C

The SMB URL

There's a fly on the frog
On the bump on the log
In the hole in the middle of the sea.

— Traditional Folk Song

C.1 The Origins of the SMB URL

The idea of a URL scheme designed specifically for use with CIFS had been kicked around before, but it was Richard Sharpe of the Samba Team who finally pushed folks into digging a foundation and pouring concrete. Richard proposed the idea to the readership of the Samba Technical mailing list, and a lively discussion ensued. It took only a short while to work out the basic design of the SMB URL, and most of those involved agreed that the rough-draft plans were a good start. Richard then began work on a prototype implementation to be included in Samba's `libsmbclient` library, and yours truly started work on an Internet Draft for submission to the IETF.

In the broader CIFS community, however, the idea received mixed reviews. Some thought that a URL scheme for use with SMB was a silly waste of time. Others liked the idea so much that they started construction before the foundation was complete, building their implementations on little more than the nominal "specification" hammered out in the mailing list discussions.

So much for the standards process...

Fortunately, the early adopters were also CIFS-savvy folk, so as *de facto* standards go, the SMB URL isn't all that bad. At the very least it can be said that the known problems with the SMB URL are rooted firmly in the bedrock of the CIFS suite itself, and that the URL scheme doesn't do anything to make matters worse.

C.2 Of Round Pegs, Square Holes, and Big Mallets

The SMB URL might have turned out to be fairly simple and straight-forward, but this is CIFS we're talking about. CIFS is a complex protocol suite, and the requirements for the new URL scheme quickly became proportionally complex. Some of the things people wanted from the SMB URL included the ability to:

- specify SMB resources available via NBT *and* naked TCP transport,
- list all available NBT Workgroups,
- list the servers within an NBT Workgroup, and
- locate Active Directory servers and list the file servers within an Active Directory (W2K) domain.

In addition, there was a general hope that the SMB URL would look and feel a lot like the older UNC format used by Windows and OS/2, while still retaining all of the virtues of the more modern, user-friendly, and familiar URL format.

That's a lot to cram into a single URL scheme.

Although the basic design of the SMB URL took only a week or so to work out, some of the finer points required a lot more discussion. In fact, as of this writing the SMB URL Internet Draft is on its fourth revision and clearly needs to be overhauled at least one more time. The need for an update is due in part to the fact that the author didn't know much about writing IETF Internet Drafts when he started. It is also true, however, that a number of fiddly issues needed to be addressed — things like ensuring that the SMB URL scheme conformed to the general URI syntax, and annoying stuff like that.

The following discussion should, therefore, be considered unreliable. See the most current SMB URL Internet Draft or (some day, hopefully) SMB URL RFC.

C.3 Form Versus Function

The basic syntax of an absolute SMB URL looks something like this:

```
smb://[[[authdomain;]user@]host[:port][/share[/path][/name]]][?context]
```

The stuff in brackets is optional, of course, and there are a lot of brackets. That means that there's a lot of potential variety in the formation of SMB URLs. Note, too, that this is the format for the *absolute* form of the URL. An implementation should also support relative URLs.[1]

One of the fiddly bits that had to be handled when designing the SMB URL was whether the scheme identifier should be "SMB" or "CIFS". There wasn't a lot of argument over this. People just started using whichever they liked, so both were declared acceptable. In other words, "`smb://`" and "`cifs://`" both mean the same thing (and implementations should support both). We'll use "`smb`" here because it is more common (and because that's the one the author likes).

`smb://`

With no host specified, this form of the SMB URL indicates the local SMB filesharing network. In practical terms, it means the set of NBT Workgroups on the local subnet.

The way to handle this is to send an NBT broadcast query for the `\x01\x02__MSBROWSE__\x02` name, thus locating any Local Master Browsers on the subnet. Query one or more of the LMBs for the list of known Workgroups, and report the results.

This form of the URL does not currently have a defined meaning in an Active Directory environment. The suggestion is that it might be used to find an Active Directory server, using the client's own fully qualified DNS domain name as a hint. If a server is found, then its W2K domain name would be returned.

1. This discussion assumes a basic knowledge of the workings of URLs and URIs (though it does not assume that you know the difference between a URL and a URI... I can't figure it out myself). For detailed information on URIs, URNs, and URLs see RFC 2396 and RFC 2732.

smb://netbios_name

If the `host` is specified using a NetBIOS name, then it *might* be the name of a Workgroup, or it *might* be the name of an SMB fileserver. The only way to know which is to send a few queries. Three queries, in fact — one for each of three different versions of the name:

- `host<1B>` (unicast),
- `host<1D>` (broadcast), and
- `host<20>`.

The `<20>` names, of course, are registered by SMB fileservers. The `<1B>` and `<1D>` names are registered by the Domain Master Browser and Local Master Browser, respectively.

Finding an SMB server is basic stuff; the browsers are a little bit trickier. LMBs can only be discovered using a broadcast query, but there may not be an LMB for the desired Workgroup on the local LAN. If the Workgroup has a DMB, it can be found by sending a query to the NBNS (assuming that the address of the NBNS is known). Not all Workgroups have a Domain Master Browser, however, so the `<1B>` query may also fail. Querying for both browser types simply increases the odds of finding something usable.

If, after all that, the `netbios_name` resolves to a Workgroup name, then the LMB or DMB should be asked for its list of member servers. If the URL resolves to an SMB fileserver, then the fileserver should be queried for the list of shares offered by the server.

There are rare cases in which the `netbios_name` may resolve to both a fileserver name and a Workgroup name. This is generally caused by a misconfigured NBT network. The recommended way to handle this situation is to issue a warning so that the user knows that there is a problem, but then go ahead and list both the servers in the Workgroup and the shares offered by the server. A tool with a graphical interface could, for example, provide different icons to distinguish the differnet object types as shown in Figure C.1.[2]

2. The network depicted in Figure C.1 is obviously poorly managed. Coffee. Pthah.

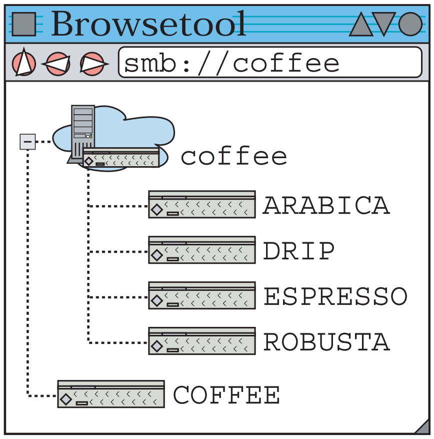

Figure C.1: *Overloading in action*

It is rare, but possible in a misconfigured NBT network, that a Workgroup and an SMB file-server will share the same NetBIOS base name. The client application should probably make an effort to help the user sort things out.

`smb://dns_name`

If the `host` is specified as a DNS name or an IP address, then it can't represent an NBT Workgroup because Workgroups can only be identified by their NetBIOS names. It might, however, be an Active Directory server (a W2K Domain name).

Once again, the implementor is faced with having to go to the wire to discover the semantics of the URL. In this case, it may be necessary to send an LDAP query to the `host` in addition to attempting SMB connections. The `host` may be a W2K Domain Controller, an SMB server, or both.

Isn't overloading fun?

`smb://host/share`
`smb://host/share/path`
`smb://host/share/name`
`smb://host/share/path/name`

The share is the root of the shared directory tree, path is a subdirectory within the share, and name is a filename. That should all be fairly familiar stuff.

C.4 Additional Parts

Those are the basics, but there are a few more fields that need explaining.

`user`

If given as part of the URL string, the username is separated from the `host` field using an "at" symbol ('@'). For example:

```
smb://cue@cleden/corgi
```

The `user` field is typically included in an SMB URL as an authentication shortcut, relieving the application from having to prompt for it. Note, though, that some SMB URL implementations support a further parsing of the user field into a username and password, e.g.:

```
smb://cue:p%40ssw0rd@cleden/corgi
```

This usage is considered bad practice, because it may encourage people to expose their passwords. Applications that handle SMB URLs should always prompt for a password, and should not support the use of the password field in the SMB URL.

`authdomain`

This is a further refinement of the authentication shortcut offered by the `user` field. The `authdomain` is separated from the user name with a semicolon, as shown in the syntax expression above. As the name suggests, the `authdomain` represents the authentication domain in which the username is valid. The authentication domain may be either a W2K or an NT Domain name.

`port`

This field, delimited by a colon, specifies the TCP port number to which to connect.

`context`

The NBT layer presents some unique problems with regard to the design of a URL scheme. URLs, of course, are intended for use on the Internet, which is an IP-based network. Internet naming is handled by the DNS, and the addresses are all of the IPv4 or IPv6 variety. NBT adds a virtual NetBIOS layer, which brings with it a whole 'nother addressing system plus a set of mechanisms to map the NetBIOS layer onto IP. The mapping requires a bit of context. In particular, the client needs to know:

- the IP address of the NBNS (WINS server), if there is one,
- the `CALLING` name (NetBIOS source address) to use, and
- the `CALLED` name (NetBIOS destination address) to use.

Clients typically gather this information from a configuration file, local host name, or destination name, but these values can be overridden using the `context` field of the SMB URL. The `context` field is set up as a URL query string. It must be at the end of the URL, separated from the rest of the string by a single question mark character. The context is given as a set of keyword/value pairs, separated by semicolons. For example:

```
smb://camarllyn/nell?called=nellie;calling=cue;nbns=10.9.7.3
```

The keywords defined in the fourth revision of the SMB URL IETF Internet Draft are:

- `NBNS` (alias `WINS`),
- `CALLED`,
- `CALLING`, and
- `WORKGROUP` (alias `NTDOMAIN`).

There is little reason to specify a Workgroup name in the `context` when it can be specified in the `host` field instead, so that one may be removed from the list. Others which may be added are:

- `BROADCAST`, to specify the broadcast address for B-mode operations,
- `NODETYPE`, to indicate B, P, M, or H mode behavior, and
- `SCOPEID`, to specify the Scope ID.

Putting the Scope ID into the context instead of including it as part of the NetBIOS name in the host field would greatly simplify the semantic interpretation of SMB URLs.

C.5 A Simple SMB URL Parser

Listing C.1 provides a simple, and not entirely robust, SMB URL parser. A better parser would consume a much larger portion of the time-space continuum than is available for example code. This one is a good place to start.

Listing C.1a: A simple SMB URL parser: `SMB_URL.h`

```
/* Typedefs.
 */

typedef struct
  {
  uchar *ntdomain;
  uchar *user;
  uchar *server;
  uchar *port;
  uchar *share;
  uchar *path;
  uchar *context;
  } smb_url;

/* Function prototypes.
 */

smb_url *smb_urlParse( char *src, smb_url *url );
  /* ---------------------------------------------------- **
   * Parse an SMB URL string into an smb_url structure.
   * The function returns NULL on error.
   * ---------------------------------------------------- **
   */

void smb_urlContent( smb_url *url );
  /* ---------------------------------------------------- **
   * Dump the contents of an smb_url structure,
   * representing a parsed SMB URL string.
   * ---------------------------------------------------- **
   */

/* ------------------------------------------------------ */
```

Listing C.1b: A simple SMB URL parser: `SMB_URL.c`

```
#include <stdio.h>
#include <string.h>

#include "smb_url.h"

smb_url *smb_urlParse( char *src, smb_url *url )
  /* ---------------------------------------------------- **
   * Parse an SMB URL string into an smb_url structure.
   *
   * This is a very, very simplistic URL parser... just
   * enough for demonstration purposes.
   * It does not handle the full syntax of SMB URLs.
   * It only handles absolute URLs, and does not do
   * enough error checking.  You can certainly do better,
   * and superior examples can be found on on the web.
   *
   * The function returns NULL on error.
   * ---------------------------------------------------- **
   */
  {
  int    pos;
  uchar *p;

  /* Clear the smb_url structure first. */
  (void)memset( url, 0, sizeof( smb_url ) );

  /* Check for a correct prefix. */
  pos = 0;
  if( 0 == strncasecmp( "smb://", src, 6 ) )
    pos = 6;
  else
    if( 0 == strncasecmp( "cifs://", src, 7 ) )
      pos = 7;
    else
      return( NULL );

  /* Check for an empty URL ("smb://"). */
  if( '\0' == src[pos] )
    return( url );

  /* Copy the original string so that we can carve it up. */
  src = strdup( &src[pos] );
```

```
/* Separate the server, share, path, and context
 * components.
 */
url->server = src;

/* Look for context. */
p = strrchr( src, '?' );
if( NULL != p )
  {
  *p = '\0';
  url->context = ++p;
  }

/* Share part next. */
p = strchr( src, '/' );
if( NULL != p )
  {
  *p = '\0';
  url->share = ++p;
  /* path part. */
  p = strchr( p, '/' );
  if( NULL != p )
    {
    *p = '\0';
    url->path = ++p;
    }
  }

/* Look for the ntdomain & username subfields
 * in the server string (the Authority field).
 */
p = strchr( url->server, '@' );
if( NULL != p )
  {
  *p = '\0';
  url->user = url->server;
  url->server = ++p;
  /* Split the user field into ntdomain;user */
  p = strchr( url->user, ';' );
  if( NULL != p )
    {
    *p = '\0';
    url->ntdomain = url->user;
    url->user     = ++p;
    }
  }
  /* Look for a port number in the server string. */
  p = strchr( url->server, ':' );
  if( NULL != p )
    {
    *p = '\0';
    url->port = ++p;
    }

  return( url );
  } /* smb_urlParse */

void smb_urlContent( smb_url *url )
  /* ---------------------------------------------------- **
   * Dump the contents of an smb_url structure,
   * representing a parsed SMB URL string.
   * ---------------------------------------------------- **
   */
  {
  if( url->ntdomain )
    (void)printf( "ntdomain: %s\n", url->ntdomain );
  if( url->user )
    (void)printf( "    user: %s\n", url->user );
  if( url->server )
    (void)printf( "  server: %s\n", url->server );
  if( url->port )
    (void)printf( "    port: %s\n", url->port );
  if( url->share )
    (void)printf( "   share: %s\n", url->share );
  if( url->path )
    (void)printf( "    path: %s\n", url->path );
  if( url->context )
    (void)printf( " context: %s\n", url->context );
  } /* smb_urlContent */

/* ------------------------------------------------------ */
```