Step To Generate Ditigal Signature in C# (Prepared by TH MOK 2024-Jul-14)

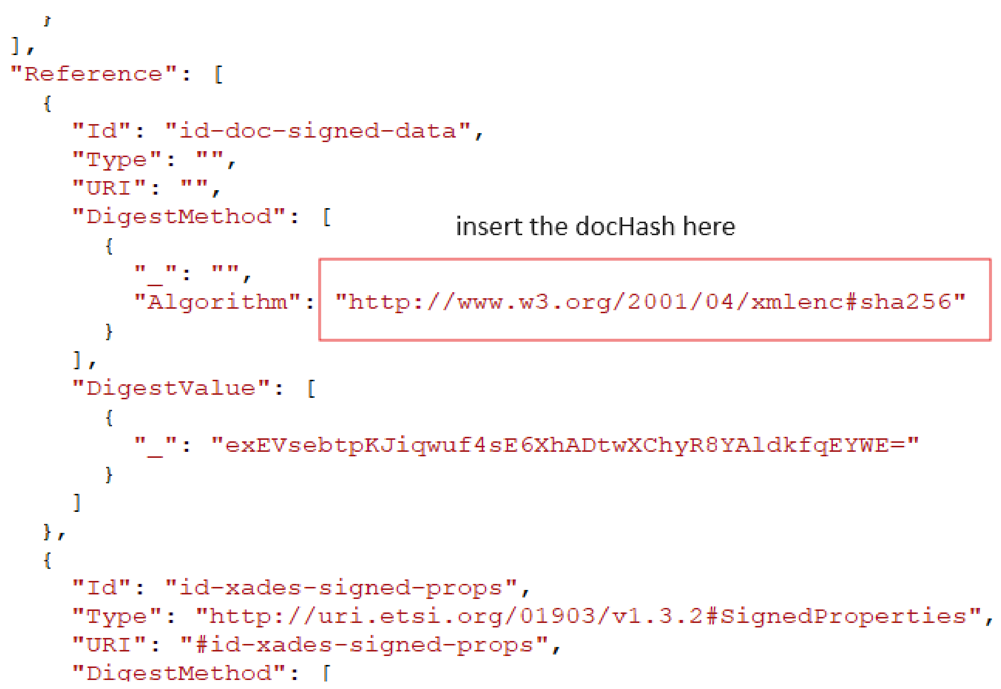
(updated on 02-08-2024)
(updated on 18-07-2024)

Open the certificate file somewhere in u local path

```
var cert = new X509Certificate2();
 cert.Import(File.ReadAllBytes(yourcertPath), yourcertPassword,
                                    X509KeyStorageFlags.MachineKeySet |
                                    X509KeyStorageFlags.PersistKeySet |
                                    X509KeyStorageFlags.Exportable);
```

1. Document hash
   - Serialize the document (class) in to 1 line string.

```
var docString = SerializeJson(jsonObject);

 var docHash = Sha256Hash(docstring);

  var docDigest = Convert.ToBase64String(docHash);

 public static string SerializeJson(object doc)
 {
     var settings = new JsonSerializerSettings
     {
         DateFormatString = "yyyy-MM-ddTH:mmZ",
         DateTimeZoneHandling = DateTimeZoneHandling.Utc,
         NullValueHandling = NullValueHandling.Ignore,
     };
     var jsonString = JsonConvert.SerializeObject(doc, settings);

     return jsonString;
 }

public static byte[] Sha256Hash(string text)
 {
     using (SHA256 sha256 = SHA256.Create())
     {
         byte[] byteData = Encoding.UTF8.GetBytes(text);
         var hashBytes = sha256.ComputeHash(byteData);
         return hashBytes;
     }
 }
```



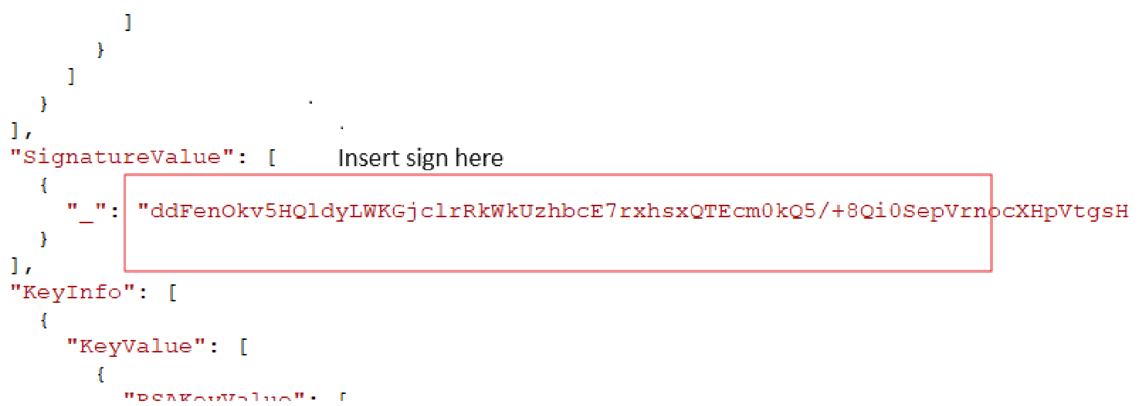
```
      ,
  ],
  "Reference": [
    {
      "Id": "id-doc-signed-data",
      "Type": "",
      "URI": "",
      "DigestMethod": [
        {
          "_": "",
          "Algorithm": "http://www.w3.org/2001/04/xmlenc#sha256"
        }
      ],
      "DigestValue": [
        {
          "_": "exEVsebtpKJiqwuf4sE6XhADtwXChyR8YAldkfqEYWE="
        }
      ]
    },
    {
      "Id": "id-xades-signed-props",
      "Type": "http://uri.etsi.org/01903/v1.3.2#SignedProperties",
      "URI": "#id-xades-signed-props",
      "DigestMethod": [
```

insert the docHash here

2. Digital Signature

```
   //use the docHash from above
   var signHash = SignData(docHash,cert);
   var sign = Convert.ToBase64String(signHash);
```

```
public static byte[] SignData(byte[] hashdata, X509Certificate2 cert)
 {
   byte[] signedData = null;
   //var hashdata= Sha256Hash(text);
   using (RSA rsa = cert.GetRSAPrivateKey())
   {
     try
     {
       var sharedParameters = rsa.ExportParameters(false);
       RSAPKCS1SignatureFormatter rsaFormatter = new RSAPKCS1SignatureFormatter(rsa);
       rsaFormatter.SetHashAlgorithm(nameof(SHA256));
       signedData= rsaFormatter.CreateSignature(hashdata);

      //or
       // signedData= rsa.SignHash(hashdata, HashAlgorithmName.SHA256,
      //                 SASignaturePadding.Pkcs1);
     }
     catch (CryptographicException)
     {

     }
   }

   return signedData;
 }
```



```
              ]
            }
          ]
        }
      ],
      "SignatureValue": [        Insert sign here
        {
          "_": "ddFenOkv5HQldyLWKGjclrRkWkUzhbcE7rxhsxQTEcm0kQ5/+8Qi0SepVrnocXHpVtgsH
        }
      ],
      "KeyInfo": [
        {
          "KeyValue": [
            {
```

3. Cert Digest

   var certRawData = cert.RawData;

   var certHash = Sha256HashBytes(certRawData);

   var certDigest = Convert.ToBase64String(certHash);

```
public static byte[] Sha256HashBytes(byte[] byteData)
 {
     using (SHA256 sha256 = SHA256.Create())
     {
         var hashBytes = sha256.ComputeHash(byteData);
         return hashBytes;
     }
}
```

4. Cert SerialNumber

```
var serialNumnber =BigInteger.Parse(cert.SerialNumber, NumberStyles.HexNumber);
```

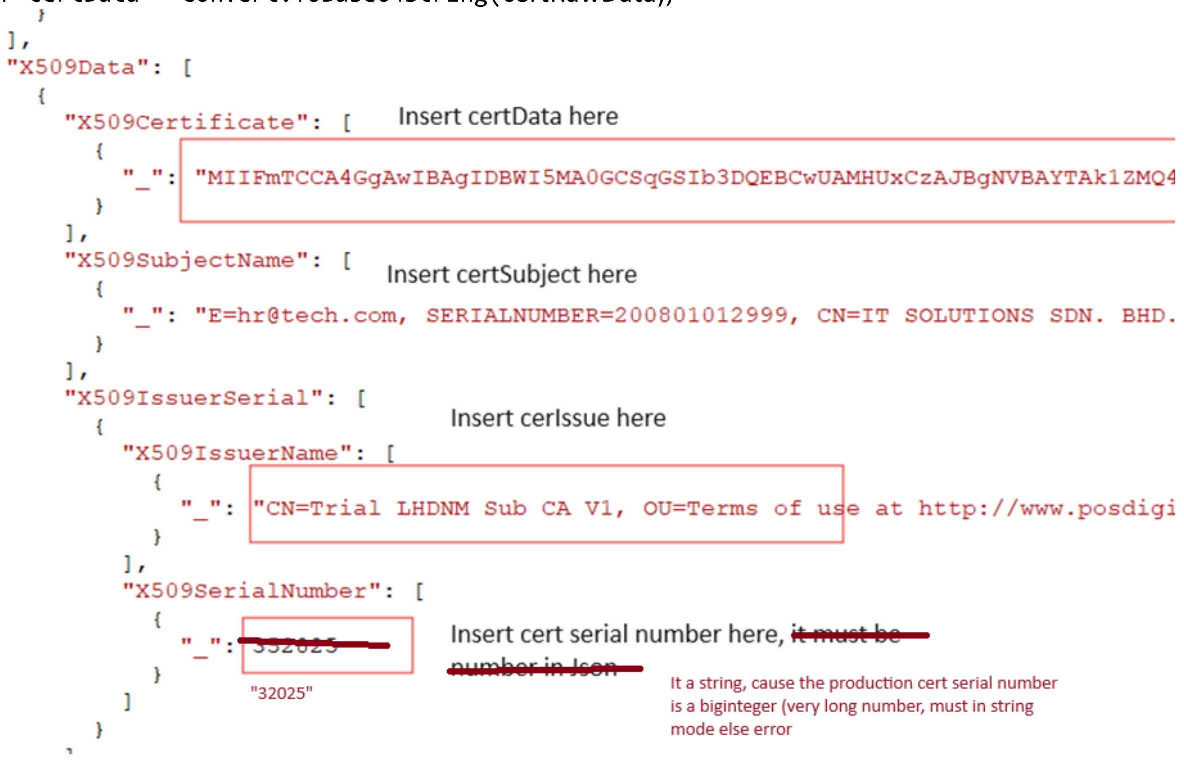
5. Cert Data

   var certRawData = cert.RawData;

   var certSubject = cert.Subject;

   vat cerissue = cert. Issuer;

   var certData = Convert.ToBase64String(certRawData);

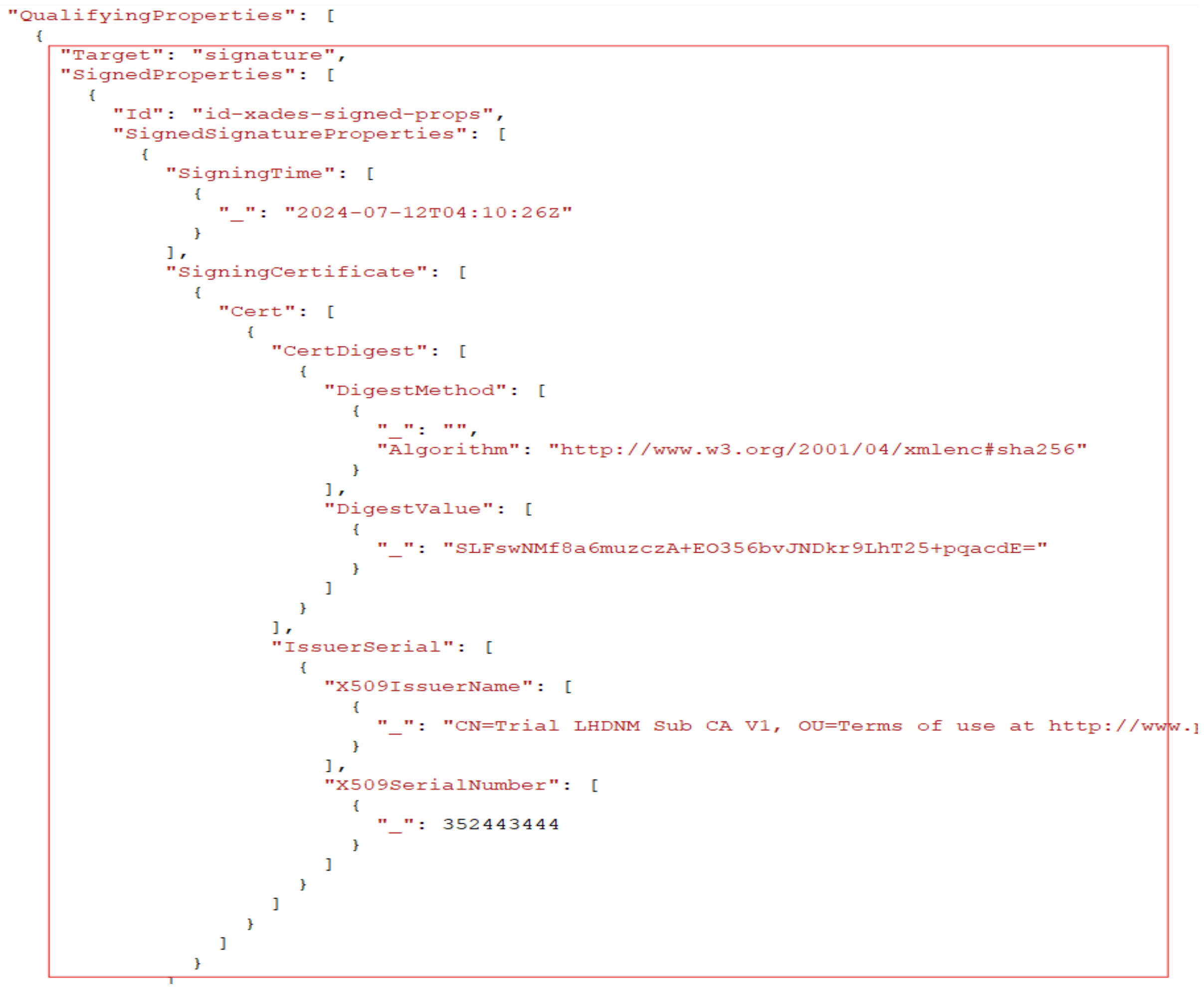


```
    }
  ],
  "X509Data": [
    {
      "X509Certificate": [     Insert certData here
        {
          "_": "MIIFmTCCA4GgAwIBAgIDBWI5MA0GCSqGSIb3DQEBCwUAMHUxCzAJBgNVBAYTAk1ZMQ4
        }
      ],
      "X509SubjectName": [     Insert certSubject here
        {
          "_": "E=hr@tech.com, SERIALNUMBER=200801012999, CN=IT SOLUTIONS SDN. BHD.
        }
      ],
      "X509IssuerSerial": [
        {                          Insert cerIssue here
          "X509IssuerName": [
            {
              "_": "CN=Trial LHDNM Sub CA V1, OU=Terms of use at http://www.posdigi
            }
          ],
          "X509SerialNumber": [
            {
              "_": ~~332025~~     Insert cert serial number here, ~~it must be
                                  number in Json~~
            }    "32025"
          ]                                   It a string, cause the production cert serial number
        }                                     is a biginteger (very long number, must in string
                                              mode else error
```

6. propCert
   - generate the UBLExtensions and populate all the above data.
   - **Note: follow exactly the structure and the sequence, if not properly u get digest not same as LHDN side.**
   - Extract out this part

(I am using class to generate the entity, so something like this)

```
var SignedProperties =      doc.UBLExtensions[0].UBLExtension[0].ExtensionContent[0].
                            UBLDocumentSignatures[0].SignatureInformation[0].
                            Signature[0].Object[0].QualifyingProperties[0];
```



```
"QualifyingProperties": [
  {
    "Target": "signature",
    "SignedProperties": [
      {
        "Id": "id-xades-signed-props",
        "SignedSignatureProperties": [
          {
            "SigningTime": [
              {
                "_": "2024-07-12T04:10:26Z"
              }
            ],
            "SigningCertificate": [
              {
                "Cert": [
                  {
                    "CertDigest": [
                      {
                        "DigestMethod": [
                          {
                            "_": "",
                            "Algorithm": "http://www.w3.org/2001/04/xmlenc#sha256"
                          }
                        ],
                        "DigestValue": [
                          {
                            "_": "SLFswNMf8a6muzczA+EO356bvJNDkr9LhT25+pqacdE="
                          }
                        ]
                      }
                    ],
                    "IssuerSerial": [
                      {
                        "X509IssuerName": [
                          {
                            "_": "CN=Trial LHDNM Sub CA V1, OU=Terms of use at http://www.
                          }
                        ],
                        "X509SerialNumber": [
                          {
                            "_": 352443444
                          }
                        ]
                      }
                    ]
                  }
                ]
              }
            ]
          }
```

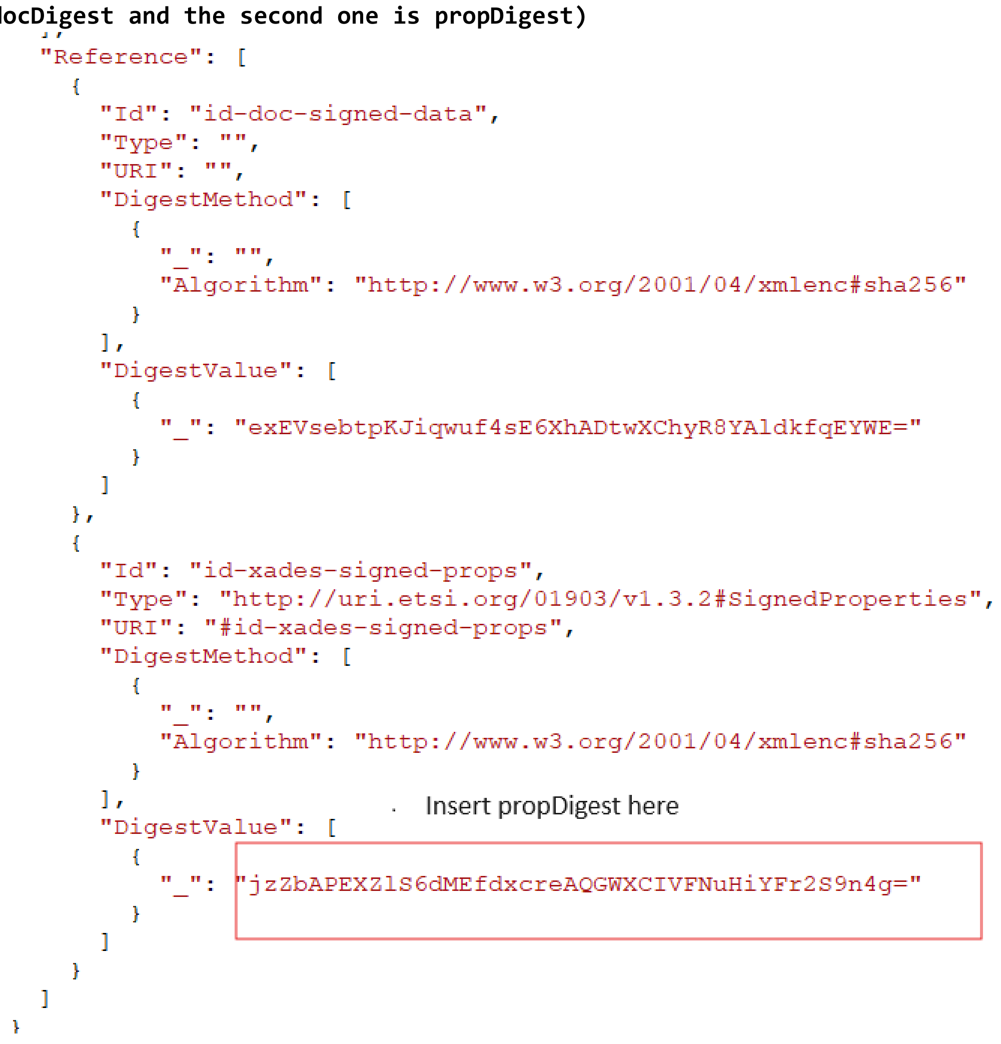
Serialize it to be 1 line string

```
public static string SerializeJsonEx(object doc)
    {
     var settings = new JsonSerializerSettings
     {
        NullValueHandling = NullValueHandling.Ignore,
        Formatting = Formatting.None,

     };
     var jsonString = JsonConvert.SerializeObject(doc, settings);
     return jsonString;
 }
```

```
Become
var propString =
“{"Target":"signature","SignedProperties":erSerial":[{"X509IssuerName":[{"_"
:"CN=Trial LHDNM SY".......}],"X509SerialNumber":[{"_":352825}]}]}]}]}]}]}”
```

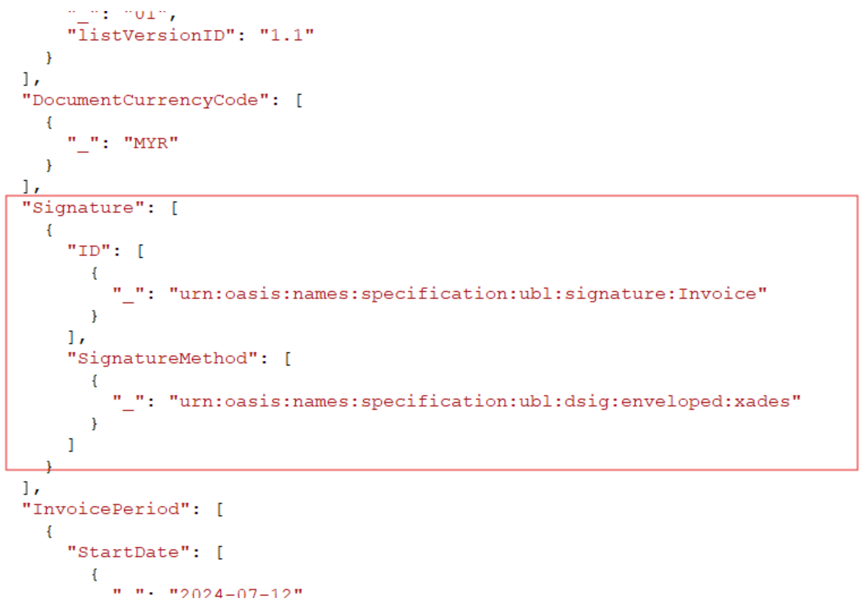
```
var propHash = Sha256Hash(propString);
var propDigest = Convert.ToBase64String(propHash);
```

(note, the reference also must follow this sequence, first reference is docDigest and the second one is propDigest)



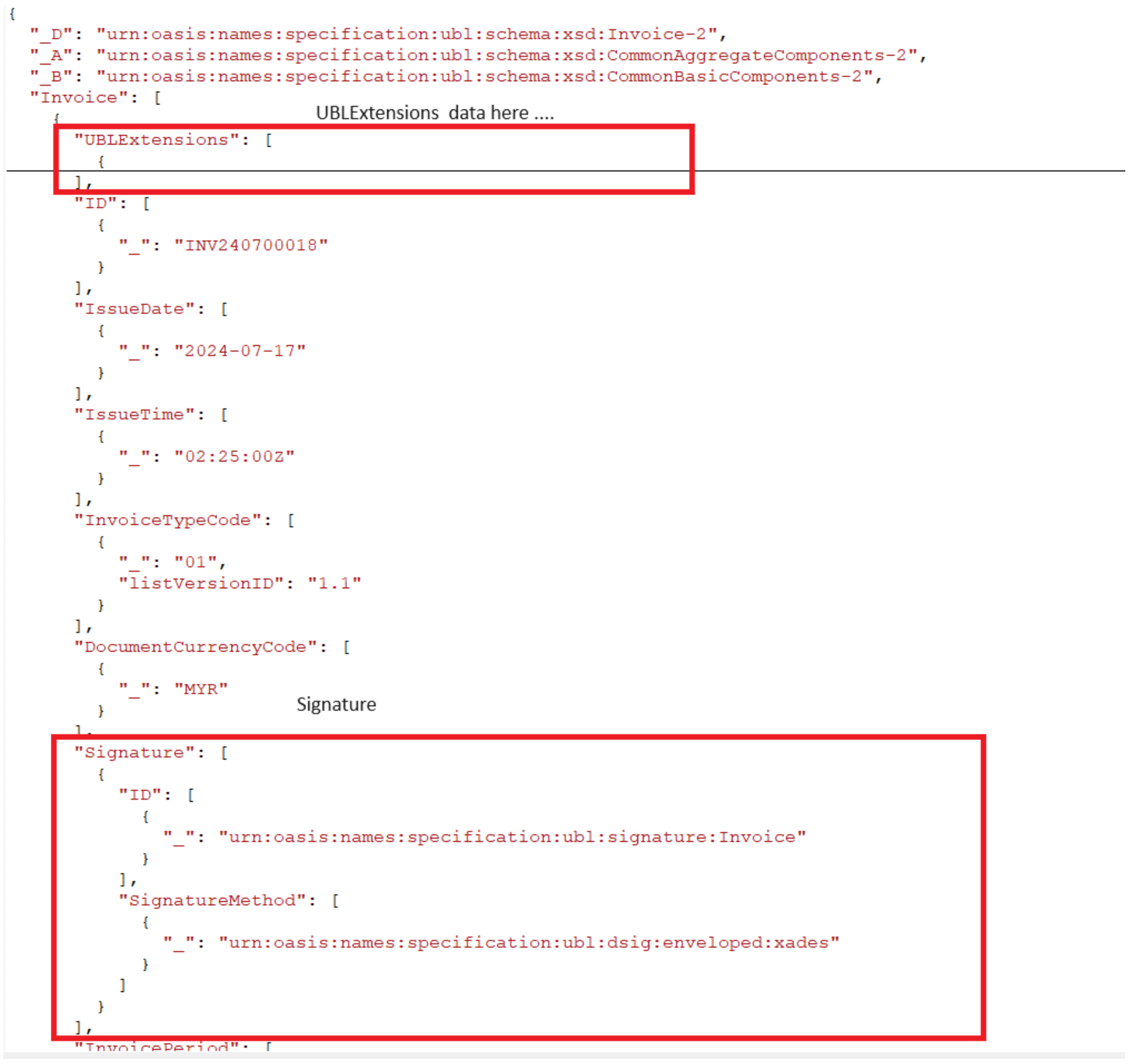
```
],
"Reference": [
  {
    "Id": "id-doc-signed-data",
    "Type": "",
    "URI": "",
    "DigestMethod": [
      {
        "_": "",
        "Algorithm": "http://www.w3.org/2001/04/xmlenc#sha256"
      }
    ],
    "DigestValue": [
      {
        "_": "exEVsebtpKJiqwuf4sE6XhADtwXChyR8YAldkfqEYWE="
      }
    ]
  },
  {
    "Id": "id-xades-signed-props",
    "Type": "http://uri.etsi.org/01903/v1.3.2#SignedProperties",
    "URI": "#id-xades-signed-props",
    "DigestMethod": [
      {
        "_": "",
        "Algorithm": "http://www.w3.org/2001/04/xmlenc#sha256"
      }
    ],
    "DigestValue": [
      {
        "_": "jzZbAPEXZlS6dMEfdxcreAQGWXCIVFNuHiYFr2S9n4g="
      }
    ]
  }
]
}
```

7. Insert the signature field in then main json doc



```
      "_": "01",
      "listVersionID": "1.1"
    }
  ],
  "DocumentCurrencyCode": [
    {
      "_": "MYR"
    }
  ],
  "Signature": [
    {
      "ID": [
        {
          "_": "urn:oasis:names:specification:ubl:signature:Invoice"
        }
      ],
      "SignatureMethod": [
        {
          "_": "urn:oasis:names:specification:ubl:dsig:enveloped:xades"
        }
      ]
    }
  ],
  "InvoicePeriod": [
    {
      "StartDate": [
        {
          "_": "2024-07-12"
```

8. Now u json file is cone with Digital Signature.



```
{
  "_D": "urn:oasis:names:specification:ubl:schema:xsd:Invoice-2",
  "_A": "urn:oasis:names:specification:ubl:schema:xsd:CommonAggregateComponents-2",
  "_B": "urn:oasis:names:specification:ubl:schema:xsd:CommonBasicComponents-2",
  "Invoice": [
    {
      "UBLExtensions": [
        {
      ],
      "ID": [
        {
          "_": "INV240700018"
        }
      ],
      "IssueDate": [
        {
          "_": "2024-07-17"
        }
      ],
      "IssueTime": [
        {
          "_": "02:25:00Z"
        }
      ],
      "InvoiceTypeCode": [
        {
          "_": "01",
          "listVersionID": "1.1"
        }
      ],
      "DocumentCurrencyCode": [
        {
          "_": "MYR"
        }
      ],
      "Signature": [
        {
          "ID": [
            {
              "_": "urn:oasis:names:specification:ubl:signature:Invoice"
            }
          ],
          "SignatureMethod": [
            {
              "_": "urn:oasis:names:specification:ubl:dsig:enveloped:xades"
            }
          ]
        }
      ],
      "InvoicePeriod": [
```

<u>Thing to take note</u>

- For date time use `ToUniversalTime()`
  `And format according eg.`
  `date.ToString("yyyy-MM-ddTHH:mm:ssZ");`

- Amount at invoice level must tally with amount at item level
- Use proper decimal point, currently based on sandbox testing, at least 1 decimal point
  Even zero, have to put 0.0 not 0 (so far my testing)
- Tax Exchange Rate only include when u currency is other than MYR
- The IssueDate and IssueTime is the Date Time when u do the submission (in UTC)
- Make sure all the code is followed LHDN Code (refer to the SDK side), eg state code, country code, tax type code…

For decimal in json, minimal is 1 decimal point

eg:

0	bad
0.0	good
1	bad
1.0	good
1.10	bad
1.1	good
3.2341	good
3.23410	bad

If u use c# Newtonsoft.Json, it will auto serialize for u.