A Common Domain Model for QDA Artifacts

BACHELOR THESIS

Benedikt Zinn

Submitted on 2 September 2026



Friedrich-Alexander-Universität Erlangen-Nürnberg
Faculty of Engineering, Department Computer Science
Professorship for Open Source Software

Supervisor:
Prof. Dr. Dirk Riehle, M.B.A.



Declaration of Originality

I confirm that the submitted thesis is original work and was written by me without further assistance. Appropriate credit has been given where reference has been made to the work of others. The thesis was not examined before, nor has it been published. The submitted electronic version of the thesis matches the printed version.

Erlangen, 2 September 2026

License

This work is licensed under the Creative Commons Attribution 4.0 International license (CC BY 4.0), see https://creativecommons.org/licenses/by/4.0/

Erlangen, 2 September 2026

Statement on the Use of AI Tools

ChatGPT (GPT-5.6 luna; OpenAI) was used to assist with the preparation of this thesis. It supported planning, file analysis, code generation and review, and language, grammar, and style assistance. AI-generated suggestions were never accepted without review. I am responsible for the technical decisions, implementation, source selection and verification, and the final version of this thesis.

Abstract

The thesis examines how project data from different Qualitative Data Analysis (QDA) tools can be exchanged through a common domain model. It asks which entities and relations recur across the analysed formats, which of them should be represented in the model, and where conversions remain lossy. The method combines project-file and documentation analysis with reverse engineering, format-specific mapping, and automated round-trip tests.

The result is the `Corpus` model and the TypeScript library `qdconvert`, which provides paths for reading and writing REFI-QDA (`.qde` and `.qdpx`), MaxQDA (`.mx20`, `.mx22`, and `.mx24`), ATLAS.ti (`.atlasproj`), and NVivo (`.nvp`) artifacts (QDArchive, 2026). The evaluation indicates that a recurring core of entities and relations can be represented in a common model for the tested properties, while target formats differ in their semantic coverage. Structured diagnostics and optional strict export failure make these differences visible.

Contents

1 Introduction **1**
- 1.1 Scope of QDA artifacts in this thesis . . . 1
- 1.2 Tools and formats . . . 2
- 1.3 Methodology . . . 3
- 1.4 Thesis structure . . . 4

2 Legality & data sources **5**

3 Related work **7**
- 3.1 REFI-QDA . . . 7
- 3.2 QuDex . . . 7
- 3.3 Vendor-provided specifications . . . 8

4 Implementation **9**
- 4.1 Implementation scope . . . 9
- 4.2 Conversion architecture . . . 9
  - 4.2.1 Format abstraction and registry . . . 10
  - 4.2.2 Import and export pipeline . . . 10
- 4.3 Shared corpus model . . . 11
  - 4.3.1 Project metadata and users . . . 11
  - 4.3.2 Code systems and graph structures . . . 12
  - 4.3.3 Sources, regions, and codings . . . 12
  - 4.3.4 Cases, variables, notes, and sets . . . 12
  - 4.3.5 Visualizations . . . 13
  - 4.3.6 Corpus serialization . . . 13
- 4.4 Format adapters . . . 13
  - 4.4.1 REFI-QDA . . . 13
  - 4.4.2 MaxQDA . . . 14
  - 4.4.3 ATLAS.ti . . . 14
  - 4.4.4 NVivo . . . 15
- 4.5 Identity and loss handling . . . 15
  - 4.5.1 Identity and reference resolution . . . 15

4.5.2 Capabilities and diagnostics . . . 15
4.5.3 Determinism and strictness . . . 16
4.6 Implementation boundary . . . 16

5 Analysis of Format Structures 17
5.1 MaxQDA (`.mx20`, `.mx22`, and `.mx24`) . . . 17
5.1.1 Storage and observed structures . . . 17
5.1.2 Native references . . . 18
5.2 REFI-QDA (`.qdpx` and `.qde`) . . . 18
5.2.1 Project structure . . . 18
5.2.2 Native references and source paths . . . 18
5.3 ATLAS.ti (`.atlasproj`) . . . 19
5.3.1 Native structures . . . 19
5.3.2 Reference chains . . . 19
5.4 NVivo (`.nvp` and `.nvpx`) . . . 19
5.4.1 Native structures . . . 20
5.4.2 Native references . . . 20
5.5 Cross-format observations . . . 20

6 Evaluation 21
6.1 Evaluation protocol . . . 21
6.2 Test structure and measured results . . . 22
6.3 Roundtrip and format tests . . . 22
6.3.1 Public in-out tests . . . 22
6.3.2 REFI-QDA roundtrip . . . 23
6.3.3 MaxQDA mapping . . . 23
6.3.4 ATLAS.ti mapping . . . 23
6.3.5 NVivo mapping . . . 24
6.4 Capability coverage . . . 24
6.5 Losses, diagnostics, and limitations . . . 25
6.5.1 Tested loss cases . . . 25
6.5.2 Interpretation limits . . . 25
6.6 Answer to the research questions . . . 26

7 Conclusion and Future Work 27
7.1 Conclusion . . . 27
7.1.1 Shared entities and relations . . . 27
7.1.2 Entities and relations in the shared model . . . 27
7.1.3 Lossy conversions and their mitigation . . . 28
7.1.4 Overall conclusion . . . 28
7.2 Future Work . . . 28
7.2.1 Extending NVivo coverage . . . 29
7.2.2 Broader format preservation . . . 29

7.2.3 Stronger evaluation and regression protection 29
7.2.4 Continuous integration 29

Appendices **31**
A Datasets 33

References **39**

List of Figures

4.1 Read-write architecture of `qdconvert`. 11

List of Tables

1.1 Analysed project file formats in this thesis 3

4.1 Core corpus types in the current implementation 11

6.1 Automated test results by implementation area 22
6.2 Roundtrip and integration evidence 24
6.3 Current test-backed capability status 25

1 Zenodo sources used in this thesis 33

Acronyms

API Application Programming Interface

CCBY40 Creative Commons Attribution 4.0 International License

CLI Command Line Interface

QDA Qualitative Data Analysis

QuDex Qualitative Data Exchange

REFI Rotterdam Exchange Format Initiative

HU Hermeneutic Unit

SQL Structured Query Language

XML Extensible Markup Language

XSD XML Schema Definition

1 Introduction

QDA tools are applications used in qualitative research to support the analysis of material such as interviews, focus groups, and textual documents (Corti & Gregory, 2011). They help organise, analyse, and identify patterns in data. Examples of such tools include ATLAS.ti and MAXQDA.

QDA tools commonly use tool-specific project formats (see Table 1.1), which complicates interoperability and can increase switching costs for users (Corti & Gregory, 2011; Farrell & Klemperer, 2007).

Open exchange formats have also been developed, including the Rotterdam Exchange Format Initiative (REFI)-QDA standard (Rotterdam Exchange Format Initiative, 2019). These formats are intended for exchange rather than for use as the primary project format of every tool.

The study is guided by the following questions:

1. Which entities and relations are shared across different QDA file formats?
2. Which of these entities and relations should be represented in the shared internal model to enable conversion through that representation?
3. Which conversions remain lossy in the end, and how can these losses be minimized in the future?

To answer them, the study analyses selected QDA file formats, reconstructs their structure and content, and maps them to a common domain model. The shared model is referred to as `Corpus` below.

1.1 Scope of QDA artifacts in this thesis

Here, a QDA artifact denotes a file that stores analysis data and may contain source files to which that data relates. In the scope of this work, the central entities are sources, codes, codings, notes, cases, variables, sets, and project metadata. These entities appear in different forms across the analysed tools and therefore provide the basis for a shared internal model.

The concepts these terms represent will be used as follows:

- A source is material that is analysed.
- A code is a named label used to classify material.
- A coding links a code to a region of a source.
- A note stores a free-text annotation.
- A case and its variables store structured contextual information.
- A set groups entities without changing their original identity.

Together, these definitions delimit the subset of entities used in this thesis. Different tools may use different terms for similar concepts; the definitions above provide consistent terminology for the comparison.

The analysed formats may also contain visualization structures such as network views. These are treated as secondary in the current implementation because their structures and semantics are more tool-specific (QDArchive, 2026).

In the scope of this work, a mapping is a transformation between two representations of data. The transformation changes the structure of the data and is intended to preserve its meaning. A mapping can be lossy, meaning that some information is not preserved during the transformation. A mapping can also introduce information that was not present in the original data, for example when a target format requires a value in its metadata that other source formats do not provide.

1.2 Tools and formats

MaxQDA, NVivo, and ATLAS.ti refer to software tools and their vendors. REFI-QDA refers to an exchange standard. File extensions such as `.mx20`, `.atlasproj`, `.qdpx`, and `.qde` refer to concrete project file formats. Terms such as SQLite, ZIP, and Extensible Markup Language (XML) refer to storage technologies used by these formats (SQLite Consortium, n.d.).

The registered format paths in `qdconvert` cover import and export for REFI-QDA, MaxQDA, ATLAS.ti and NVivo projects. The supported format identifiers are `qdpx`, `qde`, `mx20`, `mx22`, `mx24`, `atlproj`, and `nvp`. The other formats (`.nvpx` and `.hpr7`) in Table 1.1 are included as analysed, but are not registered conversion targets. The dataset inventory also contains `.qdp` and `.rqda`; their presence there does not imply that they were analysed or supported by `qdconvert` (QDArchive, 2026).

Table 1.1: Analysed project file formats in this thesis

Software or standard	Project file format(s)	Storage technology
MaxQDA	`.mx20`, `.mx22`, `.mx24`	SQLite database
NVivo	`.nvp`, `.nvpx`	Platform-specific database formats
ATLAS.ti	`.atlasproj`, `.hpr7`	ZIP archive with XML project file / Structured Query Language (SQL) database
REFI-QDA	`.qdpx`, `.qde`	ZIP package with XML project file / XML project file (Rotterdam Exchange Format Initiative, 2019)

The detailed external structures and their native reference mechanisms are discussed in Chapter 5. The rationale and design of the shared internal model are discussed in Chapter 4.

1.3 Methodology

Reverse engineering is the process of analysing a system to identify its components and their relationships, and to create representations of the system in another form or at a higher level of abstraction (Washizaki et al., 2018).

In this thesis, the analysed systems are QDA project files. First, the storage technology of a project file is identified. In practice, this can often be identified by inspecting the magic number, an invariant identifier at a fixed position near the beginning of a file (Darwin, 2025).

The analysed formats use SQLite databases, ZIP archives, and XML documents, often stored under different file extensions (ATLAS.ti Scientific Software Development GmbH, n.d.-b; MAXQDA - VERBI Software, n.d.; Rotterdam Exchange Format Initiative, 2019; SQLite Consortium, n.d.). This distinction separates extraction from the reconstruction of semantic relations.

The internal structure is then inspected. Recurring entities and references are mapped to common concepts in `Corpus`. Finally, the deserialization, mapping, and serialization paths are validated through unit tests and roundtrip tests in `qdconvert`.

1.4 Thesis structure

The thesis is organised as follows.

Chapter 2 discusses the legal basis of the reverse engineering work and the data sources used in the thesis.

Chapter 3 reviews related work on QDA interoperability, especially exchange standards and shared schemas.

Chapter 4 describes the design and implementation of the `Corpus` model and the `qdconvert` library.

Chapter 5 analyses the selected file formats and identifies their storage structures and native reference mechanisms.

Chapter 6 evaluates the current implementation and its limitations.

Chapter 7 concludes the thesis and outlines future work.

2 Legality & data sources

The legality of reverse engineering and the use of data or software obtained through reverse engineering can vary depending on the jurisdiction and the circumstances of the activity.

Within the European Union, Directive 2009/24/EC (European Parliament and Council of the European Union, 2009) concerns the legal protection of computer programs.

Article 6 of the directive addresses decompilation for improved interoperability between software. It states that the rightholder's authorisation is not required where the reproduction of code and translation of its form are indispensable to obtain the information necessary to achieve the interoperability of an independently created computer program with other programs, provided that the conditions in Article 6(1)(a)–(c) are satisfied (European Parliament and Council of the European Union, 2009).

Interoperability between QDA tools is the research goal of this thesis. The directive is included as legal context for that goal; it is not a determination that every reverse-engineering activity or every use of the resulting information is permitted.

The files analysed for this thesis and the `qdconvert` library were retrieved from Zenodo only. The records were last checked on 2 September 2026 and the files listed under the Creative Commons Attribution 4.0 International License (CC BY 4.0) (Creative Commons, 2013) at that time. This observation records the licence information displayed by the dataset records; it does not establish that every file or every possible use is covered by that licence. The CC BY 4.0 legal code also states that a licensor must be authorized to grant the relevant rights (Creative Commons, 2013). Appendix A lists all datasets used in this thesis, including their Zenodo records.

3 Related work

Interoperability and data exchange between software tools is a common challenge in many domains. QDA software is no exception, and there have been various efforts to address this issue. Related work in this area can be grouped into exchange standards, shared schemas, and vendor-provided format descriptions. They are relevant here because they address the same general problem of transferring QDA artifacts between tools.

3.1 REFI-QDA

Among these efforts, REFI-QDA is the closest match to the scope of this project (Rotterdam Exchange Format Initiative, 2019). It defines a structured exchange representation for QDA projects (Rotterdam Exchange Format Initiative, 2019), and has been adopted by several tools.

It is particularly useful as a reference point because it demonstrates that a shared representation of common QDA concepts is feasible in practice. The present work does not, however, aim to introduce another common format for direct tool adoption. Instead, it develops an internal unification layer to which existing formats can be mapped. This approach reduces dependence on vendor adoption of a single interchange standard and supports incremental extension of `qdconvert` with additional formats.

3.2 QuDex

The available Qualitative Data Exchange (QuDex) documentation describes a shared schema for QDA data (of Essex, n.d.). The present work does not assess its current adoption or implementation in commercial tools. QuDex is therefore treated as a related schema proposal rather than as an evaluated conversion target.

3.3 Vendor-provided specifications

Apart from shared interchange efforts, vendor-provided file format specifications can also support interoperability work. When such documentation is available, it can clarify the documented structure of a format. Its coverage must nevertheless be checked against the artifacts used in the analysis. ATLAS.ti, for example, provides an XML Schema Definition (XSD) for a subset of its Hermeneutic Unit (HU) XML format (ATLAS.ti Scientific Software Development GmbH, n.d.-a) from 2003. Because the current formats have changed substantially, this schema is not treated as documentation of the current format.

4 Implementation

The shared internal representation and reading-writing pipeline are implemented in the TypeScript library `qdconvert`. This chapter focuses on the design decisions relevant to conversion between the analysed QDA artifact formats. Detailed descriptions of the individual storage structures appear in Chapter 5, while test results are discussed in Chapter 6. The implementation statements in this chapter refer to the source revision recorded in the bibliography (QDArchive, 2026).

4.1 Implementation scope

`qdconvert` exposes an Application Programming Interface (API). It also provides a Command Line Interface (CLI). The library reads an input artifact into the internal `CorpusFormat` representation and writes a corpus to a selected external format. The command line interface uses the same conversion functions for direct file-to-file conversion. The registry currently contains the following format identifiers: `qde` and `qdpx` for REFI-QDA, `mx20`, `mx22`, and `mx24` for MaxQDA, `atlproj` for ATLAS.ti, and `nvp` for NVivo.

An import and export wrapper alone does not imply identical semantic coverage. Each adapter supports only the subset that can be mapped to the target representation. The capability declarations distinguish supported, adapted, limited, and unsupported concepts. Consequently, the implementation aims to preserve the common analytical content while making format-specific limitations explicit rather than treating all successful file writes as loss-free conversions.

4.2 Conversion architecture

The conversion architecture uses the internal representation as a pivot. A format adapter first deserializes the external container into an intermediate representation and then projects it into `CorpusFormat`. For export, the adapter receives a corpus, projects it into its external representation, and serializes the result. An adapter therefore does not contain a separate implementation for every possible

source format. Adding a new format requires its own deserializer, serializer, and projections to and from the shared model (QDArchive, 2026).

4.2.1 Format abstraction and registry

The abstract class `BaseFormat` defines the common adapter boundary. The `Serializable` contract provides serialization operations, and `Morphable` provides projections to and from the corpus. Concrete adapters implement these operations for their external representations. The `formatRegistry` associates each public format identifier with an adapter class. Public conversion functions can therefore select an adapter without depending on the internal data classes of a format family.

`ExportOptions` controls the target format, whether source payloads should be included, an optional platform setting, and strictness. The platform setting is part of the public interface, but no format is currently listed in `macFormats`. The public functions and the command-line interface are defined in `src/index.ts` and `src/cli.ts`, respectively (QDArchive, 2026).

4.2.2 Import and export pipeline

An import performed by `parseToCorpus()` consists of the following steps:

1. The caller supplies a filename, raw data, and a registered format identifier.
2. The adapter deserializes the raw data into its format-specific intermediate representation.
3. The adapter projects that representation into a `CorpusFormat`.
4. Registered post-processing steps are applied.
5. The result contains the corpus, diagnostics, and a conversion report.

The intermediate representation separates container parsing from semantic mapping. For example, the REFI-QDA adapter parses XML into project objects, whereas the mapper modules resolve the entities and references represented by those objects. The MaxQDA adapter first reads SQLite rows and then reconstructs the relations between them.

Export through `corpusToFile()` starts with a preflight check. The check compares the corpus with the declared capabilities of the target and reports known narrowing conversions before serialization. The adapter then projects the corpus, serializes the target representation, and returns the generated file together with diagnostics and entity counts. When `strict` is enabled, preflight or adapter diagnostics prevent a successful file result. In permissive mode, the adapter writes

the representable subset and returns warnings for information that could not be represented (QDArchive, 2026).

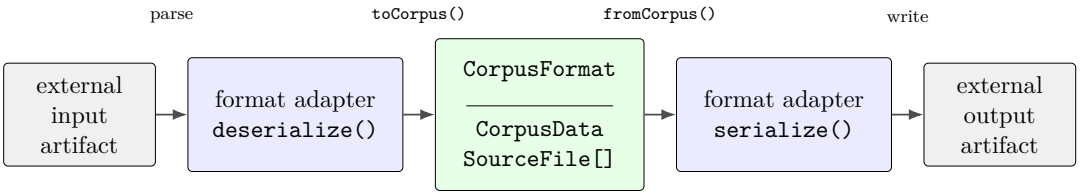


Figure 4.1: Read-write architecture of `qdconvert`.

4.3 Shared corpus model

The internal model has two layers: `CorpusData` contains the semantic entities of a project, while `CorpusFormat` wraps this data and stores the referenced source-file payloads. This distinction allows an adapter to preserve a source reference even when the corresponding bytes are not available or cannot be embedded in the target artifact.

Table 4.1: Core corpus types in the current implementation

Type	Purpose
`ArtifactMetadata`	project-level metadata
`ArtifactUser`	user identity
`CodeSystem` / `CodeNode`	hierarchical code structure
`Coding`	link from code to source region
`SourceFile`	source file metadata and content
`Case`	grouped analytical unit with references
`Variable`	typed attribute value
`Note`	free-text annotation with targets
`Graph`	reusable graph base for hierarchies and visualizations

4.3.1 Project metadata and users

`ArtifactMetadata` stores the project name and description, a `Version`, creation and modification timestamps, authorship, source software information, and the original format. `ArtifactUser` contains a GUID, a username, and an email address. These objects are referenced by codes, codings, notes, and project metadata. An adapter can construct users from a dedicated native user structure or collect authorship values when the external format does not provide one.

4.3.2 Code systems and graph structures

`CodeSystem` is a graph whose nodes are `CodeNode` objects. Each code node has a GUID and may contain a name, description, colour, creator, and timestamps. A graph node stores predecessor and successor relations. The graph validator checks that a root exists, that all nodes are reachable from it, that reciprocal edges agree, and that no cycle is encountered during traversal.

The code system normally uses a synthetic root as a common traversal point. Top-level codes can therefore be handled uniformly even when an external format does not contain an explicit root. During export, a named root is treated as a real code, whereas an empty synthetic root is omitted from the target codebook. This is a normalization rule of the internal model rather than a claim that every external format contains the same root concept (QDArchive, 2026).

4.3.3 Sources, regions, and codings

`SourceFile` separates source metadata from its optional payload. It contains a filename, GUID, content type, inclusion flag, encoding, optional binary data, optional extracted text, and optional creation and modification information. The model accepts text, PDF, image, audio, video, table, and unknown source types. The capability declarations are narrower for some targets; in particular, MaxQDA is declared for text sources only, while unsupported or ambiguous source types may be flattened or omitted by other adapters.

`RefRegion` is the common base for coding locations. `TextRegion`, `AudioRegion`, and `VideoRegion` contain start and end positions. `ImageRegion` contains polygon points and uses neutral start and end values inherited from the base class.

`Coding` links a code reference to a source reference and contains one or more regions. It also stores authorship, timestamps, and optional description, memo, colour, and weight values. The use of GUID references instead of array positions or native database row numbers allows the individual adapters to use different native identity schemes while exposing one relation to the rest of the implementation.

4.3.4 Cases, variables, notes, and sets

`Case` stores a case identity and name, optional description, references to codes and sources, variable values, and optional note references. `Variable` stores a GUID, name, type, scope, value, and description. The model distinguishes string, number, boolean, date, and categorical values. The representation can therefore retain a typed value when the source adapter provides enough information to determine its type.

`Note` stores content, authorship, timestamps, a target type, and one or more target references. The target can be a project, case, code, source, or variable. Some targets support only one native note link, so the export layer reports a narrowing conversion when the corpus contains multiple targets.

`CorpusData.sets` is a map from a set name to GUID members. A set can contain members of different entity types in the internal model. Adapters with separate native set types may preserve the members by splitting such a set, while adapters without a compatible set representation report the set as unsupported.

4.3.5 Visualizations

`Visualization` reuses the graph abstraction for network-like structures. Its vertices contain an identifier for the represented corpus entity, a name, position, shape, and scale. Its edges contain source and target vertices, a name, and a direction flag. Visualizations are kept separate from the code system because a network view is not necessarily a code hierarchy. The capability declarations currently support visualization export for REFI-QDA, do not support it for MaxQDA or NVivo, and limit the representable endpoint combinations for ATLAS.ti (QDArchive, 2026).

4.3.6 Corpus serialization

`CorpusFormat` is also a `BaseFormat` implementation. Its JSON serializer writes both `artifactData` and the `sources` list. The serializer uses explicit type markers for GUIDs, timestamps, versions, regions, maps, and sets. The code-system object provides JSON data containing node identifiers and relation references; the deserializer reconstructs the code-system graph in a separate pass. This makes the corpus representation usable as an intermediate format as well as an in-memory object (QDArchive, 2026).

4.4 Format adapters

All adapters use the same corpus boundary, but differ in how they read native structures and reconstruct relations. The following subsections summarize the current implementation strategy. They do not attempt to document every table, XML element, or archive entry; those observations belong in Chapter 5.

4.4.1 REFI-QDA

`BaseREFIFormat` provides shared mapping logic. `QDEFormat` parses and builds the XML project document. `QDPXFormat` reuses this XML logic and adds ZIP handling for the project document and contained source files.

The REFI projector delegates the mapping to modules for metadata and users, the code system, sources, codings, cases and variables, notes, sets, and visualizations. A conversion context registers identities while the mapper modules resolve references. In the reverse direction, the adapter builds the project sections from the corpus and selects archive paths for source payloads. This decomposition keeps XML parsing, semantic projection, and archive handling separate (QDArchive, 2026).

4.4.2 MaxQDA

`Mx20Format`, `Mx22Format`, and `Mx24Format` read and write SQLite databases through version-specific data classes. Their semantic projection is shared through the MaxQDA 2020 projector. The projector maps codes, sources, codings, notes, sets, and cases and variables from the corresponding native rows. On export, it creates native rows from corpus entities and assigns relational identifiers for the target database.

The case and variable mapping is an adaptation rather than a direct copy of an explicit native case model. The importer interprets `Grid` values and derives case-variable information from document and code references. The exporter writes the resulting values through MaxQDA's corresponding tables. The capability declarations record the resulting limitations: native identifiers are not stable across exports, media coding regions are limited, visualizations are unsupported, and mixed-member sets may be split (QDArchive, 2026).

4.4.3 ATLAS.ti

`AtlasprojFormat` reads an `.atlasproj` ZIP archive, parses the `.aprx` XML project document, and extracts content and sub-content entries. The mapper reconstructs users, codes, sources, codings, notes, sets, and visualizations. A coding can require several linked native objects, including a tag, quotation, document, media object, and archive content entry. The mapper resolves this chain before creating a corpus coding.

The adapter also stores an ATLAS.ti preservation state with the imported corpus. This state retains native identifiers, source and memo content, coding structures, and archive blobs that can be reused when the corpus is exported back to ATLAS.ti. The approach preserves format-specific information without adding ATLAS.ti identifiers to the shared corpus schema. Cases and variables are outside the current capability declaration, and some visualization endpoints and set combinations remain limited (QDArchive, 2026).

4.4.4 NVivo

The `NvivoFormat` adapter handles Windows `.nvp` projects through a configured SQL Server instance running in Docker. Import copies the database into the configured container, attaches it temporarily, queries a snapshot of available project tables, and maps that snapshot into the corpus. The snapshot mapping covers project metadata, users, code hierarchies, sources and payloads, codings, cases, variables, and annotations.

Export is template-based. The adapter copies a compatible `.nvp` template, attaches it to SQL Server, generates SQL for representable corpus entities, executes the script, detaches the database, and returns the resulting file. The target does not currently provide a corpus-compatible representation for sets or visualizations. Categorical values may be flattened to text, and notes or coding metadata may be narrowed. The adapter therefore reports these cases as diagnostics. The tests cover snapshot mapping and SQL export-plan generation, but do not establish a complete end-to-end `.nvp` roundtrip. The macOS `.nvpx` variant is not registered (QDArchive, 2026).

4.5 Identity and loss handling

4.5.1 Identity and reference resolution

`GUID` validates the accepted identifier forms, generates random identifiers, and derives deterministic identifiers from a seed. The `ConversionContext` keeps mapping-time registries for corpus GUIDs and native identifiers. Native identifiers are namespaced by entity type, which allows different native tables to use the same local value. The context reports duplicate registrations, invalid identifiers, missing references, and unknown references as structured diagnostics.

The adapters use the identity mechanism according to their native formats. REFI-QDA identifiers can generally be retained. ATLAS.ti combines GUIDs, local identifiers, and archive locations and therefore requires several mapping passes. MaxQDA uses relational integer identifiers and creates corpus GUIDs while building maps from native rows. NVivo uses GUID-valued item identifiers in its snapshot and export plan. Preserving valid relations is more important than retaining an invalid native identifier string because codings, cases, notes, and sets depend on those relations (QDArchive, 2026).

4.5.2 Capabilities and diagnostics

The capability module describes target support for entities, source types, coding regions, note targets, and identifier stability. `preflightExport()` uses these

declarations together with the actual corpus to identify known losses before serialization. Examples include unsupported entities, unsupported source or region types, binary-only source content for MaxQDA, multiple note targets, missing coding references, and mixed-member sets.

Adapter-specific diagnostics cover losses discovered while reconstructing or writing native structures. A diagnostic contains a severity, message, target format, affected entity and field where available, a loss reason, optional remediation, and contextual data. `createConversionReport()` retains the diagnostics and counts corpus entities and loss reasons. This separates the fact that an output file was generated from the stronger claim that all source information was preserved.

4.5.3 Determinism and strictness

The shared ordering helper sorts collections by stable keys before serialization. Code nodes, codings, notes, and several format-specific rows are therefore emitted in defined orders rather than relying on insertion order. Deterministic ordering does not make every identifier stable: identifiers generated for newly imported or reconstructed objects can differ between independent conversions.

Strictness is applied at two boundaries. The preflight check can prevent an export before an adapter is invoked, and adapter diagnostics can prevent the result after projection and serialization have been attempted. In permissive mode, the target adapter may write a representable subset, while the result contains warnings and a report. This policy makes narrowing visible to callers without requiring every conversion to fail (QDArchive, 2026).

4.6 Implementation boundary

Overall, the implementation provides a shared representation for recurring QDA entities and relations and reading-writing paths for seven registered format identifiers. It does not provide a loss-free representation of every feature in those formats. The common model preserves information where the target has a corresponding structure; otherwise, adapters transform, separate, omit, or reject information according to their capabilities and diagnostic policy. The evaluation chapter examines the tested subset and should therefore be read together with these implementation boundaries.

5 Analysis of Format Structures

This chapter examines the external structures used by the reading-writing implementation. The discussion concentrates on storage containers, native structures, and reference mechanisms. It does not repeat the design of the shared `Corpus` model or evaluate conversion capabilities; those topics are covered in Chapters 4 and 6. Claims about the current parser refer to the qdconvert source revision (QDArchive, 2026).

5.1 MaxQDA (`.mx20`, `.mx22`, and `.mx24`)

MaxQDA project files for the 2020, 2022, and 2024 releases are SQLite databases (MAXQDA - VERBI Software, n.d.; SQLite Consortium, n.d.). The three registered adapters use version-specific data classes, while their semantic projection is shared through the MaxQDA 2020 mapping logic (QDArchive, 2026).

5.1.1 Storage and observed structures

The analysis concentrates on the SQLite tables that carry the structures required by the conversion path:

- `CodeWords` stores code definitions, display properties, authorship fields, and parent references.
- `Codings` stores code-to-source applications, segment coordinates, and coding metadata.
- `Texts` stores source metadata and text content.
- `Memos` stores note-like records that can be attached to codes or sources.
- `Grid` stores document-code value pairs.
- `Textsets` and `Textlinks` store source-set membership, while `Wordsets` and `Wordlinks` store code-set membership.

- `Textgroups` stores group information used when case-like structures are written.

Several additional tables are loaded, including `GeoLinks`, `Links`, `MediaMarks`, `MemoLinks`, `MemoSets`, `QntDicCats`, and `QntDicWords`. The importer also loads `Similarity` and `SimilarityRows`. Loading a table does not mean that all of its structures are interpreted by the current semantic mapping (QDArchive, 2026).

5.1.2 Native references

MaxQDA uses relational integer identifiers rather than one global GUID scheme. The `TID` field in `CodeWords` identifies a parent code. The `TAID` and `TextID` fields in `Codings` identify the code and source associated with a coding. Segment positions such as `SegPos1X` and `SegPos2X` describe the coded range in a text.

The `Grid` table links a document identifier and a code identifier to a stored value. This structure does not form a separate, explicit case model; it is a native relation between documents, codes, and values. The distinction is important for interpreting the resulting case and variable structures in the implementation (QDArchive, 2026).

5.2 REFI-QDA (`.qdpx` and `.qde`)

REFI-QDA is an exchange representation for processed data from QDA software (Rotterdam Exchange Format Initiative, 2019). The format has two relevant containers. A `.qde` file is an XML project document, while a `.qdpx` file is a ZIP package containing a project document and, when present, source files (Rotterdam Exchange Format Initiative, 2019).

5.2.1 Project structure

The project document contains sections for project metadata, users, the codebook, sources, variables, cases, sets, notes, and coding information (Rotterdam Exchange Format Initiative, 2019). In the current implementation, `QDEFormat` handles the XML document and `QDPXFormat` reuses that logic while adding archive handling. The separation means that project structure and source payloads are read through different container layers (QDArchive, 2026).

5.2.2 Native references and source paths

The format uses explicit GUID-based references for codes, sources, cases, variables, notes, and other cross-links (Rotterdam Exchange Format Initiative, 2019). A source entry can refer to a payload through an internal archive path such as

`internal://GUID.ext`. This path connects the logical source entry in the XML document with a file in the exchange package. The importer must therefore resolve both the XML reference and the archive entry when source payloads are present (QDArchive, 2026).

5.3 ATLAS.ti (`.atlasproj`)

For ATLAS.ti, the adapter treats `.atlasproj` as a ZIP archive containing an `.aprx` XML project document and content folders (ATLAS.ti Scientific Software Development GmbH, n.d.-a, n.d.-b). The project document describes logical objects, while the archive stores the document, memo, and other content blobs to which those objects refer.

5.3.1 Native structures

The structures relevant to the current analysis include users, tags, documents, media, content entries, quotations, memos, groups, links, relations, and network views. Tags provide code-like objects; documents and media provide source-related objects; quotations and tag-to-quotation links describe coded selections; memos and tag-to-memo links describe memo associations; groups describe collections; and network views contain visualization structures (QDArchive, 2026).

5.3.2 Reference chains

ATLAS.ti combines GUIDs, local object identifiers, and archive content locations. A source payload can be reached through a chain from a document to a media object, then to a content record and an archive blob. A coding can similarly require a tag, quotation, document, and content object before its source and location are known. The parser reconstructs these relations rather than reading a coding from one isolated record (QDArchive, 2026).

5.4 NVivo (`.nvp` and `.nvpx`)

The NVivo files examined here represent platform-specific database variants. The Windows `.nvp` format is handled by the current adapter through SQL Server, while the macOS `.nvpx` format is not registered as a conversion target. Access to an `.nvp` file uses a configured Docker/SQL Server environment rather than a direct SQLite-style reader (QDArchive, 2026).

5.4.1 Native structures

For `.nvp`, the snapshot reader checks for project and user records and, when available, reads `Item`, `Role`, `Source`, and `NodeReference` records. It also reads `Annotation`, `ExtendedItem`, `CompoundSourceRegion`, and `BlobStorage` records. Items provide the native objects used for codes, sources, cases, variables, and variable values. Sources and blob records provide source content and coding-related data (QDArchive, 2026).

5.4.2 Native references

Roles connect items into hierarchies, order collections, associate cases with variable values, and identify membership relations. Parent roles connect code items, node-reference records connect code items to source items and coordinates, and item-value roles connect cases to variable values. These relations form the native reference graph that the adapter converts into the explicit references of the internal model (QDArchive, 2026).

5.5 Cross-format observations

Across the analysed artifacts, storage technology alone does not determine the semantic mapping. SQLite tables, XML elements, and archive entries can all represent codes, sources, or codings, while their identifiers and relation structures differ.

Their relations are also expressed at different levels of indirection. REFI-QDA uses explicit GUID links, MaxQDA relies on relational keys, and ATLAS.ti and NVivo reconstruct meaning from chains of linked native objects. These differences explain why the implementation separates external deserialization from projection into the shared `Corpus` model (QDArchive, 2026).

6 Evaluation

The evaluation examines the current state of `qdconvert`. The evaluation is based on the automated tests in the repository and the capability declarations used by the conversion layer (QDArchive, 2026). It assesses whether the implemented mappings preserve selected project properties and whether known limitations are reported. It does not claim complete interoperability or loss-free conversion for all entities and relations in every supported format. The implementation revision used for these claims is qdconvert `v5.0.0`.

6.1 Evaluation protocol

Three complementary checks structure the evaluation. First, unit and integration tests check individual mappers, graph operations, serialization, diagnostics, and public API behaviour. Second, in-out tests read project files and write them to target formats. Third, roundtrip tests compare selected properties after an import and export cycle.

All comparisons are semantic rather than byte-based. The tests compare properties such as entity names and counts, identifiers and references, source content, code hierarchy, region coordinates, set membership, and diagnostic messages. Byte-for-byte equality is not a goal since serializers may change container layout, ordering, or format-specific identifiers. An entity is considered *preserved* when the tested semantic property is equal after conversion. It is *transformed* when it is represented in a different structure but the tested information remains available. A property is marked as *warned* when the implementation reports that a target is narrower or that input data is incomplete. It is *lost* when the target cannot represent the information and no equivalent output is produced.

The test results reported below were obtained with `npm test` on 2 September 2026. The TypeScript implementation was compiled with `npm run build` before the test run.

6.2 Test structure and measured results

The test tree is split by implementation concern. `tests/formats/` contains format and corpus-model tests, `tests/conversion/` covers capabilities and conversion context, `tests/graph/` covers graph operations, `tests/utils/` covers shared helpers, and the top-level tests cover the public API and in-out behaviour. Table 6.1 reports the executed test suites by area.

Table 6.1: Automated test results by implementation area

Area	Suites	Test focus
REFI mapping	19	mapping
ATLAS.ti mapping	7	mapping
MaxQDA mapping	4	mapping
NVivo mapping	1	mapping
QDE parsing	4	XML parsing & mapping
Corpus model	3	graphs, JSON roundtrips, and GUIDs
Conversion API	2	capabilities, preflight checks, and context
Graph utilities	2	graph traversal and validation
Public API & in-out	3	public entry points and format smoke tests
Utilities	6	file, XML, type, and inspector helpers
Total	**51**	all tests passed

In the reported run, 382 tests were executed, and all passed.

The large REFI-QDA test group reflects the role of that format in developing the shared model; it is not interpreted as a measure of relative format complexity.

6.3 Roundtrip and format tests

6.3.1 Public in-out tests

`tests/in-out.test.ts` contains six public API tests. They import a QDE project, a QDPX project, and a MaxQDA project, then export the resulting corpus to QDE, QDPX, MX20, MX22, and MX24. The tests assert successful results and, for one QDPX case, the presence of the project entry in the generated archive. Together, these tests show that the main read-write paths can be invoked together, but they do not by themselves prove deep semantic preservation.

6.3.2 REFI-QDA roundtrip

A dedicated test in `tests/formats/refi/qdpx.roundtrip.test.ts` performs a `QDPX -> Corpus -> QDPX` cycle. For a project fixture, it compares project metadata, optional origin, users, source names and content flags, variable names and types, case names and reference counts, set names and member counts, note targets, and coding references and region counts. The test also checks that the number of top-level codes is retained.

The comparison uses a stable subset of properties. It does not require deep equality of every nested codebook field, and its source comments identify full deep equality as unfinished. This makes the test useful for detecting regressions in the common core, but it limits the strength of the claim: passing the test does not establish loss-free preservation of every REFI-QDA field.

6.3.3 MaxQDA mapping

The MaxQDA projector tests construct relational fixtures and verify both directions of the mapping. They check user collection, code hierarchy and colours, text source references, character ranges, coding descriptions and weights, source notes, sets, and the adapted case-variable mapping. They also check that multiple coding regions become separate database rows, that unreferenced sources are exported, and that the generated result is a readable SQLite database.

The roundtrip assertion uses the semantic core rather than native row identifiers. This is appropriate because the adapter reconstructs native integer identifiers when it writes a new database. The tests also verify diagnostics for unknown references, inferred variable types, unsupported media marks, and strict-mode failures.

6.3.4 ATLAS.ti mapping

The real-world ATLAS.ti tests import the `DCR Networks Evaluation.atlproj` fixture. They verify one source with text content, 69 codings, and two visualizations. The corresponding export test checks the reconstructed archive for one layer, 34 quotations, 69 tag-to-quotation links, four word lists, and 613 word-list entries. Additional fixture tests verify code and document groups, memo links, embedded source blobs, a corpus-to-ATLAS.ti-to-corpus cycle, and a corpus-to-ATLAS.ti-to-MX20 conversion.

The tests also exercise the loss policy for a visualization edge connecting a code and a source. The adapter emits a diagnostic, and strict export returns an unsuccessful result instead of writing an output that silently omits the edge. The archive-preservation test compares the complete list of non-directory files before and after the ATLAS.ti roundtrip.

6.3.5 NVivo mapping

The NVivo tests cover a broader mapper snapshot than the current file-level tests. They verify deterministic project metadata and user mapping, deterministic GUID derivation for a non-GUID user identifier, endpoint configuration, failure when the SQL Server endpoint is missing, the requirement for corpus data before export, and native SQL export-plan generation. A larger snapshot test maps native codes and parent-child relations, a text source and binary payload, text codings, cases, variables, and source annotations. An image-coding test checks conversion of native coordinates to a canonical rectangle.

These tests demonstrate mapping behaviour for the supplied snapshots. They do not establish end-to-end import or export of every NVivo version. The adapter depends on a configured SQL Server/Docker environment for reading an actual `.nvp` file and on a compatible template for export.

Table 6.2: Roundtrip and integration evidence

Path	Test evidence	Result
Public in-out	6 tests for QDE, QDPX, MX20, MX22, and MX24	All conversions succeeded; one QDPX archive was also inspected
REFI-QDA	2 QDPX roundtrip tests	Selected metadata, users, codes, sources, cases, variables, sets, notes, and codings were preserved
MaxQDA	20 projector tests	Core entities, deterministic rows, SQLite output, and a corpus roundtrip were checked
ATLAS.ti	7 mapping test suites	Archive contents, quotations, codings, visualizations, and embedded files were checked
NVivo	8 snapshot and adapter tests	Import mapping and SQL export-plan generation were checked; no end-to-end file roundtrip exists

6.4 Capability coverage

The capability matrix distinguishes representational coverage from the existence of a format wrapper. Table 6.3 summarizes the statuses declared by the capability modules in `src/conversion/`. These entries describe the tested implementation

scope, not the complete feature set of the corresponding commercial tool or standard.

Table 6.3: Current test-backed capability status

Target	I/O	IDs	Media	Cases / vars.	Graphs	Sets
QDE/QDPX	both	supported	supported	supported	supported	supported
MX20, MX22, MX24	both	limited	limited	adapted	unsupported	limited
ATLAS.ti	both	supported	supported	unsupported	limited	limited
NVivo NVP	both	supported	supported	supported	unsupported	unsupported

The matrix shows why a single "supported" label would be misleading. For example, MaxQDA has import and export paths, but its native numeric identifiers, media types, case references, and visualizations do not all have direct corpus equivalents. NVivo now maps several entity types on import and has a template-backed export path, but it does not represent sets or visualizations in the current adapter.

6.5 Losses, diagnostics, and limitations

6.5.1 Tested loss cases

The capability tests deliberately construct corpora that violate target capabilities. They verify diagnostics for unsupported source and region types, note targets and target multiplicity, binary-only source content, missing or unknown coding references, mixed-member sets, and cases with multiple code or source references. For MaxQDA and ATLAS.ti, mixed-member sets produce a warning that the target will split the members into separate native sets. For MaxQDA, a case with multiple references produces one diagnostic for each narrowed reference.

Strict export turns these representational warnings into errors before a successful output result is returned. This is important for the evaluation because a successful non-strict export does not mean that all source information was preserved. It means that the adapter wrote the representable subset and returned the diagnostics needed to inspect the loss.

6.5.2 Interpretation limits

Four limits are important when interpreting these results. First, the test suite is strongest for REFI-QDA and contains fewer deep assertions for the other for-

mats. Second, the public in-out tests mostly check successful execution, whereas the dedicated mapper tests check selected semantics. Third, NVivo end-to-end tests depend on external SQL Server, Docker, and template configuration, so its snapshot and SQL-plan tests should not be interpreted as complete file-format coverage. Fourth, the test suite reports no single numeric measure of semantic preservation across all entities. The results therefore support claims about the tested properties and diagnostic policies, not about complete or loss-free interoperability.

6.6 Answer to the research questions

The first research question asks which entities and relations are shared. The model and the tests cover a common core of metadata, users, codes, sources, codings, notes, cases, variables, and sets. Visualizations are also represented, but their coverage is target-dependent. The tests therefore support a shared core while also showing that the overlap is not uniform across formats.

The second question asks which shared entities and relations should be represented in the internal model. The shared model includes project metadata, users, codes, sources, codings, notes, cases, variables, and sets. It also represents visualizations and source payloads where the shared model can retain them. Explicit references connect codes, sources, regions, and related entities, while graph structures represent code hierarchies and visualization relations. The tests verify selected names, references, content, regions, hierarchy relations, set membership, and roundtrip properties. These results support the chosen model as a common conversion core, while format-specific structures remain outside it or are reported as limitations (QDArchive, 2026).

The third question asks which conversions remain lossy and how to handle the losses. The capability and strict-mode tests show that unsupported entities, narrowed references, and incompatible regions can be identified before or during export. These cases are handled through structured diagnostics and optional strict failure. This makes the remaining losses visible and testable, but does not remove the underlying representational differences.

7 Conclusion and Future Work

7.1 Conclusion

The thesis examined how project data from different QDA tools can be represented in a shared internal model and converted between file formats. The result is the `Corpus` model and its reading-writing implementation in the TypeScript library `qdconvert`. Together, the model and library form a common conversion core for the selected formats, while the evaluation shows that their semantic coverage is not uniform (QDArchive, 2026).

7.1.1 Shared entities and relations

The first research question asked which entities and relations are shared across the analysed formats. The format analysis identified recurring structures for project metadata, users, code systems, source files, codings, notes, cases, variables, and sets. These structures do not have identical representations in every format, but they occur often enough to form a useful common basis. Visualizations are also represented by the internal model, although their support depends on the target format (see Chapter 5).

The adapters and tests support this conclusion. The adapters map the recurring entities into the corpus model, and the tests check selected names, references, content, regions, hierarchy relations, and roundtrip properties. These results support a shared core, but they do not establish that every field or relation of every analysed format is preserved (see Chapter 6).

7.1.2 Entities and relations in the shared model

The second research question asked which shared entities and relations should be represented in the internal model. The selection in `CorpusData` contains project metadata, users, a code system, codings, cases, variables, sets, visualizations, and notes. `CorpusFormat` additionally stores source-file metadata and optional source payloads. Codings use explicit references to codes and sources, and the

graph structures represent code hierarchies and visualization relations.

These concepts were selected because they recur across the analysed formats and because their references are needed to preserve analytical meaning during conversion. Media-specific regions, typed variable values, note targets, and source payload state are included where they provide information that would otherwise be lost during a conversion. The model does not attempt to generalize every vendor-specific table, archive identifier, or visualization semantic. Such information remains in an adapter-specific representation or is handled as a reported limitation (see Sections 4.2 and 4.3).

7.1.3 Lossy conversions and their mitigation

The third research question asked which conversions remain lossy and how these losses can be minimized. The evaluation identifies several recurring causes: unsupported entities, incompatible media regions, narrower note targets, multiple references where a target accepts only one, inferred or flattened values, and native identifiers that cannot remain stable across formats. NVivo support also depends on an external SQL Server/Docker environment and a compatible template (see Chapter 6).

`qdconvert` addresses these differences through a layered loss policy. The corpus retains the richer representation where possible. Capability preflight checks identify known target limitations before export, adapters produce structured diagnostics for losses discovered during mapping, and strict mode can reject an export with diagnostics instead of returning a partial file. These mechanisms make representational loss visible and controllable; they do not make the conversions loss-free (QDArchive, 2026).

7.1.4 Overall conclusion

Taken together, these results provide a tested shared read-write core for the selected QDA artifact formats and entities. Its main contribution is not a claim of complete interoperability, but a model and conversion architecture that make common relations explicit and target-specific limitations inspectable. The evidence supports this claim for the properties covered by the implementation and tests; broader format coverage and stronger semantic comparisons remain future work.

7.2 Future Work

The priorities below follow from the limitations identified during the implementation and evaluation.

7.2.1 Extending NVivo coverage

NVivo support remains limited to import and template-based export for Windows `.nvp` projects. Validating additional `.nvp` versions and establishing complete file-level roundtrips, rather than relying only on snapshot and SQL export-plan tests, is necessary to assess the implementation (QDArchive, 2026).

Support for macOS `.nvpx` files should begin with an analysis of their database backend, schema, and native reference system. Parts of the existing snapshot-to-corpus mapping for `.nvp` may be reusable for this task, but the database access layer would need to be selected and tested separately.

7.2.2 Broader format preservation

ATLAS.ti support can be extended with cases and variables, additional memo attributes, and visualization endpoint combinations that are currently handled only partially. The analysed `.hpr7` variant should be investigated separately because its storage and reference constraints may differ from those of `.atlasproj`.

7.2.3 Stronger evaluation and regression protection

A stronger evaluation would use a stable semantic comparison procedure that reports preserved, transformed, warned, and lost properties for each fixture. Additional regression fixtures should cover identifiers, references, embedded content, media regions, mixed sets, and strict-mode diagnostics. Apart from MaxQDA, whose current fixtures cover multiple versions, the format adapters should be made version-aware where the native specifications provide version distinctions. This would reduce the risk of unexpected import errors.

7.2.4 Continuous integration

Finally, compact regression fixtures for the formats that currently depend on large project files could allow the complete test suite to run in continuous integration without requiring every large source artifact. The fixtures would need to cover the same entities, references, and loss cases as the existing tests.

Appendices

A Datasets

Table 1 lists the Zenodo-hosted source files used in this thesis together with the creator names from the corresponding Zenodo record metadata. The entries are grouped by Zenodo record, so each record identifier occurs once; multiple source files from one record are listed in the same row. Personal names are presented in family-name-first order where the metadata permits this; collective, anonymous, and otherwise non-separable entries are retained.

Querying https://zenodo.org/api/records/{ID} with the respective record ID will yield the full metadata for each source file.

The records were checked on 2 September 2026 and listed the files under the Creative Commons Attribution 4.0 International License (CC BY 4.0) (Creative Commons, 2013) at that time.

Table 1: Zenodo sources used in this thesis

Source file(s)	**Creator(s)**	**Record ID**
`GH_tagging_codebook_v.07 _2017.07.04.nvp`	Perrotta, Carlo; Persico, Donatella; Haggis, Mata; Bailey, Chris; Passarelli, Marcello; Buijntenweg, Thomas; Earp, Jeffrey	1163698
`Gaming Horizons - Copy 2.nvp`	Perrotta, Carlo; Bailey, Chris	1163707
`Midstream and downstream behavioural patterns.nvp` `Upstream behavioural patterns.nvp`	Aboah, Joshua	2600823
`MALARIA SHORT COURSE (NVivo 12).nvp`	Ajayi, Ajumobi, Ogunwale et al	3588514
`DS_education_1_review _thematic_analysis.nvp`	Vlachos, Evgenios; Wildgaard, Lorna; Hüser, Falco; Knudsen, Christian B.	3634756
`DCR Networks Evaluation.atlproj`	Andaloussi, Abbad	3724874
`SENSE-GARDEN Qualitative Interview Study 2020 Dataset.nvp`	Goodall, Gemma; André, Lara; Taraldsen, Kristin; Serrano, J Artur	4081468

Source file(s)	Creator(s)	Record ID
`Evidence of controversies.nvp` `Salini Impregilo Sustainability Discourses.nvp`	Anonymous	4392862
`Interviews.rqda`	Lai, Kenneth	4460549
`Data Labelling & Encoding.mx20`	Li, Ruiyin; Liang, Peng; Soliman, Mohamed; Avgeriou, Paris	4481565
`JBE_SocialEntrepreneurship_network analysis (Adélie Ranville 2021-05-14).atlproj`	Ranville, Adélie	4767088

`Amazon Dataset.qdpx`
`Genetic Companies Dataset.atlproj`
`Amazon Dataset.atlproj`
`Genetic Companies Dataset.qdpx`
	Bennett, Nikki E.	4774406

`SBES - Publication.mx20`	de Oliveira, Ruan Pierre; Massoni, Tiago; de Araújo, Narallynne Maciel; Sarmento, Camila Freitas; dos Santos, Francielle Silva	5095271
`1 - Constructive turn - Nvivo project file (Mac).qdpx`	Mačiulienė, Monika; Cools, Hannes	5703624
`Obesidad PUQ_anonymisiert (Franziska Pfister 2022-05-20).qdpx`	Pfister, Franziska; Pozas, Claudia	6567017
`coded-dataset.qdpx`	Swillus, Mark; Zaidman, Andy	6595110
`1.1.2. The Primary Coding.mx20` `1.2.2. The Secondary Coding.mx20`	Askari Matin, Sajjad; Montazeri Moghaddam, Reza	6836552
`Zins_(2007)_Classification_Systems_Comparison.mx20`	Köhler, Juliane	6957587
`QualitativeCodingResults.nvp`	Diezmartínez, Claudia V.; Short Gianotti, Anne G.	7008298
`Air-HSR 2020.nvp`	Rattanakijsuntorn, Waralee	7063636
`IoT Roadmap - Final Coding.qdp`	Campos Motta, Rebeca	7257727
`2-MaxQDA-Project.mx22` `3-REFI-QDA-Project.qdpx`	Sell, Johann	7330414

Source file(s)	**Creator(s)**	**Record ID**
`Populism Abbas Netanyahu Speeches at UNGA 2010-19 MAXQDA code.mx20`	Olivas Osuna, Jose Javier; Burton, Guy	7738022
`SDG in EFL.mx20` `SDG in EFL.qdpx`	Lloret-Catala, C.; Alcantud-Diaz, M.	7940122
`Ugel 02 Lima .mx20` `Ugel 02 Lima .qdpx`	Lloret-Catala, C.; Suárez-Guerrero, C.; Mateu -Luján, B.	7992180
`USMOV GDdocentes.mx20` `USMOV GDdocentes.qdpx`	Lloret-Catala, C.; Suárez-Guerrero, C.; Mateu-Luján, B.	8116354
`Appendix 1_MAXQDA 09-2023 Data coding- EU Externalization of borders - Policy Frameworks on Migration with AFrica-HoA region - Case Sudan 2015-2020.mx22`	Seurei, Jerop-Janet	8309100
`Asghari edited.mx20`	Asghari	10333562
`SLR_Analysis_updated_2023.nvp` `SLR_Analysis_updated_2023_for_MAC.nvpx`	Maqbool, Bilal; Herold, Sebastian	10557144
`DATOS_ATLAS_IMITACION_PATRICIAGASCON.hpr7`	Gascón Vera, Patricia	10619534
`SBOM data analysis_backup-20230125-232849.mx22` `SBOM data analysis_backup-20230124-215119.mx22`	Bi, Tingting	10637929
`Populism Morocco Spain border MAXQDA code v2 25 may 2023.mx20`	Olivas Osuna, Jose Javier	10638542
`Entrevistasycodificación GenMuda.atlproj`	Celia, de Marcos Fernández	10789175
`Coding.mx22`	Salas-Urbano, María; Capitán-Agudo, Carlos; Cabanillas, Cristina; Resinas, Manuel	10939676
`Memes Project.qdpx`	Walters, Alyvia	10966460
`LLM_lowcode.mx20` `Traditional_lowcode.mx20`	Liu, Yongkun	11232842

Source file(s)	Creator(s)	Record ID
voluntariado_exemplo_ MAXQDA_V4.mx24	Freitas, Francisco	11445692
POLCOR_codificacion definitiva.nvp	Táboas-Pais, María Inés; Canalez-Lacruz, Inma; Rey-Cao, Ana	12542359
2005 plan.nvp 1968 plan.nvp 2018 plan.nvp	Rios-Ocampo, Juan Pablo; Gary, Michael Shayne	13750800
LIDERAZGO TONI.nvp	de la Hoz Ruíz, Javier	13772386
201006A_restframe.qdp	Rowlinson, Antonia	13833552
eCape Energy crisis - anonymisering - final.nvp	Gram-Hanssen, Kirsten; Madsen, Line Valdorff; Nielsen, Rikke Skovgaard; Aagaard, Line Kryger	13847672
Transcrições e codificações.qdpx	Sampaio, Lucas Tito	14183330
Transcrições e codificações.qdpx	Sampaio, Lucas Tito	14195018
Análisis_Primer_ Instrumento.hpr7 Análisis_Cuali_ Segunda_Fase.hpr7	Jimmy, Ramírez	14552075
Integrative review.nvpx	Quintiens, Bert; D'Eer, Louise; Deliens, Luc; Van den Block, Lieve; Chambaere, Kenneth; De Donder, Liesbeth; Cohen, Joachim; Smets, Tinne	14650724
SPSxFic_Master_ MelinaGhasseminejad.mx24	Ghasseminejad, Melina	14780402
IO institutional resistances.nvpx	University of Namur	15040157
Trascripcion EPS Coloquio .mx20	Mosquera Ruiz, Edinso Rafael	15126328
Expert_Interviews_TAM.mx24	Schmidthaler, Eva; Himmelbauer, Michael	15322629
Tuits Amnistia-Investidura (Andrés Garcia 2025-05-13).qdpx	Garcia Latorre, Andres	15399870
dr tirta.nvp	Sulaeman, Angia	15586678

Source file(s)	Creator(s)	Record ID
TedTalks.qdpx	Leach, Tonii	15723742
Coding.mx22	Salas-Urbano, María; Capitán-Agudo, Carlos; Cabanillas, Cristina; Resinas, Manuel	15850890
ScR NViVO.nvp	Hamza, Naheeda; Bernabe, Rosemarie; Alina, Coman; Kulkarni, Uma	15860636
workshop KPI.nvpx KPI - Diagnosis cycle.nvpx KPI - cycle de validation.nvpx	Dechamps, Simon	15878416
EFL_ILP_ScopingReview_ CodedData_20250718.qdpx	Huaqiang, Li	16082705
Thermatic Analysis-Haptic.nvpx	Lan	17188839
Anonymisiert_Paludi_ Rawdata.mx24	Merkelbach, Morena	17241630
23_PLANET4B_QDPXProject-FrameAnalysis_Pisa_20250514_ Qualitative-Analysis-Project.qdpx	Navarro Gambín, Pedro; Bonetti, Marta	17347915
1_PLANET4B_QDPXProject-BusinessModels_Pisa_20250919_ Qualitative-Analysis-Project.qdpx	Navarro Gambín, Pedro; Bonetti, Marta; Villa, Matteo	17360366
FF1_B_QIA_in_ MAXQDA_public.mx24	Heysel, Jan	17384976
Analisi testuale.qdpx	Lucchese, Carolina	17416520
thesis 1.mx24	Samet, Mahya Samet; Saeedimehr, Mohammadmahdi Saeedimehr	17762610

References

ATLAS.ti Scientific Software Development GmbH. (n.d.-a). ATLAS.ti HU XSD schema. https://atlasti.com/features/xml/atlasti_hu_2.2.xsd.zip

ATLAS.ti Scientific Software Development GmbH. (n.d.-b). ATLAS.ti XML export. https://atlasti.com/de/features/xml

Corti, L., & Gregory, A. (2011). CAQDAS comparability. What about CAQDAS data exchange? *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research*, *12*(1). https://doi.org/10.17169/fqs-12.1.1634

Creative Commons. (2013). Attribution 4.0 International (CC BY 4.0). https://creativecommons.org/licenses/by/4.0/legalcode.en

Darwin, I. F. (2025, June 17). *File(1) – determine file type*. Retrieved September 2, 2026, from https://man7.org/linux/man-pages/man1/file.1.html

European Parliament and Council of the European Union. (2009). Directive 2009/24/EC of the European Parliament and of the Council of 23 April 2009 on the legal protection of computer programs. https://eur-lex.europa.eu/eli/dir/2009/24/oj

Farrell, J., & Klemperer, P. (2007). Chapter 31 coordination and lock-in: Competition with switching costs and network effects. In M. Armstrong & R. Porter (Eds.). Elsevier. https://doi.org/10.1016/S1573-448X(06)03031-7

MAXQDA - VERBI Software. (n.d.). MAXQDA file types. https://www.maxqda.com/help/technical-data-and-information/file-management

of Essex, U. (n.d.). The qualitative data exchange schema. https://www.data-archive.ac.uk/managing-data/standards-and-procedures/metadata-standards/qudex/

QDArchive. (2026). *qdconvert* (Version 5.0.0). Retrieved September 2, 2026, from https://github.com/QDArchive/qdconvert

Rotterdam Exchange Format Initiative. (2019). Exchange of processed data between qualitative data analysis software packages. https://www.qdasoftware.org/project-implementation-files

SQLite Consortium. (n.d.). Database file format. https://www.sqlite.org/fileformat.html

Washizaki, H., Guéhéneuc, Y.-G., & Khomh, F. (2018). Prometa: A taxonomy for program metamodels in program reverse engineering. *Empirical Software*

Engineering, *23*(4), 2323–2358. https://doi.org/10.1007/s10664-017-9592-3