WISE Discover Service

Description

After the devices have been found we need to check whether the device has the WISE service. This operation is done synchronously on the Maemo Platform. On the S60 Platform this operation needs to be done asynchronously using predefined interfaces.

Maemo Platform

```
int search_service(wise_connection_manager* manager,wise_device* device)
{
    uint8_t svc_uuid_int[] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0xab,0xcd };

    uuid_t svc_uuid;
    int err,result=WISE_CONNECTION_ERROR;
    bdaddr_t target;
    sdp_list_t *response_list=NULL,*search_list,*attrid_list;
    sdp_session_t *session=0;

    str2ba(device->BT_address,&target);

    /* connect to the SDP server running on the remote machine */
    session = sdp_connect( BDADDR_ANY, &target, SDP_RETRY_IF_BUSY );
    if ( session==NULL ) return WISE_CONNECTION_ERROR;

    /* specify the UUID of the application we're searching for */
    sdp_uuid128_create(&svc_uuid,&svc_uuid_int);
    search_list = sdp_list_append(NULL,&svc_uuid);

    /* specify that we want a list of all the matching applications' attributes */
    uint32_t range = 0x0000ffff;
    attrid_list = sdp_list_append( NULL, &range );

    /* get a list of service records that have the right UUID */
    err = sdp_service_search_attr_req(session,search_list,
        SDP_ATTR_REQ_RANGE,attrid_list,&response_list);

    /* go through each of the service records */
    for ( ; response_list; response_list=response_list->next )
    {
        sdp_record_t *rec = (sdp_record_t*) response_list->data;
        sdp_list_t *proto_list;

        /* get a list of the protocol sequences */
        if( sdp_get_access_protos( rec, &proto_list ) == 0 )
        {
            sdp_list_t *p = proto_list;

            /* go through each protocol sequence */
            for( ; p ; p = p->next )
            {
                sdp_list_t *pds = (sdp_list_t*)p->data;

                /* go through each protocol list of the protocol sequence */
                for( ; pds ; pds = pds->next )
                {
                    /* check the protocol attributes */
                    sdp_data_t *d = (sdp_data_t*)pds->data;
                    int proto = 0;
                    for( ; d; d = d->next )
                    {
                        switch( d->dtd )
                        {
                            case SDP_UUID16:
                            case SDP_UUID32:
                            case SDP_UUID128:
                                proto = sdp_uuid_to_proto( &d->val.uuid );
                                break;
                            case SDP_UINT8:
                                if( proto == RFCOMM_UUID )
                                {
                                    device->channel = d->val.int8;
                                    result = WISE_OK;
                                }
                                break;
                        }
                    }
                }
                sdp_list_free((sdp_list_t*)p->data,0);
            }
            sdp_list_free(proto_list,0);
        }
        sdp_record_free(rec);
    }
    sdp_close(session);
    return result;
}
```

S60 Platform

```
class CWISEConnectionManager : public CBase,
                               public MSdpAgentNotifier,
                               public MSdpAttributeValueVisitor
{
...
```


(C) Copyright Nokia 2013. All rights reserved.

```
private: // MSdpAgentNotifier

    void NextRecordRequestComplete(TInt aError, TSdpServRecordHandle aHandle,
        TInt aTotalRecordsCount);

    void AttributeRequestResult(TSdpServRecordHandle aHandle,
        TSdpAttributeID aAttrID,
        CSdpAttrValue* aAttrValue);

    void AttributeRequestComplete(TSdpServRecordHandle aHandle, TInt aError);

private: // MSdpAttributeValueVisitor

    void VisitAttributeValueL(CSdpAttrValue& aValue, TSdpElementType aType);
    void StartListL(CSdpAttrValueList &aList);
    void EndListL();
...

private: //data

        // socket server handle
        RSocketServ& iSocketServ;

        MWISEManagerObserver& iObserver;

 // service discovery agent
 CSdpAgent* iAgent;

 // service discovery search pattern
 CSdpSearchPattern* iSearchPattern;

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

 // device data record reference
 TDeviceData* iDevData;
 TInt iDeviceIdx;

 // this indicates that a port number was found in the service attributes
 // and device data record for this device needs to be updated with the
 // port number.
 TBool iServiceFound;

 TInt iServers;

 // device lists
 TDeviceDataList iFoundDevices;


};
```

```
void CWISEConnectionManager::SearchServiceL(TDeviceData* aDevData)
 {
 FinishDiscovery();

 iServiceFound=EFalse;
 iDevData=aDevData;

 // init new service discovery agent
 iAgent = CSdpAgent::NewL( *this, iDevData->iDeviceAddr );

 // set search properties for agent
 iSearchPattern = CSdpSearchPattern::NewL();

 // use our service id to filter the services discovered
 // -> will return only the services with matching service id(s)
 //TUUID serviceUUID(KServiceID);
 TUUID serviceUUID(0x0000,0x0000,0x0000,0xabcd);

 iSearchPattern->AddL(serviceUUID);
 iAgent->SetRecordFilterL(*iSearchPattern);

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

```
void CWISEConnectionManager::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..
  TRAPD(err,iAgent->AttributeRequestL(aHandle,
                        KSdpAttrIdProtocolDescriptorList) );
  }
 else
  {
  if ( iServiceFound )
   {
   AddServerL(iDevData);
   iFoundDevices.Remove(iDeviceIdx);
   }
  else
   {
   iDeviceIdx++;
   }
     if ( iDeviceIdx<iFoundDevices.Count() )
      {
      TRAPD(err,SearchServiceL(iFoundDevices[iDeviceIdx]));
      }
     else
      {
      FinishDiscovery();
      iObserver.HandleServiceDiscoveryEndedL(EFalse);
      }
     }
 }
```

```
void CWISEConnectionManager::FinishDiscovery()
```


(C) Copyright Nokia 2013. All rights reserved.

```
 {
 if( iAgent ) iAgent->Cancel();
 delete iAgent;
 iAgent=NULL;

 if( iSearchPattern ) iSearchPattern->Reset();
 delete iSearchPattern;
 iSearchPattern=NULL;
 }
```

```
void CWISEConnectionManager::AttributeRequestResult(
 TSdpServRecordHandle /*aHandle*/,
 TSdpAttributeID /*aAttrID*/,
 CSdpAttrValue* aAttrValue)
 {
 // parse attributes, will result in call to VisitAttributeValue()
 TRAPD(err,aAttrValue->AcceptVisitorL(*this) );
 }
```

```
void CWISEConnectionManager::AttributeRequestComplete(
 TSdpServRecordHandle /*aHandle*/,
 TInt aError)
 {
 if ( aError==KErrNone )
  {
  // done with attributes for this record, request next service record
  TRAPD(err, iAgent->NextRecordRequestL());
  }
 }
```

```
void CWISEConnectionManager::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)
    iDevData->iDeviceServicePort=aValue.Uint();
    // mark device data changed, so the device data record
    // in device data list will be updated.
    iServiceFound=ETrue;
   }
   break;
   }
  default:
   break;
  }
 }
```

```
void CWISEConnectionManager::StartListL(CSdpAttrValueList& /*aList*/)
 {
 }
```

```
void CWISEConnectionManager::EndListL()
 {
 }
```

Links

- Archived:Wireless Information Sharing Engine
- WISE Protocol
- WISE Discover Service

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.