Functional Specification Document (FSD_01)

Module Name: Multi-format Bulk Input Handling Engine

Audira Reference: Related to the workflow defined in *AUDIRA FILE & DATA UPLOAD SCHEMA*, *AUDIRA PRODUCT BLUEPRINT*, and *AUDIRA AGENT ONBOARDING FRAMEWORK (v1.0)*

---

1. Scope

Purpose:
This module is responsible for receiving, decoding, and preparing all types of uploaded content submitted by the user during the onboarding stage of Audira — specifically after the fixed questions and discovery questions phase (see *AUDIRA AGENT ONBOARDING FRAMEWORK*).

The module is the entry point to the full "document intelligence" pipeline, taking raw input and transforming it into clean, structured, taggable content blocks.

In-Scope Functions:

- Accept and validate document types: `.pdf`, `.docx`, `.xlsx`, `.pptx`, `.jpg`, `.png`, `.mp3`, `.wav`
- Auto-detect file type and MIME
- Extract structured text using OCR, ASR, or parser depending on format
- Generate content blocks (paragraphs, tables, captions, images, metadata) with positional and source mapping
- Associate each block with unique IDs for downstream tracing
- Ensure the format is consistent and aligned with downstream needs for segmentation and tagging

Out of Scope:

- Business meaning extraction (done in FSD_02)
- Tag detection or mapping (done in FSD_03)
- Discovery question generation (done in FSD_04)
- Response synthesis or RAG behavior (handled by prompt chain in *AUDIRA PROMPT CHAIN & LLM LOGIC FLOW*)

---

2. Supported Input Types and Format-Specific Expectations

The input handler accepts a wide variety of formats common to business use-cases:

Format	Example Use Case	Processing Logic
`.pdf`	Uploaded contract, report, invoice	Layout-aware OCR + text extraction + table parsing (see *AUDIRA FILE & DATA UPLOAD SCHEMA*)
`.docx`	Business profile or SOP	Native parser (text + heading detection + inline image flagging)
`.xlsx`	Financial model, KPIs, CRM data	Sheet-aware parsing, row/column header mapping, numeric scoring
`.pptx`	Pitch deck, business slide	Slide title → caption extraction + visual layout annotation
`.jpg` / `.png`	Scanned ID, flowchart, logo	OCR pass + optional vision model tag + metadata trace
`.mp3` / `.wav`	Voice note explaining business	ASR → text transcript + speaker diarization if multi-user

Each uploaded file is assigned a `doc_id`, and each block derived from it is mapped to a `block_id` with `block_type` (e.g., `table`, `paragraph`, `image`, `caption`, `transcript`) as per schema in *AUDIRA FILE & DATA UPLOAD SCHEMA*.

---

3. Processing Pipeline – Step-by-step Input Transformation Flow

The following pipeline applies to all files processed by the bulk input handler:

1. **File Upload Initiated**
   - Triggered after client completes Discovery Qs (per *AUDIRA AGENT ONBOARDING FRAMEWORK*)
   - Each file is assigned a `doc_id`, timestamp, and `uploaded_by` metadata
2. **MIME Type & Extension Verification**
   - Validate against allowed formats table (see Section 2)
   - If invalid: return format error to frontend
3. **Pre-processing & Format Decoding**
   - `.pdf` → OCR + PDFminer or layout-preserving parser
   - `.docx` → python-docx or equivalent
   - `.xlsx` → sheet-wise extraction with header detection
   - `.jpg` / `.png` → OCR via Tesseract or PaddleOCR
   - `.mp3` / `.wav` → ASR using Whisper or Vosk for transcription
4. **Block Detection Engine**
   - Segment file into atomic units: `paragraph`, `table`, `image`, `caption`, `transcript`
   - Tag each with `block_id`, `position`, `content_hash`, and `doc_id`
   - Capture bounding boxes if available (for table/image layout matching)
5. **Content Structuring Layer**
   - Flatten or nest structured data depending on complexity
   - Normalize output into common schema (see *AUDIRA FILE & DATA UPLOAD SCHEMA*)
   - Detect embedded tables in paragraphs and tag accordingly

6. **Metadata Enrichment**
    - Add `language`, `word_count`, `source_type`, and confidence scores
    - Flag anomalies (e.g., corrupted files, unrecognized glyphs)
7. **Output Delivery to Segment & Classify Engine (FSD_02)**
    - All `block_id` units are queued for classification and business logic tagging
    - Logs passed to `audit_log_input_parser` table (referenced in *AUDIRA PRE-LAUNCH VALIDATOR SPEC*)

---

4. Metadata Handling

Each uploaded file and extracted content block is enriched with traceable metadata to support:

- Audit logging
- Tag sourcing (for gap detection in FSD_04)
- Traceable RAG responses in downstream prompt chains

File-Level Metadata:

- `doc_id` (UUID)
- `file_name`
- `uploaded_by` (user_id)
- `timestamp_uploaded`
- `mime_type`
- `language_detected`
- `document_tags` (initial placeholder, linked in FSD_03)

Block-Level Metadata:

- `block_id` (UUID scoped to `doc_id`)
- `block_type` (paragraph, table, image, etc.)
- `position` (sequence in doc)
- `bounding_box` (if applicable)
- `content_hash`
- `word_count`, `confidence_score`, `text_direction`, `source_format`
- `origin_context` (header, body, appendix, caption, etc.)

Integration Hooks:

- Metadata is passed forward as a JSON schema via internal message bus (see *AUDIRA PRODUCT BLUEPRINT*)
- Fields are also indexed for search retrieval in *AUDIRA PROMPT CHAIN & LLM LOGIC FLOW*

---

5. Output Format

The standardized output from this engine is a **structured JSON document**, passed to downstream services for classification, tagging, question generation, and final qualification.

Each JSON output includes:

```
{
  "doc_id": "UUID",
  "file_name": "business_plan.pdf",
  "blocks": [
    {
      "block_id": "UUID",
      "block_type": "paragraph",
      "content": "We offer logistics services across Cairo and
Alexandria...",
      "position": 1,
      "bounding_box": null,
      "metadata": {
        "word_count": 12,
        "confidence_score": 0.96,
        "source_format": "pdf",
        "origin_context": "main_body"
      }
    },
    {
      "block_id": "UUID",
      "block_type": "table",
      "content": [["Region", "Revenue"], ["Cairo", "15000"], ["Alex",
"9000"]],
      "position": 2,
      "bounding_box": [100, 200, 400, 600],
      "metadata": {
        "confidence_score": 0.92,
        "source_format": "pdf",
        "origin_context": "financial_appendix"
      }
    }
  ]
}
```

Downstream Dependencies:

- Used directly by the *Segment & Classify Engine* (FSD_02)
- Indexed in RAG memory stack (per *AUDIRA PROMPT CHAIN & LLM LOGIC FLOW*)
- Supports replay in *AUDIRA AGENT SIMULATION TEST KIT* via traceability fields

---

6. Integration Points

This module is a foundational layer in Audira's intelligence stack. Integration is both real-time and persistent.

Inbound:

- Triggered after the onboarding phase via upload interface (see *AUDIRA AGENT ONBOARDING FRAMEWORK*)
- Accepts file payloads via frontend + API gateway defined in *AUDIRA INTEGRATION SCAFFOLDS GUIDE*

Outbound:

- Outputs structured block arrays to **FSD_02 Segment & Classify Engine**
- Sends indexed data to **RAG memory** used by *AUDIRA PROMPT CHAIN & LLM LOGIC FLOW*
- Logs metadata to **audit_log_input_parser** table per *AUDIRA PRE-LAUNCH VALIDATOR SPEC*
- Alerts validator if new document types are not covered in discovery tags (future enhancement for dynamic tagging coverage check)

System Hooks:

- Operates over internal event bus with message types: `doc_uploaded`, `block_ready`, `block_error`
- Downstream modules acknowledge each block ingestion to allow rollbacks or reprocessing if needed

---

7. Error Handling & Edge Cases

This module includes robust fault tolerance for malformed inputs and document anomalies.

Handled Errors:

- ❌ Unsupported file type → Return `415 Unsupported Media Type` with list of allowed extensions
- ❌ Corrupted or unreadable file → Log `doc_failed` with hash & notify user
- ❌ OCR/ASR fails or returns <50% confidence → Mark block with `low_confidence = true`
- ❌ Empty document or no extractable content → Trigger `no_data_found` status
- ❌ Encoding issues (e.g., invalid characters) → Normalize to UTF-8 and log error trace

System Resilience Features:

- All failures captured in `audit_log_input_parser` table (see *AUDIRA PRE-LAUNCH VALIDATOR SPEC*)
- User interface provides clear reason for rejection (linked to error_code lookup)
- Retry logic enabled for non-corrupted format parse failures
- Async flagging of potential data loss (e.g., missing image alt text, table borders skipped)

Edge Case Coverage:

- Large file batch (multi-MB): Automatically chunked before processing
- Mixed content (image+text): Each is handled by dedicated handler then merged
- Handwriting or stylized text: Confidence is capped and pushed to `uncertain_review` queue
- Redacted or partially obscured content: Highlighted in metadata with `visibility_issue` flag

---

8. Future Enhancements

The following upgrades are scheduled for future milestones to increase the robustness and intelligence of the input handling engine:

1. Handwriting OCR Layer

- **Why:** To support scanned handwritten notes and feedback forms, especially in industries like clinics, logistics, or creative design.
- **When:** Phase 2 after initial deployment.
- **Sample Tool:** Microsoft Read API, Google Cloud Vision OCR Handwriting Mode.
- **Audira Value:** Helps agents understand SME notes scanned from notebooks or whiteboards.

2. Multilingual Document Segmentation

- **Why:** Clients often upload content in multiple languages across pages.
- **When:** Q3 Enhancement.
- **Sample Tool:** fastText + LangDetect chain + polyglot + paddleocr multilingual
- **Audira Value:** Allows context-aware segmentation for bilingual input (e.g., Arabic/English).

3. Embedded Tag Anchoring with Visual Blocks

- **Why:** Currently text and visuals are structured separately; this adds anchors between business text and adjacent images/tables.
- **When:** As part of Smart Tag Flow upgrade (see future FSD_03+)
- **Sample Tool:** LayoutLMv3, Donut OCR
- **Audira Value:** Improves AI understanding of presentation decks or brochures.

4. Duplicate File Detection and Versioning

- **Why:** Prevent clients from uploading multiple versions of the same file with minimal changes.
- **When:** As part of Doc Upload Pipeline Revamp.
- **Sample Tool:** simhash, file signature hashing, LLM document comparison
- **Audira Value:** Avoid redundant AI workloads, maintain clean document tree.

5. Context-aware Voice Separation (multi-speaker)

- **Why:** To distinguish between customer and partner voices in calls/meetings.
- **When:** Phase 3.
- **Sample Tool:** pyannote-audio or NVIDIA NeMo
- **Audira Value:** Helps generate accurate tagged insights from roundtables, sales calls.

6. Streamed Input Parser

- **Why:** For large voice, doc, or video transcripts, allow streaming chunked parsing.
- **When:** Scaling stage post-beta.
- **Audira Value:** Enables near-real-time ingestion from API-sourced transcripts or ongoing meetings.

7. Optional Public Dataset Calibration Layer

- **Why:** Compare uploaded content structure to known domain samples (e.g., ISO reports, FDA formats)
- **When:** With Knowledge Profile Calibration (AI training phase)
- **Audira Value:** Allows Audira to auto-profile structure using reference patterns from business sectors.

Licensing Note: All enhancements must use MIT or open-source components (see *AUDIRA PRODUCT BLUEPRINT*) to protect Dev Roadmap and scalability ownership.