Windows Search Protocol & Samba

Noel Power

noel.power@suse.com

Agenda

- Overview
- History
- Some WSP protocol info
- Approach to a WIP Implementation
- Demo
- Questions

Windows Search (some background)

Overview

“Windows Search is a desktop search platform that has instant search capabilities for most common file and data types such as email, contacts, calendar appointments, documents, photos, multimedia etc. These capabilities enable users to find, manage, and organize the increasing amount of data common in home and enterprise environments.” - MSDN

History

There have been a number incarnations of indexing technologies on Windows.

- Indexing service shipped with Windows NT 4.0 (option pack) (first shipped in the late 90's), Windows 2000 & later.
- Windows Desktop Search (Windows XP, Windows 2000 & Windows server 2003 (shipped as an addin)
- Instant search on Vista
- Windows Search (an addin for XP) and included standard from Vista onwards

Windows Search

- Windows Search Service (WSS)
- Development platform
- User interface

Windows Search Service (WSS)

- Builds an index (from a selected location(s)) of a collection of documents by
  - Analyzing files
  - Extracting content, properties & meta data
- Maintains a single index shared by all users
- Maintains security restrictions on content access
- Process remote queries from client computers on the network.

Windows Search UI

- Integrated into all Windows explorer windows (Vista & later)
- Incremental search (“search as you type”) that continuously refines your search as you type
- Enhanced column headers in Windows Explorer views enable sorting and grouping documents in different ways. For example, results can be sorted according to name, date modified, type, size, and tag
- Searches can be saved (to be retrieved later). The results will be dynamically repopulated based on the original criteria when the saved search is opened.

Windows Search UI

- Preview handlers and thumbnail handlers enable users to preview documents in Windows Explorer without having to open the application that created them.

Windows Search Queries

- Advanced Query Syntax (AQS)
- Natural Query Syntax (NQS)
- Structured Query Language (SQL) (well actually Windows Search SQL)
- Structured query interfaces (programmatic support for building queries)
- search-ms protocol, allows queries to be expressed in terms of parameter/value arguments and supports all of the above

Query examples

- Advanced Query Syntax
  - *“search phrase(s)" System.Author:(npower OR noel) System.ItemFolderNameDisplay:C:”\MyDocs”*
- Windows Search SQL
  - *"SELECT Path FROM UserA-4.SystemIndex.Scope() WHERE "SCOPE"= 'file://UserA-4/Users/UserA/Pictures' AND CONTAINS(*, '"flowers"')"*
- Natural Query Syntax (NQS)
  - *"Documents created last week by npower"*
- search-ms protocol
  - *search-ms::query=flowers&crumb=kind:pics*

WSP Protocol

Windows Search Protocol

- Allows a client to issue queries to a server hosting the Windows Search service.
- The protocol is primarily intended to be used for full-text queries.
- Presents the query in a binary representation of what looks most like Windows Search SQL than the other query dialects.
- Uses SMB pipe protocol
- Has a dedicated pipe \pipe\MSFTEWDS allocated for this protocol

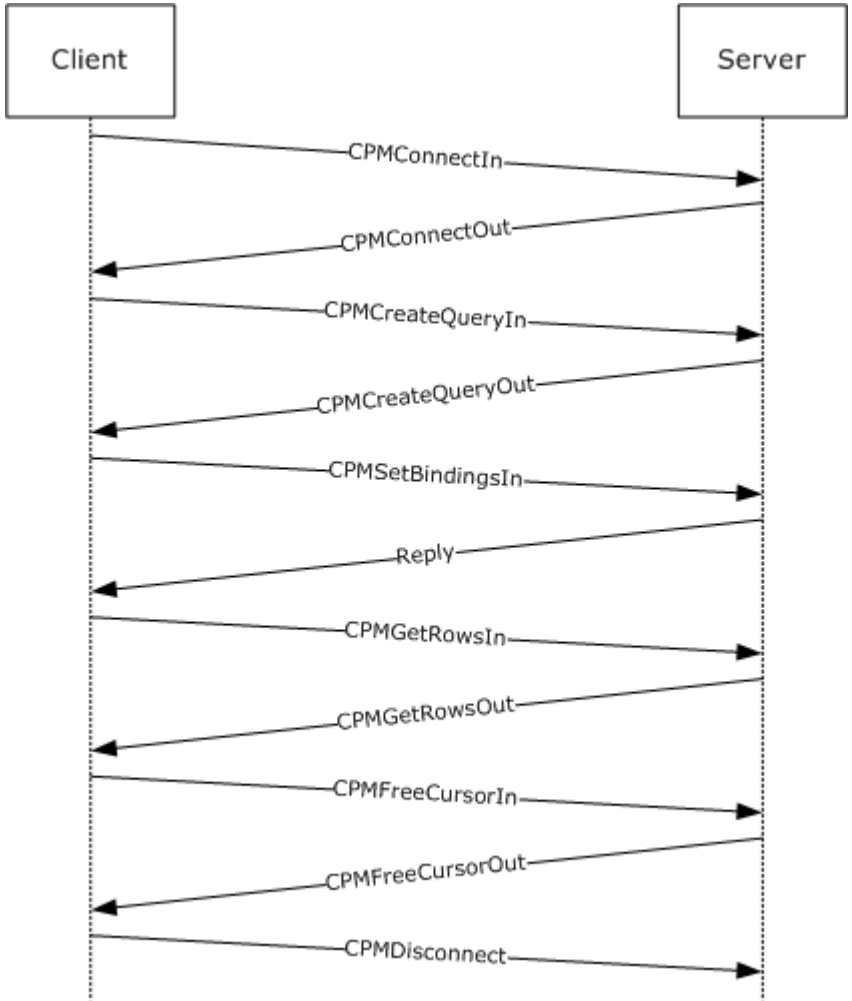
The Protocol (some details)

- Most messages are client initiated
- In general simple request/response model (there seems maybe some exceptions e.g. GetRowNotify)
- Most of the useful info for the server probably encapsulated by the following messages
  - CPMConnectin
  - CPMCreateQueryIn
  - CPMSetBindingsIn
  - CPMGetRows

Query messages

- CPMConnectin
  - Specifies catalog name and configuration information (query-type, locale, search in folders.
- CPMCreateQuery
  - Specifies the restrictions, groupings, sorting, other query related config info
- CPMSetBindings
  - Specifies the columns to be returned
- CPMGetRows
  - Returns rows for a specific cursor, allows seeking to specific bookmarks

The Protocol (simplified exchange)



Why?

- Simply curiosity! and an urge to learn more about samba.
- Choosing WSP as a protocol to implement was mostly random (result of seeing some question on a mailinglist)
- Samba already talks named pipes (albeit a layer on top e.g. DCERPC).
- On linux there are already indexers such as Beagle &Tracker that seem conceptually similar enough satisfy at least basic search requests.

Why?

- If either the location searched isn't indexed or connecting to remote WSS fails then search falls back to directly accessing files. This means we have nothing to lose if we can't handle some queries but lots to gain (speed/bandwidth etc.) - Not strictly true yet (still missing some detail to make this work consistently)
- Seamless integration with Windows Search UI
- We should get the use of AQS & Windows SQL for free (will be already converted to the binary query by the time the server gets it) NQS ? (at least some experiments indicate client does convert that)

WIP Prototype

Approach

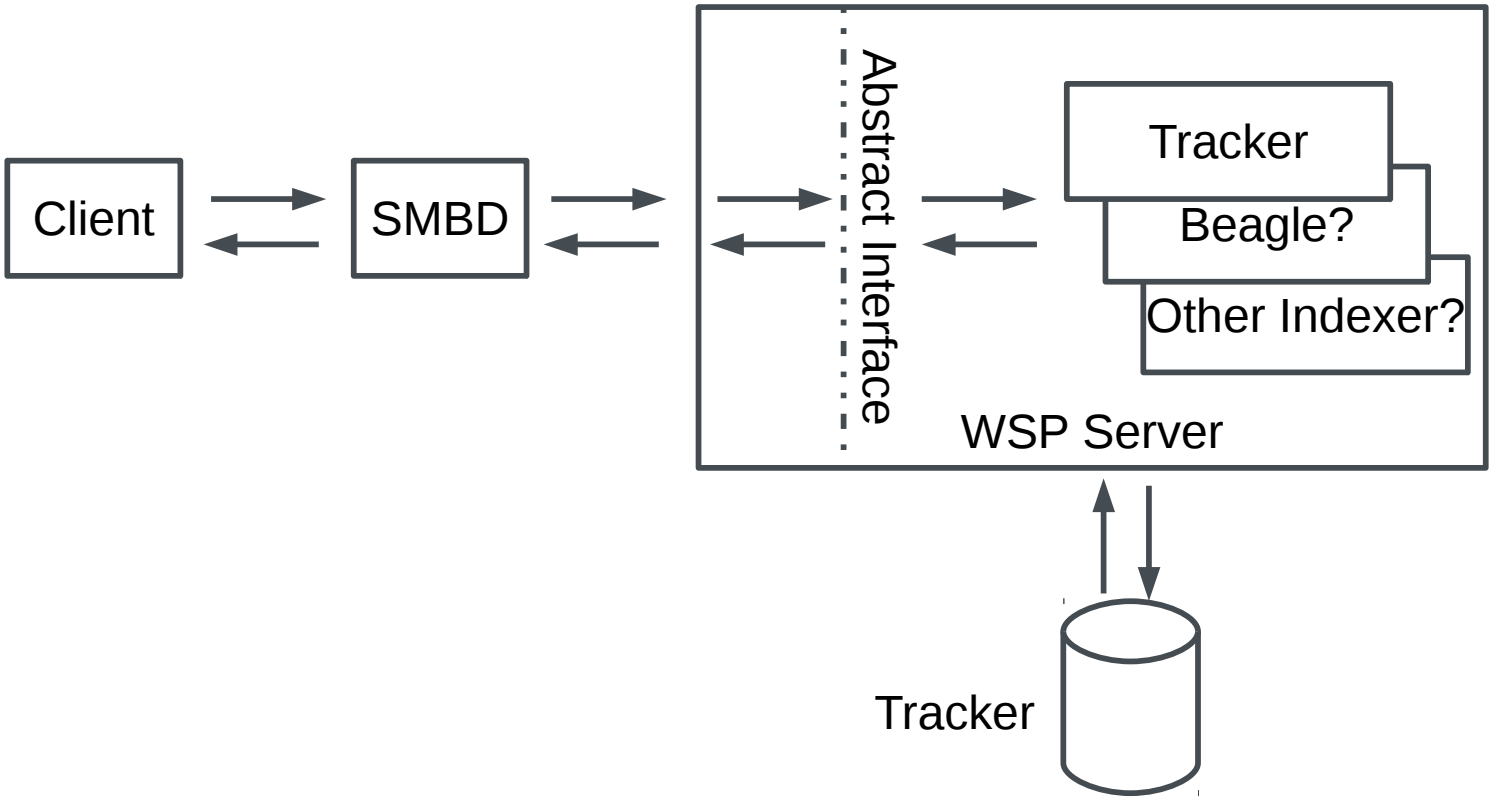
- Create marshalling and unmarshalling routines for all the messages and structures described in the protocol document
- Some tools/code to parse the WSP message structures in order extract useful (and human readable) info.
- Create a simple client to inject messages to a windows server to verify both message content and structure.
- Try and identify the elements from a search that can be translated into a tracker query

Approach

- Gather lots of wireshark dumps of the standard searches available for the search UI
- Create a simple WIP server to achieve the following goals
  - Understand better message and data interaction
  - Allow piecemeal integration with say 'tracker' with the goal of providing basic search capability for some standard queries (e.g. search Documents, Pictures. Videos, Music)
  - Develop mapping/translation functionality to convert the Windows Search query to 'tracker' sparql
  - Prove viability of using tracker to satisfy standard queries from windows integrated search UI.
  - Provide a basis for a generic extensible solution for a 'real' server

Approach

- Architecture for WSP server daemon



Tracker Challenges

- Converting binary query related messages into sparql
  - Extract restriction expression tree from CPMCreateQuery msg
  - Extract column bindings from CPMSetBindings
- Integration of SMBD/tevent with libtracker-sparql with uses glib event loop
- Tracker essentially runs 'per' user but we really need a 'system' wide tracker.
- Tracker dbus-context means smbd needs to launched 'specially' otherwise smbd can't communicate with tracker

Converting WSP query to SPARQL

- Simple SPARQL

```
SELECT ?name
WHERE {
  ?x foaf:name ?name .
}
FILTER (fn:starts-with(?name,'foo'))
```

- Extract the elements of the 'SELECT' statement from both the CPMCreateQuery & CPMSetBindings messages
- Extract the 'WHERE' & 'FILTER' clauses from the CPMCreateQuery message
- For simplicity I use a 'catch-all' matching graph pattern
  - WHERE { ?urn nie:url ?url .} and then append a FILTER generated from the binary query restriction set

General Properties

Windows	***Value returned WSP server***
System.ParsingName	Generated from nfo:fileName
System.ItemDisplay	Generated from nfo:fileName
Path	Generated from nie:url
System.DateModified	nfo:FileLastModified
System.DateAccessed	nfo:FileLastAccessed
System,DateCreated	nfo:FileCreated
System.Size	nfo:fileSize
System.ItemType	Generated from nfo:fileName
System.Kind	Generated from nie:mimeType
System.EntryID	Generated by the server
And many more.....	

Window8.1 Search Videos example

infix restriction expression

"(RTPROPERTY System.Kind = 'video' && (!RTPROPERTY System.Shell.SFGAOFlagsStrings = 'hidden' && RTPROPERTY System.Shell.OmitFromView != 'true') && RTPROPERTY Scope = ' file://old-trouble/testshare/')"

Converted tracker sparql FILTER expression

"(?type IN (nfo:Video) && regex(nie:url(?u),'^ file:///data7/test-share-smaller/'))"

And finally into full tracker sparql query

"SELECT nie:isStoredAs(?u) nfo:fileName(?u) nie:mimeType(?u) nie:url(?u) nfo:fileLastModified(?u) nfo:fileLastAccessed(?u) nfo:fileSize(?u) WHERE{?u nie:url ?url . ?u rdf:type ?type FILTER(?type IN (nfo:Video) && regex(nie:url(?u),'^file:///data7/test-share-smaller/'))}"

Windows8.1 Freetext search example

Infix restriction expression

"((((((((((((((RTPROPERTY System.ItemNameDisplay = 'john' || RTPROPERTY System.ItemAuthors = 'john') || RTPROPERTY System.Keywords = 'john') || RTPROPERTY f29f85e0-4ff9-1068-ab91-08002b27b3d9/24 = 'john') || RTPROPERTY f29f85e0-4ff9-1068-ab91-08002b27b3d9/26 = 'john') || RTPROPERTY System.Music.AlbumTitle = 'john') || RTPROPERTY System.Title = 'john') || RTPROPERTY System.Music.Genre = 'john') || RTPROPERTY System.Message.FromName = 'john') || RTPROPERTY System.Subject = 'john') || RTPROPERTY System.Contact.FullName = 'john') || RTCONTENT 00000000-0000-0000-0000-000000000000/#MRPROPS equals john) || RTCONTENT 00000000-0000-0000-0000-000000000000/#MRPROPS starts with john) || RTCONTENT All equals john) || RTCONTENT All starts with john) && insert expression for WHEREID = 36)"

Note: Whereid refers to a previously encountered restriction set that is to be reused (but not shown here)

Windows8.1 Freetext search example

Full tracker sparql query

"SELECT nie:isStoredAs(?u) nfo:fileName(?u) nie:mimeType(?u) nie:url(?u) nfo:fileLastModified(?u) nfo:fileLastAccessed(?u) nfo:fileSize(?u) WHERE{?u nie:url ?url FILTER((((((((((((((nfo:fileName(?u) = 'john')))) || nmm:musicAlbum(?u) = 'john') || nie:title(?u) = 'john') || nmm:genre(?u) = 'john') || nmo:from(?u) = 'john') || nmo:messageSubject(?u) = 'john')))) || nie:plainTextContent(?u) = 'john' || nie:title(?f) = 'john') || fn:contains(fn:lower-case(nie:plainTextContent(?u)), 'john') || fn:contains(fn:lower-case(nie:title(?u)),'john') || fn:starts-with(fn:lower-case(nfo:fileName(?u)), 'john')) && (regex(nie:url(?u),'^file:///data7/test-share-smaller/'))}"

Note: WhereId expression has been expanded above

Windows Search Protocol Implementation

- Issues
  - Lots of structures > 50 defined (but actually even more due to implementation issues).
  - Quite a few messages built on top.
  - Protocol is biased towards the windows search service implementation.
- Protocol documentation errors and/or ambiguities
  - Feature mismatch between linux indexer and WSS
    - Complex restrictions (vector modeling, probalistic ranking etc.)
    - complexity of mapping 'handles', Document IDs, and cursor iteration
  - Some SMB DCERPC related problems
    - WaitNamedPipe, not handled for wsp pipe
    - max_data, transaction layer hardcoded to DCERPC fragment size

Windows Search Protocol Implementation

- Issues
  - Bookmarks
    - A marker that uniquely identifies a row within a set of rows
  - Chapter
    - A range of rows within a set of rows.
  - WorkId
    - a document ID identifying a document within a result set

Documentation troubles

Some documentation issues

- Some structures have extra (or different layout) data e.g. CSortSet seems to have an extra unexplained 8 bytes
- Some critical messages structures e.g. CRowVariant documentation is at worst incorrect at best ambiguous regarding real implementation usage.
- First documentation problem is the requirement of a impersonationlevel of SECURITY_IDENTIFICATION while it seem that actually SECURITY_IMPERSONATION (unfortunate consequence of this is things work 'till a point and then fail for inexplicable reasons)
- Protocol documentation issues

IDL TROUBLES

Marshalling and UnMarshalling

- The > 30 messages that make up the protocol are based on > 50 interconnected and interdependent structures
- To hand code or not to handcode that is the question?
  - Well, I totally Failed to handcode, too error prone
  - Alternatives? Use pidl from samba – unfortunately the following make representing the structures... troublesome
    - Elements of messages structures that depend on dynamic runtime info
    - Padding
    - Recurstive/nested structures
- Because of the interdependence between the structures, generating is an all or nothing task.

Where are we

Progress

- A nearly complete Wireshark dissector based on Gregor Beck's original version [1]
- A simple server [2] that
  - Has all structures and messages represented in idl (working with patched pidl)
  - Implements the basic messages of WSP protocol
  - Has a specific concrete Tracker implementation
  - Is capable of servicing some standard queries (video, music, document, pictures)

[1] branch: wsp-hacking-v2 repo: ssh://people.freedesktop.org/~noelp/noelp-wireshark-wsp
[2] branch: wsp-hacking-v2 repo: ssh://people.freedesktop.org/~noelp/noelp-wireshark-wsp

Demo

Conclusion

- Was interesting to look into this
- Learned quite a lot (but possibly not exactly what I intended to learn about)
- It certainly is possible to service basic WSP queries with Tracker and the indexer.
- There are concerns
  - Scalablilty
  - Performance (e.g cursor iteration with large datasets)
  - Difficulty in translating queries to tracker-sparql

TODO

- Make wsp server async
- Increase WSP/Windows property mapping to tracker conversion
- Smarter/Better Tracker Sparql generation
- Decide on server architecture
  - Child per client or just one instance
  - Tracker part in separate process or thread
  - Single tracker connection or tracker connection per child
  - Special Tracker user ?

Questions?