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. 概要 (Abstract)

1.1 目的

ClearTrace Protocol (CTP) は、アルゴリズム取引の「意思決定」と「実行結果」を、改ざん不可能かつ検証可能な形式 **(Immutable & Verifiable)** で記録する世界標準規格である。

1.2 バージョニング

Semantic Versioning 2.0.0 を採用。v1.x 間では完全な後方互換性を保証する。

---

2. 実装準拠レベル (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. イベントモデル (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. データモデル: CTP-CORE (Header)

必須ヘッダーフィールド

Tag	Field Name	Type	Description
1001	EventID	UUID	イベント固有ID (v4)
1002	TraceID	UUID	取引ルートID (CAT Rule 613準拠)
1010	Timestamp	Int64	UTC Nanoseconds (Epoch)

1011	EventType	Enum	SIG, ORD, ACK, EXE, PRT, REJ, CXL, MOD, CLS, HBT, ERR
1020	VenueID	String	ブローカー/取引所ID
1030	Symbol	String	銘柄コード

---

5. 拡張ペイロード (Extensions)

5.4 CTP-HEALTH: System Health Extensions

使用条件: *EventType=ERR* 時

Tag	Field Name	Type	Description
8001	ErrorCode	String	エラーコード (例: "ERR_TIMEOUT")
8002	ErrorMessage	String	詳細メッセージ
8003	Component	String	発生源 (例: "Bridge", "RiskEngine")

(CTP-AI, CTP-ALG, CTP-DETECT は 省略)

---

6. Integrity & Security Layer (CTP-SEC)

6.1 Hash Chain Fields

Tag	Field Name	Type	Description
9001	PrevHash	SHA-256	直前イベントの EventHash (初回: 0x0...0)
9002	EventHash	SHA-256	下記6.5節に従い計算されたハッシュ
9003	Signature	Base64	下記6.6節に従い生成された署名
9004	SignAlgo	Enum	ECDSA_SECP256K1, ED25519, RSA_2048
9006	SignedBy	Enum	CLIENT, DELEGATED

6.4 Merkle Tree Anchoring Fields (New!)

使用条件: アンカリングイベント発生時（定期的）

Tag	Field Name	Type	Description
9010	MerkleRoot	SHA-256	対象期間のイベント群のルートハッシュ
9011	AnchorTxHash	String	ブロックチェーンTxID (例: "0x...")
9012	AnchorTime	Int64	アンカリング実行時刻 (UTC ns)
9013	AnchorProvider	Enum	BITCOIN, ETHEREUM,

			OPENTIMESTAMPS, TSA_RFC3161

6.5 EventHash Calculation (正規化ルール)

異なる言語間でのハッシュ一致を保証するため、**RFC 8785 (JSON Canonicalization Scheme)** に準拠する。

1. **Canonicalization:** HeaderとPayloadオブジェクトをRFC 8785に従い正規化する。
   - キーのソート: アルファベット昇順
   - 空白除去: 改行・インデント・スペースを完全削除
   - Unicode正規化: NFC形式
2. Input Construction:
   HashInput = Canonical(Header) + Canonical(Payload) + PrevHash
3. Hashing:
   EventHash = SHA256(HashInput) (Hex Stringとして出力)

6.6 Signature Generation Process

1. **Sign Input:** EventHash (64文字のHex文字列) を **UTF-8バイト列** として扱う。
   - 例: EventHash = "f2ca..." -> SignInput = b"f2ca..." (64 bytes)
   - 注意: EventHashは既にSHA-256ハッシュ済みであるため、署名関数内で再度ハッシュ化しないこと(または HashNone モードを使用すること)。
2. **Execution:** 選択されたアルゴリズム (SignAlgo) で署名し、DER形式をBase64エンコードして Tag 9003 に格納する。

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)

MQL5におけるUNIXタイムスタンプ生成とJSON送信のサンプル。

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

  // 注意: MQL5ではナノ秒精度の時刻取得は不可能。
  // Silver Tierでは「Best-effort」とされているため、
  // GetTickCount64() (ミリ秒) を 1,000,000倍してナノ秒に変換する。
  long GetNavTime() {
    return (long)GetTickCount64() * 1000000;
  }

  string GenerateUUID() {
    // 簡易実装 (本番ではWinAPI等でRFC4122準拠UUIDを生成推奨)
    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,"  // 64bit整数として出力
        "\"EventType\":\"SIG\","
```

```
            "\"Symbol\":\"%s\""
        "},"
        "\"Security\":{\"SignedBy\":\"DELEGATED\"}"
        "}",
        GenerateUUID(), traceID, GetNavTime(), symbol
    );
    //... (WebRequest送信処理)
    return true;
  }
};
```

Appendix E: Python Implementation Guide (JCS Compliance)

RFC 8785 (JCS) に準拠した正規化と署名の実装例。

```
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"
          }
      }
```