

## A new DCERPC infrastructure for Samba

<https://wiki.samba.org/index.php/DCERPC>

Stefan Metzmacher <metze@samba.org>

Samba Team / SerNet

2014-09-17

- ▶ What is DCERPC?
- ▶ Where is DCERPC used?
- ▶ Implementations in Samba
- ▶ Why do we need a new implementation?
- ▶ Samba3 vs. Samba4 ...
- ▶ Testing
- ▶ MS-SWN: witness\_AsyncNotify
- ▶ MS-FRS2: frstrans\_BytePipe
- ▶ Project Status
- ▶ Thanks!
- ▶ Questions?

# What is DCE-RPC?

- ▶ Distributed Computing Environment / Remote Procedure Calls
  - ▶ It is an infrastructure to call a function on a remote server
  - ▶ "remote" is connected via some kind of socket (tcp/ip, named pipes, ...)
- ▶ As development environment
  - ▶ Function stubs are typically autogenerated from an Interface Definition Language (IDL)
- ▶ As network protocol defines how:
  - ▶ marshalling of payloads work - transfer syntax (NDR/NDR64)
  - ▶ marshalling of PDUs
  - ▶ PDUs are ordered
  - ▶ authentication and encryption works

# What is DCE-RPC?

- ▶ Distributed Computing Environment / Remote Procedure Calls
  - ▶ It is an infrastructure to call a function on a remote server
  - ▶ "remote" is connected via some kind of socket (tcp/ip, named pipes, ...)
- ▶ As development environment
  - ▶ Function stubs are typically autogenerated from an Interface Definition Language (IDL)
- ▶ As network protocol defines how:
  - ▶ marshalling of payloads work - transfer syntax (NDR/NDR64)
  - ▶ marshalling of PDUs
  - ▶ PDUs are ordered
  - ▶ authentication and encryption works

# What is DCE-RPC?

- ▶ Distributed Computing Environment / Remote Procedure Calls
  - ▶ It is an infrastructure to call a function on a remote server
  - ▶ "remote" is connected via some kind of socket (tcp/ip, named pipes, ...)
- ▶ As development environment
  - ▶ Function stubs are typically autogenerated from an Interface Definition Language (IDL)
- ▶ As network protocol defines how:
  - ▶ marshalling of payloads work - transfer syntax (NDR/NDR64)
  - ▶ marshalling of PDUs
  - ▶ PDUs are ordered
  - ▶ authentication and encryption works

# Where is DCERPC used?

- ▶ DCERPC was designed with a reference implementation and documentation
  - ▶ Most of it is available from [www.opengroup.org](http://www.opengroup.org)
  - ▶ Also see <http://en.wikipedia.org/wiki/DCE/RPC>
- ▶ Adaption of DCERPC in Windows
  - ▶ Windows has extended it, see [MS-RPCE], <http://msdn.microsoft.com/en-us/library/cc243560.aspx>
  - ▶ The SDK is available to everyone to use it, see "RPC Technical Reference", <http://technet.microsoft.com/en-us/library/cc759499.aspx>
- ▶ Typically used for:
  - ▶ Remote administration: Authentication-, File-, DNS-, ... -servers
  - ▶ Printing (Spoolss)
  - ▶ Replication protocols (WINS-REPL, Active Directory, SYSVOL)
  - ▶ Distributed Component Object Model (DCOM), see [https://en.wikipedia.org/wiki/Distributed\\_Component\\_Object\\_Model](https://en.wikipedia.org/wiki/Distributed_Component_Object_Model)

# Where is DCERPC used?

- ▶ DCERPC was designed with a reference implementation and documentation
  - ▶ Most of it is available from [www.opengroup.org](http://www.opengroup.org)
  - ▶ Also see <http://en.wikipedia.org/wiki/DCE/RPC>
- ▶ Adaption of DCERPC in Windows
  - ▶ Windows has extended it, see [MS-RPCE], <http://msdn.microsoft.com/en-us/library/cc243560.aspx>
  - ▶ The SDK is available to everyone to use it, see "RPC Technical Reference", <http://technet.microsoft.com/en-us/library/cc759499.aspx>
- ▶ Typically used for:
  - ▶ Remote administration: Authentication-, File-, DNS-, ... -servers
  - ▶ Printing (Spoolss)
  - ▶ Replication protocols (WINS-REPL, Active Directory, SYSVOL)
  - ▶ Distributed Component Object Model (DCOM), see [https://en.wikipedia.org/wiki/Distributed\\_Component\\_Object\\_Model](https://en.wikipedia.org/wiki/Distributed_Component_Object_Model)

# Where is DCERPC used?

- ▶ DCERPC was designed with a reference implementation and documentation
  - ▶ Most of it is available from [www.opengroup.org](http://www.opengroup.org)
  - ▶ Also see <http://en.wikipedia.org/wiki/DCE/RPC>
- ▶ Adaption of DCERPC in Windows
  - ▶ Windows has extended it, see [MS-RPCE], <http://msdn.microsoft.com/en-us/library/cc243560.aspx>
  - ▶ The SDK is available to everyone to use it, see "RPC Technical Reference", <http://technet.microsoft.com/en-us/library/cc759499.aspx>
- ▶ Typically used for:
  - ▶ Remote administration: Authentication-, File-, DNS-, ... -servers
  - ▶ Printing (Spoolss)
  - ▶ Replication protocols (WINS-REPL, Active Directory, SYSVOL)
  - ▶ Distributed Component Object Model (DCOM), see [https://en.wikipedia.org/wiki/Distributed\\_Component\\_Object\\_Model](https://en.wikipedia.org/wiki/Distributed_Component_Object_Model)



# Implementations in Samba (Part 1)

- ▶ Samba 3.0 (and before)
  - ▶ hand written marshalling code
  - ▶ only implemented what was strictly required by Windows clients
- ▶ Samba pre 4.0 (development branch)
  - ▶ Start from scratch after 3.0.0 was released
  - ▶ First start of an async client library
  - ▶ New server infrastructure allows async execution
  - ▶ Invention of an IDL compiler, pidl
  - ▶ Marshalling, client and server code is now autogenerated
  - ▶ The server is single threaded for all services together!

# Implementations in Samba (Part 1)

- ▶ Samba 3.0 (and before)
  - ▶ hand written marshalling code
  - ▶ only implemented what was strictly required by Windows clients
- ▶ Samba pre 4.0 (development branch)
  - ▶ Start from scratch after 3.0.0 was released
  - ▶ First start of an async client library
  - ▶ New server infrastructure allows async execution
  - ▶ Invention of an IDL compiler, pidl
  - ▶ Marshalling, client and server code is now autogenerated
  - ▶ The server is single threaded for all services together!

# Implementations in Samba (Part 2)

- ▶ Samba 3.2
  - ▶ pidl merged back to the 3.X release stream
  - ▶ generating bindings for the 3.X infrastructure
- ▶ Samba 3.4
  - ▶ Services can be moved to external processes
  - ▶ This `named_pipe_auth` abstraction uses unix sockets to implement SMB named pipes
- ▶ Samba 3.6
  - ▶ pidl generates only one set of client stubs
  - ▶ They're based on a `struct dcerpc_binding_handles` abstraction, with different implementations (`s3`, `s4`, `irpc`, `wbint`)
- ▶ Samba 4.0
  - ▶ The single threaded server also hosts the OpenChange services as an externally provided plugin

# Implementations in Samba (Part 2)

- ▶ Samba 3.2
  - ▶ pidl merged back to the 3.X release stream
  - ▶ generating bindings for the 3.X infrastructure
- ▶ Samba 3.4
  - ▶ Services can be moved to external processes
  - ▶ This named\_pipe\_auth abstraction uses unix sockets to implement SMB named pipes
- ▶ Samba 3.6
  - ▶ pidl generates only one set of client stubs
  - ▶ They're based on a struct dcerpc\_binding\_handles abstraction, with different implementations (s3, s4, irpc, wbint)
- ▶ Samba 4.0
  - ▶ The single threaded server also hosts the OpenChange services as an externally provided plugin

# Implementations in Samba (Part 2)

- ▶ Samba 3.2
  - ▶ pidl merged back to the 3.X release stream
  - ▶ generating bindings for the 3.X infrastructure
- ▶ Samba 3.4
  - ▶ Services can be moved to external processes
  - ▶ This named\_pipe\_auth abstraction uses unix sockets to implement SMB named pipes
- ▶ Samba 3.6
  - ▶ pidl generates only one set of client stubs
  - ▶ They're based on a struct dcerpc\_binding\_handles abstraction, with different implementations (s3, s4, irpc, wbint)
- ▶ Samba 4.0
  - ▶ The single threaded server also hosts the OpenChange services as an externally provided plugin

# Implementations in Samba (Part 2)

- ▶ Samba 3.2
  - ▶ pidl merged back to the 3.X release stream
  - ▶ generating bindings for the 3.X infrastructure
- ▶ Samba 3.4
  - ▶ Services can be moved to external processes
  - ▶ This named\_pipe\_auth abstraction uses unix sockets to implement SMB named pipes
- ▶ Samba 3.6
  - ▶ pidl generates only one set of client stubs
  - ▶ They're based on a struct dcerpc\_binding\_handles abstraction, with different implementations (s3, s4, irpc, wbint)
- ▶ Samba 4.0
  - ▶ The single threaded server also hosts the OpenChange services as an externally provided plugin

# Why do we need a new implementation? (Part 1)

- ▶ A lot of newer services require async processing in the server
  - ▶ Service Witness Protocol [MS-SWN] used for SMB3 fileserver clusters
  - ▶ Print System Asynchronous Remote Protocol [MS-PAR]
  - ▶ Distributed File System Replication Protocol [MS-FRS2] (for SYSVOL)
- ▶ Some services need support for association groups
  - ▶ Multiple transport connections can be bound to an association
  - ▶ Similar to SMB3 Multi-Channel
- ▶ Some services need support for DCERPC [pipe]
  - ▶ A remote procedure call passes a "pipe" as argument to a function
  - ▶ A "pipe" is a stream of chunks (arrays of fixed size elements)
  - ▶ It's terminated by a zero length chunk.
  - ▶ Order: [in] parameters, [in] pipes, [out] pipes, [out] parameters
  - ▶ Used in [MS-FRS2] e.g. RdcGetFileDataAsync() with byte elements.
- ▶ DCERPC callbacks (optional for now)
  - ▶ It's possible for the server to call a callback function to the client
  - ▶ This implies a DCERPC infrastructure needs to be client and server at the same time

# Why do we need a new implementation? (Part 1)

- ▶ A lot of newer services require async processing in the server
  - ▶ Service Witness Protocol [MS-SWN] used for SMB3 fileserver clusters
  - ▶ Print System Asynchronous Remote Protocol [MS-PAR]
  - ▶ Distributed File System Replication Protocol [MS-FRS2] (for SYSVOL)
- ▶ Some services need support for association groups
  - ▶ Multiple transport connections can be bound to an association
  - ▶ Similar to SMB3 Multi-Channel
- ▶ Some services need support for DCERPC [pipe]
  - ▶ A remote procedure call passes a "pipe" as argument to a function
  - ▶ A "pipe" is a stream of chunks (arrays of fixed size elements)
  - ▶ It's terminated by a zero length chunk.
  - ▶ Order: [in] parameters, [in] pipes, [out] pipes, [out] parameters
  - ▶ Used in [MS-FRS2] e.g. RdcGetFileDataAsync() with byte elements.
- ▶ DCERPC callbacks (optional for now)
  - ▶ It's possible for the server to call a callback function to the client
  - ▶ This implies a DCERPC infrastructure needs to be client and server at the same time



# Why do we need a new implementation? (Part 1)

- ▶ A lot of newer services require async processing in the server
  - ▶ Service Witness Protocol [MS-SWN] used for SMB3 fileserver clusters
  - ▶ Print System Asynchronous Remote Protocol [MS-PAR]
  - ▶ Distributed File System Replication Protocol [MS-FRS2] (for SYSVOL)
- ▶ Some services need support for association groups
  - ▶ Multiple transport connections can be bound to an association
  - ▶ Similar to SMB3 Multi-Channel
- ▶ Some services need support for DCERPC [pipe]
  - ▶ A remote procedure call passes a "pipe" as argument to a function
  - ▶ A "pipe" is a stream of chunks (arrays of fixed size elements)
  - ▶ It's terminated by a zero length chunk.
  - ▶ Order: [in] parameters, [in] pipes, [out] pipes, [out] parameters
  - ▶ Used in [MS-FRS2] e.g. RdcGetFileDataAsync() with byte elements.
- ▶ DCERPC callbacks (optional for now)
  - ▶ It's possible for the server to call a callback function to the client
  - ▶ This implies a DCERPC infrastructure needs to be client and server at the same time

# Why do we need a new implementation? (Part 1)

- ▶ A lot of newer services require async processing in the server
  - ▶ Service Witness Protocol [MS-SWN] used for SMB3 fileserver clusters
  - ▶ Print System Asynchronous Remote Protocol [MS-PAR]
  - ▶ Distributed File System Replication Protocol [MS-FRS2] (for SYSVOL)
- ▶ Some services need support for association groups
  - ▶ Multiple transport connections can be bound to an association
  - ▶ Similar to SMB3 Multi-Channel
- ▶ Some services need support for DCERPC [pipe]
  - ▶ A remote procedure call passes a "pipe" as argument to a function
  - ▶ A "pipe" is a stream of chunks (arrays of fixed size elements)
  - ▶ It's terminated by a zero length chunk.
  - ▶ Order: [in] parameters, [in] pipes, [out] pipes, [out] parameters
  - ▶ Used in [MS-FRS2] e.g. RdcGetFileDataAsync() with byte elements.
- ▶ DCERPC callbacks (optional for now)
  - ▶ It's possible for the server to call a callback function to the client
  - ▶ This implies a DCERPC infrastructure needs to be client and server at the same time

# Why do we need a new implementation? (Part 2)

- ▶ Easier to maintain security
  - ▶ The authentication implementations are abstracted by gensec now
- ▶ Header signing
  - ▶ ready for Samba 4.2 in the old infrastructure
  - ▶ depends on support in the gensec backend
- ▶ dcerpc\_sec\_verification\_trailer
  - ▶ Header signing negotiation is protected via a hidden structure after the payload
- ▶ Bindtime feature negotiation
  - ▶ SecurityContextMultiplexingSupported - like multiple session setups
  - ▶ KeepConnectionOnOrphanSupported

# Why do we need a new implementation? (Part 2)

- ▶ Easier to maintain security
  - ▶ The authentication implementations are abstracted by gensec now
- ▶ Header signing
  - ▶ ready for Samba 4.2 in the old infrastructure
  - ▶ depends on support in the gensec backend
- ▶ dcerpc\_sec\_verification\_trailer
  - ▶ Header signing negotiation is protected via a hidden structure after the payload
- ▶ Bindtime feature negotiation
  - ▶ SecurityContextMultiplexingSupported - like multiple session setups
  - ▶ KeepConnectionOnOrphanSupported

# Why do we need a new implementation? (Part 2)

- ▶ Easier to maintain security
  - ▶ The authentication implementations are abstracted by gensec now
- ▶ Header signing
  - ▶ ready for Samba 4.2 in the old infrastructure
  - ▶ depends on support in the gensec backend
- ▶ dcerpc\_sec\_verification\_trailer
  - ▶ Header signing negotiation is protected via a hidden structure after the payload
- ▶ Bindtime feature negotiation
  - ▶ SecurityContextMultiplexingSupported - like multiple session setups
  - ▶ KeepConnectionOnOrphanSupported

# Why do we need a new implementation? (Part 2)

- ▶ Easier to maintain security
  - ▶ The authentication implementations are abstracted by gensec now
- ▶ Header signing
  - ▶ ready for Samba 4.2 in the old infrastructure
  - ▶ depends on support in the gensec backend
- ▶ dcerpc\_sec\_verification\_trailer
  - ▶ Header signing negotiation is protected via a hidden structure after the payload
- ▶ Bindtime feature negotiation
  - ▶ SecurityContextMultiplexingSupported - like multiple session setups
  - ▶ KeepConnectionOnOrphanSupported

- ▶ Goals for Samba:
  - ▶ We have four separate (all incomplete) implementations of DCERPC (two servers and two clients).
  - ▶ The aim is to merge the good parts of all implementations together and extend the result to be more feature complete.
  - ▶ Base the whole infrastructure on talloc, tevent and tstream.
  - ▶ All internals should be fully async.
  - ▶ Flexible process models (single process, pre-fork)
  - ▶ Implement everything we need within Samba
- ▶ Goals useful for others
  - ▶ Make it easier for external projects e.g. OpenChange to use
  - ▶ Try to provide a stable ABI for them
  - ▶ struct dcerpc\_binding became a private structure recently
  - ▶ We will include ncacn\_http support in 4.2 (contributes by the OpenChange developers)

- ▶ Goals for Samba:
  - ▶ We have four separate (all incomplete) implementations of DCERPC (two servers and two clients).
  - ▶ The aim is to merge the good parts of all implementations together and extend the result to be more feature complete.
  - ▶ Base the whole infrastructure on talloc, tevent and tstream.
  - ▶ All internals should be fully async.
  - ▶ Flexible process models (single process, pre-fork)
  - ▶ Implement everything we need within Samba
- ▶ Goals useful for others
  - ▶ Make it easier for external projects e.g. OpenChange to use
  - ▶ Try to provide a stable ABI for them
  - ▶ struct dcerpc\_binding became a private structure recently
  - ▶ We will include ncacn\_http support in 4.2 (contributes by the OpenChange developers)



- ▶ Low-level protocol testing
  - ▶ `python/samba/tests/dcerpc/raw_protocol.py`
  - ▶ This uses our python bindings to marshall PDUs and use raw sockets
- ▶ Rely on our existing application level tests
  - ▶ all `smbtorture` rpc tests
  - ▶ additional python tests

- ▶ Low-level protocol testing
  - ▶ `python/samba/tests/dcerpc/raw_protocol.py`
  - ▶ This uses our python bindings to marshall PDUs and use raw sockets
- ▶ Rely on our existing application level tests
  - ▶ all `smbtorture` rpc tests
  - ▶ additional python tests

# MS-SWN: witness\_AsyncNotify (IDL)

The IDL function definition:

```
WERROR witness_AsyncNotify(  
    [in] policy_handle context_handle,  
    [out] witness_notifyResponse **response  
);
```

The generated C structure:

```
struct witness_AsyncNotify {  
    struct {  
        struct policy_handle context_handle;  
    } in;  
  
    struct {  
        struct witness_notifyResponse **response; /* [ref] */  
        WERROR result;  
    } out;  
};
```

# MS-SWN: witness\_AsyncNotify (IDL)

The IDL function definition:

```
WERROR witness_AsyncNotify(  
    [in] policy_handle context_handle,  
    [out] witness_notifyResponse **response  
);
```

The generated C structure:

```
struct witness_AsyncNotify {  
    struct {  
        struct policy_handle context_handle;  
    } in;  
  
    struct {  
        struct witness_notifyResponse **response; /* [ref] */  
        WERROR result;  
    } out;  
};
```

# MS-SWN: witness\_AsyncNotify (Client)

The structure based client stubs:

```
struct tevent_req *dcerpc_witness_AsyncNotify_r_send(TALLOC_CTX *mem_ctx,
                                                    struct tevent_context *ev,
                                                    struct dcerpc_binding_handle *h,
                                                    struct witness_AsyncNotify *r);
NTSTATUS dcerpc_witness_AsyncNotify_r_recv(struct tevent_req *req, TALLOC_CTX *mem_ctx);
NTSTATUS dcerpc_witness_AsyncNotify_r(struct dcerpc_binding_handle *h,
                                     TALLOC_CTX *mem_ctx,
                                     struct witness_AsyncNotify *r);
```

The argument based client stubs:

```
struct tevent_req *dcerpc_witness_AsyncNotify_send(TALLOC_CTX *mem_ctx,
                                                    struct tevent_context *ev,
                                                    struct dcerpc_binding_handle *h,
                                                    struct policy_handle context_handle /*[in]*/,
                                                    struct witness_notifyResponse **response /*[out,ref]*/);
NTSTATUS dcerpc_witness_AsyncNotify_recv(struct tevent_req *req,
                                       TALLOC_CTX *mem_ctx,
                                       WERROR *result);
NTSTATUS dcerpc_witness_AsyncNotify(struct dcerpc_binding_handle *h,
                                   TALLOC_CTX *mem_ctx,
                                   struct policy_handle context_handle /*[in]*/,
                                   struct witness_notifyResponse **response /*[out,ref]*/,
                                   WERROR *result);
```

# MS-SWN: witness\_AsyncNotify (Client)

The structure based client stubs:

```
struct tevent_req *dcerpc_witness_AsyncNotify_r_send(TALLOC_CTX *mem_ctx,
                                                    struct tevent_context *ev,
                                                    struct dcerpc_binding_handle *h,
                                                    struct witness_AsyncNotify *r);
NTSTATUS dcerpc_witness_AsyncNotify_r_recv(struct tevent_req *req, TALLOC_CTX *mem_ctx);
NTSTATUS dcerpc_witness_AsyncNotify_r(struct dcerpc_binding_handle *h,
                                     TALLOC_CTX *mem_ctx,
                                     struct witness_AsyncNotify *r);
```

The argument based client stubs:

```
struct tevent_req *dcerpc_witness_AsyncNotify_send(TALLOC_CTX *mem_ctx,
                                                    struct tevent_context *ev,
                                                    struct dcerpc_binding_handle *h,
                                                    struct policy_handle context_handle /*[in]*/,
                                                    struct witness_notifyResponse **response /*[out,ref]*/);
NTSTATUS dcerpc_witness_AsyncNotify_recv(struct tevent_req *req,
                                       TALLOC_CTX *mem_ctx,
                                       WERROR *result);
NTSTATUS dcerpc_witness_AsyncNotify(struct dcerpc_binding_handle *h,
                                   TALLOC_CTX *mem_ctx,
                                   struct policy_handle context_handle /*[in]*/,
                                   struct witness_notifyResponse **response /*[out,ref]*/,
                                   WERROR *result);
```

# MS-SWN: witness\_AsyncNotify (Server Part 1)

The '\_state' structure:

```
struct _witness_AsyncNotify_state {
    struct _witness_AsyncNotify_state *prev, *next;
    struct tevent_context *ev;
    struct dcerpc_call_handle *call;
    struct witness_AsyncNotify *r;
    struct tevent_req *req;
};
```

The \_send function:

```
static struct tevent_req *_witness_AsyncNotify_send(TALLOC_CTX *mem_ctx,
                                                    struct tevent_context *ev,
                                                    struct dcerpc_call_handle *call,
                                                    struct witness_AsyncNotify *r)
{
    struct tevent_req *req;
    struct _witness_AsyncNotify_state *state;
    struct witness_service *service = witness_get_service(call);

    /* TODO: ... */

    DLIST_ADD_END(service->pending, state);
    tevent_req_deferr_callback(req, ev);
    return req;
}
```

# MS-SWN: witness\_AsyncNotify (Server Part 1)

The '\_state' structure:

```
struct _witness_AsyncNotify_state {
    struct _witness_AsyncNotify_state *prev, *next;
    struct tevent_context *ev;
    struct dcerpc_call_handle *call;
    struct witness_AsyncNotify *r;
    struct tevent_req *req;
};
```

The \_send function:

```
static struct tevent_req *_witness_AsyncNotify_send(TALLOC_CTX *mem_ctx,
                                                    struct tevent_context *ev,
                                                    struct dcerpc_call_handle *call,
                                                    struct witness_AsyncNotify *r)
{
    struct tevent_req *req;
    struct _witness_AsyncNotify_state *state;
    struct witness_service *service = witness_get_service(call);

    /* TODO: ... */

    DLIST_ADD_END(service->pending, state);
    tevent_req_deferr_callback(req, ev);
    return req;
}
```



# MS-SWN: witness\_AsyncNotify (Server Part 2)

The 'service' structure:

```
struct witness_service {
    /* list of pending requests */
    struct _witness_AsyncNotify_state *pending;
};
```

The service monitor function:

```
static void witness_service_ipchange_handler(struct witness_service *service,
                                             /* TODO: ... */)
{
    while (service->pending != NULL) {
        /* TODO: ... */
        tevent_req_done(service->pending->req);
        /* TODO: ... */
    }
}
```

The \_recv function:

```
static NTSTATUS _witness_AsyncNotify_recv(struct tevent_req *req)
{
    return tevent_req_simple_recv_ntstatus(req);
}
```

# MS-SWN: witness\_AsyncNotify (Server Part 2)

The 'service' structure:

```
struct witness_service {
    /* list of pending requests */
    struct _witness_AsyncNotify_state *pending;
};
```

The service monitor function:

```
static void witness_service_ipchange_handler(struct witness_service *service,
                                             /* TODO: ... */)
{
    while (service->pending != NULL) {
        /* TODO: ... */
        tevent_req_done(service->pending->req);
        /* TODO: ... */
    }
}
```

The \_recv function:

```
static NTSTATUS _witness_AsyncNotify_recv(struct tevent_req *req)
{
    return tevent_req_simple_recv_ntstatus(req);
}
```

# MS-SWN: witness\_AsyncNotify (Server Part 2)

The 'service' structure:

```
struct witness_service {
    /* list of pending requests */
    struct _witness_AsyncNotify_state *pending;
};
```

The service monitor function:

```
static void witness_service_ipchange_handler(struct witness_service *service,
                                             /* TODO: ... */)
{
    while (service->pending != NULL) {
        /* TODO: ... */
        tevent_req_done(service->pending->req);
        /* TODO: ... */
    }
}
```

The \_recv function:

```
static NTSTATUS _witness_AsyncNotify_recv(struct tevent_req *req)
{
    return tevent_req_simple_recv_ntstatus(req);
}
```

# DCERPC server glue

The entry point (dispatch) functions pointers (an async function pair):

```
typedef struct tevent_req *(*dcerpc_call_entry_point_send_fn_t)(TALLOC_CTX *mem_ctx,
                                                              struct tevent_context *ev,
                                                              struct dcerpc_call_handle *call,
                                                              void *r);

typedef NTSTATUS (*dcerpc_call_entry_point_rcv_fn_t)(struct tevent_req *req);
struct dcerpc_call_entry_point_fns {
    dcerpc_call_entry_point_send_fn_t fn_send;
    dcerpc_call_entry_point_rcv_fn_t fn_rcv;
};
```

The entry point vector which represent a logic behind a "manager":

```
struct dcerpc_call_entry_point_vector {
    const char *name;
    const struct ndr_interface_table *table;
    uint32_t num_fns;
    const struct dcerpc_call_entry_point_fns *fns;
};
```

Some structures available on dcerpc\_call\_handle:

```
struct dcerpc_server_context; /* top level abstract context for a server instance */
struct dcerpc_server_auth_type; /* an auth_type including the server credentials state */
struct dcerpc_server_endpoint; /* endpoints for the server to listen on */
struct dcerpc_server_manager; /* registered interface with the entry point vector */
struct dcerpc_server_object; /* object to allow multiple managers per interface */
```

# DCERPC server glue

The entry point (dispatch) functions pointers (an async function pair):

```
typedef struct tevent_req *(*dcerpc_call_entry_point_send_fn_t)(TALLOC_CTX *mem_ctx,
                                                                struct tevent_context *ev,
                                                                struct dcerpc_call_handle *call,
                                                                void *r);

typedef NTSTATUS (*dcerpc_call_entry_point_recv_fn_t)(struct tevent_req *req);
struct dcerpc_call_entry_point_fns {
    dcerpc_call_entry_point_send_fn_t fn_send;
    dcerpc_call_entry_point_recv_fn_t fn_recv;
};
```

The entry point vector which represent a logic behind a "manager":

```
struct dcerpc_call_entry_point_vector {
    const char *name;
    const struct ndr_interface_table *table;
    uint32_t num_fns;
    const struct dcerpc_call_entry_point_fns *fns;
};
```

Some structures available on dcerpc\_call\_handle:

```
struct dcerpc_server_context; /* top level abstract context for a server instance */
struct dcerpc_server_auth_type; /* an auth_type including the server credentials state */
struct dcerpc_server_endpoint; /* endpoints for the server to listen on */
struct dcerpc_server_manager; /* registered interface with the entry point vector */
struct dcerpc_server_object; /* object to allow multiple managers per interface */
```

# DCERPC server glue

The entry point (dispatch) functions pointers (an async function pair):

```
typedef struct tevent_req *(*dcerpc_call_entry_point_send_fn_t)(TALLOC_CTX *mem_ctx,
                                                                struct tevent_context *ev,
                                                                struct dcerpc_call_handle *call,
                                                                void *r);

typedef NTSTATUS (*dcerpc_call_entry_point_recv_fn_t)(struct tevent_req *req);
struct dcerpc_call_entry_point_fns {
    dcerpc_call_entry_point_send_fn_t fn_send;
    dcerpc_call_entry_point_recv_fn_t fn_recv;
};
```

The entry point vector which represent a logic behind a "manager":

```
struct dcerpc_call_entry_point_vector {
    const char *name;
    const struct ndr_interface_table *table;
    uint32_t num_fns;
    const struct dcerpc_call_entry_point_fns *fns;
};
```

Some structures available on dcerpc\_call\_handle:

```
struct dcerpc_server_context; /* top level abstract context for a server instance */
struct dcerpc_server_auth_type; /* an auth_type including the server credentials state */
struct dcerpc_server_endpoint; /* endpoints for the server to listen on */
struct dcerpc_server_manager; /* registered interface with the entry point vector */
struct dcerpc_server_object; /* object to allow multiple managers per interface */
```

# MS-FRS2: frstrans\_BytePipe

The IDL function definition:

```
typedef [flag(NDR_PAHEX)] pipe uint8 frstrans_BytePipe;
```

The generated pipe push/pull functions:

```
struct frstrans_BytePipe_chunk {
    uint32_t count;
    uint8_t *array;
} /* [flag(LIBNDR_PRINT_ARRAY_HEX)] */;
struct frstrans_BytePipe *dcerpc_frstrans_BytePipe_create(TALLOC_CTX *mem_ctx);

struct tevent_req *dcerpc_frstrans_BytePipe_chunk_push_send(TALLOC_CTX *mem_ctx,
    struct tevent_context *ev,
    struct frstrans_BytePipe *p,
    const struct frstrans_BytePipe_chunk *chunk);
NTSTATUS dcerpc_frstrans_BytePipe_chunk_push_recv(struct tevent_req *req);

struct tevent_req *dcerpc_frstrans_BytePipe_chunk_pull_send(TALLOC_CTX *mem_ctx,
    struct tevent_context *ev,
    struct frstrans_BytePipe *p);
NTSTATUS dcerpc_frstrans_BytePipe_chunk_pull_recv(struct tevent_req *req,
    TALLOC_CTX *mem_ctx,
    struct frstrans_BytePipe_chunk **chunk);
};
```

# MS-FRS2: frstrans\_BytePipe

The IDL function definition:

```
typedef [flag(NDR_PAHEX)] pipe uint8 frstrans_BytePipe;
```

The generated pipe push/pull functions:

```
struct frstrans_BytePipe_chunk {
    uint32_t count;
    uint8_t *array;
}/* [flag(LIBNDR_PRINT_ARRAY_HEX)] */;
struct frstrans_BytePipe *dcerpc_frstrans_BytePipe_create(TALLOC_CTX *mem_ctx);

struct tevent_req *dcerpc_frstrans_BytePipe_chunk_push_send(TALLOC_CTX *mem_ctx,
    struct tevent_context *ev,
    struct frstrans_BytePipe *p,
    const struct frstrans_BytePipe_chunk *chunk);
NTSTATUS dcerpc_frstrans_BytePipe_chunk_push_recv(struct tevent_req *req);

struct tevent_req *dcerpc_frstrans_BytePipe_chunk_pull_send(TALLOC_CTX *mem_ctx,
    struct tevent_context *ev,
    struct frstrans_BytePipe *p);
NTSTATUS dcerpc_frstrans_BytePipe_chunk_pull_recv(struct tevent_req *req,
    TALLOC_CTX *mem_ctx,
    struct frstrans_BytePipe_chunk **chunk);
};
```



# MS-FRS2: frstrans\_RawGetFileDataAsync

The IDL function definition:

```
WERROR frstrans_RawGetFileDataAsync(  
    [in,ref] policy_handle *server_context,  
    [out,ref] frstrans_BytePipe *byte_pipe  
);
```

The generated client stub:

```
struct tevent_req *dcerpc_frstrans_RdcGetFileDataAsync_send(TALLOC_CTX *mem_ctx,  
    struct tevent_context *ev,  
    struct dcerpc_binding_handle *h,  
    struct policy_handle *_server_context /*[in,ref]*/,  
    struct frstrans_BytePipe *_byte_pipe /*[out,ref]*/);  
NTSTATUS dcerpc_frstrans_RdcGetFileDataAsync_recv(struct tevent_req *req,  
    TALLOC_CTX *mem_ctx,  
    WERROR *result);
```

# MS-FRS2: frstrans\_RawGetFileDataAsync

The IDL function definition:

```
WERROR frstrans_RawGetFileDataAsync(  
    [in,ref] policy_handle *server_context,  
    [out,ref] frstrans_BytePipe *byte_pipe  
);
```

The generated client stub:

```
struct tevent_req *dcerpc_frstrans_RdcGetFileDataAsync_send(TALLOC_CTX *mem_ctx,  
    struct tevent_context *ev,  
    struct dcerpc_binding_handle *h,  
    struct policy_handle *_server_context /*[in,ref]*/,  
    struct frstrans_BytePipe *_byte_pipe /*[out,ref]*/);  
NTSTATUS dcerpc_frstrans_RdcGetFileDataAsync_recv(struct tevent_req *req,  
    TALLOC_CTX *mem_ctx,  
    WERROR *result);
```

# MS-FRS2: pull service (Part 1)

The pull service code:

```
state->byte_pipe = dcerpc_frstrans_BytePipe_create(state);
if (state->byte_pipe == NULL) {
    return;
}

subreq = dcerpc_frstrans_RawGetFileDataAsync_send(state,
                                                session->service->task->event_ctx,
                                                session->conn->dcerpc_pipe->
                                                binding_handle,
                                                &state->server_context,
                                                state->byte_pipe);

if (subreq == NULL) {
    return;
}
tevent_req_set_callback(subreq, dfsrsrv_download_loop_read_done, state);

subreq2 = dcerpc_frstrans_BytePipe_chunk_pull_send(state,
                                                  session->service->task->event_ctx,
                                                  state->byte_pipe);

if (subreq2 == NULL) {
    return;
}
tevent_req_set_callback(subreq2, dfsrsrv_download_loop_chunk_done, state);
```

## MS-FRS2: pull service (Part 2)

The pull service code:

```
/* TODO: ... */

error = dcerpc_frstrans_BytePipe_chunk_pull_rcv(subreq2, state, &chunk);
TALLOC_FREE(subreq2);
if (!NT_STATUS_IS_OK(error)) {
    return;
}

/* TODO: ... */

if (chunk->count == 0) {
    TALLOC_FREE(chunk);
    return;
}
TALLOC_FREE(chunk);

subreq2 = dcerpc_frstrans_BytePipe_chunk_pull_send(state,
                                                    session->service->task->event_ctx,
                                                    state->byte_pipe);

if (subreq2 == NULL) {
    return;
}
tevent_req_set_callback(subreq2, dfsrsrv_download_loop_chunk_done, state);
```

- ▶ See <https://wiki.samba.org/index.php/DCERPC>
- ▶ Some code is already upstream
- ▶ A lot more in work in progress branches
- ▶ Sadly not much progress since my talk at the SambaXP Conference
- ▶ This is currently a sparetime project
- ▶ Hopefully more progress in the next months

# Thanks!

- ▶ People who assist:
  - ▶ Gregor Beck
  - ▶ Andreas Schneider
  - ▶ Günther Deschner
  - ▶ David Disseldorp
  - ▶ Others

# Questions?

- ▶ Stefan Metzmacher, [metze@samba.org](mailto:metze@samba.org), [sm@sernet.de](mailto:sm@sernet.de)
- ▶ <http://www.sernet.com>

→ **SerNet** sponsor booth