WATS Client MAC Address Determination

Complete Technical Reference for Client Registration and MAC Address Selection

Table of Contents

1. Overview

Purpose

The WATS Client uses a **MAC address** (or custom identifier) as a unique machine identifier for:

- Client authentication
- Passcode encryption/decryption
- License management
- Client tracking

Key Principles

1. **First Connection**: Client searches for the “best” MAC address
2. **Subsequent Connections**: Client validates the registered MAC address still exists
3. **Storage**: MAC address stored in Windows Registry

4. **Security**: Passcode encrypted using MAC address as entropy

2. MAC Address Selection Algorithm

2.1 The Search Priority

When the client needs to find a MAC address for the **first time** (`validateOnly = false`), it searches in this **exact order**:

```
Priority 1: Wireless Interface (Wireless80211)
    ? (if not found)
Priority 2: Ethernet Interface (Up and Connected)
    ? (if not found)
Priority 3: Any Interface (excluding Loopback & Tunnel)
```

2.2 Step-by-Step Algorithm

Located in: `WATS Client API\Core\REST\RestHelper.cs` (lines 447-509)

```csharp
public static string GetMACAddress(bool validateOnly = true, bool rememberResult = true)
{
    // 1. If validateOnly is true, try to use cached MAC
    string mac = validateOnly ? _mac : null;

    if (mac == null || !rememberResult)
    {
        if (IdentifierType == ClientIdentifierType.MacAddress)
        {
            // 2. Get the last registered MAC from registry
            string lastUsedMAC = Env.MACAddressRegistered;

            // 3. Try to find the network interface with that MAC
            var iface = NetworkInterface.GetAllNetworkInterfaces()
                .Where(nic => nic.GetPhysicalAddress().ToString() == lastUsedMAC)
                .FirstOrDefault();

            if (iface == null)
                lastUsedMAC = null;  // Registered MAC not found
            else
                mac = iface.GetPhysicalAddress().ToString();

            // 4. If no registered MAC exists and not validating, search for one
            if (String.IsNullOrEmpty(lastUsedMAC) && !validateOnly)
            {
                // PRIORITY 1: Wireless Interface
                // PRIORITY 2: Ethernet Interface
                // PRIORITY 3: Any Interface
                // (detailed below)
            }
        }
        else  // Custom identifier
        {
            mac = Env.MACAddressRegistered;
        }

        if (rememberResult)
            _mac = mac;  // Cache for future calls
    }

    return mac;
}
```

2.3 Priority 1: Wireless Interface

```
// Get first (by IPv4's index) wireless interface with:
// - Type: Wireless80211
// - Has MAC address
// - Has IP properties
// - Has gateway address (not 0.0.0.0)

iface = NetworkInterface.GetAllNetworkInterfaces()
    .Where(nic =>
        nic.NetworkInterfaceType == NetworkInterfaceType.Wireless80211 &&
        nic.GetPhysicalAddress() != null &&
        nic.GetIPProperties() != null &&
        (nic.GetIPProperties().GatewayAddresses.Count > 0 &&
         nic.GetIPProperties().GatewayAddresses.First()?.Address?.ToString() != "0.0.0.0"))
    .OrderBy(nic => nic.GetIPProperties().GetIPv4Properties().Index)
    .FirstOrDefault();
```

Criteria:

- ? Must be `NetworkInterfaceType.Wireless80211`
- ? Must have a physical address (MAC)
- ? Must have IP properties
- ? Must have at least one gateway address
- ? Gateway address cannot be `0.0.0.0`
- ? Ordered by IPv4 interface index (lowest = preferred)

Why Wireless First?

- Wireless MAC addresses are typically built-in to the motherboard
- Less likely to change or be removed
- More stable identifier than USB/external adapters

2.4 Priority 2: Ethernet Interface

```
// Get first (by IPv4's index) ethernet interface with:
// - Type: Ethernet
// - Status: Up (operational)
// - Has MAC address
// - Has IP properties
// - Has gateway address (not 0.0.0.0)

iface = NetworkInterface.GetAllNetworkInterfaces()
    .Where(nic =>
        nic.NetworkInterfaceType == NetworkInterfaceType.Ethernet &&
        nic.OperationalStatus == OperationalStatus.Up &&
        nic.GetPhysicalAddress() != null &&
        nic.GetIPProperties() != null &&
        (nic.GetIPProperties().GatewayAddresses.Count > 0 &&
         nic.GetIPProperties().GatewayAddresses.First()?.Address?.ToString() != "0.0.0.0"))
    .OrderBy(nic => nic.GetIPProperties().GetIPv4Properties().Index)
    .FirstOrDefault();
```

Criteria:

- ? Must be `NetworkInterfaceType.Ethernet`
- ? Must be `OperationalStatus.Up` (connected and operational)
- ? Must have a physical address (MAC)
- ? Must have IP properties
- ? Must have at least one gateway address
- ? Gateway address cannot be `0.0.0.0`
- ? Ordered by IPv4 interface index (lowest = preferred)

Why Second?

- Ethernet is typically built-in
- Status must be “Up” (actively connected)
- Ensures a working network connection

2.5 Priority 3: Any Interface (Fallback)

```
// Get first (by IPv4's index) interface with:
// - NOT Tunnel
// - NOT Loopback
// - Has MAC address
// - Has IP properties
// (No gateway requirement - might return VMWare/Virtual interfaces)

iface = NetworkInterface.GetAllNetworkInterfaces()
    .Where(nic =>
        nic.NetworkInterfaceType != NetworkInterfaceType.Tunnel &&
        nic.NetworkInterfaceType != NetworkInterfaceType.Loopback &&
        nic.GetPhysicalAddress() != null &&
        nic.GetIPProperties() != null)
    .OrderBy(nic => nic.GetIPProperties().GetIPv4Properties().Index)
    .First();  // Throws if none found
```

Criteria:

- ? Must NOT be `NetworkInterfaceType.Tunnel`
- ? Must NOT be `NetworkInterfaceType.Loopback`
- ? Must have a physical address (MAC)
- ? Must have IP properties
- ?? **NO gateway requirement** (relaxed criteria)
- ? Ordered by IPv4 interface index (lowest = preferred)

Warning:

- May select virtual adapters (VMWare, VirtualBox, etc.)
- Less stable than physical adapters
- Only used if no wireless or ethernet found

2.6 No Interface Found

```
if (iface == null)
    throw new ApplicationException("No MAC address found");
```

If **all three searches fail**, an exception is thrown.

2.7 MAC Address Format

```
mac = iface.GetPhysicalAddress().ToString();
```

Format: Uppercase hexadecimal string without separators

- ? Example: `"00112233445566"`
- ? NOT: `"00:11:22:33:44:55"` or `"00-11-22-33-44-55"`

3. Storage Mechanism

3.1 Windows Registry

Location:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Virinco\WATS
```

Key:

```
MACAddressRegistered = "00112233445566"
```

3.2 Env.MACAddressRegistered Property

Located in: `Core\Env.cs` (lines 478-485)

```
public static string MACAddressRegistered
{
    get
    {
        return getValue(MACAddressRegisteredKey, PersistValues);
    }
    set
    {
        setValue(MACAddressRegisteredKey, value.ToString(), PersistValues);
    }
}
```

Read:

```
string mac = Env.MACAddressRegistered;
// Returns: "00112233445566" or null
```

Write:

```
Env.MACAddressRegistered = "00112233445566";
// Saves to registry: HKEY_LOCAL_MACHINE\SOFTWARE\Virinco\WATS\MACAddressRegistered
```

3.3 Passcode Encryption

Located in: `WATS Client API\Core\REST\RestHelper.cs` (line 129, 595)

The MAC address is used as **entropy** for encrypting the client passcode:

```
// During LoadSettings() - Encrypting passcode
if (_settings.ClientPasscode != null)
{
    _settings.EncryptedClientPasscode = Convert.ToBase64String(
        ProtectedData.Protect(
            Encoding.UTF8.GetBytes(_settings.ClientPasscode),
            Encoding.UTF8.GetBytes(GetMACAddress() ?? " "),  // ? MAC as entropy
            DataProtectionScope.LocalMachine
        )
    );
    _settings.ClientPasscode = null;  // Clear plaintext
}


// During RegisterClient() - Saving encrypted passcode
_settings.EncryptedClientPasscode = Convert.ToBase64String(
    ProtectedData.Protect(
        Encoding.UTF8.GetBytes(newPasscode),
        Encoding.UTF8.GetBytes(mac),  // ? MAC as entropy
        DataProtectionScope.LocalMachine
    )
);

Env.MACAddressRegistered = mac;  // Save to registry
```

3.4 Passcode Decryption

Located in: `WATS Client API\Core\REST\RestHelper.cs` (lines 200-211)

```
protected string GetClientToken()
{
    if (_settings.EncryptedClientPasscode == null)
        throw new CryptographicException("Passcode is null.");

    string mac = GetMACAddress();  // ? Validate registered MAC

    if (mac == null)
        throw new CryptographicException("Registered mac address is null.");

    // Decrypt using MAC as entropy
    string pc = Encoding.UTF8.GetString(
        ProtectedData.Unprotect(
            Convert.FromBase64String(_settings.EncryptedClientPasscode),
            Encoding.UTF8.GetBytes(mac),  // ? MAC as entropy
            DataProtectionScope.LocalMachine
        )
    );

    return string.IsNullOrEmpty(pc) ? null : GetB64String("{0}:{1}", mac, pc.Trim('\"'));
}
```

Security Implications:

- ? Passcode encrypted using `ProtectedData` (Windows DPAPI)
- ? MAC address used as additional entropy
- ? `LocalMachine` scope (all users on this machine)
- ?? If MAC changes, passcode **CANNOT** be decrypted
- ?? Client must re-register if network card changes

4. Validation Process

4.1 On Every API Call

When the client makes any API call:

```
public responseType GetJson<responseType>(string query, ...)
{
    // Authentication header includes MAC and passcode
    request.Headers.Authorization = new AuthenticationHeaderValue(
        "Basic",
        GetClientToken()  // ? Validates MAC and decrypts passcode
    );

    // Send request...
}
```

4.2 GetClientToken() Validation

```
protected string GetClientToken()
{
    // 1. Check encrypted passcode exists
    if (_settings.EncryptedClientPasscode == null)
        throw new CryptographicException("Passcode is null.");

    // 2. Get and validate registered MAC
    string mac = GetMACAddress();  // validateOnly = true (default)

    // 3. Check MAC found
    if (mac == null)
        throw new CryptographicException("Registered mac address is null.");

    // 4. Decrypt passcode using MAC
    string pc = Encoding.UTF8.GetString(
        ProtectedData.Unprotect(
            Convert.FromBase64String(_settings.EncryptedClientPasscode),
            Encoding.UTF8.GetBytes(mac),
            DataProtectionScope.LocalMachine
        )
    );

    // 5. Build token: "MAC:Passcode" (Base64 encoded)
    return GetB64String("{0}:{1}", mac, pc.Trim('\"'));
}
```

4.3 GetMACAddress() Validation Mode

```
public static string GetMACAddress(bool validateOnly = true, ...)
{
    if (validateOnly)  // ? Default behavior
    {
        // 1. Get registered MAC from registry
        string lastUsedMAC = Env.MACAddressRegistered;

        // 2. Search for network interface with that exact MAC
        var iface = NetworkInterface.GetAllNetworkInterfaces()
            .Where(nic => nic.GetPhysicalAddress().ToString() == lastUsedMAC)
            .FirstOrDefault();

        if (iface == null)
        {
            // Registered MAC not found on this machine!
            return null;  // ? Triggers CryptographicException
        }

        return iface.GetPhysicalAddress().ToString();
    }
}
```

Validation Checks:

1. ? Registered MAC exists in registry
2. ? Network interface with that MAC exists on machine
3. ? MAC can decrypt the passcode

Failure Scenarios:

- ? Network card removed
- ? Network card replaced
- ? MAC address changed (spoofing)
- ? Registry value corrupted/deleted

5. Client Registration Flow

5.1 First-Time Registration

```
User initiates registration
    ?
RegisterClient(BaseUrl, Username, Password)
    ?
1. Get MAC address (validateOnly = false)
    ? Priority 1: Wireless
    ? Priority 2: Ethernet (Up)
    ? Priority 3: Any (not Tunnel/Loopback)
    ?
2. Build registration URL
   api/internal/Client/Register?mac={mac}&name={station}&...
    ?
3. Send POST with Basic Auth (Username:Password)
    ?
4. Server returns new passcode
    ?
5. Test connection with new passcode
   GetServerInfo(BaseUrl, mac, newPasscode)
    ?
6. Encrypt passcode using MAC as entropy
    ?
7. Save to settings.json:
   - TargetURL
   - EncryptedClientPasscode
    ?
8. Save MAC to registry:
   Env.MACAddressRegistered = mac
    ?
Registration Complete
```

5.2 Code Flow

Located in: `WATS Client API\Core\REST\RestHelper.cs` (lines 535-599)

```
public void RegisterClient(string BaseUrl, string Username, string Password)
{
    // 1. Get MAC address for FIRST TIME
    var mac = GetMACAddress(false);  // ? validateOnly = false

    if (string.IsNullOrWhiteSpace(mac))
    {
        throw new Exception("No network card found. " +
            "The client needs to bind to a network card with a fixed mac address.");
    }

    // 2. Build registration URL
    string fullurl = $"{BaseUrl}api/internal/Client/Register?" +
        $"mac={mac}&" +
        $"name={Env.StationName}&" +
        $"location={Env.Location}&" +
        $"purpose={Env.Purpose}&" +
        $"utcOffset={new decimal(DateTimeOffset.Now.Offset.TotalHours)}&" +
        $"version={Assembly.GetExecutingAssembly().GetName().Version}";

    // 3. Create authentication token
    string tmpToken = string.IsNullOrWhiteSpace(Username)
        ? Password
        : GetB64String("{0}:{1}", Username, Password);

    // 4. Send POST request
    HttpRequestMessage request = new HttpRequestMessage();
    request.Method = HttpMethod.Post;
    request.RequestUri = new Uri(fullurl);
    request.Headers.Authorization = new AuthenticationHeaderValue("Basic", tmpToken);

    var response = client.SendAsync(request).Result;

    if (!response.IsSuccessStatusCode)
        throw new HttpRequestException(...);

    // 5. Read new passcode from response
    string newPasscode;
    using (var responseReader = new StreamReader(response.Content.ReadAsStreamAsync().Result))
        newPasscode = responseReader.ReadToEnd().Trim('\"');

    // 6. Test connection with new passcode
    GetServerInfo(BaseUrl, mac, newPasscode);
    // 7. Save settings
    _settings.TargetURL = BaseUrl;
    _settings.EncryptedClientPasscode = Convert.ToBase64String(
        ProtectedData.Protect(
            Encoding.UTF8.GetBytes(newPasscode),
            Encoding.UTF8.GetBytes(mac),  // ? Encrypt with MAC
            DataProtectionScope.LocalMachine
        )
    );
    _settings.ClientPasscode = null;

    // 8. Save MAC to registry
    Env.MACAddressRegistered = mac;
}
```

5.3 Server-Side Storage

The server stores:

- **MAC Address**: Client unique identifier
- **Passcode**: Authentication credential
- **Station Name**: Machine name
- **Location**: Physical location
- **Purpose**: Client purpose/role
- **UTC Offset**: Timezone
- **Version**: Client API version

6. Code Reference

6.1 Key Files

File	Location	Purpose
`RestHelper.cs`	`WATS Client API\Core\REST\`	MAC selection, validation, encryption
`Env.cs`	`Core\`	Registry access for MAC storage
`ClientSettings.cs`	`Configuration\`	Settings.json structure

6.2 Key Methods

Method	File	Line	Purpose
`GetMACAddress()`	`RestHelper.cs`	447-509	Find/validate MAC address
`RegisterClient()`	`RestHelper.cs`	535-599	Register client with server
`GetClientToken()`	`RestHelper.cs`	200-211	Decrypt passcode for auth
`MACAddressRegistered`	`Env.cs`	478-485	Registry access

6.3 Network Interface Checks

All Checks:

```
// Check 1: Wireless
NetworkInterfaceType.Wireless80211
    + Has PhysicalAddress
    + Has IPProperties
    + Has Gateway (not 0.0.0.0)

// Check 2: Ethernet
NetworkInterfaceType.Ethernet
    + OperationalStatus.Up
    + Has PhysicalAddress
    + Has IPProperties
    + Has Gateway (not 0.0.0.0)

// Check 3: Any
NOT Tunnel
NOT Loopback
    + Has PhysicalAddress
    + Has IPProperties
```

7. Troubleshooting

7.1 Common Issues

Issue 1: “Registered mac address is null”

Cause:

- Network card removed/replaced
- Registry value deleted
- MAC address changed

Solution:

1. Check registry: `HKEY_LOCAL_MACHINE\SOFTWARE\Virinco\WATS\MACAddressRegistered`
2. Re-register client with server
3. New MAC will be detected and saved

Issue 2: “No network card found”

Cause:

- No physical network adapters
- All adapters are virtual (VMWare, VirtualBox)
- All adapters offline

Solution:

1. Install/enable a physical network adapter
2. Connect network cable (for ethernet)
3. Ensure adapter is “Up” and has gateway

Issue 3: “Cannot decrypt passcode”

Cause:

- MAC changed after registration
- Passcode encrypted with different MAC

Solution:

1. Delete encrypted passcode from `settings.json`
2. Re-register client

3. New encryption will use current MAC

7.2 Diagnostic Commands

Check Current MAC

```
using Virinco.WATS.REST;

string mac = ServiceProxy.GetCurrentMACAddress();
Console.WriteLine($"Current MAC: {mac}");
// Output: "00:11:22:33:44:55" (formatted)
```

Check Registered MAC

```
using Virinco.WATS;

string registeredMac = Env.MACAddressRegistered;
Console.WriteLine($"Registered MAC: {registeredMac}");
// Output: "00112233445566" (raw format)
```

List All Network Interfaces

```
using System.Net.NetworkInformation;

foreach (var nic in NetworkInterface.GetAllNetworkInterfaces())
{
    Console.WriteLine($"Name: {nic.Name}");
    Console.WriteLine($"Type: {nic.NetworkInterfaceType}");
    Console.WriteLine($"Status: {nic.OperationalStatus}");
    Console.WriteLine($"MAC: {nic.GetPhysicalAddress()}");

    var ip = nic.GetIPProperties();
    if (ip != null && ip.GatewayAddresses.Count > 0)
    {
        Console.WriteLine($"Gateway: {ip.GatewayAddresses.First().Address}");
    }
    Console.WriteLine();
}
```

7.3 Manual MAC Override

Not recommended, but possible:

```
// Set custom MAC manually (use with caution!)
Env.MACAddressRegistered = "001122334455666";

// Or use Custom Identifier instead
Env.IdentifierType = ClientIdentifierType.Custom;
Env.MACAddressRegistered = Guid.NewGuid().ToString();
```

Summary Diagram

```
???????????????????????????????????????????????????????????????
? Client Registration Flow                                    ?
???????????????????????????????????????????????????????????????

1. First Registration
   ?
   GetMACAddress(validateOnly: false)
   ?
   ?? Check 1: Wireless80211 + Gateway
   ?  ? (not found)
   ?? Check 2: Ethernet (Up) + Gateway
   ?  ? (not found)
   ?? Check 3: Any (not Tunnel/Loopback)
   ?
   MAC Found: "00112233445566"
   ?
   Send to Server: api/internal/Client/Register?mac=00112233445566&...
   ?
   Server Returns: "newPasscode123"
   ?
   Encrypt: ProtectedData.Protect(passcode, MAC, LocalMachine)
   ?
   Save Registry: MACAddressRegistered = "00112233445566"
   ?
   Save JSON: EncryptedClientPasscode = "base64..."

2. Subsequent API Calls
   ?
   GetClientToken()
   ?
   GetMACAddress(validateOnly: true)
   ?
   Read Registry: MACAddressRegistered = "00112233445566"
   ?
   Find Interface: Where(MAC == "00112233445566")
   ?
   ? Found ? Decrypt passcode
   ? Not Found ? Throw CryptographicException
   ?
   Return Token: Base64("00112233445566:passcode123")
   ?
   Use in Authorization Header

3. MAC Validation Failure
   ?
   Network Card Removed/Changed
   ?
   GetMACAddress() returns null
   ?
   CryptographicException: "Registered mac address is null"
   ?
   User Must Re-Register Client
```

Document Version: 1.0

Last Updated: 2024

For WATS Client API Version: 5.0+

Status: ? VERIFIED AGAINST SOURCE CODE