Phase 1: Enable CRAM 3.1 "Normal" Profile Writing - Detailed Plan

Overview

Phase 1 enables htsjdk to **write** valid CRAM 3.1 files using a "normal" profile equivalent to htslib's default. This uses rANS Nx16 and Name Tokenisation — codecs that already have working encoders. No new codec implementations are required. FQZComp encoding is explicitly out of scope (Phase 2).

A development branch `cn_cram_3_1_write` already contains 4 commits (164 insertions, 64 deletions across 11 files) that implement this work. The plan below is based on that branch's changes, with additional review and testing considerations.

Pre-Work: Review the Existing Branch

Before writing code, review the 4 commits on `origin/cn_cram_3_1_write` to determine whether to cherry-pick/merge or re-implement from scratch on a clean branch:

Commit	Description	Files
`881163d75`	Enable a naive CRAM 3.1 write profile	8 files, +67/-48
`d516bce7b`	Changes based on instrumented trial runs	NameTokenisationEncode tuning
`0135312a4`	CRAM 3.1 write tests and code cleanup	Tests + HtsCRAMCodec31Test
`7ff46c4c0`	Temp fix for preSorted=false default	ReadsEncoderOptions workaround

Decision needed: Merge the branch as-is, squash-merge, or cherry-pick individual changes. The branch is 4 commits ahead of master with no conflicts currently.

Task 1: Remove the CRAMEncoderV3_1 Write Block

File: `src/main/java/htsjdk/beta/codecs/reads/cram/cramV3_1/CRAMEncoderV3_1.java`

Change: Delete the `throw new CRAMException("CRAM v3.1 encoding is not yet supported")` on line 25 of the constructor. The class otherwise extends `CRAMEncoder` correctly and returns the right version.

Risk: None — the encoder inherits the full write path from `CRAMEncoder`. The only thing blocking it was this explicit throw.

Task 2: Change the Default CRAM Version to 3.1

File: `src/main/java/htsjdk/samtools/cram/common/CramVersions.java`

Change: `DEFAULT_CRAM_VERSION = CRAM_v3` → `DEFAULT_CRAM_VERSION = CRAM_v3_1`

Impact: All code paths that create a CRAM writer without specifying a version will now produce CRAM 3.1 files instead of 3.0. This affects:

- `CRAMFileWriter` (low-level SAMTools API)
- `SAMFileWriterFactory.makeWriter()` (standard API)
- The HTS plugin framework codec resolution

Consideration: Downstream users who depend on 3.0 output will need to explicitly request 3.0. This is the same approach htslib took (htslib defaults to 3.1 already). Users can override via `SAMFileWriterFactory` or `CRAMEncodingStrategy`.

Task 3: Remove the ReadsResolver CRAM 3.1 Filter Override

File: `src/main/java/htsjdk/beta/plugin/registry/ReadsResolver.java`

Change: Delete the `filterByVersion()` override (lines ~212-235) that explicitly excludes CRAM 3.1 when the requested version is `NEWEST_VERSION`. This method was a temporary workaround to prevent the HTS plugin framework from selecting the 3.1 encoder (which would throw). With the write block removed, this filter is no longer needed.

Side effect: Also removes unused imports for `CRAMCodecV3_1`, `ReadsFormats`, `List`, `Collectors`.

Task 4: Update CompressionHeaderEncodingMap for CRAM 3.1 Codecs

File: `src/main/java/htsjdk/samtools/cram/structure/CompressionHeaderEncodingMap.java`

4a: Switch all rANS 4x8 usages to rANS Nx16

The private helper methods `putExternalRansOrderOneEncoding()` and `putExternalRansOrderZeroEncoding()` currently use `BlockCompressionMethod.RANS` with `RANS4x8Params.ORDER`. Change to `BlockCompressionMethod.RANSNx16` with `RANSNx16Params.ORDER`.

This affects these data series (no change to their order-0/order-1 assignments, just the codec):

- `AP_AlignmentPositionOffset` → RANSNx16 order-0
- `BA_Base` → RANSNx16 order-1
- `BF_BitFlags` → RANSNx16 order-1
- `CF_CompressionBitFlags` → RANSNx16 order-1
- `NS_NextFragmentReferenceSequenceID` → RANSNx16 order-1
- `QS_QualityScore` → RANSNx16 order-1
- `RG_ReadGroup` → RANSNx16 order-1
- `RI_RefId` → RANSNx16 order-0
- `RL_ReadLength` → RANSNx16 order-1
- `TS_InsertSize` → RANSNx16 order-1

All other data series (BS, DL, FC, FN, FP, HC, IN, MF, MQ, NF, NP, PD, RS, SC, TL) remain GZIP — no change.

4b: Switch RN_ReadName to Name Tokeniser

In `initializeDefaultEncodings()`, change the `RN_ReadName` line from:

```
putExternalByteArrayStopTabGzipEncoding(encodingStrategy,
DataSeries.RN_ReadName);
```

to call a new method:

```
putByteArrayStopNameTokEncoding(encodingStrategy, DataSeries.RN_ReadName);
```

Add the new private method `putByteArrayStopNameTokEncoding()` that uses:

- `ByteArrayStopEncoding` with `NameTokenisationDecode.NAME_SEPARATOR` as the stop byte
- `BlockCompressionMethod.NAME_TOKENISER` as the compressor

4c: Update getBestExternalCompressor() to use RANSNx16

This method tries GZIP, rANS order-0, rANS order-1 and picks the smallest. Change the rANS candidates from `BlockCompressionMethod.RANS` / `RANS4x8Params` to `BlockCompressionMethod.RANSNx16` / `RANSNx16Params`. This method is used for tag block compression (not the main data series).

Import changes

Remove `import htsjdk.samtools.cram.compression.rans.rans4x8.RANS4x8Params`; add `import htsjdk.samtools.cram.compression.rans.ransnx16.RANSNx16Params` and `import htsjdk.samtools.cram.compression.nametokenisation.NameTokenisationDecode`.

Task 5: Tune NameTokenisationEncode.tryCompress()

File: `src/main/java/htsjdk/samtools/cram/compression/nametokenisation/NameTokenisationEncode.java`

5a: Narrow the flag combinations tried

The current code tries 7 combinations of RANSNx16 flags for each sub-stream. The branch narrows this to 4 (`{0, RLE, PACK, PACK|ORDER}`) based on empirical testing showing these yield the best results. The removed combinations (`ORDER` alone, `RLE|ORDER`, `PACK|RLE|ORDER`) were rarely winners and slowed encoding.

Same change for the Range encoder path (used in tests but not the default write profile).

5b: Add CAT fallback

When no compression combination beats the uncompressed size, fall back to CAT (uncompressed passthrough) rather than using a compression that expands the data. Add this check after the flag-set loop for both the rANS and Range paths.

Task 6: Update CompressorCache Comments

File: `src/main/java/htsjdk/samtools/cram/structure/CompressorCache.java`

Minor: Update comments for the RANS and RANSNx16 cache cases to clarify the caching approach (cosmetic, no behavioral change).

Task 7: Update CompressionUtils Error Message

File: `src/main/java/htsjdk/samtools/cram/compression/CompressionUtils.java`

Change error message from `"Failed to allocate sufficient buffer size for RANS coder."` to `"Failed to allocate sufficient buffer size for CRAM codec."` since this utility is now shared across multiple codecs.

Task 8: Add CRAM 3.1 Write Tests

8a: Update CRAM31Tests.java (low-level API write test)

File: `src/test/java/htsjdk/samtools/cram/CRAM31Tests.java`

- Rename `testCRAM31RoundTrip` → `testCRAM31SamtoolsFidelity`
- Extend the test to also write CRAM 3.1 from htsjdk (via `SAMFileWriterFactory.makeWriter`) and perform a 3-way comparison: original 3.0 vs samtools-generated 3.1 vs htsjdk-generated 3.1
- Add public static helper `doHTSJDKWriteCRAM(inputPath, outputPath, referencePath)` for reuse
- Make `getCRAMVersion()` public for reuse in other test classes
- Assert that htsjdk-written files have version 3.1
- Assert that samtools-written files have version 3.1

8b: Add HtsCRAMCodec31Test.testRoundTripCRAM31() (HTS plugin framework write test)

File: `src/test/java/htsjdk/beta/codecs/reads/cram/HtsCRAMCodec31Test.java`

- Add a new test that uses the HTS plugin framework (not the low-level API) to write CRAM 3.1
- Uses `HtsDefaultRegistry.getReadsResolver().getReadsEncoder()` to get a 3.1 encoder
- Round-trips: read source CRAM 3.0 → write CRAM 3.1 → read back CRAM 3.1 → compare all records
- Must set `.setPreSorted(true)` on `ReadsEncoderOptions` (see Task 9)
- Verify the output file is actually version 3.1

8c: Update CRAMVersionTest.java

File: `src/test/java/htsjdk/samtools/cram/CRAMVersionTest.java`

Update `test_V3()` → `test_V3_1()` to verify that the default `CRAMFileWriter` now produces 3.1 files (change expected version from `CRAM_v3` to `CRAM_v3_1`).

8d: Fix SliceBlockWriteStreamTest.java for NameTokeniser

File: `src/test/java/htsjdk/samtools/cram/structure/SliceBlockWriteStreamTest.java`

The Name Tokeniser requires input data to be in NAME_SEPARATOR-delimited format. The existing test writes raw data series names as content for all data series. For `RN_ReadName`, append the name separator character so the Name Tokeniser can parse it correctly.

Task 9: Known Issue — preSorted Default

File: Addressed in `HtsCRAMCodec31Test` test code (not a source fix)

The HTS plugin framework's `ReadsEncoderOptions` defaults `preSorted` to `false`, which causes CRAM writers to sort records before writing. This interacts poorly with CRAM in general (not just 3.1). The branch workaround is to explicitly set `.setPreSorted(true)` in the test. This is a pre-existing issue, not introduced by Phase 1.

What Phase 1 Does NOT Change

- **FQZCompEncode.java** — remains a stub that throws. Quality scores use RANSNx16 order-1.
- **RANSNx16 Stripe encoding** — still throws on stripe flag. Not used by the normal profile.
- **Range Stripe encoding** — still throws on stripe flag. Range coding is not used in the normal profile.
- **CRAMCodecModelContext** — remains empty. Not needed until FQZComp encoding.
- **CRAMEncodingStrategy** — no profile concept added. The single default encoding map handles 3.1.
- **NameTokenisationEncode** — the "naive" implementation is unchanged (no DUP_PREVIOUS_STREAM optimization, no all-MATCH stream elision). These are compression ratio improvements, not correctness issues.

Verification Checklist

1. htsjdk writes CRAM 3.1 files (version bytes = `\3\1` in header)
2. htsjdk-written CRAM 3.1 files are readable by htsjdk itself (round-trip)
3. htsjdk-written CRAM 3.1 files are readable by samtools/htslib (interop)
4. samtools-written CRAM 3.1 files remain readable by htsjdk (existing tests)
5. Record fidelity: all SAMRecord fields match between 3.0 input and 3.1 round-trip output
6. Default version is 3.1 (CRAMFileWriter, SAMFileWriterFactory, HTS plugin framework all produce 3.1)
7. Explicit 3.0 version requests still produce valid 3.0 files
8. Existing CRAM 3.0 tests continue to pass (no regression)

Implementation Approach

The recommended approach is to **merge or cherry-pick from the existing `cn_cram_3_1_write` branch**, since the work is already implemented and tested there. The branch is 4 commits ahead of master with 164

insertions / 64 deletions across 11 files. A squash merge would be cleanest for history.

If a clean re-implementation is preferred, the changes are small enough to apply manually following this plan (the branch diff is the reference implementation).