Archived:Discovering Bluetooth services using Symbian C++

Archived: This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

Overview

This article demonstrates how to use `CSdpAgent` to discover Bluetooth services.

`CSdpAgent` creates Bluetooth service discovery protocol (SDP) requests to a remote device.

Construct a CSdpAgent object with the Bluetooth address of the remote device to query.

Set the classes of service that you want to query for. The classes have predefined UUIDs, which are specified with `SetRecordFilterL()`.

Get results through `NextRecordRequestL()`. Information on services is stored as records, which can be queried in turn for attribute values for the service through `AttributeRequestL()`.

The user of this class must implement MSdpAgentNotifier to receive the responses to queries.

Note that queries are asynchronous, although this is hidden in the API. The implication is that the interface functions that receive responses are only called when the thread's active scheduler can schedule handling the completion of the query.

`CSdpSearchPattern` A list of Bluetooth service classes, represented as Universal Unique Identifiers (UUIDs), to be matched in SDP Service Search Requests.

MMP file

The following capabilities and libraries are required:

```
CAPABILITY LocalServices
LIBRARY sdpagent.lib
LIBRARY sdpdatabase.lib
```

Header file

```
#include <btsdp.h>

// The service ID that identifies the service. This ID is
// used when advertising the service and discovering the service.
#define KBT_serviceID 0x10ff

class CMyServiceDiscoverer : public CBase,
                             public MSdpAgentNotifier,
                             public MSdpAttributeValueVisitor
{
...

// Service discovery agent
CSdpAgent*          iAgent;

// Service discovery search pattern
CSdpSearchPattern*  iSpat;

// Index of device in device data list used for service discovery
TInt iDeviceIdx;

// Last discovered uuid in the service attributes
TUUID iLastUUID;

// Port (comm channel) found in the service attributes
TUint iPort;
```

Source file

Discover services on a given device. The service discovery agent will be started to do the discovery of services on given remote device. Service discovery will be limited to search only for services with the service ID.

(C) Copyright Nokia 2013. All rights reserved.

```
void CMyServiceDiscoverer::DiscoverServicesOnDeviceL(TBTDevAddr* aDevAddr)
    {
    // Init new service discovery agent
    iAgent = CSdpAgent::NewL( *this, aDevAddr);

    // Set search properties for agent
    iSpat = CSdpSearchPattern::NewL();

    // Use own service id to filter the services discovered
    // -> will return only the services with matching service id(s)
    TUUID serviceUUID(KBT_serviceID);
    iSpat->AddL(serviceUUID);
    iAgent->SetRecordFilterL(*iSpat);

    // Initiate search
    // this will result in call to NextRecordRequestComplete()
    iAgent->NextRecordRequestL();
    }
```

Called when the service discovery agent has completed discovering services on device.

```
// from MSdpAgentNotifier
void CMyServiceDiscoverer::NextRecordRequestComplete(
    TInt aError,
    TSdpServRecordHandle aHandle,
    TInt aTotalRecordsCount)
    {
    if ( aError==KErrNone && aTotalRecordsCount>0 )
        {
        // We got records, retrieve attributes for record
        // request protocol descriptor from remote device records,
        // we need this to retrieve remote port to connect to later on.
        // Calls AttributeRequestResult()
        TRAPD(err,iAgent->AttributeRequestL(aHandle, KSdpAttrIdProtocolDescriptorList));
        if( err )
            {
            // TODO: Handle error
            }
        }
    else
        {
        // Done with this device

        // Discover services on next device, if any left
        iDeviceIdx++;
        if ( iDeviceIdx<iDevDataList->Count() )
            {
            // More devices to probe, proceed
            // TODO: Call DiscoverServicesOnDeviceL() again with
            // next TBTDevAddr
            }
        else
            {
            // All devices done
            FinishDiscovery();
            }
        }
    }
```

Called when the service attributes for the service record have been retrieved.

```
// from MSdpAgentNotifier
void CMyServiceDiscoverer::AttributeRequestResult(
    TSdpServRecordHandle aHandle,
    TSdpAttributeID aAttrID,
    CSdpAttrValue* aAttrValue)
    {
    // Parse attributes, will result in call to VisitAttributeValue()
    TRAPD(err,aAttrValue->AcceptVisitorL(*this) );
    if( err )
        {
        // TODO: Handle error
        }
    delete aAttrValue;
    }
```

Called for processing of each service attribute. Check for attributes of UUID type. If the UUID is RFCOMM UUID, resolve the value for this attribute, which will be the channel number to be used for connecting to the remote device.

```
// From MSdpAttributeValueVisitor
void CMyServiceDiscoverer::VisitAttributeValueL(
    CSdpAttrValue &aValue,
    TSdpElementType aType)
    {
    switch (aType)
        {
        case ETypeUUID:
            {
            TPtrC8 uuid(aValue.UUID().ShortestForm());
            iLastUUID.SetL(uuid);
            break;
            }
        case ETypeUint:
            {
            if ( iLastUUID==KRFCOMM )
                {
                // Previous call to this method with rfcomm UUID, therefore
                // this one will be the value, rfcomm service channel (port)
                iPort=aValue.Uint();
                }
            break;
            }
        default:
            // the rest don't really matter..
            break;
        }
    }
```

Called when the request to resolve the service attributes for the service record completes. If there are more service records, proceed by resolving the next

 (C) Copyright Nokia 2013. All rights reserved.

service record.

```
// From MSdpAgentNotifier
void CServiceDiscoverer::AttributeRequestComplete(
    TSdpServRecordHandle aHandle,
    TInt aError)
    {
    if ( aError==KErrNone )
        {
        // Done with attributes for this record, request next
        // service record, this calls NextRecordRequestComplete()
        TRAPD(err,iAgent->NextRecordRequestL());
        if( err )
            {
            // TODO: Handle error
            }
        }
    else
        {
        // TODO: Handle error
        }
    }
```

```
void CMyServiceDiscoverer::FinishDiscovery()
    {
    if(iAgent)
        {
        iAgent->Cancel();
        }
    delete iAgent;
    if(iSpat)
        {
        iSpat->Reset();
        }
    delete iSpat;
    }
```

Postconditions

CSdpAgent finds the specific service (UUID = KBT_serviceID).

See also

Archived:Discovering Bluetooth devices using Symbian C++

Advertising Bluetooth services using Symbian C++

Archived:Establishing a Bluetooth connection using Symbian C++

Archived:RHostResolver and redundant display of access point selection dialog (Known Issue)

Example application

This code snippet has been used in the S60 Platform: Bluetooth Point-to-multipoint Example application.

(C) Copyright Nokia 2013. All rights reserved.

 (C) Copyright Nokia 2013. All rights reserved.