

# How to Use StreamSpin for Creating Your Own Services

Kim Christensen  
kcdc@cs.aau.dk

Department of Computer Science  
Aalborg University, Denmark

September 25, 2007

## 1 Introduction

The center for Data-intensive Systems (Daisy) at Aalborg University is currently working on a large project in which they combine the two worlds of *Web 2.0* and *Location-based services*.

The project known as the StreamSpin-project currently employs numerous developers. Furthermore it is planned that students will have the opportunity to develop their own services for the system during the coming semesters in order to mature the system and its service deliveries.

It is the overall purpose of the StreamSpin project to “*be for Location-based services what YouTube is for video*”. Thereby, the vision is that virtually anybody should be able to create their own services and publish them to the masses or limit the publication to close friends.

The motivation behind the project is the vision that the web now is becoming more and more mobile. Hence the need for Location-based services will grow massively in the near future. People using mobile phones with GPS or other positioning systems will have the possibility of knowing their position at a given time.

This opens up for numerous advantageous scenarios in which a user of a mobile phone could retrieve information based on position, time and a customizable profile. Systems that displays this behavior already exist, however the StreamSpin-project proposes that services are pushed to the mobile device rather than requested by the mobile device itself [1, 3].

The aim of this short paper is to cover the most important issues involving the creation of services for the StreamSpin-project. This includes understanding the StreamSpin server structure and creating a service for publishing and consuming these services. Furthermore it is highly advisable to read the material available on the StreamSpin web page before continuing onwards in this paper as some concepts might stand clearer afterwards.

## 2 Creating Services for StreamSpin

Initially it is important to understand what StreamSpin is capable of and what it consist of. As mentioned in the introduction StreamSpin is a middle-layer service that allows information and services to be pushed directly to a mobile device. Hence only the following requirements are needed:

- Mobile device either with Windows Mobile 5 or Java MIDP 2.0 and GPS receiver [2]
- Access to a server with a server-side HTML scripting language (like PHP or ASP.Net)

The team behind StreamSpin provides a client application for use with the mobile device on both platforms. This client is most helpful for testing purposes. The client can be downloaded on the StreamSpin web site. However it is also possible to test StreamSpin without the usage of a mobile device. This is done by creating locations for the users profile and simulates “moves” by selecting these in an emulation of the Windows Mobile 5 client. When the service to be created needs to be pushed according to the users location, some script must be executed to reload the content which must be pushed to the mobile device.

### 2.1 Interacting with StreamSpin

When a user interacts with StreamSpin the first important issue should be to register a profile at the project web page. From the web page it is possible to setup name, occupation, new locations, preferred types of services and other information needed for the personal profile. The most important information stored in the profile is the *Symmetric Key* and *User ID*. These two keys identify the user when interacting with the StreamSpin server. This can be done in a number of ways including a .NET API, a web service, PHP API, Java API and a query API. All of these ways of communication provide the same functionality. However the .NET API has a built in mechanism that handles users automatically. This makes the coding simpler, however this depends on always using the newest library. When interacting (both publishing and consuming content) with StreamSpin through the web service or Query API it is needed to authenticate the callee to avoid wrongfully use of the StreamSpin server.

#### 2.1.1 Publishing a Simple Service in StreamSpin

To publish simple content in StreamSpin it is of course necessary to have an active account which can be created at the project web page. When the account has been created it is possible to create new services for use in StreamSpin. StreamSpin allows the user to create services both from the web page and through API calls in the available StreamSpin web services that handles creation of services. The content of a simple service contains a *description* and a *URL*. Simply, the description can be translated as the headline of the content to be pushed to the mobile device. The resource located at the URL on the other hand contains the actual changeable content.

A simple service in StreamSpin does not contain any time or geographical context. Thereby, examples of simple services could be notification on emails,

news feeds etc. As a consequence of this, the StreamSpin server does not have to obtain any knowledge on this type of information about the user. Hence whenever some content is received by the StreamSpin server it is pushed directly to the users who have subscribed to this particular service. Whenever some content is received, the StreamSpin server will refresh the page pointed to by the URL (e.g. the PageLoad-method) and this contains the actual code for setting up the content for the mobile device.

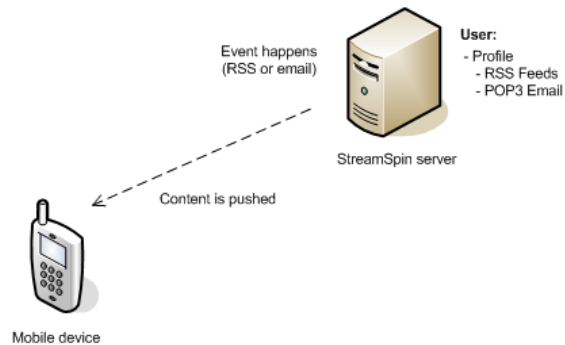


Figure 1: A simple service in StreamSpin [5]

A simple service like the one shown in Figure 1 could be implemented with Listing 1. As the Listing shows the use of the CryptoLib.dll library it is a call through the web service located at:

<http://serverservices.streamspin.com/contentpublishing.asmx>

The *ServerServices*-class is available from the web service. This class allows the programmer to create a *ContentPublishing*-object for publishing content.

```

1 using CryptoLib.Signatures;
2
3 static void Main(string[] args)
4 {
5     private const int sID = THE ID OF YOUR SERVICE;
6     private const string key = YOUR SYMMETRIC KEY;
7
8     //create a streamspin publish object
9     ServerServices.ContentPublishing publisher = new
10    ServerServices.ContentPublishing();
11
12    //create a streamspin content object
13    ServerServices.Content content = new ServerServices.Content();
14
15    //set the description of the content object
16    content.Description = "Streamspin_demo";
17
18    //set the body of the content object
19    content.Url = "Hello_World"
20
21    //now we need to sign the content, first we create a timestamp
22    string tstamp = Signatures.CreateTimeStamp(DateTime.Now);
23
24    //then we create the textstring to sign
25    string toSign =
26        content.Description + content.Url + sID + tstamp;
27

```

```

28 //finally we create the signature using the key from MyInfo
29 string signature = Signatures.Sign(toSign, key);
30
31 //we then publish the content using the publisher object
32 publish.PublishPrivateContent(content, sID,
33                               signature, timestamp);
34 }

```

Listing 1: Hello world in Streamspin

### 2.1.2 The StreamSpin .NET API

When a user uses the Streamspin .NET API it is not necessary to sign the messages each time a service is published or manipulated. The user security is ensured by the API itself. Hence it is not needed with the signing of messages for the StreamSpin server. This simplifies the code that needs to be written substantially. Listing 2 shows the details on how to use the StreamSpin .NET API for publishing a simple service.

```

1 using Com.Streamspin;
2
3 static void Main(string[] args)
4 {
5     private const int sID = THE ID OF YOUR SERVICE;
6     private const string key = YOUR SYMMETRIC KEY;
7
8     //create a streamspin publish object
9     Publisher pub = new Publisher();
10
11    //create a streamspin content object
12    Content con = new Content();
13
14    //set the description of the content object
15    con.Description = "Streamspin_demo";
16
17    //set the body of the content object
18    con.UrlOrBody = YOUR URL OR BODY eg. 'Hello World';
19
20    //we then publish the content using the publisher object
21    pub.PublishContent(con, sID, new Key(key));
22 }

```

Listing 2: Hello world with StreamSpin .NET API

As Code Snippet 2 clearly shows, the advantage of using the .NET API is that the user does not need to sign the content before publishing. Henceforth the .NET API will be used in the examples of this article.

### 2.1.3 Publishing a Location-based Service in StreamSpin

Compared to simple services the location-based services are much more interesting to deal with from the consumer viewpoint. However publishers can also have interest in publishing location based content to the masses. From the consumer viewpoint it is highly favorable to receive content only when the user is in the vicinity of some point of interest (POI). Furthermore it could also be of interest of the publisher to make use of vicinity based offers to customers within short ranges.

A location based service is not much different from a conventional service. However it is needed to handle the movement of users at server side to be able to push content to the mobile device. To use StreamSpins location based

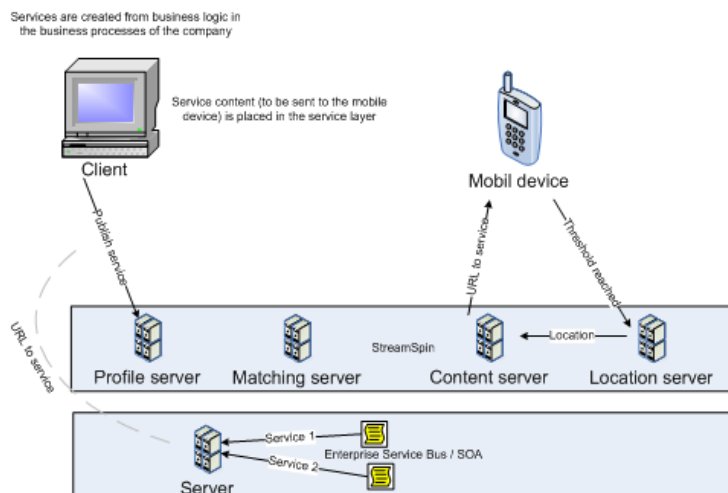


Figure 2: A location based service in StreamSpin [5]

functionality for realistic use it is necessary to create a web application that handles the content which must be pushed to the user. This is beyond the scope of this paper, however it is possible to obtain ideas for creating your own business logic by visiting the web page given in [4].

When this web application has been made it is possible to subscribe to the location based service in StreamSpin. This is done the same way as in a conventional service (see Listing 2). However it is also necessary to notice the location server in StreamSpin to monitor the movement of a user. This is handled by monitoring the movement of the mobile device on which the user is logged on. This is simply done by creating a console application with a main-method that executes the following:

```

1 using Com.Streamspin;
2
3 static void Main(string[] args)
4 {
5     private const int sID = THE ID OF YOUR SERVICE;
6     private const string key = YOUR SYMMETRIC KEY;
7
8     //create key from the your key string
9     Key myKey = new Key(key);
10
11    //set the url of where your web application is located
12    string url = YOUR URL OR BODY;
13
14    //create new userlocationrequest
15    UserLocationRequest userReq = new UserLocationRequest();
16
17    //call AddRequest with above parameters and an int threshold
18    userReq.AddRequest(userid, serviceid, 25, url, myKey);
19 }

```

## 3 Understanding the StreamSpin .NET API

To understand more of StreamSpin it is needed to dig into its components. At the surface StreamSpin consists of five main components or classes[4]:

- `Content`
- `Key`
- `Publisher`
- `Services`
- `UserLocationRequest`

Of these five classes the most interesting are the `Publisher`, `Services` and `UserLocationRequest`. These are interesting as they are the classes that enables StreamSpin to create and publish services. Furthermore the `UserLocationRequest`-class presents the user of the functionality to make a service location dependent.

### 3.1 The Publisher-class

The `Publisher`-class is responsible for publishing content on the StreamSpin system. It has one special method that handles the publishing, namely the `PublishContent`-method.

**The `PublishContent`-method** takes three parameters (*Content*, *serviceID* and *key*). These three parameters identifies the user and makes the system aware of which content that is to be published. In advance the programmer must have constructed the relevant content to be published. The method returns a *bool* to show whether the content has been published correctly or not.

### 3.2 The Services-class

The `Services`-class handles the creation and deletion of services. Before digging into the details of the `Services`-class it is important to understand the principle of a *template*. A template is a service that other services can be created from. Using object-oriented terms a template is a superclass of all the services which are created with this service as template. An example is a service template like a shopping list. A service provider have created a general template for how the shopping list is handled. When a programmer wants to use this service he does only use the template to create his own specialized service and not the template directly. In this way the template and the actual service(s) are linked in a parent-child relationship. The *Services*-class has four important methods:

- *CreateService*
- *CreateServiceCopy*

- *DeleteService*
- *SubscribeUserToService*

**The CreateService-method** does as the name suggests. It creates a service in the StreamSpin system. The method takes numerous parameters and returns the id of the newly created service. The parameters that the method takes includes the usual parts for identifying the service and its creator. Besides these somewhat mandatory parameters the method has a number of special parameters which applies in special cases of service creation. Tagging the service with customized tags is possible via the *tags* string array. The *allowMultiSubscription* boolean-parameter applies when a service is created as a template. If this parameter is set to true there exist a setupURL which must be given in the *url*-parameter. This is due to the fact that the created service is only a template and thereby needs to be “instantiated” to be used. The instantiation is done in the web application found at the location of the setupURL. If the service allows multiple subscriptions and has a URL, the boolean parameter *requiresSetup* should also be set to true for the given service. One last interesting parameter is the *startServiceURL*-parameter. If this URL is set the service can be started directly from the mobile device. An example of such a service is a guided tour.

**The CreateServiceCopy-method** is responsible for “instantiating” (or personalizing) the actual services that are created from a template service. The parameters for this method offers to personalize the service template. The user can add his own distinction to the name by passing the *addToTitle*-parameter. This string is prefixed to the template service name. Another interesting parameter is the *useParent*-parameter. This parameter consists of the id of the parent to the newly created service copy. This is practical as it important to know which service is the parent of the personalized service. Especially this is important when services are deleted. In the case where the personalized service uses functionality provided in its parent it is crucial that the service will be deleted if the parent is also deleted. Either way, the service copy will not function without its parent.

**The DeleteService-method** deletes a given service in the StreamSpin system. As mentioned above the deletion of a service template will cause the service copies created from the template to be deleted **if** the *useParent* is set. Otherwise the deletion of a service template will not cascade down through its “children”.

**The SubscribeUserToService-method** subscribes a specific user to a service created in advance. However the mobile device will not instantly receive the information provided by the service. This is only achieved if the user subscribes manually to the service (can be done using the StreamSpin web site). Furthermore a method for subscribing and approving a service has been created. However this method is just for quick testing and multiple user subscription purposes. Further details on the subscribe approval method can be found in [4].

### 3.3 The UserLocationRequest-class

The UserLocationRequest-class encapsulates user location requests. This means that this class notifies the StreamSpin location server whenever it needs to listen for movement of a user.

**The AddRequest-method** notifies the location server that a given user must be monitored. The method returns an integer that indicates whether the request has been acknowledged (0) or some error has occurred (like web service error codes). Besides the usual service and user credentials the arguments of this method include *threshold* and *callbackURL*. The *threshold* specifies how far (in meters) the device must move before it receives information via StreamSpin. The *callbackURL* indicates where the actual service specific code is located. The URL will be called with the *userID*, *latitude* and *longitude*. These arguments are named `userId`, `lat` and `lng` if they are to be used in the service specific code.

**The CancelRequest-method** removes a request for device movement. This method takes the same arguments as the *AddRequest*-method except that it takes no *threshold*.

### 3.4 Summarizing the Important Elements

This section is dedicated to giving a short summary of the most useful methods in the different classes of the StreamSpin .NET API. This is presented in Table 2. Furthermore a summary of the most useful constructors is also presented in Table 1.

Constructor	Arguments
Content	<i>string</i> description, <i>string</i> urlOrBody
Key	<i>string</i> keyValue
Publisher	
Services	
UserLocationRequest	

Table 1: Summary of important constructors in the StreamSpin .NET API

Table 1 summarizes the most important constructors in the StreamSpin .NET API and their arguments. In the cases where no arguments are given means that only a default constructor is available. Table 2 gives an overview of the methods in the different classes and their arguments and return values.

Method	Class	Arguments	Return Value
PublishContent	Publisher	<i>Content</i> Content, <i>int</i> serviceID, <i>Key</i> key	<i>bool</i>
CreateService	Services	<i>int</i> userID, <i>int</i> groupID, <i>string</i> name, <i>string</i> description, <i>string</i> subscriptionPassword, <i>bool</i> allowMultiSubscription, <i>string</i> url, <i>string</i> startServiceURL, <i>bool</i> requiresSetup, <i>bool</i> isPublic, <i>Key</i> key, <i>string[]</i> tags	<i>int</i>
CreateServiceCopy	Services	<i>string</i> addToTitle, <i>int</i> editorID, <i>bool</i> isPublic, <i>int</i> serviceID, <i>int</i> userID, <i>string</i> url, <i>bool</i> useParent, <i>Key</i> key	<i>int</i>
DeleteService	Services	<i>int</i> serviceID, <i>Key</i> key	<i>int</i>
SubscribeUser ToService	Services	<i>int</i> serviceID, <i>int</i> providerID, <i>int</i> userToSubscribeID, <i>Key</i> key	<i>bool</i>
AddRequest	UserLocationRequest	<i>int</i> userID, <i>int</i> serviceID, <i>int</i> threshold, <i>string</i> callbackURL, <i>Key</i> key	<i>int</i>
CancelRequest	UserLocationRequest	<i>int</i> userID, <i>int</i> serviceID, <i>string</i> callbackURL, <i>Key</i> key	<i>int</i>

Table 2: Summary of important methods in the StreamSpin .NET API

## References

- [1] G. Borriello, M. Chalmers, A. LaMarca, and P. Nixon. Delivering real-world ubiquitous location systems. *Communications of the ACM*, 48(3):36–41, 2005.
- [2] A. Civilis, C. S. Jensen, and S. Pakalnis. Techniques for efficient road-network-based tracking of moving objects. *IEEE Transactions on Knowledge and Data Engineering*, 17(5):698–712, 2005.
- [3] E. Jeannine. User needs for location-aware mobile services. *Personal and Ubiquitous Computing*, 7:70–79, 2003.
- [4] Rico Wind and Kenneth H. Pedersen. *StreamSpin Website*. Daisy, Center for DATA Intensive SYstems, AAU, 2007. URL <http://www.streamspin.com>.
- [5] Rico Wind, Christian S. Jensen, and Kristian Torp. An Open Platform for the Creation and Deployment of Transport-Related Mobile Data Services. 2007.