Paper ET09

RIP – Inconsistent Metadata Specifications with RAP (R And Python). Generate Specs Faster and Detect Early Errors to Speed Up Submissions.

Kairav Tarmaster, Sycamore Informatics Inc., Surat, India

ABSTRACT

Define-XML specifications are reverse-engineered from datasets and manually updated. The Define-XML is validated for conformance to CDISC and FDA rules, and errors and warnings are reported. Authoring the defined specifications with open-source technologies speeds up the process with real-time feedback. As you author the specs, early detection of errors will save time and reduce the validation cost by avoiding multiple iterations.

Like SDTM specs, metadata generation and validation are enhanced with plug-and-play applications. It addresses a common and frequent challenge with manual metadata entry. Manual entries are time-consuming and error prone. Metadata is created and validated based on rules well defined by the industry (CDISC and Regulatory Conformance Rules), companies, specific business functions, or systems.

This presentation will show the use of R and Python applications with examples, to evolve our thoughts on handling and automating metadata.

INTRODUCTION

In the clinical trial ecosystem, metadata management is essential to ensure data integrity, regulatory compliance, and streamlined operations. Metadata defines data structure, lineage, and meaning, making it a key enabler for efficient trial execution. However, traditional approaches to metadata management are often manual, error-prone, and time-intensive. Clinical trial systems are transforming with the rise in the adoption of Generative AI (Gen AI) and open-source technologies like R and Python.

Gen AI can automate metadata generation by applying pre-defined business rules, regulatory standards such as CDISC, and historical study data. This significantly reduces manual efforts and ensures accuracy in metadata creation. Additionally, AI models can suggest mappings for SDTM and ADaM datasets by learning from previous study metadata, enabling faster and more efficient data standardization.

Open-source technologies like R and Python provide flexible and powerful frameworks for building applications that generate, validate, and manage clinical metadata. Organizations can use low code-no code platforms to develop dashboards that allow clinical teams to manage study specifications, validate metadata against compliance rules, and monitor data quality.

This paper explores how integrating Gen AI with R and Python with a metadata repository (MDR) drives clinical trial metadata management innovation. Examples will demonstrate how automated tools can streamline the metadata lifecycle, improve regulatory compliance, and accelerate study timelines. The future of clinical metadata management lies in leveraging AI-powered applications to reduce human errors, ensure data accuracy, and enhance decision-making processes.

INDUSTRY CHALLENGES IN METADATA MANAGEMENT

Across various encounters with sponsors and CROs, metadata has been characterized by many adjectives.

- Boring and Repetitive
- Lengthy and Time-Consuming
- Inevitable yet Trivial
- Standardized yet Novel

These characteristics have emerged due to the current ways of handling metadata and the challenges in establishing and maintaining robust metadata management systems. The primary problems include:

1. **Manual Metadata Creation and Entry:** Human-based metadata creation is prone to errors, time-consuming, and often inconsistent. Metadata is usually created using spreadsheets or legacy systems without sufficient validation checks, leading to human errors and discrepancies.
2. **Fragmented Metadata across disparate systems:** Data often resides in multiple systems across an organization, making it difficult to access and manage siloed metadata. Different business functions maintain their metadata differently and disjointed, resulting in inconsistencies and challenges in cross-functional data analysis.
3. **Lack of Metadata Standardization:** Inconsistent data definitions and the absence of standard metadata models hinder interoperability across systems. Without standardized metadata, integrating data from different sources becomes complex and increases the risk of misinterpretation.
4. **Metadata Quality and Governance:** Poor metadata quality, missing data lineage, and lack of ownership lead to unreliable data insights and non-compliance. Metadata errors can propagate across data pipelines, impacting downstream analytics and decision-making.
5. **Limited Metadata Utilization:** Many organizations fail to leverage metadata for data discovery, analytics, and decision-making. Metadata remains underutilized when organizations lack the tools and processes to extract actionable insights.
6. **Inefficient Regulatory Compliance Management:** In regulated industries like healthcare and pharmaceuticals, ensuring compliance with standards such as CDISC requires accurate and validated metadata. Manual compliance checks are labor-intensive and prone to errors.
7. **Lack of Real-Time Metadata Insights:** Organizations often struggle to monitor metadata in real-time, leading to delays in issue identification and resolution. Real-time monitoring can provide proactive insights into data quality and compliance issues.

ADOPTION OF METADATA REPOSITORY AND GENAI

The clinical trial industry is rapidly adopting Metadata Repository (MDR), Gen AI, and open-source technologies like R and Python to enhance operational efficiency, ensure compliance, and accelerate trial timelines.

Companies recognize that a centralized metadata repository is necessary to manage the clinical trials' metadata. While the clinical data originates across multiple sources (EDC, Labs, Vendors, PKPD, etc.), is transformed into Review and Tabulation Models, and analyzed to produce Tables, Listings, and Figures, the metadata for all the data states with their transformation and derivation logic should be managed in a single repository for end-to-end traceability. Our vision extends beyond this to use the AI capabilities with MDR to generate, validate, and adorn metadata.

The need to automate metadata is real. It completes a full cycle from Protocol to CSR with multiple business functions involved, including Protocol Planning and Design, Clinical Data Management, Statistical Programming, and CSR Medical Writing. The common objective is speeding up the trials with optimal study design and fewer amendments.

Each function evaluates GenAI's use for the metadata and data use cases that best suit their business needs.

- For Protocol Design, GenAI assists in optimizing the objectives, endpoints, trial design, and the schedule of assessment (SoA) from past protocols and scientific journals for novel yet complete and consistent study designs.
- Clinical Data Management uses Retrieval Augmented Generated (RAG) to recommend new form design and to speed up data review, cleaning, and reconciliation by interpreting data and identifying outliers or anomalies.
- Statistical Programming automates the generation of analysis datasets by predicting appropriate data transformation specifications and creating statistical models based on previous trial data. The use case extends to suggest programming logic for standard data derivations, validating statistical outputs

by comparing generated results.

- CSR Medical Writing uses GenAI to reuse clinical content from multiple study documents to author CSRs and AI-powered assistants to interpret outputs and create summaries.

Open-source platforms like R and Python further facilitate the implementation of AI models, integrating seamlessly with existing clinical trial workflows and reducing the burden of generating and validating clinical trial metadata and data to achieve faster analysis cycles and improved accuracy in study reporting.

Using Gen AI and emerging technologies with R and Python, the challenges commonly faced can be addressed -

- **Gen AI for Metadata Generation and Validation:** Using Generative AI models, organizations can automate metadata generation based on business rules. For example, when designing a new Case Report Form (CRF) in a clinical study, Gen AI can generate necessary metadata using R or Python applications.
- **Rule-Based Metadata Generation:** Implement rule-based engines using R and Python to generate metadata automatically. Applications can read minimal input from study teams and produce metadata aligned with standards such as CDISC or company-specific rules.
- **AI-Driven Metadata Recommendations:** Extend the metadata workflow by using machine learning models to recommend SDTM mappings and metadata suggestions based on previous studies and existing standards.
- **Metadata Validation and Conformance:** Develop Python and R validation engines that apply predefined conformance rules to ensure metadata quality and compliance. Dashboards can provide real-time validation feedback.
- **Low-Code-No-Code Dashboards:** Provide end-users with an intuitive low-code-no-code dashboard interface that enables metadata management without extensive programming knowledge. Platforms like Shiny (R) or Dash (Python) can be used for this purpose.
- **Scalable Metadata Management Platforms:** Leverage cloud-native metadata management solutions that support horizontal scaling to handle large-scale data environments.
- **Centralized Metadata Repository:** Implementing a centralized metadata repository using modern data cataloging tools allows organizations to maintain a unified view of their metadata assets.
- **Standardization and Interoperability:** Establish standardized metadata models as released by CDISC and other standards governing bodies.

While these are much broader benefits for using metadata repositories with GenAI capabilities, this paper describes the solution with specific examples of clinical data management (CRF) and SDTM programming (SDTM) workflows.

SOLUTION

Every organization and system often has well-defined rules that can be leveraged to generate and validate metadata. Using GenAI, these rules can be easily written in open-source technologies like R and Python and managed as RShiny and Streamlit applications, respectively. Within a framework of MDR, these apps can be triggered on demand or in real-time to generate and validate metadata for a study.

Rules-based metadata brings consistency to improve the quality of metadata in Standard Libraries. Study specifications built using Standard Libraries in MDR ensure conformance to the standards and promote consistency across studies. When there is a study-specific CRF / Domain, the rules-based metadata continues to drive consistency, thus increasing the overall quality of study specifications.

Similarly, integration with CDISC CORE and Pinnacle21 with MDR provides the framework to validate the metadata and get real-time feedback as users edit and work with it.

For different teams with common goals and objectives, these applications can be published on a common dashboard accessible to all the teams, maximizing their use.

SCENARIOS

CRF.01 - For a new CRF / Data Collection Instrument (DCI) in a study, the study teams finalize the science and biostatistics parts of the CRF. A lot of metadata needs to be entered for the new CRF to instantiate it in an EDC system.

The EDC programmer instantiates the new CRF in the EDC system and creates the metadata. Even though there are prescribed rules and standardization for entering the metadata, manual data entry could be error-prone due to human interpretation and the EDC programmer's experience and skill sets.

SDTM.01 - The SDTM specification author develops the SDTM mapping specifications and the define.xml metadata for the new CRF. The define.xml specifications have a much-delayed start by reverse-engineering the SDTM datasets. This gives a good starting point, but the SDTM programmers manually enter a lot of metadata for every variable in each domain. The define.xml specifications are validated, errors are detected, and the cycle repeats. Often, the define.xml specifications do not precisely portray the SDTM mapping specifications and the transformations. For example, the origin of an SDTM variable may be incorrectly reported as Assigned / Protocol instead of CRF.

The errors and inconsistencies are often detected much later, causing rework, thus adding cost and increasing timelines.

RULES

The rules mentioned below refer to the properties of a Field (item) on CRF and DataDictionary (Codelist) in Medidata Rave EDC, specifically the Rave ALS file, due to its familiarity. However, the idea and concept apply to metadata for any data model or system used in clinical trials. The Rules follow a naming convention: Metadata Generation (MG) and Metadata Validation (MV).

MG.CRF.Rule01

General Rule: When an Item on a Form uses a Codelist, its length should be the maximum of the EDC Display Value of all the Code Terms.

EDC Specific Rule: For a study conducted in Medidata Rave, when a Field (Item) uses a Data Dictionary (Codelist), the Data Format should be $n, where n is the max length of the User Data String in Data Dictionary Entries.

MG.CRF.Rule02

General Rule: When an Item on a Form uses a Codelist, the user interface control should be driven by the number of Code Terms.

EDC Specific Rule: For a study conducted in Medidata Rave, when a Field (Item) is associated with a Data Dictionary (Codelist), the Control Type should be determined based on the number of Data Dictionary Entries.

- When the count of Data Dictionary Entries < 5 in a Data Dictionary, Control Type for a Field = RadioButton
- When the count of DataDictionaryEntries => 5 in a DataDictionary, ControlType for a Field = Checkbox.

MG.CRF.Rule03

General Rule: My organization should follow CDASH Naming conventions for Item OIDs.

EDC Specific Rule: For a study being conducted in Medidata Rave, and when my organization standard follows CDASH Standards or a specific naming convention for FieldOIDs –

- When FieldOID ends with “DAT”, Controltype = “DateTime” and DataFormat = “dd- MMM- yyyy”
- When FieldOID ends with “TIM”, Controltype = “DateTime” and DataFormat = “dd MMM yyyy hh:nn:ss”

MV. SDTM.Rule01

For a derived SDTM variable, i.e., when Origin = Derived, the variable must have a Method Definition.

MV.SDTM.Rule02

For an SDTM variable with Origin = CRF, the variable must have Page Numbers specified.

MV. SDTM.Rule03

For an SDTM variable mapped to a CRF variable, i.e., when Origin = CRF, the SDTM Mapping specification must mention the Form.Item, where Form.Item is the 2-level convention for specifying mappings.

For an SDTM variable with an assigned value, i.e., when Origin = Assigned, the SDTM Mapping specification must mention a constant value in double quotes “value.”

MV.SDTM.Rule04

For a study, validate the SDTM specifications using CDISC CORE or Pinnacle21.

IMPLEMENT THE RULES PROGRAMMATICALLY USING GENAI

A programmer can program all the example rules above in R and Python languages. To develop programs for such rules, one could simply rely on GenAI these days to generate the programs to check/create the metadata managed in a Rave ALS file or, more appropriately, metadata managed in a Metadata Repository (MDR) that manages the metadata for all the data models in a study such as CRF, DTS, SDTM, ADaM and any custom data models in use at your company. Programming these rules is simplified using GenAI capabilities, and they can be published as applications managed within a Metadata Repository (MDR).

- Navigate to ChatGPT, or more appropriately a Locally available Large Language Model interface
- For ***MG.CRF.Rule01***, Enter the prompt as shown in the box below and attach the metadata file. The prompt entered in GPT will look as shown in Figure 1: Enter the prompt in GPT with the Metadata file, the resultant program will look as shown in Program 1 and the documentation for the program will look as shown in Figure 2 –

```
I am attaching a Rave ALS file in xlsx format. I want a [streamlit application /] python
program that reads the Fields tab of the file. For a FormOID-FieldOID, the program should
identify whether it is associated with the DataDictionary. If the FormOID-FieldOID is
associated with the DataDictionary, generate the DataFormat by referencing the maximum
length of the UserDataString in the DataDictionaryEntries tab for the DataDictionary
associated with the FormOID-FieldOID. The program should generate a Rave ALS file as
output with the updated DataFormat for each FormOID-FieldOID with DataDictionary
associated and not updating the DataFormat for the FormOID-FieldOID without
DataDictionary associated.
```

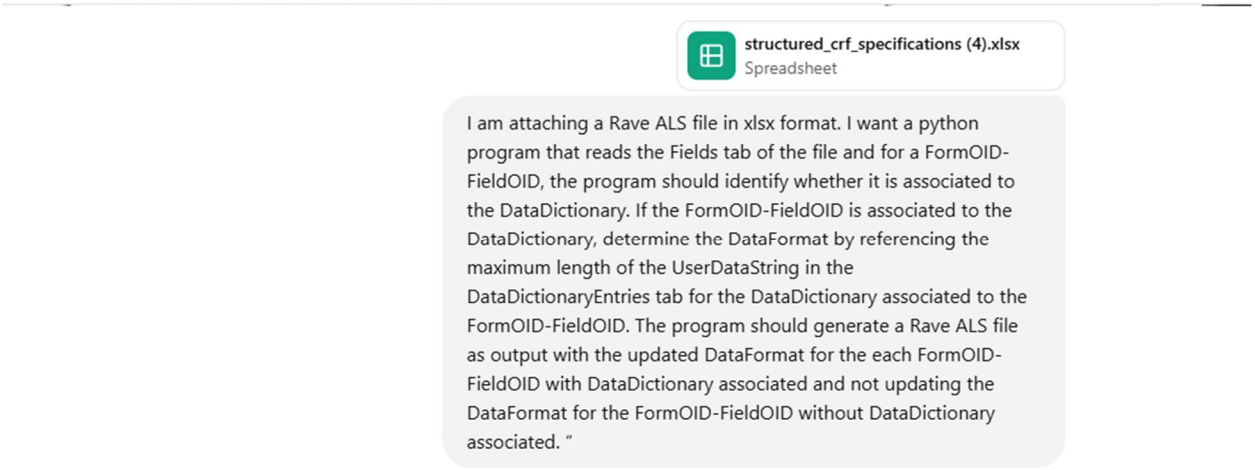
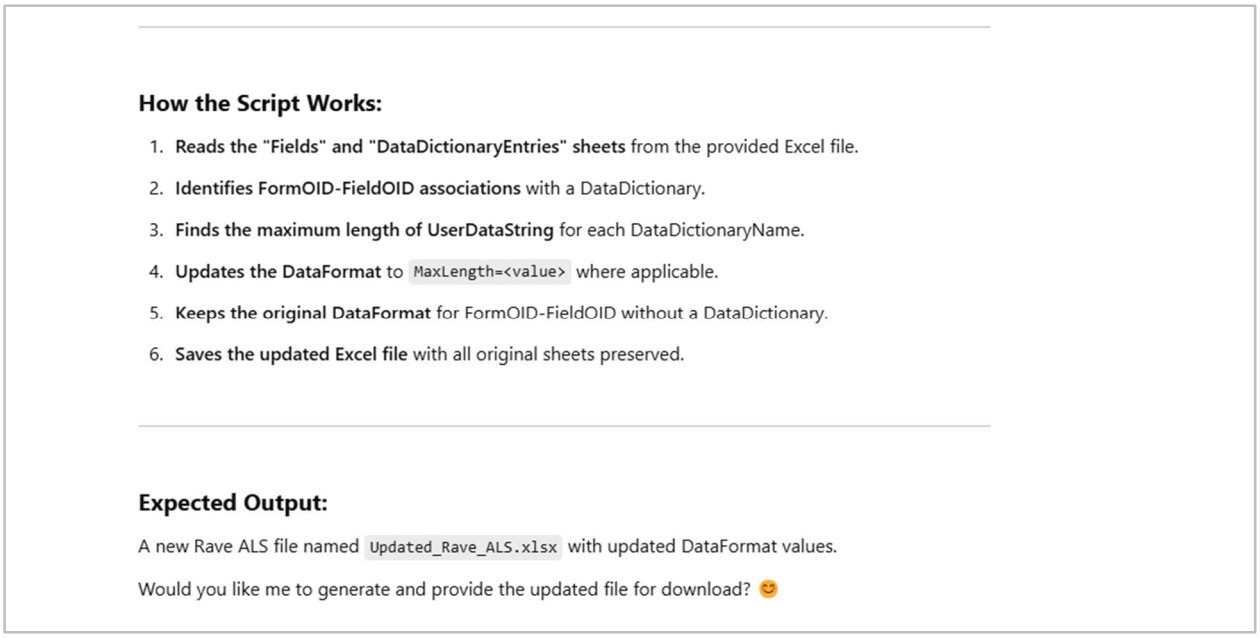


Figure 1: Enter the prompt in GPT with the Metadata file

```
import pandas as pd

def update_rave_als(input_file, output_file):
    """
    Reads the Fields tab from the Rave ALS file, updates DataFormat for FormOID-FieldOID
    associated with a DataDictionary using the max length of UserDataString from
    DataDictionaryEntries, and writes the updated file.
    """
    # Load the Excel file
    xls = pd.ExcelFile(input_file)

    # Read necessary sheets
    fields_df = pd.read_excel(xls, sheet_name="Fields")
    data_dict_entries_df = pd.read_excel(xls, sheet_name="DataDictionaryEntries")

    # Ensure required columns exist
    required_fields_columns = {"FormOID", "FieldOID", "DataDictionaryName", "DataFormat"}
    required_dd_entries_columns = {"DataDictionaryName", "UserDataString"}

    if not required_fields_columns.issubset(fields_df.columns):
        raise ValueError(f"Missing required columns in 'Fields' sheet:
{required_fields_columns - set(fields_df.columns)}")

    if not required_dd_entries_columns.issubset(data_dict_entries_df.columns):
        raise ValueError(f"Missing required columns in 'DataDictionaryEntries' sheet:
{required_dd_entries_columns - set(data_dict_entries_df.columns)}")

    # Compute max length of UserDataString for each DataDictionaryName
    max_lengths = (
        data_dict_entries_df.dropna(subset=["UserDataString"])
        .groupby("DataDictionaryName")["UserDataString"]
        .apply(lambda x: x.astype(str).map(len).max())
        .to_dict()
    )

    # Update DataFormat based on DataDictionary association
    def update_data_format(row):
        if pd.notna(row["DataDictionaryName"]) and row["DataDictionaryName"] in
max_lengths:
            return f"MaxLength={max_lengths[row['DataDictionaryName']]}"
        return row["DataFormat"]  # Keep original if no association

    fields_df["DataFormat"] = fields_df.apply(update_data_format, axis=1)
```

```
    # Write the updated data back to a new Excel file
    with pd.ExcelWriter(output_file, engine="openpyxl") as writer:
        for sheet_name in xls.sheet_names:
            sheet_df = pd.read_excel(xls, sheet_name=sheet_name)
            if sheet_name == "Fields":
                fields_df.to_excel(writer, sheet_name=sheet_name, index=False)
            else:
                sheet_df.to_excel(writer, sheet_name=sheet_name, index=False)

    print(f"Updated Rave ALS file saved as: {output_file}")

# Input and Output file paths
input_file_path = "/mnt/data/structured_crf_specifications (4).xlsx"
output_file_path = "/mnt/data/Updated_Rave_ALS.xlsx"

# Run the function
update_rave_als(input_file_path, output_file_path)

# Return the output file path
Output_file_path
```

Program 1. Python code provided by GPT



How the Script Works:

1. **Reads the "Fields" and "DataDictionaryEntries" sheets** from the provided Excel file.
2. **Identifies FormOID-FieldOID associations** with a DataDictionary.
3. **Finds the maximum length of UserDataString** for each DataDictionaryName.
4. **Updates the DataFormat** to `MaxLength=<value>` where applicable.
5. **Keeps the original DataFormat** for FormOID-FieldOID without a DataDictionary.
6. **Saves the updated Excel file** with all original sheets preserved.

Expected Output:

A new Rave ALS file named `Updated_Rave_ALS.xlsx` with updated DataFormat values.

Would you like me to generate and provide the updated file for download? 😊

Figure 2: The Documentation for the Program and the Expected Output

PUBLISHING AND EXECUTING THE APPS

Programmers develop these applications and make them available for use in company-vide. Developing and using such applications in isolation is neither beneficial nor commercially viable. The ROI benefits are realized when these applications are used at scale across studies. To make such applications available across the organization, the programmers should be able to publish them for the benefit of a wider audience.

PERSPECTIVE ON INDUSTRY-WIDE IMPLEMENTATION AND UTILIZATION

While Gen AI and open-source technologies offer significant benefits, organizations face several concerns during adoption.

Data privacy and security remain paramount, as sensitive study designs and metadata must be safeguarded to comply with regulations like GDPR and HIPAA. Ensuring AI models maintain data privacy while processing metadata is essential.

Regulatory compliance also presents challenges, as AI algorithms need to produce transparent and auditable results to meet the scrutiny of regulatory bodies that require explainability and validation of AI-generated metadata.

Metadata quality and model bias pose risks since poor-quality information can lead to biased AI models and inaccurate predictions. Continuous monitoring and retraining are necessary to ensure model reliability.

Integration complexity further complicates adoption, as legacy systems may lack compatibility with AI-powered applications, requiring seamless integration with existing metadata management systems.

User adoption and training are significant concerns. Teams may need comprehensive training to utilize AI-driven solutions effectively, and organizations should foster a culture of innovation to drive adoption.

Finally, **cost management** remains a consideration, with substantial initial investments in AI and infrastructure. Companies must carefully evaluate the long-term return on investment (ROI) against the initial implementation costs.

By addressing these concerns through robust data governance, transparent AI models, and effective change management, the clinical trial industry can maximize the benefits of Gen AI and open-source technologies in metadata management.

CONCLUSION

Effective metadata management is the cornerstone of data governance and ensures organizations derive maximum value from their data assets. Companies can reduce manual errors by using Gen AI, R, and Python for metadata generation, validation, and recommendation, improve data quality, and ensure compliance with industry standards. Implementing centralized repositories, adopting metadata standards, enforcing governance policies, and utilizing scalable platforms will drive better data discovery, compliance, and analytics outcomes. This paper provides actionable insights for organizations seeking to enhance their metadata management capabilities and build a strong foundation for data-driven success.

REFERENCES

www.cdisc.org/core

www.sycamoreinformatics.com/mdr

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Kairav Tarmaster
Sycamore Informatics Inc.
ktarmaster@sycamoreinformatics.com
https://www.linkedin.com/in/kairavtarmaster
+91-90999-17133