EOPF-ZARR GDAL DRIVER: SPRINT EXECUTION GUIDE

Quick-Start for Sprint Leads & Contributors

How to Use This Document

1. **Sprint Leads**: Use this for planning & capacity allocation
2. **Individual Contributors**: Check your assigned stories & acceptance criteria
3. **QA/Testers**: Use DoD checklist before marking stories done
4. **Stakeholders**: Track progress via GitHub Projects board

Sprint Cadence (2-week cycles)

```
Week 1                    Week 2
Mon  - Sprint Planning     Mon  - Mid-sprint check-in
Tue  - Daily standups      Tue  - Daily standups
Wed  - Dev continues       Wed  - Dev continues
Thu  - Dev continues       Thu  - Sprint Review (4pm CET)
Fri  - Dev + planning      Fri  - Sprint Retro (5pm CET)
       next sprint
```

Sprint 1: Foundations (Weeks 1-2)

Sprint Goal

"Establish basic Zarr reading and GDAL plugin registration so that developers can load EOPF SLC products."

Team Capacity

- Lead Developer: 25 points
- Contributor 1: 15 points
- QA Lead: 10 points
- **Total**: 50 points

Stories Assigned

Story	Points	Dev	QA	Status
1.1.1: SLC Burst Reading	8	Lead	QA	TO-DO
1.2.1: Driver Registration	3	Contrib1	QA	TO-DO
1.2.2: Identify() Method	3	Contrib1	QA	TO-DO
1.3.1: CI/CD Setup	8	QA	Contrib1	TO-DO
1.3.2: Unit Test Suite (part 1)	13	Lead + QA	QA	TO-DO
1.1.6: Metadata Extraction (part 1)	5	Contrib1	QA	TO-DO

Success Metrics

- [ ] SLC burst reading working with `gdalinfo`
- [ ] Driver visible in `gdalinfo --formats`
- [ ] 3+ unit tests passing
- [ ] GitHub Actions CI running on PRs
- [ ] Sample Zarr file in repo accessible

Key Dependencies

- GDAL 3.x development headers installed on all dev machines
- Access to EOPF sample Zarr files (public link: zarr.eopf.copernicus.eu)
- CMake 3.10+ configured

Blockers & Risks

- Zarr library not installed? → Add to build instructions
- GDAL version mismatch? → Document requirements in README
- GitHub Actions quota? → Start simple, expand later

Definition of Done Checklist (for each story)

```
☑ Implementation complete
☑ Code compiles without warnings (-Wall)
☑ Unit tests added and passing
☑ Documentation/comments added
☑ Code reviewed (peer + lead)
☑ Integrated with main branch (no conflicts)
☑ CI/CD green on all platforms
☑ GitHub issue closed with description of changes
```

Demo Content (Sprint Review)

- **Live Demo 1**: Open SLC Zarr with `gdalinfo`, show burst names
- **Live Demo 2**: Read single burst data in Python via gdal
- **Show**: GitHub Actions passing on all platforms
- **Announce**: Public GitHub repo with v0.1-alpha tag

Resources & Documentation Links

- GDAL Raster Driver Tutorial
- GDAL Development Practices
- EOPF Zarr Structure
- Sample Data: `https://s3.example.com/eopf-samples/sentinel1_slc_sample.zarr`

Sprint 2: Advanced Reading (Weeks 3-4)

Sprint Goal

"Add GRD support, CPM version validation, and burst selection so users can work with specific areas of interest."

Team Capacity

- Lead Developer: 25 points
- Contributor 1: 15 points
- QA Lead: 10 points

Stories Assigned

Story	Points	Dev	QA	Status
1.1.2: GRD Product Reading	5	Contrib1	QA	TO-DO
1.1.3: CPM Version Validation	3	Lead	QA	TO-DO
1.1.4: Burst Selection (open options)	5	Lead	QA	TO-DO
1.2.3: Error Handling & Logging	5	Contrib1	QA	TO-DO
1.3.2: Unit Test Suite (part 2)	13	QA	Lead	TO-DO
1.3.3: Integration Tests (real EOPF data)	8	QA	Lead	TO-DO

Success Metrics

- ☐ GRD products readable with two polarization bands
- ☐ Burst selection via `-oo BURST=IW1_VV_001` working
- ☐ CPM version check prevents loading incompatible data
- ☐ Integration tests pass with real EOPF data
- ☐ Error messages helpful (log level DEBUG shows context)

Key Dependencies

- Sprint 1 complete (driver registered)

- Real GRD sample from EOPF
- Real SLC sample for version testing

Demo Content

- **Demo 1**: Load GRD product in QGIS, show VV + VH bands separately
- **Demo 2**: Select single burst, show memory savings
- **Demo 3**: Try incompatible product, show clear error + fix suggestion
- **Metrics**: Build time, test run time on all platforms

Sprint 3: Memory Efficiency (Weeks 5-6)

Sprint Goal

"Optimize chunk streaming so large bursts don't require full load, enabling processing of massive datasets."

Stories Assigned

Story	Points	Dev	QA	Status
1.1.5: Zarr Chunk Streaming	8	Lead	QA	TO-DO
1.2.4: PAM Support	3	Contrib1	QA	TO-DO
Perf Benchmarking (1.1.5 related)	8	QA	Lead	TO-DO

Success Metrics

- [ ] Burst reads < 2 seconds for 10MB block
- [ ] Memory constant regardless of burst size
- [ ] Benchmark report published
- [ ] Documentation includes performance notes

Demo Content

- **Benchmark chart**: Read time vs burst size
- **Memory graph**: Constant memory footprint demo
- **Live test**: Load 100MB burst with < 500MB RAM used

Sprint 4: Metadata Mastery (Weeks 7-8)

Sprint Goal

"Extract all EOPF metadata so users understand product characteristics and integrate with downstream workflows."

Stories Assigned

Story	Points	Dev	QA	Status
1.1.6: Metadata Extraction (part 2)	5	Contrib1	QA	TO-DO
Metadata documentation examples	8	Lead	Contrib1	TO-DO

Version

v0.2-MVP Released at end of Sprint 4

Demo Content

- Show metadata extraction example in Python
- Integration with xarray workflows
- QGIS metadata display

Sprint 5-8: Phase 2 Planning

Defer detailed planning until Phase 1 complete. Update roadmap based on learnings.

General Phase 2 Focus

- Georeferencing (Stories 2.1.x)
- GCP extraction and QGIS integration
- Calibration fundamentals (sigma-nought)
- Basic multilooking

Team Additions

- SAR domain expert needed for Stories 2.2.x+

GitHub Project Management

Issue Template

```
## Story: [Story ID] [Story Title]

### User Story
**As a** [persona]
**I want** [capability]
**So that** [value]

### Acceptance Criteria
- [ ] Criterion 1
- [ ] Criterion 2

### Tasks
- [ ] Task 1
- [ ] Task 2

### Definition of Done
- [ ] All criteria met
- [ ] Tests passing
- [ ] Reviewed
- [ ] Documented

### Estimates
- Story points: [X]
- Sprint: [N]

### Labels
`priority:p0` `phase:1` `component:zarr-reading` `testing:unit`
```

GitHub Project Board Columns

```
Backlog → Planned → In Progress → Code Review → Testing → Done
```

Labels to Use

```
priority:p0/p1/p2
phase:1/2/3/4
component:zarr-reading/gdal-standards/testing/georeferencing/calibration/preprocessing/interferometry
testing:unit/integration/manual
blocked:yes
```

Code Review Checklist

Before approving a PR:

```
Correctness
- [ ] Logic is correct
- [ ] No buffer overflows
- [ ] Error handling complete
- [ ] Edge cases handled

Quality
- [ ] Code follows GDAL conventions
- [ ] No unused variables
- [ ] Meaningful variable names
- [ ] Comments for complex logic

Testing
- [ ] Unit tests added
```

```
- [ ] Test coverage > 70%
- [ ] Integration tests pass
- [ ] No regressions

Documentation
- [ ] Code comments clear
- [ ] README/docs updated
- [ ] Examples included
- [ ] Docstrings/function docs complete

Build & CI
- [ ] Compiles on all platforms
- [ ] GitHub Actions green
- [ ] No new compiler warnings
- [ ] Build time reasonable
```

Common Development Tasks

Setting Up Dev Environment

```
# Clone repo
git clone https://github.com/EOPF-Sample-Service/GDAL-ZARR-EOPF.git
cd GDAL-ZARR-EOPF

# Install dependencies (Ubuntu)
sudo apt-get install -y libgdal-dev cmake g++ python3-pip
pip install pytest numpy xarray zarr

# Build
mkdir build && cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
cmake --build . -j$(nproc)

# Run tests
ctest --verbose
pytest ../tests/
```

Creating a Feature Branch

```
# From main, pull latest
git checkout main
git pull origin main

# Create feature branch (sprint_number-story_id-description)
git checkout -b sprint1-1_1_1-slc-burst-reading

# Make changes, commit frequently
git add .
git commit -m "1.1.1: Implement burst reading - parse Zarr structure"

# Push when ready for review
git push origin sprint1-1_1_1-slc-burst-reading

# Create GitHub PR with story details
```

Running Tests Locally

```
# All tests
cd build && ctest --verbose

# Specific test file
pytest ../tests/test_slc_reading.py -v

# Specific test
pytest ../tests/test_slc_reading.py::test_burst_open -v

# With coverage
pytest --cov=gdal_eopfzarr ../tests/ --cov-report=html
```

Creating Sample Test Data

```
# tests/create_sample_zarr.py
import zarr
import numpy as np
```

```
def create_sample_slc_zarr(output_path):
    """Create minimal EOPF SLC Zarr for testing"""
    root = zarr.open(output_path, mode='w')

    # Metadata
    root.attrs['product_type'] = 'S1_SLC'
    root.attrs['cpm_version'] = '2.6.0'

    # Single burst (IW1_VV_001)
    burst = root.create_group('SENTINEL1_IW1_VV_001')

    # Complex data (1024 x 1024)
    data = np.array(
        [np.random.rand(1024) + 1j * np.random.rand(1024)
         for _ in range(1024)],
        dtype=np.complex64
    )
    burst.create_dataset('measurement', data=data, chunks=(256, 256))

    # Metadata arrays
    burst.create_dataset('latitude', data=np.arange(45.0, 45.01, 0.00001))
    burst.create_dataset('longitude', data=np.arange(10.0, 10.01, 0.00001))

    # Calibration LUT
    qual = burst.create_group('quality')
    cal = qual.create_group('calibration')
    cal.create_dataset('sigma_nought',
                       data=np.ones((257, 257)) * -10.0)

    return output_path
```

Common Issues & Troubleshooting

Issue: "gdal library not found"

```
Solution: Install GDAL dev headers
$ apt-get install libgdal-dev
```

Issue: Complex data not displaying in QGIS

```
Solution: QGIS may not support CComplex64 well
Workaround: Create virtual band with amplitude instead
```

Issue: CI build failing on Windows

```
Solution: Check GDAL paths in CMakeLists.txt
Verify vcpkg integration or manual include/lib paths
```

Issue: Zarr chunks not aligned with GDAL blocks

```
Solution: Profile and benchmark chunk sizes
Consider custom block size matching Zarr chunks
```

Retrospective Template

What went well?

- Development pace
- Team communication
- Testing coverage

What could improve?

- Estimation accuracy
- Code review turnaround
- Documentation clarity

Action items for next sprint:

- [ ] Action 1
- [ ] Action 2

Release Checklist

Before v0.2 (MVP) Release

```
Code Quality
- [ ] All Phase 1 stories done
- [ ] Code coverage > 70%
- [ ] No critical bugs open
- [ ] All tests passing on all platforms

Documentation
- [ ] README complete
- [ ] Installation guide written
- [ ] API documentation generated
- [ ] Example notebooks created

Community
- [ ] GitHub repo public
- [ ] License file present (MIT)
- [ ] CONTRIBUTING.md written
- [ ] Code of Conduct added

Release Artifacts
- [ ] Git tag created (v0.2)
- [ ] Release notes written
- [ ] Binary builds available
- [ ] PyPI package (if applicable)

Announcement
- [ ] EURAC news post
- [ ] EOPF community forum post
- [ ] GitHub discussions enabled
```

Metrics to Track

Per-Sprint

```
- Story completion rate (%)
- Velocity (points completed)
- Burndown chart
- Code review cycle time
- Test pass rate
```

Per-Release

```
- Build time (each platform)
- Test runtime
- Code coverage (%)
- Issue density (bugs per 1000 LOC)
```

Long-term

```
- Feature adoption (GH stars, forks)
- Community contributions
- Performance (read speed, memory)
- Documentation quality
```

Helpful Links

- **GDAL Docs**: https://gdal.org/
- **EOPF Site**: https://zarr.eopf.copernicus.eu/
- **EOPF Webinar Recording**: https://www.youtube.com/watch?v=_JC9cVoomDY
- **Sample Data**: https://zarr.eopf.copernicus.eu/stac/
- **GitHub Project**: https://github.com/EOPF-Sample-Service/GDAL-ZARR-EOPF

Last Updated: December 5, 2025
Next Review: End of Sprint 2