условиях препятствия показывают разный уровень поглощения волн, амплитуда колебаний достигает 2%;

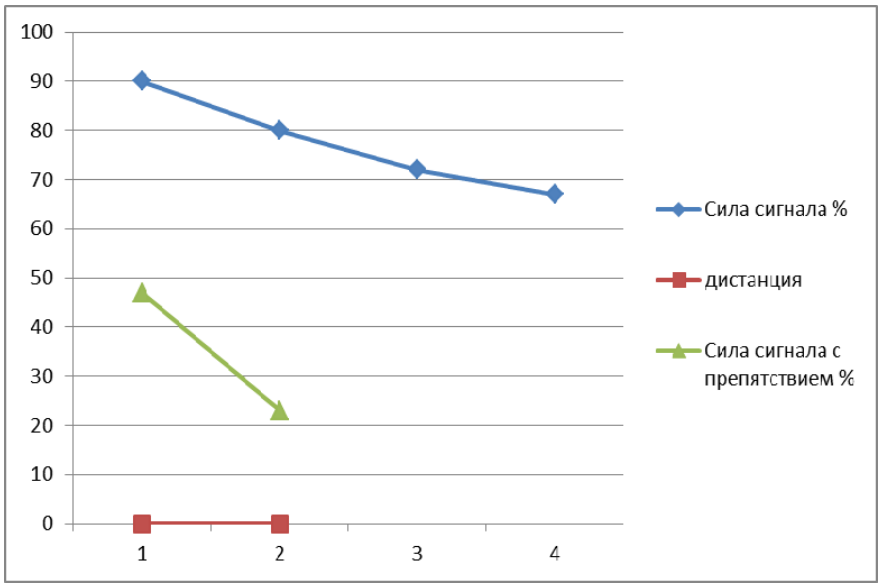


Рисунок 5 – Изменения силы сигнала

ЗАКЛЮЧЕНИЕ

В результате проделанной работы была разработана технология определения местоположения объектов с помощью технологии Wi-Fi с высокой точностью и быстротой позиционирования. Данная технология может использоваться в системах предлагающих LBS-сервисы нового поколения. Одним из вариантов использования данной системы является создание средства навигации для слепых и слабовидящих людей. Другими вариантами могут быть обеспечение навигации на выставках, внутри крупных торговых центров, автоматизация различных рутинных бизнес-процессов требующих информации о точном местоположении.

Данный проект является финалистом и обладателем приза зрительских симпатий регионального конкурса Microsoft Imagine Cup, а так же участником инновационных сессий проектов XIII и XIV Томского инновационного форума, финалистом конкурсов УМНИК 2010, Майкрософт Старт-2010 и i2com - 2011.

Список использованных источников

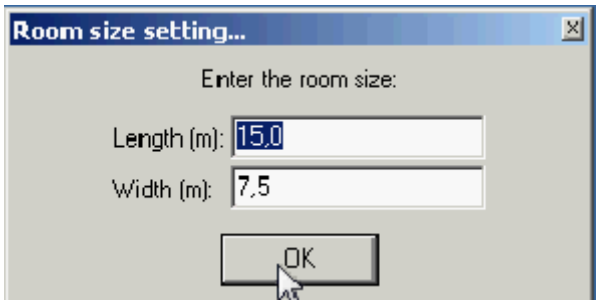
1. Аникин А. Определение местоположения мобильного объекта с помощью приемопередатчиков nanoLOC фирмы Nanotron. // Интеграция. – 2007 - № 9.
2. Anwar A.K. Evaluation of indoor location based on combination of AGPS/HSGPS. / Anwar A.K., Ioannis G., Pavlidou F.N. //Procs of 3rd symp on wireless pervasive computing, pp 383-387.
3. Bahl P., Balachandran A., Padmanabhan V. Enhancements to the radar user location and tracking system. // Technical report. - 2000.
4. Brunatto M. Transparent location fingerprinting for wireless services/ Brunatto M., Kallo C.K. // Technical report, University of Trento. - 2002.
5. Castro P. A probabilistic room location service for wireless networked environments. / Castro P., Chiu P., Kremenek T., Muntz R. // Proceedings of Med-Hoc-Net, Mediterranean workshop on ad-hoc networks, Baia Chia, Cagliari. – 2002.
6. Charlet D. Mobility prediction for multimedia services. / Charlet D., Lassabe F., Canalda P., Chatonnay P., Spies F. // Ibrahim IK, Johannes Kepler University Linz Handbook of research in mobile multimedia, pp491-506. Linz. – 2006
7. Furey E., Curran K., Mc Kevitt P. HABITS A History Aware Based Wi-Fi Indoor Tracking System / Furey E., Curran K., Mc Kevitt P. HABITS // University of Ulster. – 2010
8. Tauber J.A. Indoor location systems for pervasive computing. Area exam report. Massachusetts Institute of Technology. – 2007

ПРИЛОЖЕНИЕ А
РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

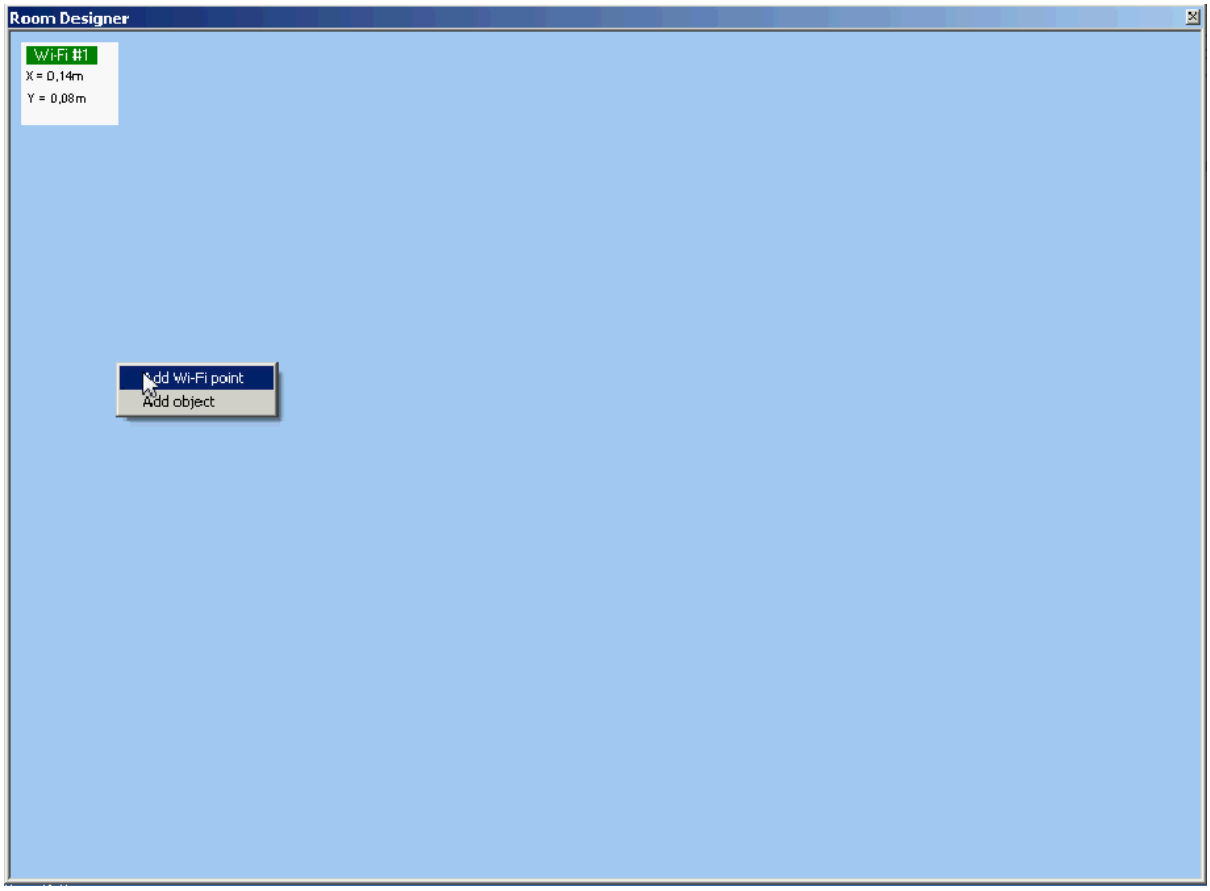
Система содержит конструктор комнат, который позволяет быстро изменять содержимое комнаты и обеспечивать LBS-сервисы в реальном времени.

Рассмотрим работу конструктора на примере наполнения комнаты картинной галереи экспонатами.

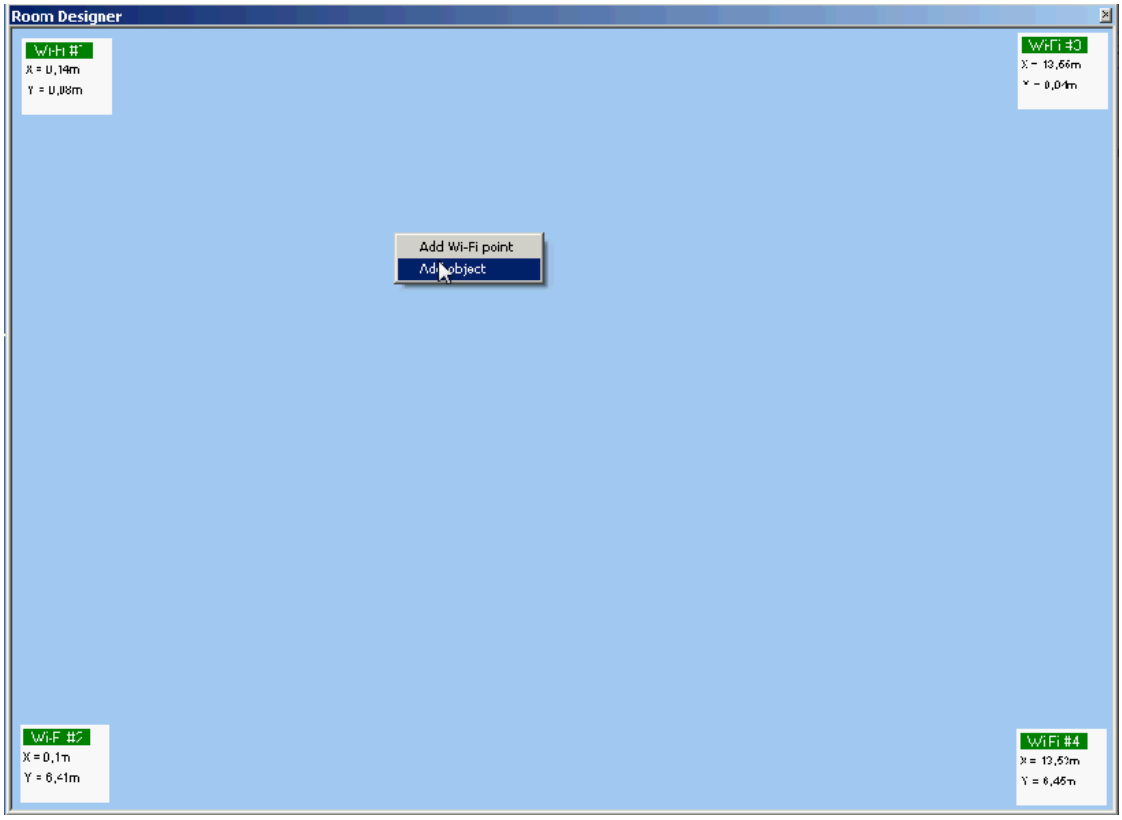
1) Указываем размеры комнаты



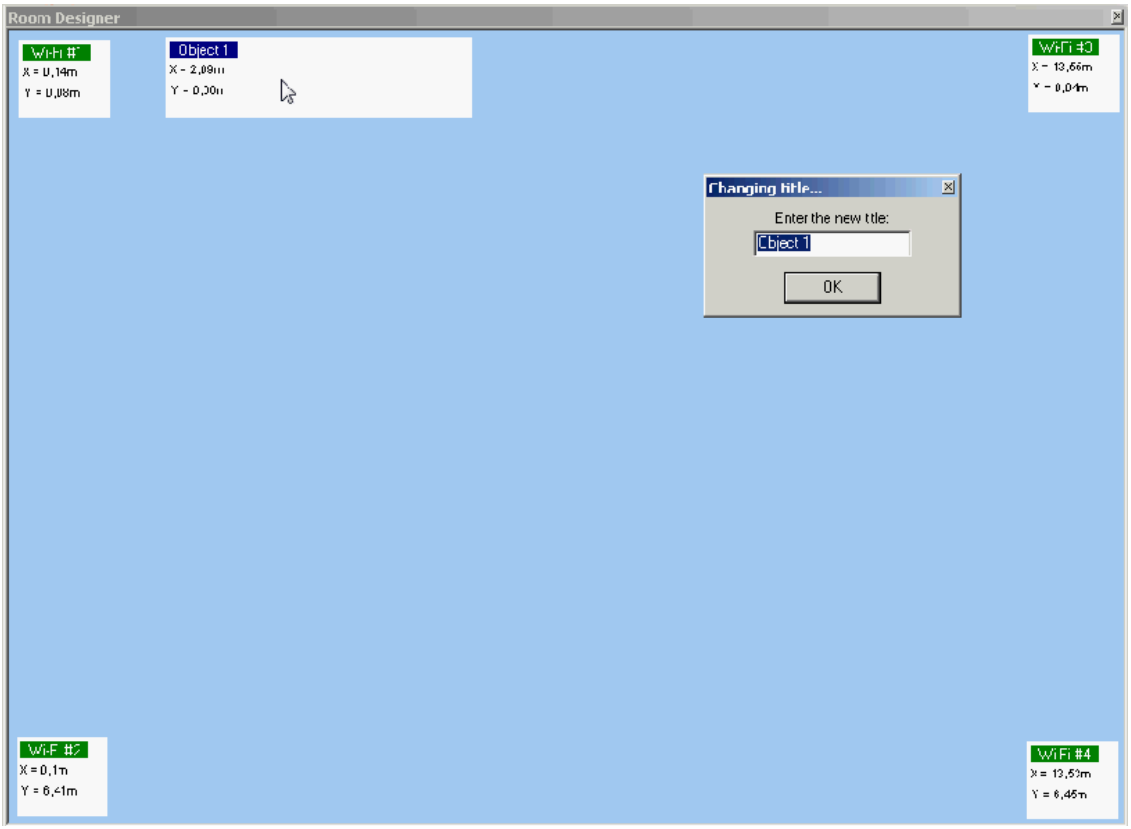
2) Указываем расположение точек доступа Wi-Fi. Для этого необходимо сделать клик правой кнопкой мыши на любом свободном месте и выбрать из контекстного меню «Add Wi-Fi point».



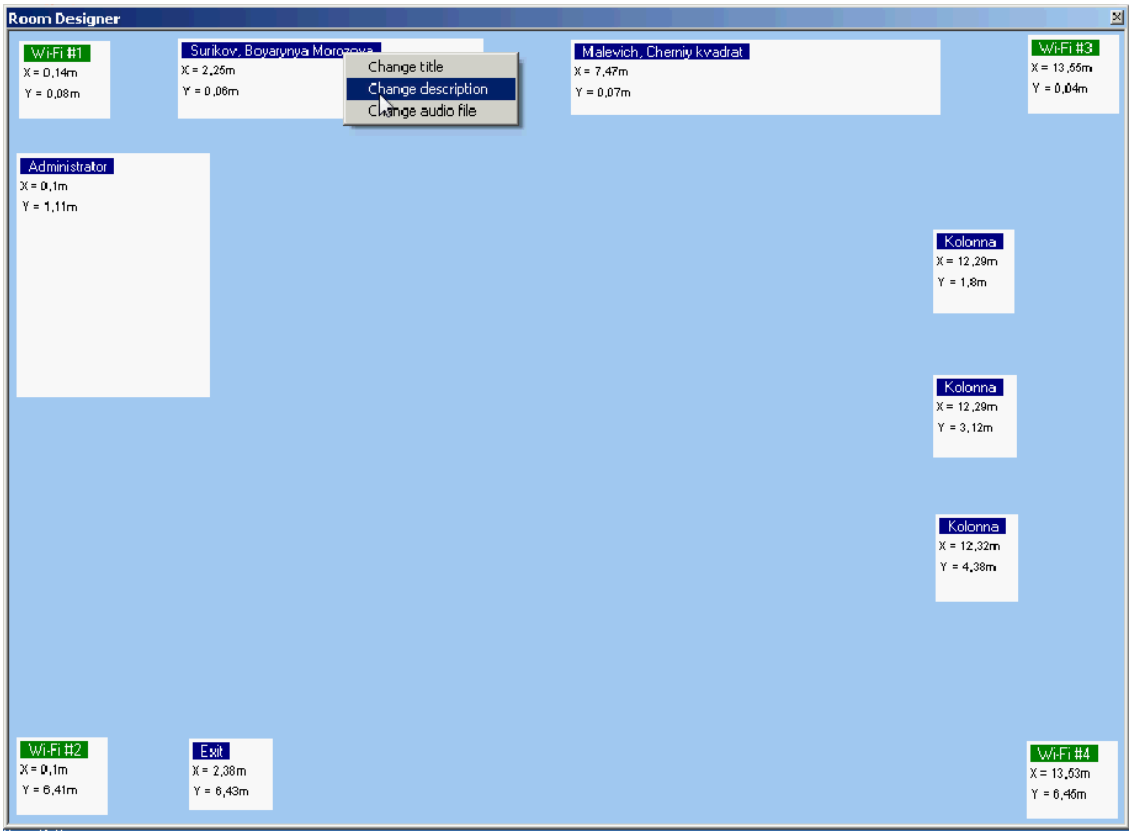
3) Указываем размещение объектов. Для этого необходимо сделать клик правой кнопкой мыши на любом свободном месте и выбрать из контекстного меню «Add object»



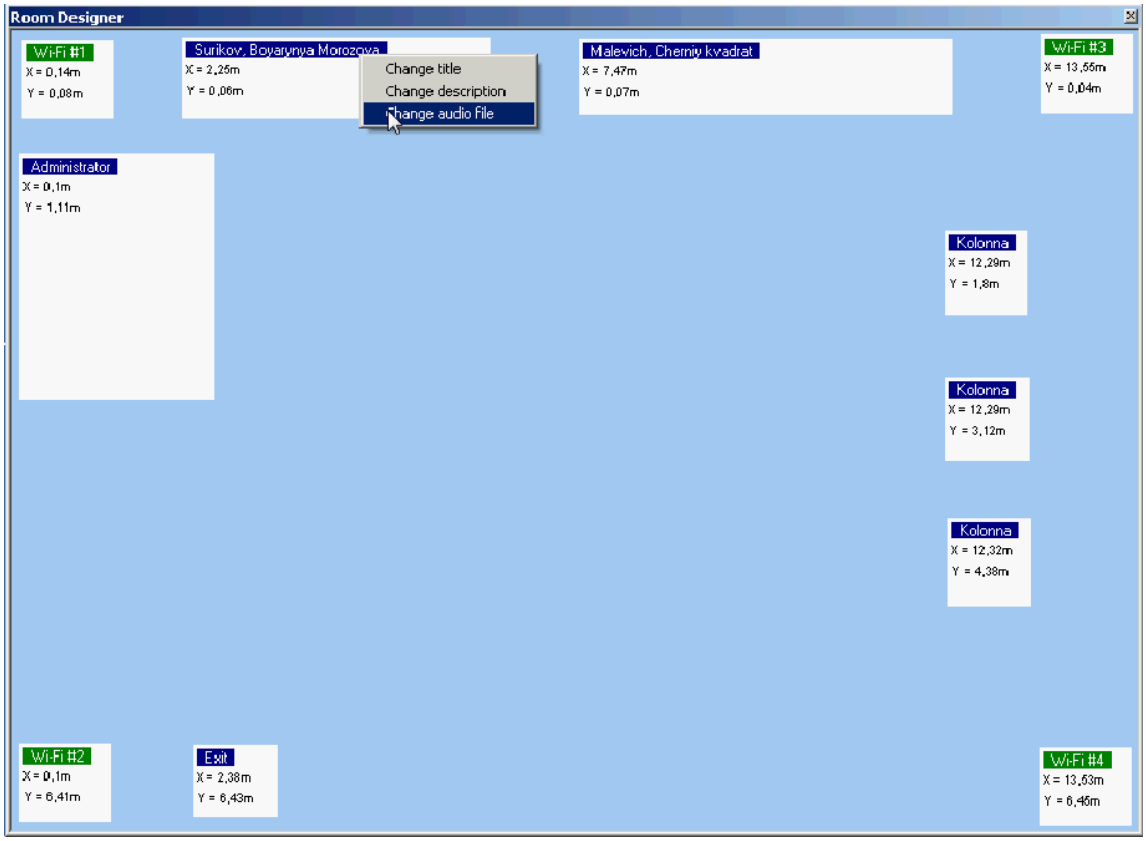
4) Вводим имя объекта



5) У каждого объекта имеется своё описание, которое, например, может появляться при приближении пользователя к этому объекту. Для добавления описания нужно сделать клик правой кнопкой мыши по объекту и выбрать «Change description».



6) Аналогично можно изменить аудио-файл



ПРИЛОЖЕНИЕ Б

Программный интерфейс для работы с Wi-Fi

```
{
        /// <summary>
        /// Represents a client to the Zeroconf (Native Wifi) service.
        /// </summary>
        /// <remarks>
        /// This class is the entrypoint to Native Wifi management. To manage WiFi settings, create an instance
        /// of this class.
        /// </remarks>
        public class WlanClient
        {
                /// <summary>
                /// Represents a Wifi network interface.
                /// </summary>
                public class WlanInterface
                {
                        private WlanClient client;
                        private Wlan.WlanInterfaceInfo info;

                        #region Events
                        /// <summary>
                        /// Represents a method that will handle <see cref="WlanNotification"/> events.
                        /// </summary>
                        /// <param name="notifyData">The notification data.</param>
                        public delegate void WlanNotificationEventHandler(Wlan.WlanNotificationData
notifyData);

                        /// <summary>
                        /// Represents a method that will handle <see cref="WlanConnectionNotification"/>
events.
                        /// </summary>
                        /// <param name="notifyData">The notification data.</param>
                        /// <param name="connNotifyData">The notification data.</param>
                        public delegate void
WlanConnectionNotificationEventHandler(Wlan.WlanNotificationData notifyData,
Wlan.WlanConnectionNotificationData connNotifyData);

                        /// <summary>
                        /// Represents a method that will handle <see cref="WlanReasonNotification"/> events.
                        /// </summary>
                        /// <param name="notifyData">The notification data.</param>
                        /// <param name="reasonCode">The reason code.</param>
                        public delegate void WlanReasonNotificationEventHandler(Wlan.WlanNotificationData
notifyData, Wlan.WlanReasonCode reasonCode);

                        /// <summary>
                        /// Occurs when an event of any kind occurs on a WLAN interface.
                        /// </summary>
                        public event WlanNotificationEventHandler WlanNotification;

                        /// <summary>
                        /// Occurs when a WLAN interface changes connection state.
                        /// </summary>
                        public event WlanConnectionNotificationEventHandler WlanConnectionNotification;

                        /// <summary>
                        /// Occurs when a WLAN operation fails due to some reason.
                        /// </summary>
                        public event WlanReasonNotificationEventHandler WlanReasonNotification;
```

```
#endregion

#region Event queue
private bool queueEvents;
private AutoResetEvent eventQueueFilled = new AutoResetEvent(false);
private Queue<object> eventQueue = new Queue<object>();

private struct WlanConnectionNotificationEventData
{
	public Wlan.WlanNotificationData notifyData;
	public Wlan.WlanConnectionNotificationData connNotifyData;
}
private struct WlanReasonNotificationData
{
	public Wlan.WlanNotificationData notifyData;
	public Wlan.WlanReasonCode reasonCode;
}
#endregion

internal WlanInterface(WlanClient client, Wlan.WlanInterfaceInfo info)
{
	this.client = client;
	this.info = info;
}

/// <summary>
/// Sets a parameter of the interface whose data type is <see cref="int"/>.
/// </summary>
/// <param name="opCode">The opcode of the parameter.</param>
/// <param name="value">The value to set.</param>
private void SetInterfaceInt(Wlan.WlanIntfOpcode opCode, int value)
{
	IntPtr valuePtr = Marshal.AllocHGlobal(sizeof(int));
	Marshal.WriteInt32(valuePtr, value);
	try
	{
		Wlan.ThrowIfError(
			Wlan.WlanSetInterface(client.clientHandle, info.interfaceGuid,
opCode, sizeof(int), valuePtr, IntPtr.Zero));
	}
	finally
	{
		Marshal.FreeHGlobal(valuePtr);
	}
}

/// <summary>
/// Gets a parameter of the interface whose data type is <see cref="int"/>.
/// </summary>
/// <param name="opCode">The opcode of the parameter.</param>
/// <returns>The integer value.</returns>
private int GetInterfaceInt(Wlan.WlanIntfOpcode opCode)
{
	IntPtr valuePtr;
	int valueSize;
	Wlan.WlanOpcodeValueType opcodeValueType;
	Wlan.ThrowIfError(
		Wlan.WlanQueryInterface(client.clientHandle, info.interfaceGuid,
opCode, IntPtr.Zero, out valueSize, out valuePtr, out opcodeValueType));
	try
	{
		return Marshal.ReadInt32(valuePtr);
```

```
                }
                finally
                {
                        Wlan.WlanFreeMemory(valuePtr);
                }
        }

        /// <summary>
        /// Gets or sets a value indicating whether this <see cref="WlanInterface"/> is
automatically configured.
        /// </summary>
        /// <value><c>true</c> if "autoconf" is enabled; otherwise, <c>false</c>.</value>
        public bool Autoconf
        {
                get
                {
                        return GetInterfaceInt(Wlan.WlanIntfOpcode.AutoconfEnabled) != 0;
                }
                set
                {
                        SetInterfaceInt(Wlan.WlanIntfOpcode.AutoconfEnabled, value ? 1 : 0);
                }
        }

        /// <summary>
        /// Gets or sets the BSS type for the indicated interface.
        /// </summary>
        /// <value>The type of the BSS.</value>
        public Wlan.Dot11BssType BssType
        {
                get
                {
                        return (Wlan.Dot11BssType)
GetInterfaceInt(Wlan.WlanIntfOpcode.BssType);
                }
                set
                {
                        SetInterfaceInt(Wlan.WlanIntfOpcode.BssType, (int)value);
                }
        }

        /// <summary>
        /// Gets the state of the interface.
        /// </summary>
        /// <value>The state of the interface.</value>
        public Wlan.WlanInterfaceState InterfaceState
        {
                get
                {
                        return
(Wlan.WlanInterfaceState)GetInterfaceInt(Wlan.WlanIntfOpcode.InterfaceState);
                }
        }

        /// <summary>
        /// Gets the channel.
        /// </summary>
        /// <value>The channel.</value>
        /// <remarks>Not supported on Windows XP SP2.</remarks>
        public int Channel
        {
                get
                {
```

```
                        return GetInterfaceInt(Wlan.WlanIntfOpcode.ChannelNumber);
                }
        }

        /// <summary>
        /// Gets the RSSI.
        /// </summary>
        /// <value>The RSSI.</value>
        /// <remarks>Not supported on Windows XP SP2.</remarks>
        public int RSSI
        {
                get
                {
                        return GetInterfaceInt(Wlan.WlanIntfOpcode.RSSI);
                }
        }

        /// <summary>
        /// Gets the current operation mode.
        /// </summary>
        /// <value>The current operation mode.</value>
        /// <remarks>Not supported on Windows XP SP2.</remarks>
        public Wlan.Dot11OperationMode CurrentOperationMode
        {
                get
                {
                        return (Wlan.Dot11OperationMode)
GetInterfaceInt(Wlan.WlanIntfOpcode.CurrentOperationMode);
                }
        }

        /// <summary>
        /// Gets the attributes of the current connection.
        /// </summary>
        /// <value>The current connection attributes.</value>
        /// <exception cref="Win32Exception">An exception with code 0x0000139F (The group
or resource is not in the correct state to perform the requested operation.) will be thrown if the interface is not
connected to a network.</exception>
        public Wlan.WlanConnectionAttributes CurrentConnection
        {
                get
                {
                        int valueSize;
                        IntPtr valuePtr;
                        Wlan.WlanOpcodeValueType opcodeValueType;
                        Wlan.ThrowIfError(
                                Wlan.WlanQueryInterface(client.clientHandle,
info.interfaceGuid, Wlan.WlanIntfOpcode.CurrentConnection, IntPtr.Zero, out valueSize, out valuePtr, out
opcodeValueType));
                        try
                        {
                                return
(Wlan.WlanConnectionAttributes)Marshal.PtrToStructure(valuePtr, typeof(Wlan.WlanConnectionAttributes));
                        }
                        finally
                        {
                                Wlan.WlanFreeMemory(valuePtr);
                        }
                }
        }

        /// <summary>
        /// Requests a scan for available networks.
```

```
                /// </summary>
                /// <remarks>
                /// The method returns immediately. Progress is reported through the <see
cref="WlanNotification"/> event.
                /// </remarks>
                public void Scan()
                {
                        Wlan.ThrowIfError(
                                Wlan.WlanScan(client.clientHandle, info.interfaceGuid, IntPtr.Zero,
IntPtr.Zero, IntPtr.Zero));
                }

                /// <summary>
                /// Converts a pointer to a available networks list (header + entries) to an array of
available network entries.
                /// </summary>
                /// <param name="bssListPtr">A pointer to an available networks list's header.</param>
                /// <returns>An array of available network entries.</returns>
                private Wlan.WlanAvailableNetwork[] ConvertAvailableNetworkListPtr(IntPtr
availNetListPtr)
                {
                        Wlan.WlanAvailableNetworkListHeader availNetListHeader =
(Wlan.WlanAvailableNetworkListHeader)Marshal.PtrToStructure(availNetListPtr,
typeof(Wlan.WlanAvailableNetworkListHeader));
                        long availNetListIt = availNetListPtr.ToInt64() +
Marshal.SizeOf(typeof(Wlan.WlanAvailableNetworkListHeader));
                        Wlan.WlanAvailableNetwork[] availNets = new
Wlan.WlanAvailableNetwork[availNetListHeader.numberOfItems];
                        for (int i = 0; i < availNetListHeader.numberOfItems; ++i)
                        {
                                availNets[i] =
(Wlan.WlanAvailableNetwork)Marshal.PtrToStructure(new IntPtr(availNetListIt),
typeof(Wlan.WlanAvailableNetwork));
                                availNetListIt +=
Marshal.SizeOf(typeof(Wlan.WlanAvailableNetwork));
                        }
                        return availNets;
                }

                /// <summary>
                /// Retrieves the list of available networks.
                /// </summary>
                /// <param name="flags">Controls the type of networks returned.</param>
                /// <returns>A list of the available networks.</returns>
                public Wlan.WlanAvailableNetwork[]
GetAvailableNetworkList(Wlan.WlanGetAvailableNetworkFlags flags)
                {
                        IntPtr availNetListPtr;
                        Wlan.ThrowIfError(
                                Wlan.WlanGetAvailableNetworkList(client.clientHandle,
info.interfaceGuid, flags, IntPtr.Zero, out availNetListPtr));
                        try
                        {
                                return ConvertAvailableNetworkListPtr(availNetListPtr);
                        }
                        finally
                        {
                                Wlan.WlanFreeMemory(availNetListPtr);
                        }
                }

                /// <summary>
                /// Converts a pointer to a BSS list (header + entries) to an array of BSS entries.
```

```
                /// </summary>
                /// <param name="bssListPtr">A pointer to a BSS list's header.</param>
                /// <returns>An array of BSS entries.</returns>
                private Wlan.WlanBssEntry[] ConvertBssListPtr(IntPtr bssListPtr)
                {
                        Wlan.WlanBssListHeader bssListHeader =
(Wlan.WlanBssListHeader)Marshal.PtrToStructure(bssListPtr, typeof(Wlan.WlanBssListHeader));
                        long bssListIt = bssListPtr.ToInt64() +
Marshal.SizeOf(typeof(Wlan.WlanBssListHeader));
                        Wlan.WlanBssEntry[] bssEntries = new
Wlan.WlanBssEntry[bssListHeader.numberOfItems];
                        for (int i=0; i<bssListHeader.numberOfItems; ++i)
                        {
                                bssEntries[i] = (Wlan.WlanBssEntry)Marshal.PtrToStructure(new
IntPtr(bssListIt), typeof(Wlan.WlanBssEntry));
                                bssListIt += Marshal.SizeOf(typeof(Wlan.WlanBssEntry));
                        }
                        return bssEntries;
                }

                /// <summary>
                /// Retrieves the basic service sets (BSS) list of all available networks.
                /// </summary>
                public Wlan.WlanBssEntry[] GetNetworkBssList()
                {
                        IntPtr bssListPtr;
                        Wlan.ThrowIfError(
                                Wlan.WlanGetNetworkBssList(client.clientHandle, info.interfaceGuid,
IntPtr.Zero, Wlan.Dot11BssType.Any, false, IntPtr.Zero, out bssListPtr));
                        try
                        {
                                return ConvertBssListPtr(bssListPtr);
                        }
                        finally
                        {
                                Wlan.WlanFreeMemory(bssListPtr);
                        }
                }

                /// <summary>
                /// Retrieves the basic service sets (BSS) list of the specified network.
                /// </summary>
                /// <param name="ssid">Specifies the SSID of the network from which the BSS list is
requested.</param>
                /// <param name="bssType">Indicates the BSS type of the network.</param>
                /// <param name="securityEnabled">Indicates whether security is enabled on the
network.</param>
                public Wlan.WlanBssEntry[] GetNetworkBssList(Wlan.Dot11Ssid ssid,
Wlan.Dot11BssType bssType, bool securityEnabled)
                {
                        IntPtr ssidPtr = Marshal.AllocHGlobal(Marshal.SizeOf(ssid));
                        Marshal.StructureToPtr(ssid, ssidPtr, false);
                        try
                        {
                                IntPtr bssListPtr;
                                Wlan.ThrowIfError(
                                        Wlan.WlanGetNetworkBssList(client.clientHandle,
info.interfaceGuid, ssidPtr, bssType, securityEnabled, IntPtr.Zero, out bssListPtr));
                                try
                                {
                                        return ConvertBssListPtr(bssListPtr);
                                }
                                finally
```

```
                {
                        Wlan.WlanFreeMemory(bssListPtr);
                }
        }
        finally
        {
                Marshal.FreeHGlobal(ssidPtr);
        }
}

/// <summary>
/// Connects to a network defined by a connection parameters structure.
/// </summary>
/// <param name="connectionParams">The connection paramters.</param>
protected void Connect(Wlan.WlanConnectionParameters connectionParams)
{
        Wlan.ThrowIfError(
                Wlan.WlanConnect(client.clientHandle, info.interfaceGuid, ref
connectionParams, IntPtr.Zero));
}

/// <summary>
/// Requests a connection (association) to the specified wireless network.
/// </summary>
/// <remarks>
/// The method returns immediately. Progress is reported through the <see
cref="WlanNotification"/> event.
/// </remarks>
public void Connect(Wlan.WlanConnectionMode connectionMode, Wlan.Dot11BssType
bssType, string profile)
{
        Wlan.WlanConnectionParameters connectionParams = new
Wlan.WlanConnectionParameters();
        connectionParams.wlanConnectionMode = connectionMode;
        connectionParams.profile = profile;
        connectionParams.dot11BssType = bssType;
        connectionParams.flags = 0;
        Connect(connectionParams);
}

/// <summary>
/// Connects (associates) to the specified wireless network, returning either on a success to
connect
/// or a failure.
/// </summary>
/// <param name="connectionMode"></param>
/// <param name="bssType"></param>
/// <param name="profile"></param>
/// <param name="connectTimeout"></param>
/// <returns></returns>
public bool ConnectSynchronously(Wlan.WlanConnectionMode connectionMode,
Wlan.Dot11BssType bssType, string profile, int connectTimeout)
{
        queueEvents = true;
        try
        {
                Connect(connectionMode, bssType, profile);
                while (queueEvents && eventQueueFilled.WaitOne(connectTimeout,
true))
                {
                        lock (eventQueue)
                        {
                                while (eventQueue.Count != 0)
```

```
                                        {
                                                object e = eventQueue.Dequeue();
                                                if (e is
WlanConnectionNotificationEventData)
                                                {

        WlanConnectionNotificationEventData wlanConnectionData = (WlanConnectionNotificationEventData)e;
                                                        // Check if the conditions are good to
indicate either success or failure.
                                                        if
(wlanConnectionData.notifyData.notificationSource == Wlan.WlanNotificationSource.ACM)
                                                        {
                                                                switch
((Wlan.WlanNotificationCodeAcm)wlanConnectionData.notifyData.notificationCode)
                                                                {
                                                                        case
Wlan.WlanNotificationCodeAcm.ConnectionComplete:
                                                                                if
(wlanConnectionData.connNotifyData.profileName == profile)

        return true;
                                                                                break;
                                                                }
                                                        }
                                                        break;
                                                }
                                        }
                                }
                        }
                        finally
                        {
                                queueEvents = false;
                                eventQueue.Clear();
                        }
                        return false; // timeout expired and no "connection complete"
                }

                /// <summary>
                /// Connects to the specified wireless network.
                /// </summary>
                /// <remarks>
                /// The method returns immediately. Progress is reported through the <see
cref="WlanNotification"/> event.
                /// </remarks>
                public void Connect(Wlan.WlanConnectionMode connectionMode, Wlan.Dot11BssType
bssType, Wlan.Dot11Ssid ssid, Wlan.WlanConnectionFlags flags)
                {
                        Wlan.WlanConnectionParameters connectionParams = new
Wlan.WlanConnectionParameters();
                        connectionParams.wlanConnectionMode = connectionMode;
                        connectionParams.dot11SsidPtr = Marshal.AllocHGlobal(Marshal.SizeOf(ssid));
                        Marshal.StructureToPtr(ssid, connectionParams.dot11SsidPtr, false);
                        connectionParams.dot11BssType = bssType;
                        connectionParams.flags = flags;
                        Connect(connectionParams);
                        Marshal.DestroyStructure(connectionParams.dot11SsidPtr, ssid.GetType());
                        Marshal.FreeHGlobal(connectionParams.dot11SsidPtr);
                }

                /// <summary>
                /// Deletes a profile.
                /// </summary>
```

```
                /// <param name="profileName">
                /// The name of the profile to be deleted. Profile names are case-sensitive.
                /// On Windows XP SP2, the supplied name must match the profile name derived
automatically from the SSID of the network. For an infrastructure network profile, the SSID must be supplied for
the profile name. For an ad hoc network profile, the supplied name must be the SSID of the ad hoc network
followed by <c>-adhoc</c>.
                /// </param>
                public void DeleteProfile(string profileName)
                {
                        Wlan.ThrowIfError(
                                Wlan.WlanDeleteProfile(client.clientHandle, info.interfaceGuid,
profileName, IntPtr.Zero));
                }

                /// <summary>
                /// Sets the profile.
                /// </summary>
                /// <param name="flags">The flags to set on the profile.</param>
                /// <param name="profileXml">The XML representation of the profile. On Windows XP
SP 2, special care should be taken to adhere to its limitations.</param>
                /// <param name="overwrite">If a profile by the given name already exists, then specifies
whether to overwrite it (if <c>true</c>) or return an error (if <c>false</c>).</param>
                /// <returns>The resulting code indicating a success or the reason why the profile wasn't
valid.</returns>
                public Wlan.WlanReasonCode SetProfile(Wlan.WlanProfileFlags flags, string
profileXml, bool overwrite)
                {
                        Wlan.WlanReasonCode reasonCode;
                        Wlan.ThrowIfError(
                                        Wlan.WlanSetProfile(client.clientHandle, info.interfaceGuid,
flags, profileXml, null, overwrite, IntPtr.Zero, out reasonCode));
                        return reasonCode;
                }

                /// <summary>
                /// Gets the profile's XML specification.
                /// </summary>
                /// <param name="profileName">The name of the profile.</param>
                /// <returns>The XML document.</returns>
                public string GetProfileXml(string profileName)
                {
                        IntPtr profileXmlPtr;
                        Wlan.WlanProfileFlags flags;
                        Wlan.WlanAccess access;
                        Wlan.ThrowIfError(
                                Wlan.WlanGetProfile(client.clientHandle, info.interfaceGuid,
profileName, IntPtr.Zero, out profileXmlPtr, out flags,
                                        out access));
                        try
                        {
                                return Marshal.PtrToStringUni(profileXmlPtr);
                        }
                        finally
                        {
                                Wlan.WlanFreeMemory(profileXmlPtr);
                        }
                }

                /// <summary>
                /// Gets the information of all profiles on this interface.
                /// </summary>
                /// <returns>The profiles information.</returns>
                public Wlan.WlanProfileInfo[] GetProfiles()
```

```
                {
                        IntPtr profileListPtr;
                        Wlan.ThrowIfError(
                                Wlan.WlanGetProfileList(client.clientHandle, info.interfaceGuid,
IntPtr.Zero, out profileListPtr));
                        try
                        {
                                Wlan.WlanProfileInfoListHeader header =
(Wlan.WlanProfileInfoListHeader) Marshal.PtrToStructure(profileListPtr,
typeof(Wlan.WlanProfileInfoListHeader));
                                Wlan.WlanProfileInfo[] profileInfos = new
Wlan.WlanProfileInfo[header.numberOfItems];
                                long profileListIterator = profileListPtr.ToInt64() +
Marshal.SizeOf(header);
                                for (int i=0; i<header.numberOfItems; ++i)
                                {
                                        Wlan.WlanProfileInfo profileInfo = (Wlan.WlanProfileInfo)
Marshal.PtrToStructure(new IntPtr(profileListIterator), typeof(Wlan.WlanProfileInfo));
                                        profileInfos[i] = profileInfo;
                                        profileListIterator += Marshal.SizeOf(profileInfo);
                                }
                                return profileInfos;
                        }
                        finally
                        {
                                Wlan.WlanFreeMemory(profileListPtr);
                        }
                }

                internal void OnWlanConnection(Wlan.WlanNotificationData notifyData,
Wlan.WlanConnectionNotificationData connNotifyData)
                {
                        if (WlanConnectionNotification != null)
                                WlanConnectionNotification(notifyData, connNotifyData);

                        if (queueEvents)
                        {
                                WlanConnectionNotificationEventData queuedEvent = new
WlanConnectionNotificationEventData();
                                queuedEvent.notifyData = notifyData;
                                queuedEvent.connNotifyData = connNotifyData;
                                EnqueueEvent(queuedEvent);
                        }
                }

                internal void OnWlanReason(Wlan.WlanNotificationData notifyData,
Wlan.WlanReasonCode reasonCode)
                {
                        if (WlanReasonNotification != null)
                                WlanReasonNotification(notifyData, reasonCode);
                        if (queueEvents)
                        {
                                WlanReasonNotificationData queuedEvent = new
WlanReasonNotificationData();
                                queuedEvent.notifyData = notifyData;
                                queuedEvent.reasonCode = reasonCode;
                                EnqueueEvent(queuedEvent);
                        }
                }

                internal void OnWlanNotification(Wlan.WlanNotificationData notifyData)
                {
                        if (WlanNotification != null)
```

```
                                        WlanNotification(notifyData);
                        }

                        /// <summary>
                        /// Enqueues a notification event to be processed serially.
                        /// </summary>
                        private void EnqueueEvent(object queuedEvent)
                        {
                                lock (eventQueue)
                                        eventQueue.Enqueue(queuedEvent);
                                eventQueueFilled.Set();
                        }

                        /// <summary>
                        /// Gets the network interface of this wireless interface.
                        /// </summary>
                        /// <remarks>
                        /// The network interface allows querying of generic network properties such as the
interface's IP address.
                        /// </remarks>
                        public NetworkInterface NetworkInterface
                        {
                                get
                                {
                        // Do not cache the NetworkInterface; We need it fresh
                        // each time cause otherwise it caches the IP information.
                                        foreach (NetworkInterface netIface in
NetworkInterface.GetAllNetworkInterfaces())
                                        {
                                                Guid netIfaceGuid = new Guid(netIface.Id);
                                                if (netIfaceGuid.Equals(info.interfaceGuid))
                                                {
                                                        return netIface;
                                                }
                                        }
                        return null;
                                }
                        }

                        /// <summary>
                        /// The GUID of the interface (same content as the <see
cref="System.Net.NetworkInformation.NetworkInterface.Id"/> value).
                        /// </summary>
                        public Guid InterfaceGuid
                        {
                                get { return info.interfaceGuid; }
                        }

                        /// <summary>
                        /// The description of the interface.
                        /// This is a user-immutable string containing the vendor and model name of the adapter.
                        /// </summary>
                        public string InterfaceDescription
                        {
                                get { return info.interfaceDescription; }
                        }

                        /// <summary>
                        /// The friendly name given to the interface by the user (e.g. "Local Area Network
Connection").
                        /// </summary>
                        public string InterfaceName
                        {
```

```
                        get { return NetworkInterface.Name; }
                    }
                }

                private IntPtr clientHandle;
                private uint negotiatedVersion;
                private Wlan.WlanNotificationCallbackDelegate wlanNotificationCallback;

                private Dictionary<Guid,WlanInterface> ifaces = new Dictionary<Guid,WlanInterface>();

                /// <summary>
                /// Creates a new instance of a Native Wifi service client.
                /// </summary>
                public WlanClient()
                {
                    Wlan.ThrowIfError(
                        Wlan.WlanOpenHandle(Wlan.WLAN_CLIENT_VERSION_XP_SP2,
IntPtr.Zero, out negotiatedVersion, out clientHandle));
                    try
                    {
                        Wlan.WlanNotificationSource prevSrc;
                        wlanNotificationCallback = new
Wlan.WlanNotificationCallbackDelegate(OnWlanNotification);
                        Wlan.ThrowIfError(
                            Wlan.WlanRegisterNotification(clientHandle,
Wlan.WlanNotificationSource.All, false, wlanNotificationCallback, IntPtr.Zero, IntPtr.Zero, out prevSrc));
                    }
                    catch
                    {
                        Wlan.WlanCloseHandle(clientHandle, IntPtr.Zero);
                        throw;
                    }
                }

                ~WlanClient()
                {
                    Wlan.WlanCloseHandle(clientHandle, IntPtr.Zero);
                }

                private Wlan.WlanConnectionNotificationData? ParseWlanConnectionNotification(ref
Wlan.WlanNotificationData notifyData)
                {
                    int expectedSize = Marshal.SizeOf(typeof(Wlan.WlanConnectionNotificationData));
                    if (notifyData.dataSize < expectedSize)
                        return null;

                    Wlan.WlanConnectionNotificationData connNotifyData =
                        (Wlan.WlanConnectionNotificationData)
                        Marshal.PtrToStructure(notifyData.dataPtr,
typeof(Wlan.WlanConnectionNotificationData));
                    if (connNotifyData.wlanReasonCode == Wlan.WlanReasonCode.Success)
                    {
                        IntPtr profileXmlPtr = new IntPtr(
                            notifyData.dataPtr.ToInt64() +
                            Marshal.OffsetOf(typeof(Wlan.WlanConnectionNotificationData),
"profileXml").ToInt64());
                        connNotifyData.profileXml = Marshal.PtrToStringUni(profileXmlPtr);
                    }
                    return connNotifyData;
                }

                private void OnWlanNotification(ref Wlan.WlanNotificationData notifyData, IntPtr context)
                {
```

```
                        WlanInterface wlanIface = ifaces.ContainsKey(notifyData.interfaceGuid) ?
ifaces[notifyData.interfaceGuid] : null;

                        switch(notifyData.notificationSource)
                        {
                                case Wlan.WlanNotificationSource.ACM:
                                        switch((Wlan.WlanNotificationCodeAcm)notifyData.notificationCode)
                                        {
                                                case Wlan.WlanNotificationCodeAcm.ConnectionStart:
                                                case Wlan.WlanNotificationCodeAcm.ConnectionComplete:
                                                case Wlan.WlanNotificationCodeAcm.ConnectionAttemptFail:
                                                case Wlan.WlanNotificationCodeAcm.Disconnecting:
                                                case Wlan.WlanNotificationCodeAcm.Disconnected:
                                                        Wlan.WlanConnectionNotificationData?
connNotifyData = ParseWlanConnectionNotification(ref notifyData);
                                                        if (connNotifyData.HasValue)
                                                                if (wlanIface != null)

        wlanIface.OnWlanConnection(notifyData, connNotifyData.Value);
                                                        break;
                                                case Wlan.WlanNotificationCodeAcm.ScanFail:
                                                        {
                                                                int expectedSize = Marshal.SizeOf(typeof
(Wlan.WlanReasonCode));
                                                                if (notifyData.dataSize >= expectedSize)
                                                                {
                                                                        Wlan.WlanReasonCode reasonCode
= (Wlan.WlanReasonCode) Marshal.ReadInt32(notifyData.dataPtr);
                                                                        if (wlanIface != null)

        wlanIface.OnWlanReason(notifyData, reasonCode);
                                                                }
                                                        }
                                                        break;
                                        }
                                        break;
                                case Wlan.WlanNotificationSource.MSM:
                                        switch((Wlan.WlanNotificationCodeMsm)notifyData.notificationCode)
                                        {
                                                case Wlan.WlanNotificationCodeMsm.Associating:
                                                case Wlan.WlanNotificationCodeMsm.Associated:
                                                case Wlan.WlanNotificationCodeMsm.Authenticating:
                                                case Wlan.WlanNotificationCodeMsm.Connected:
                                                case Wlan.WlanNotificationCodeMsm.RoamingStart:
                                                case Wlan.WlanNotificationCodeMsm.RoamingEnd:
                                                case Wlan.WlanNotificationCodeMsm.Disassociating:
                                                case Wlan.WlanNotificationCodeMsm.Disconnected:
                                                case Wlan.WlanNotificationCodeMsm.PeerJoin:
                                                case Wlan.WlanNotificationCodeMsm.PeerLeave:
                                                case Wlan.WlanNotificationCodeMsm.AdapterRemoval:
                                                        Wlan.WlanConnectionNotificationData?
connNotifyData = ParseWlanConnectionNotification(ref notifyData);
                                                        if (connNotifyData.HasValue)
                                                                if (wlanIface != null)

        wlanIface.OnWlanConnection(notifyData, connNotifyData.Value);
                                                        break;
                                        }
                                        break;
                        }

                        if (wlanIface != null)
                                wlanIface.OnWlanNotification(notifyData);
```

```
        }

        /// <summary>
        /// Gets the WLAN interfaces.
        /// </summary>
        /// <value>The WLAN interfaces.</value>
        public WlanInterface[] Interfaces
        {
                get
                {
                        IntPtr ifaceList;
                        Wlan.ThrowIfError(
                                Wlan.WlanEnumInterfaces(clientHandle, IntPtr.Zero, out ifaceList));
                        try
                        {
                                Wlan.WlanInterfaceInfoListHeader header =
                                        (Wlan.WlanInterfaceInfoListHeader)
Marshal.PtrToStructure(ifaceList, typeof (Wlan.WlanInterfaceInfoListHeader));
                                Int64 listIterator = ifaceList.ToInt64() + Marshal.SizeOf(header);
                                WlanInterface[] interfaces = new
WlanInterface[header.numberOfItems];
                                List<Guid> currentIfaceGuids = new List<Guid>();
                                for (int i = 0; i < header.numberOfItems; ++i)
                                {
                                        Wlan.WlanInterfaceInfo info =
                                                (Wlan.WlanInterfaceInfo)
Marshal.PtrToStructure(new IntPtr(listIterator), typeof (Wlan.WlanInterfaceInfo));
                                        listIterator += Marshal.SizeOf(info);
                                        WlanInterface wlanIface;
                                        currentIfaceGuids.Add(info.interfaceGuid);
                                        if (ifaces.ContainsKey(info.interfaceGuid))
                                                wlanIface = ifaces[info.interfaceGuid];
                                        else
                                                wlanIface = new WlanInterface(this, info);
                                        interfaces[i] = wlanIface;
                                        ifaces[info.interfaceGuid] = wlanIface;
                                }

                                // Remove stale interfaces
                                Queue<Guid> deadIfacesGuids = new Queue<Guid>();
                                foreach (Guid ifaceGuid in ifaces.Keys)
                                {
                                        if (!currentIfaceGuids.Contains(ifaceGuid))
                                                deadIfacesGuids.Enqueue(ifaceGuid);
                                }
                                while(deadIfacesGuids.Count != 0)
                                {
                                        Guid deadIfaceGuid = deadIfacesGuids.Dequeue();
                                        ifaces.Remove(deadIfaceGuid);
                                }

                                return interfaces;
                        }
                        finally
                        {
                                Wlan.WlanFreeMemory(ifaceList);
                        }
                }
        }

        /// <summary>
        /// Gets a string that describes a specified reason code.
        /// </summary>
```

```
                /// <param name="reasonCode">The reason code.</param>
                /// <returns>The string.</returns>
                public string GetStringForReasonCode(Wlan.WlanReasonCode reasonCode)
                {
                        StringBuilder sb = new StringBuilder(1024); // the 1024 size here is arbitrary; the
WlanReasonCodeToString docs fail to specify a recommended size
                        Wlan.ThrowIfError(
                                Wlan.WlanReasonCodeToString(reasonCode, sb.Capacity, sb, IntPtr.Zero));
                        return sb.ToString();
                }
        }
}
```