Understanding MAPI

Julien Kerihuel, <j.kerihuel@openchange.org>

Contents

1. Introduction
2. MAPI Overview
3. MAPI Concepts

1 Introduction

1

Introduction

- This introduction provides a quick overview of the underlying technology involved in Exchange protocols:

  - Overview of RPC
  - IDL, NDR
  - MSRPC and Samba4
  - OpenChange

  - This is a mix between existing OpenChange presentation and online resources

Introduction

What is RPC?

- RPC stands for **Remote Procedure Call**
- Technology used for Inter Process Communication (IPC)
- Allows a procedure to be executed on a remote resource:
  - The client send a request with parameters
  - The remote server executes the procedure with client parameters
  - The remote server returns the result back to the client
- Distributed client-server model
- Popular RPC based protocols include NFS, CIFS, ExchangeRPC etc.
- Two main implementation of RPC: ONC RPC and **DCE/RPC**
- Microsoft RPC implementation: **MSRPC**

What is RPC?

- **RPC uses a portmapper service**:
  - First RPC service started
  - Have an assigned port (111 or 135 for Microsoft endpoint mapper)
  - **Perform service lookup**
  - Dynamically assign port for a service
  - Maintain a database of available services

- **RPC is transport independent** and (depending on the implementation) can run on top of:

ncacn_ip_tcp:	TCP/IP transport
ncacn_ip_udp:	UDP/IP transport
ncacn_np:	Named pipes transport
ncacn_http:	HTTP transport
ncalrpc:	Local Inter Process Communication

- **ncacn** is an acronym for **N**etwork **C**omputing **A**rchitecture **C**onnection-Oriented RPC Protocol and these

  - ncacn_* are transport sequences

Introduction

What is a RPC interface?

- **Set of procedures that can invoked remotely**

- Each interface requires:
  - **uuid:** universal unique identifier (e.g. a4f1db00-ca47-1067-b31f-00dd010662da is the EMSMDB uuid)
  - **Version number:** the interface version (e.g. 0.81)

- Furthermore an interface needs to define **endpoints**
  - An endpoint is the name for an entity on one side of a transport connection
  - Defines how the client can connect to the service

- **endpoint("ncacn_ip_tcp:")** specifies the interface accepts TCP/IP transport

- **ncacn_ip_tcp:192.168.102.236[print] is a binding string**

Introduction

Interface definitions

- RPC protocols are not written manually but specified using an IDL
- **IDL stands for Interface Definition Language**
- An IDL **describes a protocol/interface** in a language-neutral way
- An IDL file is processed by a compiler (**midl** for Windows MSRPC, **pidl** for Samba MSRPC):
  - The compiler take a description of an interface as their input
  - use it to generate C code (or any other languages) that can use this interface.
- The compiler can generate stubs for DCE/RPC server code, DCE/RPC client code and even Wireshark dissectors (with pidl)!

Introduction

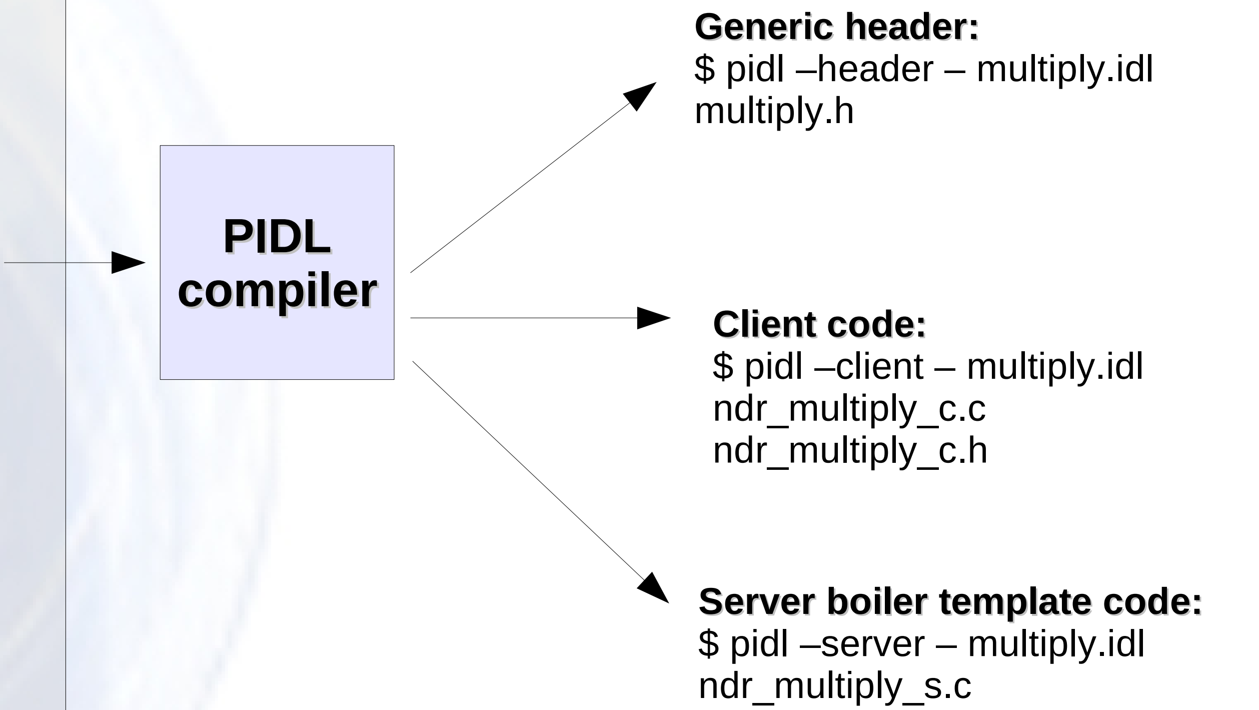
Samba IDL Example

multiply.idl

```
[
  uuid("e2fcfeaa-3730-43de-ad54-6a19daa979f8"),
  pointer_default(unique),
  endpoint("ncacn_ip_tcp:","ncacn_ip_udp"),
  version(1.0),
  helpstring("My Multiply Protocol")
] interface multiply
{
  /* Function 0x0 */
  uint32 Connect (
    [out] policy_handle *handle
  );

  /* Function 0x1 */
  uint32 Multiply (
   [in] policy_handle *handle,
   [in] uint32 a,
   [in] uint32 b,
   [out,ref] uint32 *result
  );

  /* Function 0x2 */
  uint32 Disconnect (
  [in,out] policy_handle *handle
  );
}
```



PIDL compiler

Generic header:
$ pidl –header – multiply.idl
multiply.h

Client code:
$ pidl –client – multiply.idl
ndr_multiply_c.c
ndr_multiply_c.h

Server boiler template code:
$ pidl –server – multiply.idl
ndr_multiply_s.c

Header + client + server boiler template = 448 lines of C code generated

Introduction

Policy Handles

- A policy handle is a connection context
- also known as binding handle
- It is used by the server to uniquely identify a client session
- Hold a handle type and a GUID
- This handle persists all along the session

[in] [out] [ref]

- **In:**
  - identify parameters passed to the calling procedure
  - from the client to the server
- **Out:**
  - identify parameters returned by a remote procedure
  - from the server to the client
- **Ref:**
  - reference a pointer, level of indirection

Introduction

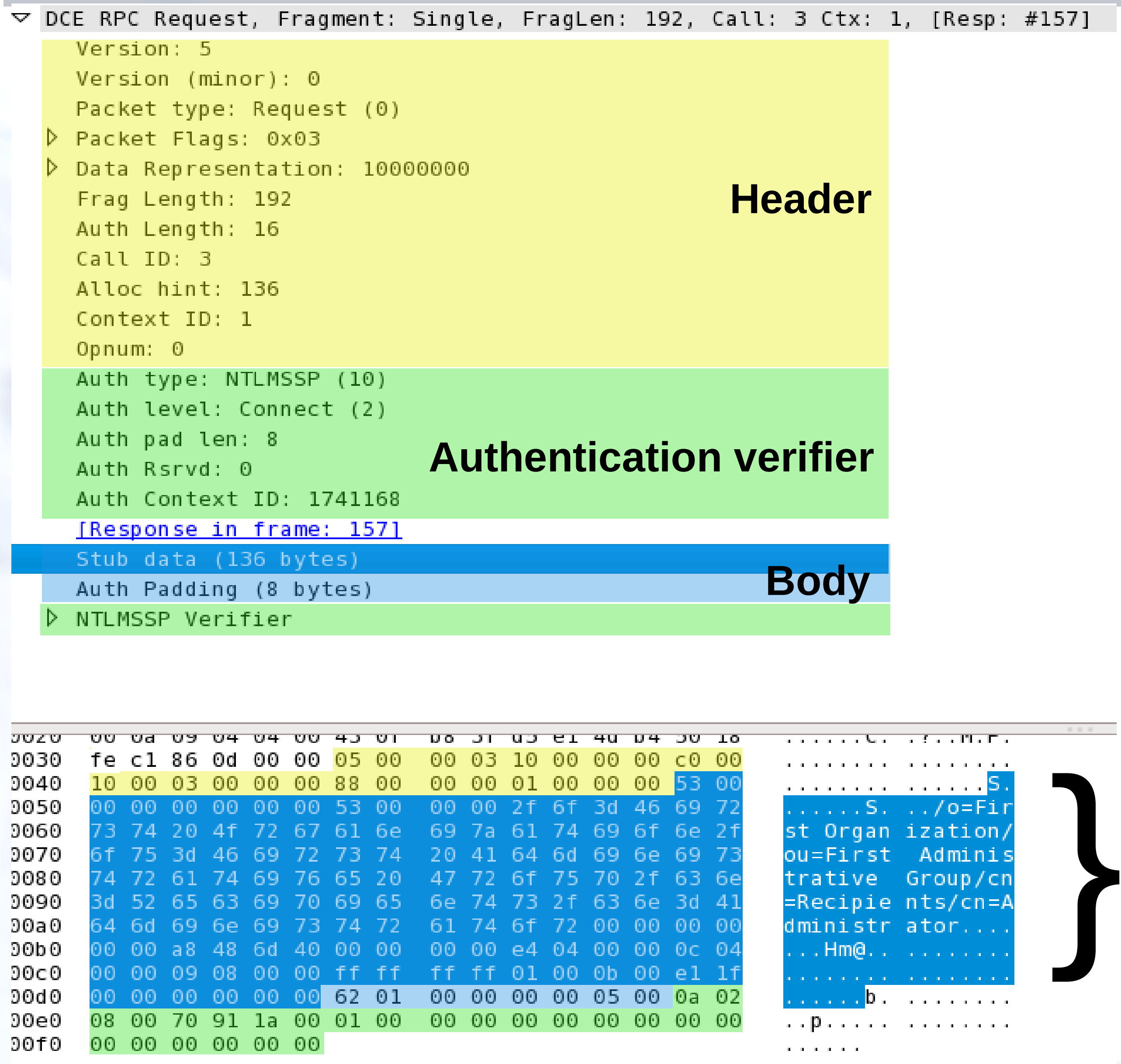
NDR

- NDR is the acronym for **N**etwork **D**ata **R**epresentation
- Implementation of the presentation layer (OSI model)
- **Map IDL data types into octet streams**
- 13 primitives types: (signed and unsigned) int8, int16, int32, int64, float, utf8 strings...
- Can map varying strings, array of strings, blobs, pointers etc.
- Can represent any kind of complex data structure
- NDR is a **Transfer Syntax**
- **It's the only existing transfer syntax for DCE/RPC**
- **NDR encoding negotiated by the client**
- NDR alignment (1,2,4,8)

- NDR blob is a part of the **RPC PDU**:
  - PDU = Protocol Data Units
  - Composed of:
    - Header
    - Body (NDR data)
    - Authentication verifier

- Other representations include: XDR, ASN.1

Introduction



```
▽ DCE RPC Request, Fragment: Single, FragLen: 192, Call: 3 Ctx: 1, [Resp: #157]
     Version: 5
     Version (minor): 0
     Packet type: Request (0)
  ▷ Packet Flags: 0x03
  ▷ Data Representation: 10000000
     Frag Length: 192
     Auth Length: 16
     Call ID: 3
     Alloc hint: 136
     Context ID: 1
     Opnum: 0
     Auth type: NTLMSSP (10)
     Auth level: Connect (2)
     Auth pad len: 8
     Auth Rsrvd: 0
     Auth Context ID: 1741168
     [Response in frame: 157]
     Stub data (136 bytes)
     Auth Padding (8 bytes)
  ▷ NTLMSSP Verifier
```

Header

Authentication verifier

Body

```
0020  00 0a 09 04 04 00 43 01  b8 3f d3 e1 4d b4 30 18   .......C. .?..M.F.
0030  fe c1 86 0d 00 00 05 00  00 03 10 00 00 00 c0 00   ........ ........
0040  10 00 03 00 00 00 88 00  00 00 01 00 00 00 53 00   ........ ......S.
0050  00 00 00 00 00 00 53 00  00 00 2f 6f 3d 46 69 72   ......S. ../o=Fir
0060  73 74 20 4f 72 67 61 6e  69 7a 61 74 69 6f 6e 2f   st Organ ization/
0070  6f 75 3d 46 69 72 73 74  20 41 64 6d 69 6e 69 73   ou=First  Adminis
0080  74 72 61 74 69 76 65 20  47 72 6f 75 70 2f 63 6e   trative  Group/cn
0090  3d 52 65 63 69 70 69 65  6e 74 73 2f 63 6e 3d 41   =Recipie nts/cn=A
00a0  64 6d 69 6e 69 73 74 72  61 74 6f 72 00 00 00 00   dministr ator....
00b0  00 00 a8 48 6d 40 00 00  00 00 e4 04 00 00 0c 04   ...Hm@.. ........
00c0  00 00 09 08 00 00 ff ff  ff ff 01 00 0b 00 e1 1f   ........ ........
00d0  00 00 00 00 00 00 62 01  00 00 00 00 05 00 0a 02   ......b. ........
00e0  08 00 70 91 1a 00 01 00  00 00 00 00 00 00 00 00   ..p..... ........
00f0  00 00 00 00 00 00                                  ......
```

} NDR encoded

Introduction

NDR Stub Data

```
0000   53 00 00 00 00 00 00 00 53 00 00 00 2f 6f 3d 46   S.......S.../o=F
0010   69 72 73 74 20 4f 72 67 61 6e 69 7a 61 74 69 6f   irst Organizatio
0020   6e 2f 6f 75 3d 46 69 72 73 74 20 41 64 6d 69 6e   n/ou=First Admin
0030   69 73 74 72 61 74 69 76 65 20 47 72 6f 75 70 2f   istrative Group/
0040   63 6e 3d 52 65 63 69 70 69 65 6e 74 73 2f 63 6e   cn=Recipients/cn
0050   3d 41 64 6d 69 6e 69 73 74 72 61 74 6f 72 00 00   =Administrator..
0060   00 00 00 00 a8 48 6d 40 00 00 00 00 e4 04 00 00   .....Hm@........
0070   0c 04 00 00 09 08 00 00 ff ff ff ff 01 00 0b 00   ................
0080   e1 1f 00 00 00 00 00 00                           ........
```

NDR string example:

- an ascii string prefixed with [size] [offset] [length], all 32 bits and null terminated
- **`[size 32 bits][offset 32 bits][length 32 bits][/o=First Organiza....\0]`**
  - `size:   53 00 00 00 -> 0x53 -> 83`
  - `offset: 00 00 00 00`
  - `length: 53 00 00 00 -> 0x53 -> 83`
  - `string: /o=First Organization/ou=First Administrative Group/cn=Recipients/cn=Administrator`

Introduction

- **Marshalling and Unmarshalling**
  - Similar to **serialization**
  - To **marshall** means transforming the memory representation of an object to a data format suitable for storage or transmission (wikipedia)
  - To **unmarshall** is the opposite process

Introduction

What is MSRPC?

- Microsoft implementation of the DCE RPC mechanism
- A few words about MSRPC history ...
- Implement additional transport such as **ncacn_np**
- MSRPC uses the **Windows SSPI** (Security Support Provider Interface) to provide security services such as authentication and confidentiality.
- Security services include among others:
  - SPNEGO
  - NTLM
  - Schannel (SSL, PCT, TLS)
  - MS Kerberos
  - MSN SSP

Introduction

MSRPC and Samba4

- Finally we get back to our initial topic
- Samba4 provides an interoperable implementation of the MSRPC stack and Windows protocols
- Samba4 includes PIDL (Perl IDL compiler) with a syntax similar to MIDL (gets closer with time)
- Almost 50% of all Samba4 code is auto-generated
- Makes the implementation more reliable and easier to update/extend/fix
- Samba4 provides client code, but also server implementation for numerous services:
  - Endpoint mapper, Active Directory, MS Kerberos, NBT, SMB, WINS etc.

Introduction

- **What about Microsoft Exchange?**

  - What is Microsoft Exchange?
    - Groupware server working with Outlook
    - Provide features such as messaging, shared calendars, contact databases, public folders, notes, tasks, journal etc.

  - Outlook-Exchange communications are RPC-based

  - They use a set of protocols called ExchangeRPC

  - These ExchangeRPC protocols uses the MSRPC implementation

Introduction

Finally what is OpenChange?

- **Portable implementation of Microsoft Exchange protocols and Microsoft Exchange server**
- What we provide:
  - **Library for interoperability with Microsoft Exchange**
  - **Alternative to Microsoft Exchange Server:**
    - using native Exchange protocols
    - provides exactly equivalent functionality when viewed from Microsoft Outlook clients
- Brief OpenChange history
- Works on top of Samba4. Use a subset of its library:
  - dcerpc and ndr for the dcerpc stack and ndr encapsulation
  - talloc for memory allocation
  - ldb and tdb for database management

2 MAPI overview

MAPI Overview

- **MAPI is the Mail Application Programming Interface for Windows clients:**
  - set of C function calls (think of it as the library interface)
  - existed prior to Exchange being developed
  - **NOT a network protocol**

- **ExchangeRPC:**
  - Proprietary transport protocol for MAPI
  - Closely matches the MAPI calling interface

- 2 main protocols used in MAPI communications:
  - **NSPI**: Address Book Protocol
  - **EMSMDB**: Exchange transport

- There is also the **RFR** protocol used to locate the NSPI server

MAPI Overview

- **NSPI Protocol**
  - Acronym for **N**ame **S**ervice **P**rovider **I**nterface
  - Used while creating a **MAPI profile**:
    - Connection (binding strings)
    - Credentials (username, password, domain, realm)
    - User information (mailbox path)
  - Mainly used by MAPI clients to perform **username lookup**
  - This is a RPC wrapper/proxy over Exchange Active Directory
  - Its interface implements 21 procedures but only 50% are commonly/really used
  - Its data structures are similar to the EMSMDB ones

MAPI Overview

- **EMSMDB Protocol**
  - Exchange Message Provider
  - **99% of the Outlook-Exchange traffic is performed through a single EMSMDB procedure** (EcDoRpc or EcDoRpcEx depending on pipe version)
  - Outlook-Exchange data does not use NDR but a custom and non-aligned encoding
  - EMSMDB pipe version evolved over years

MAPI Overview

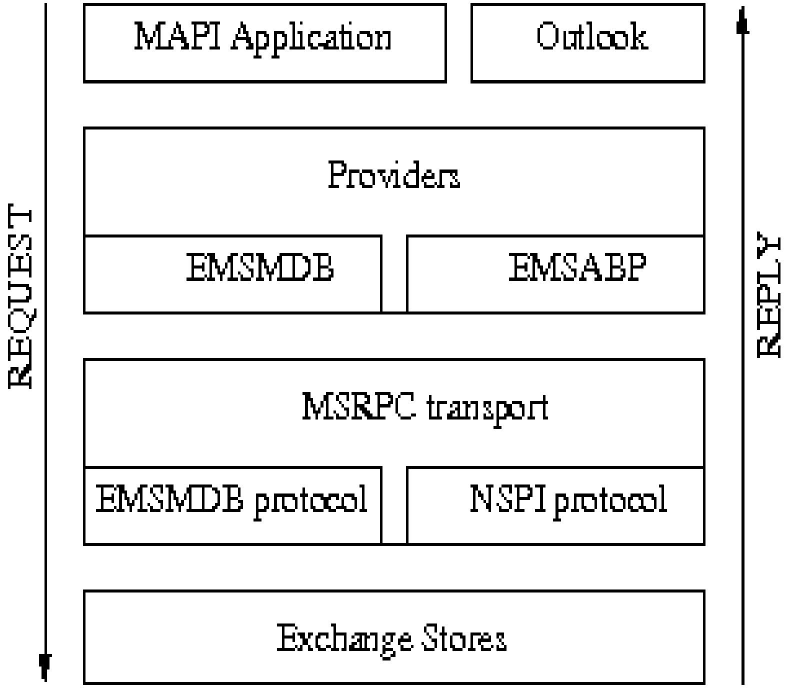
- **Exchange 5.5 to Exchange 2000:**
  - Obfuscated content (xor 0xa5)
  - Use EcDoConnect (0x1) and **EcDoRpc** (0x2)
  - Implemented in libmapi

- **Exchange 2003 to 2007:**
  - AirMAPI compression algorithm introduced (LZXPRESS)
  - New pipe functions introduced:
    - EcDoConnectEx (0xA)
    - **EcDoRpcEx** (0xB)
  - Some packets are still obfuscated with ***xor 0xA5*** while others are compressed

MAPI Overview

Describing a MAPI conversation:

- **Client-side:**
  - MAPI applications call MAPI providers, using the API to pass data
  - MAPI providers pack the client or server MAPI information in a blob
  - ExchangeRPC protocol is used to transport the MAPI information, using one of two MAPI-specific protocols:
    - **EMSMDB Message Store Protocol**
    - **NSPI Address book Protocol**
- **Store provider on server side:**
  - extracts the MAPI blob from RPC
  - protocol functions analyzes its content
  - performs operations embedded within it



3 MAPI Concepts

3

MAPI Concepts

- Exposing all MAPI concepts within few slides wouldn't be reasonable
- Instead fundamental MAPI concepts will be exposed and some example of the protocol scope provided
- This section covers MAPI object, handles, properties and expose some part of the implementation for MAPI tables and streams.

MAPI Concepts

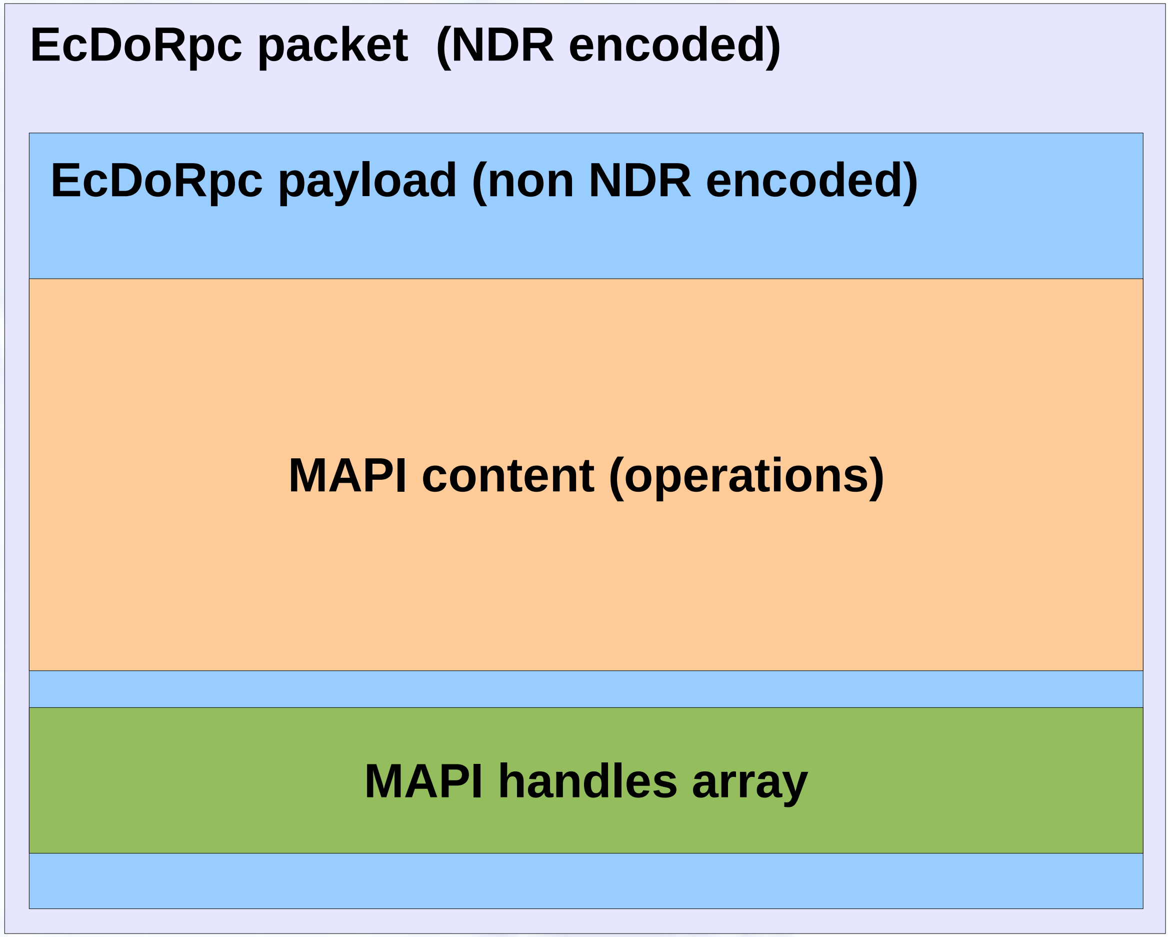
- **MAPI Objects:**
  - **Any MAPI data you access is associated to an object**
  - **Objects are generic**:
    - They can be considered as an array of rows with 2 columns:
      - Column 1: property (unsigned 32 bit integer)
      - Column 2: property data
    - Only difference between objects: methods you can use – depends on the context (table, folder, message, streams etc.)
    - **Objects are within a hierarchy** and are all (at different levels) children of the top Message Store object.
  - **MAPI Handles**:
    - temporary identifiers returned by Exchange when you access or create objects on the server.
    - make reference to a particular object all along its session lifetime
    - **only links between objects** accessed on the client side and efficiently stored on the server side

MAPI Concepts

- ## MAPI Objects:
  - **“Free” a MAPI object:**
    - Exchange providers handle memory management on their own
    - You can tell Exchange you do not need an object anymore by **releasing its handle using the Release (0x1) call**
    - MAPI hierarchy semantics implies that if you release a parent object, it recursively applies to children objects.
  - **MAPI object uniqueness:**
    - MAPI objects are not permanent
    - They change when you move them from a folder to another etc.
    - However you can generally uniquely identify a given object using its **FID/MID**:
      - **FID**: Folder Identifier (uint64_t) unique on a given Exchange store
      - **MID**: Message Identifier (uint64_t) unique for a given message

MAPI Concepts

- **Sample EcDoRpc packet:**



EcDoRpc packet (NDR encoded)

EcDoRpc payload (non NDR encoded)

MAPI content (operations)

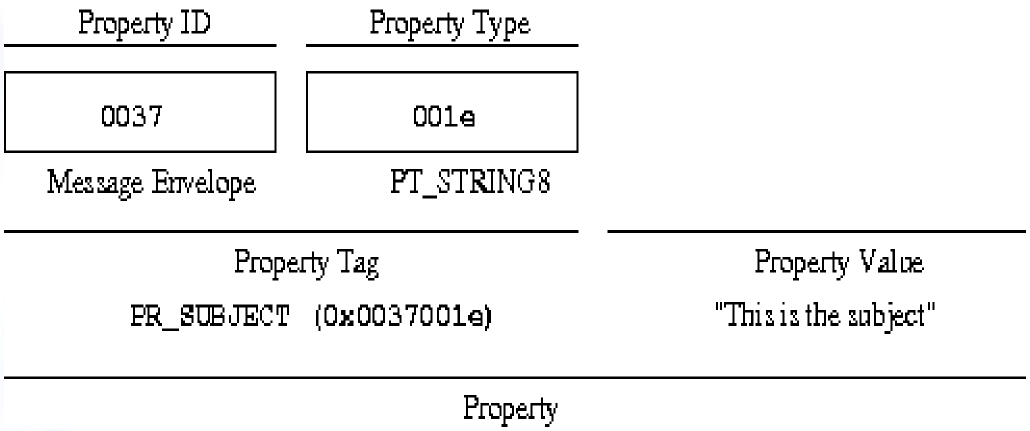
MAPI handles array

MAPI Concepts

- **MAPI Properties:**

  - **Attributes of a MAPI object used to describe something associated with the object.**

- Composed of:
  - **Property Tag**:
    - Property Type
    - Property ID
  - **Property Value**
    - Value matching property type



Property ID

Property Type

0037

001e

Message Envelope

PT_STRING8

Property Tag

PR_SUBJECT (0x0037001e)

Property Value

"This is the subject"

Property

  - **Property value exceeding 32kb are stored into streams**

  - Related functions: GetProps / SetProps
  - Related structure: SPropValue

MAPI Concepts

- **MAPI Tables:**
  - **MAPI tables are used to view MAPI objects as a set of rows and columns; where objects are rows and MAPI properties columns of the table.**
  - 4 kind of tables associated to specific MAPI calls:
  - **GetHierarchyTable**
    - Collect information about child containers
  - **GetContentsTable**
    - Collect information about objects within a container
  - **GetAttachmentTable**
    - Collect information about attachment objects within a message
  - **GetTable**
    - Retrieve information about permissions for a given container or object

MAPI Concepts

- **MAPI Tables:**
  - MAPI provides several method to customize, browse and seek the table view:
    - **SetColumns**
      - Specify the columns of a table
    - **QueryRows**
      - Query the table and retrieve records
    - **QueryColumns**
    - **SeekRow**
    - **Bookmark operations**
    - **SortTable, Restrict** to apply filters
    - **FindRow** to find a specific row

MAPI Concepts

MAPI Table Example:

- **OpenFolder**
  - Give FID 0xdeadbeef00000001 as input parameter
  - Retrieve a folder handle
- **GetHierarchyTable**
  - Use the folder parent handle
  - retrieve a table handle
- **SetColumns**
  - Use the table handle
  - I only want to fetch PR_MID, PR_FID and PR_SUBJECT
- **QueryRows** to retrieve rows
- Release the table handle
- Release the folder handle

MAPI Concepts

- **MAPI Streams:**
  - **MAPI streams are used to retrieve large contents (exceeding 32kb)**
  - Stream operations:
    - **OpenStream**
    - **ReadStream / WriteStream**
    - **CommitStream**
    - **SeekStream**
    - **GetStreamSize / SetStreamSize**
    - **CopyToStream**
  - How to fetch a stream?
    - Get the attachment table, customize the view and find the attachment object
    - Open the stream and get its size
    - Read the stream until read size is NULL
    - Release the stream

Bibliography

- MSRPC, a.k.a. Microsoft implementation of DCE RPC, 2003-2006 Jean-Baptiste Marchand:
http://www.hsc.fr/ressources/articles/win_net_srv/chap_msrpc.html

- Transfer Syntax NDR specifications, 1997 The Open Group :
http://www.opengroup.org/onlinepubs/9629399/chap14.htm

- How Samba was written, 2003 Andrew Tridgell:
http://samba.org/ftp/tridge/misc/french_cafe.txt

Thanks

- **Brad Hards and Jelmer Vernooij for review**