Sequence Listing Report for Application No: 63805241

File Name: ImplementationListings.xml (6637 B), **Created:** May 13, 2025, 09:23 PM
USPTO Receipt Date: May 13, 2025, 09:19 PM
Validation Date: May 13, 2025, 09:23 PM
Reviewed by: NL2bdE5uK0BX8j2lcMI8RA==

DEFECTIVE: This validation report concerns the sequence listing filed 05-13-2025 21:19:34 and indicates the errors found by the USPTO during validation of the sequence listing. Any resolution of the errors identified in this validation report should not be construed as meaning that the sequence listing will comply with all of the requirements of 37 CFR §§ 1.821- 1.825, WIPO Standard ST.25, 37 CFR §§ 1.831-1.835 and 1.839, or WIPO Standard ST.26.
For any sequence listing related questions or concerns, the Applicant is invited to contact the USPTO Sequence Help Desk at 571-272-2510 or SequenceHelpDesk@USPTO.GOV

Reviewer's Comments

The submitted file is not a compliant ST.26 sequence listing.

A sequence listing is required when nucleotide and/or amino acid sequence(s) are disclosed in the application. If no sequence(s) are disclosed in the application, a sequence listing is not required.

1. E261 ==
Validation Report for Application: 63805241 cannot be generated. This file is not compliant with the ST.26 DTD. Timestamp: Tue May 13 21:23:12 EDT 2025
==

Validated By CRFValidator v 1.5.6

Application No: 63805241 Version No: 1.0

CRF Copy Request from application : Version :

Input Set: **/slic/CRF_WORKFLOW/63/805/241/1.0/ImplementationListings.xml**

Output Set: **/slic/CRF_WORKFLOW/63/805/241/1.0/ImplementationListings.xml.pdf**

Started:	**2025-05-13 21:23:12.059**
Finished:	**2025-05-13 21:23:12.059**
Elapsed:	**0 hr(s) 0 min(s) 0 sec(s) 0 ms**
Total Warnings:	**0**
Total Errors:	**0**
No. of SeqIDs Defined:	**0**
Actual SeqID Count:	**0**

Error code	Error Description
E 261	**==** **Validation Report for Application: 63805241 cannot be generated.** **This file is not compliant with the ST.26 DTD.** **Timestamp: Tue May 13 21:23:12 EDT 2025** **==**

```
<?xml version="1.0" encoding="US-ASCII"?>
<ComputerProgramListing applicantFile="ImplementationListings.xml">
  <Code id="Listing1" language="PHP"><![CDATA[
<?php
namespace App\Livewire;

use Livewire\Component;
use App\Services\CartridgeBus;   // handles IC / RS-485 I/O
use App\Models\AuditLedger;      // Eloquent model  Fabric client

/**
 * Reference numerals:
 *   110  sensor input
 *   130  micro-pump
 *   160  UID / EEPROM chip
 */
class DoseController extends Component
{
    /** @var array<float,int>  pH threshold    L dose */
    protected array $thresholdMap = [
        7.10 =>  5,
        7.00 =>  8,
        6.90 => 12,
    ];

    /** Poll sensor, apply rule, dose, and audit the event */
    public function pollAndDose(): void
    {
        $uid = CartridgeBus::readUid(160);          // (a) identify cartridge
        $pH  = CartridgeBus::readSensor(110);       // (b) read sensor value

        foreach (array_reverse($this->thresholdMap, true) as $limit => $dose) {
            if ($pH < $limit) {
                  // (c) compare to threshold
                CartridgeBus::pumpDose(130, $dose); // (d) dispense micro-dose
                $this->audit($uid, $pH, $dose, $limit);
                break;
            }
        }
    }

    protected function audit(string $uid, float $pH, int $dose, float $limit): void
    {
        AuditLedger::record([                       // (e) tamper-evident log
            'uid'      => $uid,
            'sensor'   => $pH,
            'dose'     => $dose,
            'rule_hit' => "pH < {$limit}",
```

```
        ]);
    }
}

  ]]></Code>
  <Code id="Listing2" language="PHP"><![CDATA[
<?php
namespace App\Lib;

use Illuminate\Support\Facades\Hash;

/**
 * HashChainHelper
 *  lightweight append-only ledger: 64-byte SHA-256 link +
 *   ISO-8601 timestamp + JSON payload.
 *
 * Usage:
 *   $hc = new HashChainHelper(storage_path('ledger.hash'));
 *   $hc->append(['event' => 'dose', 'slot' => 'pump_05', 'L' => 12.4]);
 */
class HashChainHelper
{
    protected string $file;
    protecte
d string $lastHash;

    public function __construct(string $file)
    {
        $this->file = $file;
        $this->lastHash = file_exists($file)
            ? trim(last(file($file)))
            : str_repeat('0', 64);                 // genesis hash
    }

    public function append(array $payload): void
    {
        $line = [
            'ts'   => now()->toIso8601String('microsecond'),
            'prev' => $this->lastHash,
            'data' => $payload,
        ];
        $json = json_encode($line, JSON_UNESCAPED_SLASHES);
        $hash = hash('sha256', $json);
        file_put_contents($this->file, $json . PHP_EOL, FILE_APPEND);
        file_put_contents($this->file, $hash . PHP_EOL, FILE_APPEND);
        $this->lastHash = $hash;
    }
```

```
    public function verify(): bool
    {
        $lines = file($this->file, FILE_IGNORE_NEW_LINES);
        $prev  = str_repeat('0', 64);
        for ($i = 0; $i < count($lines); $i += 2) {
            $json = $lines[$i];
            $hash = $lines[$i + 1] ?? '';

      $calc = hash('sha256', $json);
            if ($calc !== $hash) return false;
            $rec  = json_decode($json, true);
            if ($rec['prev'] !== $prev) return false;
            $prev = $hash;
        }
        return true;
    }
}

  ]]></Code>
  <Code id="Listing3" language="PHP"><![CDATA[
<?php
namespace App\Livewire;

use Livewire\Component;
use Illuminate\Support\Collection;
use Illuminate\Support\Facades\Http;

/**
 * MLPredict Component
 *  wires sensor vector to an async ML inference endpoint.
 *   Replace the HTTP call with your actual model API.
 */
class MLPredict extends Component
{
    public Collection $vector;     // bound via Alpine/MQTT

    public string $prediction = 'n/a';
    public string $status     = '';

    protected $listeners = ['sensor-update' => 'updateVector'];

    public function mount()
    {
        $this->vector = collect();
    }

    public function updateVector(string $slug, float $value): void
    {
```

```
        $this->vector->put($slug, $value);
        //
 throttle inference to 1/5 s
        $this->dispatch('ml-infer')->debounce(5000);
    }

    public function mlInfer(): void
    {
        $this->status = 'predicting';
        try {
            $resp = Http::timeout(1.5)->post(
                config('services.ml.url'),
                $this->vector->all()
            )->json();

            $this->prediction = $resp['label'] ?? 'n/a';
            $this->status     = 'ok';
        } catch (\Throwable $e) {
            $this->status = 'error';
        }
    }

    public function render()
    {
        return <<<'BLADE'
<div class="p-2 border rounded">
    <span class="font-semibold">ML Prediction:</span>
    <span class="ml-2">{{ $prediction }}</span>
    <span class="text-xs text-gray-500 ml-2">({{ $status }})</span>
</div>
BLADE;
    }
}

  ]]></Code>
  <Code id="Listing4" language="PHP"><![CDATA[
<?php
namespace App\Console\Commands;

use Illuminate\Console\Command;
use Illuminate\Support\Facades\Http;

/**
 * Runs an offline simulation: replays 24 h of h
istorical sensor
 * data against the digital twin REST API, compares actuator
 * outputs to golden-path CSV, and prints accuracy stats.
 */
```

```
class DigitalTwinValidate extends Command
{
    protected $signature = 'digital-twin:validate {--file=storage/app/replay.csv}';
    protected $description = 'Offline validation of the digital twin against replay data';

    public function handle(): int
    {
        $file = $this->option('file');
        if (!is_readable($file)) {
            $this->error("Replay file not found: $file");
            return self::FAILURE;
        }

        $lines   = file($file, FILE_IGNORE_NEW_LINES);
        $mape    = 0; $count = 0;

        foreach ($lines as $row) {
            [$ts, $slug, $value] = str_getcsv($row);
            $resp = Http::post('http://localhost:9000/twin/infer', [
                'timestamp' => $ts,
                'slug'      => $slug,
                'value'     => (float)$value,
            ])->json();

            $expected = $this->goldenPath($ts, $slug);
            $mape    += abs($resp['actuator'] - $expected) / max(1e-6, $expected);
            $count++;
        }
        $mape = $mape / $count * 100;
        $this->info("Digital-twin MAPE: " . number_format($mape, 2) . " %");
        return $mape < 5 ? self::SUCCESS : self::FAILURE;
    }

    private function goldenPath(string $ts, string $slug): float
    {
        // Placeholder: replace with lookup from CSV or DB
        return 42.0;
    }
}

  ]]></Code>
</ComputerProgramListing>
```