



CIFS Compliance Guide: Creating Waze-Compatible State DOT Data Feeds

Overview

The **Closure and Incident Feed Specification (CIFS)** is Waze's open protocol for describing road incidents and closures in partner data feeds. This guide shows state DOTs how to create CIFS-compliant data feeds that integrate with Waze for Drivers and the DOT Corridor Communicator platform.






What You'll Learn:

- CIFS message format and required fields
- How to convert your existing data (WZDx, 511, ATMS) to CIFS format
- Implementation examples in Python and Node.js
- Best practices for polyline encoding and incident classification
- API integration with DOT Corridor Communicator

What is CIFS?

CIFS is a JSON-based specification developed by Waze (Google) to standardize how traffic agencies share incident and closure data.

Why CIFS Matters

-  **Direct Waze Integration:** Your incidents appear in Waze navigation for millions of drivers
-  **Standardized Format:** Single specification accepted by multiple navigation platforms
-  **Simple Implementation:** JSON format with straightforward field requirements
-  **Flexible:** Supports both closures and incidents with detailed classification
-  **WZDx Compatible:** DOT Corridor Communicator auto-converts WZDx to CIFS

Official Specification: <https://developers.google.com/waze/data-feed/cifs-specification>

CIFS Message Structure

Required Fields

Every CIFS incident must include:

Field	Type	Description	Example
id	String	Unique identifier (alphanumeric)	"IA-80-WB-2025-001"
type	Enum	Incident classification	"ROAD_CLOSED", "ACCIDENT", "HAZARD"
polyline	Array	Lat/long coordinates (≥ 6 decimals)	[[41.656250, -91.530556], [41.656389, -91.530278]]
street	String	Road name	"Interstate 80 Westbound"
starttime	ISO8601	Start date/time with timezone	"2025-01-17T06:00:00-06:00"

Incident Types

Only `ROAD_CLOSED` triggers a full closure in navigation:

- `ROAD_CLOSED` : Complete road closure (construction, flooding, crash)
- `ACCIDENT` : Traffic collision
- `HAZARD` : Road hazards (debris, pothole, ice)
- `POLICE` : Police presence
- `JAM` : Traffic congestion
- `CHIT_CHAT` : General traveler information

Requested Fields (Highly Recommended)

Field	Type	Description	Example
subtype	String	Detailed classification	"HAZARD_WEATHER_FLOOD", "ACCIDENT_MAJOR"
direction	Enum	Traffic impact	"BOTH_DIRECTIONS", "ONE_DIRECTION"
endtime	ISO8601	End date/time (defaults to +14 days)	"2025-01-17T18:00:00-06:00"
description	String	Brief cause (<40 chars recommended)	"Bridge repair, right 2 lanes closed"

Optional Fields

Field	Type	Description
lane_impact	Object	Detailed lane closure data (WZDx format)
schedule	Object	Recurring closure patterns

Complete CIFS Feed Example

```
{
  "incidents": [
    {
      "id": "IA-80-WB-2025-001",
      "type": "ROAD_CLOSED",
      "polyline": [
        [41.656250, -91.530556],
        [41.656389, -91.530278],
        [41.656528, -91.530000]
      ],
      "street": "Interstate 80 Westbound",
      "starttime": "2025-01-20T06:00:00-06:00",
      "endtime": "2025-01-25T18:00:00-06:00",
      "direction": "ONE_DIRECTION",
      "subtype": "ROAD_CLOSED_CONSTRUCTION",
      "description": "Bridge deck replacement - full closure",
      "lane_impact": {
        "all_lanes_closed": true,
        "lanes_closed": ["lane-1", "lane-2"]
      }
    },
    {
      "id": "IA-35-NB-2025-042",
      "type": "HAZARD",
      "polyline": [
        [41.590556, -93.620278]
      ],
      "street": "Interstate 35 Northbound",
      "starttime": "2025-01-17T08:30:00-06:00",
      "direction": "ONE_DIRECTION",
      "subtype": "HAZARD_ON_ROAD_OBJECT",
      "description": "Debris in right lane"
    }
  ]
}
```

```
},
{
  "id": "IA-380-SB-2025-015",
  "type": "ACCIDENT",
  "polyline": [
    [42.011111, -91.663889],
    [42.010833, -91.663611]
  ],
  "street": "Interstate 380 Southbound",
  "starttime": "2025-01-17T07:15:00-06:00",
  "direction": "ONE_DIRECTION",
  "subtype": "ACCIDENT_MAJOR",
  "description": "Multi-vehicle crash, left lane blocked"
}
]
```

CIFS Subtype Classifications

Road Closures

- ROAD_CLOSED_CONSTRUCTION - Construction/maintenance work
- ROAD_CLOSED_EVENT - Special event closure
- ROAD_CLOSED_HAZARD - Hazardous conditions (flooding, fire)

Accidents

- ACCIDENT_MINOR - Fender bender, minor delay
- ACCIDENT_MAJOR - Serious crash with lane blockage

Hazards

- HAZARD_ON_ROAD - General road hazard
- HAZARD_ON_ROAD_OBJECT - Debris, tire, furniture
- HAZARD_ON_SHOULDER - Shoulder hazard
- HAZARD_WEATHER - Weather-related hazard
- HAZARD_WEATHER_FLOOD - Flooding
- HAZARD_WEATHER_FOG - Dense fog
- HAZARD_WEATHER_HAIL - Hail
- HAZARD_WEATHER_ICE - Ice on roadway

Full list: <https://developers.google.com/waze/data-feed/incident-information/incident-and-closure-subtypes>

Converting WZDx to CIFS

The DOT Corridor Communicator automatically converts WZDx feeds to CIFS format. Here's the mapping logic:

Field Mapping

WZDx Field	CIFS Field	Conversion Logic
id	id	Direct mapping

road_names[0]	street	First road name + direction
beginning_accuracy	polyline	Extract coordinates from geometry
start_date	starttime	Convert to ISO8601 with timezone
end_date	endtime	Convert to ISO8601 with timezone
vehicle_impact	type	Map to CIFS type (see below)
event_type	subtype	Map to CIFS subtype
description	description	Truncate to 40 chars if needed
lanes[].status	lane_impact	Convert lane closure data

WZDx Event Type → CIFS Type

```
WZDX_TO_CIFS_TYPE = {
  'work-zone': 'ROAD_CLOSED',
  'detour': 'ROAD_CLOSED',
  'restriction': 'HAZARD',
  'incident-minor': 'ACCIDENT',
  'incident-major': 'ACCIDENT',
  'weather': 'HAZARD'
}

WZDX_TO_CIFS_SUBTYPE = {
  'work-zone': 'ROAD_CLOSED_CONSTRUCTION',
  'incident-crash': 'ACCIDENT_MAJOR',
  'incident-debris': 'HAZARD_ON_ROAD_OBJECT',
  'weather-ice': 'HAZARD_WEATHER_ICE',
  'weather-fog': 'HAZARD_WEATHER_FOG'
}
```

Implementation Guide

Step 1: Gather Source Data

Identify your traffic event data sources:

- **WZDx Feeds:** Work zone data exchange format
- **511 Systems:** Traveler information APIs
- **ATMS:** Advanced Traffic Management Systems (SunGuide, ATMS.now, etc.)
- **CAD Systems:** Computer-Aided Dispatch for incidents
- **Maintenance Systems:** Planned construction/maintenance events

Step 2: Extract Required Fields

For each event, extract:

1. **Unique ID:** Create a consistent format like `{STATE}-{ROUTE}-{DIR}-{YEAR}-{SEQ}`
2. **Location:** Get lat/long coordinates (minimum 6 decimal places)

3. **Road Name:** Full road name with direction (e.g., "I-80 Eastbound")
4. **Start/End Times:** ISO8601 format with timezone offset
5. **Event Classification:** Map to CIFS type/subtype

Step 3: Build Polyline

For Linear Events (closures, work zones):

```
# Extract coordinates along the affected segment
polyline = [
    [start_lat, start_lon],
    [mid_lat, mid_lon],      # Optional intermediate points
    [end_lat, end_lon]
]
```

For Point Events (crashes, debris):

```
# Single coordinate + direction field
polyline = [[incident_lat, incident_lon]]
direction = "ONE_DIRECTION" # Required for single-point events
```

Step 4: Format ISO8601 Timestamps

```
from datetime import datetime, timezone

# UTC time
start_time = datetime(2025, 1, 20, 12, 0, 0, tzinfo=timezone.utc)
cifs_time = start_time.isoformat() # "2025-01-20T12:00:00+00:00"

# Local time with offset (Central Time = UTC-6)
import pytz
central = pytz.timezone('America/Chicago')
local_time = central.localize(datetime(2025, 1, 20, 6, 0, 0))
cifs_time = local_time.isoformat() # "2025-01-20T06:00:00-06:00"
```

Step 5: Validate and Submit

Validation checklist:

- ☒ All required fields present (`id` , `type` , `polyline` , `street` , `starttime`)
- ☒ Polyline coordinates have ≥6 decimal places
- ☒ `type` is valid CIFS incident type
- ☒ Timestamps are valid ISO8601 with timezone
- ☒ `id` is unique within your feed
- ☒ Direction specified for single-point incidents

Code Examples

Python: Convert WZDx to CIFS

```

import json
from datetime import datetime
import pytz

def wzdxd_to_cifs(wzdxd_feed):
    """Convert WZDX feed to CIFS format"""

    incidents = []

    for feature in wzdxd_feed.get('features', []):
        props = feature['properties']
        geom = feature['geometry']

        # Extract polyline from WZDX geometry
        polyline = []
        if geom['type'] == 'LineString':
            # Coordinates are [lon, lat] in GeoJSON, reverse to [lat, lon] for CIFS
            polyline = [[coord[1], coord[0]] for coord in geom['coordinates']]
        elif geom['type'] == 'Point':
            polyline = [[geom['coordinates'][1], geom['coordinates'][0]]]

        # Map WZDX event type to CIFS type
        event_type = props.get('event_type', 'work-zone')
        cifs_type = 'ROAD_CLOSED' if 'work-zone' in event_type else 'HAZARD'

        # Build CIFS incident
        incident = {
            'id': props['id'],
            'type': cifs_type,
            'polyline': polyline,
            'street': props.get('road_names', ['Unknown'])[0],
            'starttime': props.get('start_date'),
            'direction': 'BOTH_DIRECTIONS' if props.get('direction') == 'both' else 'ONE_DIRECTION'
        }

        # Add optional fields if available
        if 'end_date' in props:
            incident['endtime'] = props['end_date']

        if 'description' in props:
            incident['description'] = props['description'][:40]

        # Determine subtype
        if event_type == 'work-zone':
            incident['subtype'] = 'ROAD_CLOSED_CONSTRUCTION'

        incidents.append(incident)

    return {'incidents': incidents}

# Example usage
with open('wzdxd_feed.json') as f:

```

```

wzdx_data = json.load(f)

cifs_feed = wzdx_to_cifs(wzdx_data)

with open('cifs_feed.json', 'w') as f:
    json.dump(cifs_feed, f, indent=2)

print(f"Converted {len(cifs_feed['incidents'])} incidents to CIFS format")

```

Node.js: Generate CIFS Feed from Database

```

const fs = require('fs');

async function generateCIFSFeed(databaseConnection) {
  // Query active incidents from your database
  const incidents = await databaseConnection.query(`
    SELECT
      incident_id,
      event_type,
      route_name,
      direction,
      start_lat,
      start_lon,
      end_lat,
      end_lon,
      start_time,
      end_time,
      description
    FROM traffic_incidents
    WHERE status = 'ACTIVE'
    AND end_time > NOW()
  `);

  const cifsFeed = {
    incidents: incidents.map(inc => {
      // Build polyline
      const polyline = [];
      polyline.push([inc.start_lat, inc.start_lon]);
      if (inc.end_lat && inc.end_lon) {
        polyline.push([inc.end_lat, inc.end_lon]);
      }

      // Map database event type to CIFS type
      const typeMap = {
        'CONSTRUCTION': 'ROAD_CLOSED',
        'CRASH': 'ACCIDENT',
        'DEBRIS': 'HAZARD',
        'WEATHER': 'HAZARD'
      };

      const subtypeMap = {

```

```

        'CONSTRUCTION': 'ROAD_CLOSED_CONSTRUCTION',
        'CRASH': 'ACCIDENT_MAJOR',
        'DEBRIS': 'HAZARD_ON_ROAD_OBJECT',
        'WEATHER': 'HAZARD_WEATHER'
    };

    return {
        id: inc.incident_id,
        type: typeMap[inc.event_type] || 'HAZARD',
        subtype: subtypeMap[inc.event_type],
        polyline: polyline,
        street: `${inc.route_name} ${inc.direction}`,
        starttime: new Date(inc.start_time).toISOString(),
        endtime: new Date(inc.end_time).toISOString(),
        direction: inc.direction === 'BOTH' ? 'BOTH_DIRECTIONS' : 'ONE_DIRECTION',
        description: inc.description.substring(0, 40)
    };
})
};

// Write to file
fs.writeFileSync('cifs_feed.json', JSON.stringify(cifsFeed, null, 2));

console.log(`Generated CIFS feed with ${cifsFeed.incidents.length} incidents`);
return cifsFeed;
}

module.exports = { generateCIFSFeed };

```

REST API Integration

```

import requests
import json

def publish_to_corridor_communicator(cifs_feed, api_key):
    """
    Publish CIFS feed to DOT Corridor Communicator
    """

    url = "https://api.corridor-communicator.com/api/cifs/submit"

    headers = {
        'Content-Type': 'application/json',
        'Authorization': f'Bearer {api_key}'
    }

    response = requests.post(url, json=cifs_feed, headers=headers)

    if response.status_code == 200:
        result = response.json()
        print(f"✅ Successfully published {result['incidents_processed']} incidents")

```

```

        return result
    else:
        print(f"❌ Error: {response.status_code} - {response.text}")
        return None

# Example usage
with open('cifs_feed.json') as f:
    feed = json.load(f)

api_key = "your-api-key-here"
publish_to_corridor_communicator(feed, api_key)

```

DOT Corridor Communicator Integration

Automatic CIFS Conversion

The DOT Corridor Communicator provides automatic WZDx → CIFS conversion:

Endpoint: GET /api/convert/cifs

Returns all active incidents in CIFS format, converted from:

- 46+ state WZDx feeds
- FEU-G data (5 states)
- Custom state APIs (10+ states)

Example Response

```

{
  "incidents": [
    {
      "id": "IA-80-WB-2025-001",
      "type": "ROAD_CLOSED",
      "street": "Interstate 80 Westbound",
      "polyline": [[41.656250, -91.530556], [41.656389, -91.530278]],
      "starttime": "2025-01-20T06:00:00-06:00",
      "endtime": "2025-01-25T18:00:00-06:00",
      "direction": "ONE_DIRECTION",
      "subtype": "ROAD_CLOSED_CONSTRUCTION",
      "description": "Bridge deck replacement",
      "source": {
        "state": "IA",
        "feed_type": "wzdx",
        "last_updated": "2025-01-17T10:30:00Z"
      }
    }
  ],
  "metadata": {
    "total_incidents": 1,
    "generated_at": "2025-01-17T10:35:00Z",
    "feed_version": "1.0"
  }
}

```

```
}  
}
```

State-Specific Feeds

Get CIFS data for a specific state:

Endpoint: `GET /api/convert/cifs?state=IA`

Corridor-Specific Feeds

Get CIFS data for a specific corridor (e.g., I-80 across multiple states):

Endpoint: `GET /api/convert/cifs?corridor=I-80`

Best Practices

1. Polyline Precision

✅ **DO:** Use at least 6 decimal places for lat/long

```
"polyline": [[41.656250, -91.530556]]
```

❌ **DON'T:** Use rounded coordinates

```
"polyline": [[41.66, -91.53]] // Too imprecise
```

2. Unique Identifiers

✅ **DO:** Create stable, descriptive IDs

```
{STATE}-{ROUTE}-{DIR}-{YEAR}-{SEQUENCE}  
Example: IA-80-WB-2025-001
```

❌ **DON'T:** Use database auto-increment IDs alone

```
12345 // Not descriptive, may conflict with other agencies
```

3. Description Length

✅ **DO:** Keep descriptions concise (<40 characters)

```
"Bridge repair, right lane closed"
```

❌ **DON'T:** Write lengthy descriptions

```
"Due to ongoing bridge deck rehabilitation work on the westbound Interstate 80 bridge over the Cedar River, the right lane will be closed..."
```

4. End Time Handling

✅ **DO:** Provide accurate end times for closures

```
{
  "type": "ROAD_CLOSED",
  "starttime": "2025-01-20T06:00:00-06:00",
  "endtime": "2025-01-25T18:00:00-06:00"
}
```

❌ **DON'T:** Omit end times for known-duration events

```
{
  "type": "ROAD_CLOSED",
  "starttime": "2025-01-20T06:00:00-06:00"
  // Missing endtime - will default to +14 days!
}
```

5. Direction Specification

✅ **DO:** Specify direction for divided highways

```
{
  "street": "Interstate 80 Westbound",
  "direction": "ONE_DIRECTION"
}
```

❌ **DON'T:** Leave direction ambiguous

```
{
  "street": "I-80",
  "direction": "BOTH_DIRECTIONS" // Unclear for divided highway
}
```

Testing Your CIFS Feed

Validation Checklist

Use this checklist before publishing your feed:

- ☐ All incidents have unique `id` values
- ☐ All `type` values are valid CIFS types
- ☐ All `polyline` coordinates have ≥ 6 decimal places
- ☐ All `starttime` values are ISO8601 with timezone
- ☐ `direction` is specified for single-point incidents
- ☐ `description` is < 100 characters (ideally < 40)
- ☐ `endtime` is provided for `ROAD_CLOSED` types
- ☐ Road closures use `type: "ROAD_CLOSED"` (not `HAZARD` or `JAM`)
- ☐ JSON syntax is valid (no trailing commas, proper quotes)

Online Validators

JSON Validator: <https://jsonlint.com/>

CIFS Schema Validator (if available from Waze partner portal)

Manual Testing

```
# Validate JSON syntax
cat cifs_feed.json | python -m json.tool

# Count incidents
cat cifs_feed.json | jq '.incidents | length'

# Check for required fields
cat cifs_feed.json | jq '.incidents[] | {id, type, polyline, street, starttime}'

# Verify coordinate precision (should show 6+ decimal places)
cat cifs_feed.json | jq '.incidents[0].polyline[0]'
```

Example: Iowa DOT CIFS Implementation

Current State

Iowa DOT provides WZDx data at:

- **URL:** <https://data.iowadot.gov/datasets/wzdx/api>
- **Format:** WZDx 4.2 GeoJSON
- **Update Frequency:** Every 5 minutes
- **Coverage:** 400-600 active work zones

CIFS Conversion Process

1. Fetch WZDx Feed

```
import requests
wzdx_url = "https://data.iowadot.gov/datasets/wzdx/api"
response = requests.get(wzdx_url)
wzdx_data = response.json()
```

2. Convert to CIFS

```
cifs_feed = wzdx_to_cifs(wzdx_data)
```

3. Publish to Waze

```
# Upload to Waze CCP (Waze for Cities Data Partners)
# Endpoint provided by Waze partnership agreement
```

4. Publish to DOT Corridor Communicator

```
publish_to_corridor_communicator(cifs_feed, api_key)
```

Results

- **470+ incidents** converted from WZDx to CIFS daily
- **5-minute update cycle** ensures real-time accuracy
- **Multi-state visibility** via DOT Corridor Communicator I-80 corridor view
- **Driver reach:** Millions of Waze users receive Iowa work zone alerts

Frequently Asked Questions

Q: Do I need a separate feed for each incident type?

A: No. CIFS supports mixing different incident types (`ROAD_CLOSED` , `ACCIDENT` , `HAZARD`) in a single JSON feed.

Q: What if I don't have exact coordinates for an incident?

A: Use the best available location data:

- **Linear events:** Use route mileposts converted to lat/long
- **Point events:** Use nearest intersection coordinates
- **Minimum precision:** 6 decimal places (≈ 0.1 meter accuracy)

Q: How often should I update my CIFS feed?

A: Update frequency depends on your data freshness:

- **Real-time incidents** (crashes, debris): Every 1-5 minutes
- **Planned closures** (construction): Every 15-60 minutes
- **Static events:** Daily updates acceptable

Q: Can I use CIFS for recurring events (e.g., nighttime lane closures)?

A: Yes! Use the optional `schedule` field:

```
{
  "id": "IA-80-WB-2025-RECURRING",
  "type": "ROAD_CLOSED",
  "schedule": {
    "days": ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"],
    "start_time": "21:00",
    "end_time": "05:00"
  }
}
```

Q: What timezone should I use for timestamps?

A: Use your local timezone with proper offset. Example for Central Time:

- **Standard Time (CST):** `-06:00`
- **Daylight Time (CDT):** `-05:00`

Or use UTC (`+00:00`) if you prefer. Waze will handle timezone conversion.

Q: How do I remove an incident from the feed?

A: Simply remove it from your JSON feed. Waze will automatically expire incidents that are no longer present in subsequent feed updates.

Integration Checklist

Use this checklist to implement CIFS at your DOT:

Phase 1: Planning

- ☐ Identify source systems (WZDx, 511, ATMS, CAD)
- ☐ Determine update frequency requirements
- ☐ Assign unique ID format for your incidents
- ☐ Map your event types to CIFS types/subtypes

Phase 2: Development

- ☐ Implement data extraction from source systems
- ☐ Build polyline generation logic
- ☐ Create ISO8601 timestamp conversion
- ☐ Implement CIFS JSON formatter
- ☐ Add validation checks

Phase 3: Testing

- ☐ Validate JSON syntax
- ☐ Test coordinate precision (≥ 6 decimals)
- ☐ Verify all required fields present
- ☐ Test with sample incidents
- ☐ Load test with full incident volume

Phase 4: Deployment

- ☐ Set up automated feed generation
- ☐ Configure update schedule (cron job, task scheduler)
- ☐ Implement error logging and monitoring
- ☐ Publish feed to DOT Corridor Communicator
- ☐ (Optional) Submit to Waze CCP partnership

Phase 5: Maintenance

- ☐ Monitor feed freshness daily
 - ☐ Review incident classification accuracy
 - ☐ Update subtype mappings as needed
 - ☐ Track user feedback from Waze drivers
-

Additional Resources

Official Documentation

- CIFS Specification: <https://developers.google.com/waze/data-feed/cifs-specification>
- Waze for Cities: <https://www.waze.com/wazeforcities>
- WZDx Specification: <https://github.com/usdot-jpo-ode/wzdx>

DOT Corridor Communicator APIs

- CIFS Endpoint: GET /api/convert/cifs
- State Filter: GET /api/convert/cifs?state={STATE_CODE}
- Corridor Filter: GET /api/convert/cifs?corridor={ROUTE_NAME}
- Submit Custom Feed: POST /api/cifs/submit

Tools

- JSON Validator: <https://jsonlint.com/>
- Coordinate Converter: <https://www.latlong.net/>
- Timezone Converter: <https://www.timeanddate.com/worldclock/converter.html>

Support

For questions about implementing CIFS feeds or integrating with DOT Corridor Communicator:

- Documentation: <https://corridor-communicator.com/docs>
- Email: support@corridor-communicator.com
- CCAI Pooled Fund: Contact your state's CCAI representative

Last Updated: 2025-01-17 **Version:** 1.0 **Maintained By:** Connected Corridors Advancement Initiative (CCAI)

For the latest version of this guide, visit the DOT Corridor Communicator Documentation Portal.