ClearTrace Protocol (CTP) Specification v1.0

Status: Production Ready
Category: Financial Technology / Audit Standards
Date: 2025-11-25
Maintainer: Aegis ClearTrace Standards Committee (ACSC)

---

1. Abstract

1.1 Purpose

The ClearTrace Protocol (CTP) is a global standard specification for recording the "decision-making" and "execution results" of algorithmic trading in an Immutable & Verifiable format.

1.2 Versioning

Adopts Semantic Versioning 2.0.0. Full backward compatibility is guaranteed within the v1.x series. 2. Compliance Tiers

---

2. Compliance Tiers

Level	Target	Clock Sync	Serialization	Signature	Anchor
Platinum	HFT / Exchange	**PTPv2** (<1µs)	**SBE**	Hardware	10 min
Gold	Prop Firm	**NTP** (<1ms)	JSON	Client Local	1 hour
Silver	**MT4/5 Retail**	**Best-effort**	**JSON**	**Delegated**	24 hours

3. Event Lifecycle

```
stateDiagram-v2
    [*] --> SIG: Signal
    SIG --> ORD: Order Sent
    ORD --> ACK: Acknowledged
    ACK --> EXE: Filled (Full)
    ACK --> PRT: Partial Fill
    ACK --> CXL: Cancelled
    SIG --> REJ: Rejected
    EXE --> MOD: Modified
    EXE --> CLS: Closed

    state "System Events" as Sys {
        HBT: Heartbeat
        ERR: Error
    }
```

4. Data Model: CTP-CORE (Header)

Required Header Fields

Tag	Field Name	Type	Description
1001	EventID	UUID	Unique Event ID (v4)
1002	TraceID	UUID	Trade Route ID (CAT Rule 613 compliant)
1010	Timestamp	Int64	UTC Nanoseconds (Epoch)
1011	EventType	Enum	SIG, ORD, ACK,

			EXE, PRT, REJ, CXL, MOD, CLS, HBT, ERR
1020	VenueID	String	Broker/Exchange ID
1030	Symbol	String	Symbol Code

---

5. Extensions

5.4 CTP-HEALTH: System Health Extensions

Condition: When EventType=ERR Tag

Tag	Field Name	Type	Description
8001	ErrorCode	String	Error code (e.g., "ERR_TIMEOUT")
8002	ErrorMessage	String	Detailed message
8003	Component	String	Source (e.g., "Bridge", "RiskEngine")

(CTP-AI, CTP-ALG, CTP-DETECT are omitted)

---

6. Integrity & Security Layer (CTP-SEC)

6.1 Hash Chain Fields

Tag	Field Name	Type	Description

9001	PrevHash	SHA-256	EventHash of the immediately preceding event (Initial: 0x0...0)
9002	EventHash	SHA-256	Hash calculated according to Section 6.5
9003	Signature	Base64	Signature generated according to Section 6.6
9004	SignAlgo	Enum	ECDSA_SECP256K1, ED25519, RSA_2048
9006	SignedBy	Enum	CLIENT, DELEGATED

6.4 Merkle Tree Anchoring Fields

Condition: When an anchoring event occurs (periodically)

Tag	**Field Name**	**Type**	**Description**
9010	MerkleRoot	SHA-256	Root hash of the event group for the target period
9011	AnchorTxHash	String	Blockchain TxID (e.g., "0x...")
9012	AnchorTime	Int64	Anchoring execution

			timestamp (UTC ns)
9013	AnchorProvider	Enum	BITCOIN, ETHEREUM, OPENTIMESTAMPS, TSA_RFC3161

6.5 EventHash Calculation (Canonicalization Rules)

To ensure hash consistency across different languages, CTP complies with RFC 8785 (JSON Canonicalization Scheme).

1. **Canonicalization:** Normalize the Header and Payload objects according to RFC 8785.
   - Key Sorting: Alphabetical ascending order
   - Whitespace Removal: Complete removal of newlines, indentation, and spaces
   - Unicode Normalization: NFC format
2. Input Construction:
   HashInput = Canonical(Header) + Canonical(Payload) + PrevHash
3. Hashing:
   EventHash = SHA256(HashInput) (Output as Hex String)

6.6 Signature Generation Process

1. **Sign Input**: The input to the signature function must be the raw digest (32 bytes) resulting from the SHA256 calculation in Section 6.5.
   - *Example: EventHash = "f2ca..." -> SignInput = b"f2ca..." (64 bytes)*
   - *Note: Since the input is already the SHA-256 hash digest, do not re-hash it within the signature function (use functions designed to sign digests directly, often referred to as "HashNone" mode).*
2. **Execution:** Sign using the selected algorithm (SignAlgo), encode the DER format in Base64, and store it in Tag 9003.

Appendix C: SBE XML Schema (Template)

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<sbe:messageSchema xmlns:sbe="http://fixprotocol.io/2016/sbe"
          package="com.aegis.ctp" id="1" version="1" semanticVersion="1.0.0">
  <types>
    <composite name="Header">
      <type name="EventID" primitiveType="char" length="36"/>
      <type name="Timestamp" primitiveType="int64"/>
      <enum name="EventType" encodingType="uint8">
        <validValue name="SIG">1</validValue>
        <validValue name="ERR">99</validValue>
      </enum>
    </composite>
  </types>
  <sbe:message name="CTP_Event" id="100">
    <field name="Header" id="1" type="Header"/>
    <data name="Payload" id="2" type="varDataEncoding"/>
    <data name="Signature" id="3" type="varDataEncoding"/>
  </sbe:message>
</sbe:messageSchema>
```

Appendix D: MQL5 Implementation Guide (Silver Tier)

Sample implementation for generating UNIX timestamps and sending JSON in MQL5.

```
// CTPLogger.mqh
class CTPLogger {
  string m_endpoint;
  string m_apiKey;
public:
  CTPLogger(string endpoint, string key) : m_endpoint(endpoint), m_apiKey(key) {}

// Note: Nanosecond precision time acquisition is impossible in MQL5.
// Since Silver Tier is defined as "Best-effort",
// convert GetTickCount64() (milliseconds) to nanoseconds by multiplying by 1,000,000.
  long GetNavTime() {
    return (long)GetTickCount64() * 1000000;
  }

  string GenerateUUID() {
    // Simple implementation (For production, recommend generating RFC4122 compliant UUID using WinAPI etc.)
    return "550e8400-e29b-41d4-a716-" + IntegerToString(GetTickCount());
  }

  bool SendSignal(string traceID, string symbol) {
    string json = StringFormat(
      "{"
      "\"CTP_Version\":\"1.0.0\","
      "\"ComplianceTier\":\"Silver\","
      "\"Header\":{"
        "\"EventID\":\"%s\","
        "\"TraceID\":\"%s\","
        "\"Timestamp\":%I64d,"  // Output as 64bit integer
        "\"EventType\":\"SIG\","
        "\"Symbol\":\"%s\""
      "},"
      "\"Security\":{\"SignedBy\":\"DELEGATED\"}"
      "}",
      GenerateUUID(), traceID, GetNavTime(), symbol
    );
    //... (WebRequest sending process)
    return true;
  }
};
```

Appendix E: Python Implementation Guide (JCS Compliance)

Example implementation of normalization and signing compliant with RFC 8785 (JCS).

```
import json
import hashlib
import time
import uuid
import base64 # Import base64

# pip install canonicaljson ecdsa
from canonicaljson import encode_canonical_json
from ecdsa import SigningKey, SECP256k1
# Import specific encoding for DER format as required by Section 6.6
from ecdsa.util import sigencode_der

class CTPEvent:
    # Fixed syntax: def_init__ -> def __init__
    # Fixed syntax: prev_hash $=^{\prime\prime}0^{\prime\prime}64$ -> prev_hash="0"*64
    def __init__(self, trace_id, event_type, symbol, prev_hash="0"*64):
        self.header = {
            "EventID": str(uuid.uuid4()),
            "TraceID": trace_id,
            "Timestamp": int(time.time() * 1e9),
            "EventType": event_type,
            "Symbol": symbol
        }
        self.payload = {}
        self.prev_hash = prev_hash

    def sign(self, private_key_pem):
        # 1. Canonicalize (RFC 8785)
        # Fixed variable assignments (removed $c=$ etc.)
        header_c = encode_canonical_json(self.header)
        payload_c = encode_canonical_json(self.payload)

        # 2. Calculate EventHash
        # Input = Canonical(Header) + Canonical(Payload) + PrevHash(UTF-8)
        hash_input = header_c + payload_c + self.prev_hash.encode('utf-8')

        # Calculate the raw digest (32 bytes) and the hex representation.
        event_hash_digest = hashlib.sha256(hash_input).digest()
        event_hash_hex = event_hash_digest.hex()

        # 3. Sign the Hash Digest (Standard Practice)
        # Fixed variable assignment (removed $sk=$)
        sk = SigningKey.from_pem(private_key_pem)

        # Sign the digest directly (HashNone mode) using sign_digest() instead of sign().
        # Encode in DER format.
        signature_der = sk.sign_digest(event_hash_digest, sigencode=sigencode_der)

        # Encode the DER signature in Base64 as required by Section 6.6, Tag 9003.
        signature_b64 = base64.b64encode(signature_der).decode('utf-8')

        return {
            "CTP_Version": "1.0.0",
            "Header": self.header,
            "Payload": self.payload,
            "Security": {
                "PrevHash": self.prev_hash,
                "EventHash": event_hash_hex,  # Store the Hex representation
                "Signature": signature_b64,   # Store the Base64 DER signature
                "SignAlgo": "ECDSA_SECP256K1",
                "SignedBy": "CLIENT"
            }
        }
```