Paper ET17

Orchestrating Efficiency: Symphony and Composer in Clinical Reporting

Margaret Wishart, Bristol Myers Squibb, Princeton, United States
Karma Tarap, Bristol Myers Squibb, Boudry, Switzerland

ABSTRACT

Symphony is a next-generation solution that streamlines the creation of mock shells in clinical data workflows by digitizing and automating traditional manual processes. It features a user-friendly Shiny-based interface for building and reviewing mock shells, generating table layouts with underlying R code and structured metadata. Once ADaM datasets are available, users can leverage Symphony to instantly produce submission-ready outputs, reducing redundant programming and speeding up delivery. The innovation is powered by two internally developed R packages: Symphony, which provides the Shiny UI for mock shell creation and review, and Composer, the engine that converts metadata into executable R code for both mock-ups and final tables, listings, and figures (TLFs). Both packages are actively being developed and will be open-sourced once stable, encouraging community collaboration and adoption for more efficient clinical reporting.

INTRODUCTION

Mock shells serve as a foundational component of clinical reporting, functioning as the primary requirements specification for regulatory submission outputs. Because they define the structure, content, and expectations of every table, listing, and figure (TLF), they undergo extensive cross-functional review across biostatistics, statistical programming, medical, and clinical operations teams. Ensuring their accuracy is critical, not only for quality, but for downstream operational efficiency.

Historically, however, mock shell creation has relied on fragmented, manual workflows. Teams typically build mock shells using Word templates, and Excel files, then reviewed and tracked through email exchanges. While significant effort is invested upfront to ensure consistency and correctness, such as aligning to therapeutic standards, agreeing on codelists, or resolving repeated review cycles, these efforts cannot be reused downstream. Word documents lack structured metadata, and manual formatting does not translate cleanly into statistical programming environments. As a result, programmers must reinterpret specifications, recreate structure from scratch, and manually encode metadata already reviewed in earlier stages. This disconnect leads to duplicative work, inconsistent interpretation, prolonged timelines, and unnecessary operational risk.

Symphony was designed to address these challenges by unifying mock shell design, review, and execution within a single, metadata-driven platform. By providing a unified, Shiny-based platform for building, reviewing, and operationalizing mock shells, Symphony enables users to create high-quality, metadata-rich specifications that are both human-readable and machine-executable. The tool streamlines cross-functional review, standardizes expectations, and bridges the gap between mock design and downstream programming. Symphony not only simplifies the creation and review process but also accelerates the transition into programming by generating structured metadata and executable R code that reflect the approved specifications. This represents a significant modernization of the clinical reporting workflow and unlocks efficiencies that were previously inaccessible due to tool limitations and manual processes.

MOCK SHELL CREATION

Symphony provides a next-generation, flexible user experience designed to modernize how teams create and maintain mock shells. The application integrates global standards, therapeutic-area conventions, automated validation, and structured metadata generation into a single interface that supports both scientific clarity and operational efficiency.

FLEXIBLE UI FOR RAPID MOCK SHELL CREATION

Symphony offers a user-friendly interface that enables rapid creation of table shell layouts. All global and therapeutic-area standards are preloaded, allowing users to begin from known templates rather than starting from scratch. While standards are encouraged, Symphony also offers flexible mechanisms to support novel or study-specific outputs.

A key innovation is AI-assisted mock generation: users can upload an image of a table, whether from a publication, a protocol, or a previous study and Symphony automatically converts it into a structured mock shell within the system.

This significantly reduces manual formatting work and helps ensure shells reflect intended designs from the earliest stages.
Once a table is created, users can format it to closely resemble the final expected output. This visual alignment helps improve cross-functional understanding and reduces ambiguity during review.

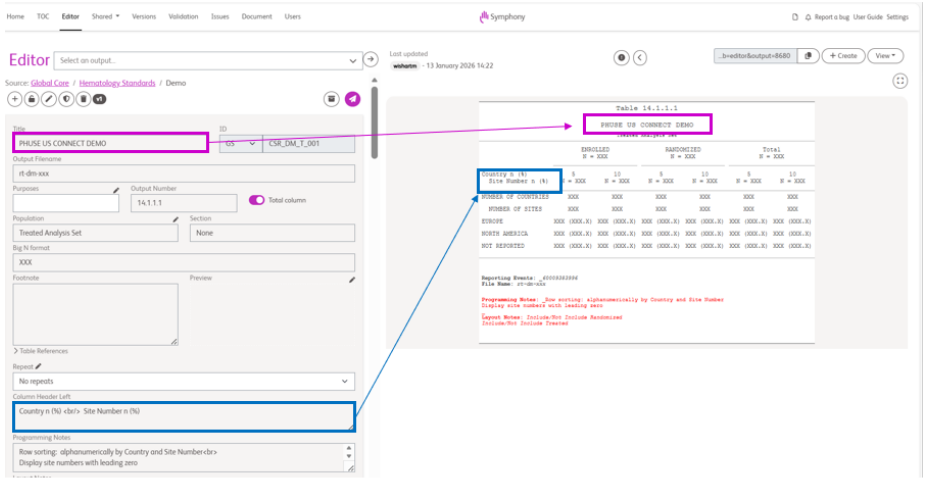

Figure 1: Mock shell custom editor

GLOBAL CONTROLS

The Global Settings panel centralizes the definition of reusable components such as:

- **Codelists:** Codelists enable users to identify common variables and their potential values within the data. This allows those variables to be incorporated directly into mock-up creation, helping replicate the final TLF structure as closely as possible before data are available and ensuring a smoother transition from mock-up to production outputs.
- **Repeats:** The repeat functionality allows users to easily define cases where a table has the same structure repeated with only small changes to population, grouping, or other parameters. By specifying these repeats at a global level, users can efficiently apply them to any tables where this pattern is needed.
- **Footnotes:** Footnotes provide essential clarifications, methodological notes, or regulatory references relevant to the data presented in tables, listings, and figures. Defining these centrally enables consistent messaging and ensures critical information is communicated accurately across all outputs.
- **Sections:** Sections act as organizational units that help group related outputs and control how they appear within the overall document

This reduces repeated manual work and ensures consistency across all outputs. Instead of redefining common components for each shell, teams can configure these once and apply them across multiple TLFs. This also guarantees that metadata is standardized across a study, facilitating downstream use in programming.

REVIEW FUNCTIONALITY AND AUTOMATED QUALITY CHECKS

Symphony includes built-in features to streamline cross-functional review cycles:

- **In-tool review and commenting**: Symphony enables users to conduct reviews and provide feedback directly within the application interface. This eliminates the need for external emails or offline documents, allowing all stakeholders to view, add, and respond to comments in real time. The centralized system streamlines collaboration, ensures that input is captured in context, and reduces the risk of feedback being overlooked or lost.**-tool review and commenting**
- **Issue identification and resolution tracking:** The platform provides built-in tools for flagging issues, assigning them to team members, and monitoring their status through to resolution. This structured approach ensures that all identified problems are tracked systematically, with clear accountability and visibility for each issue's progress, ultimately helping teams resolve concerns efficiently before finalizing outputs.

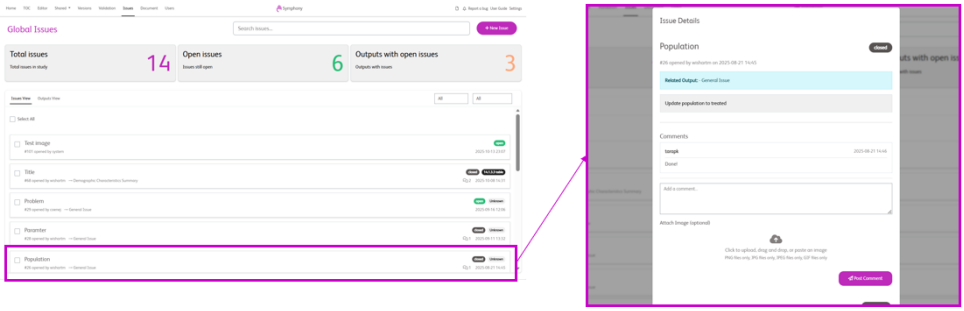

Figure 2: Issue management portal

- **Automated checks to detect inconsistencies, omissions, or structural mistakes:** Symphony performs automated quality control by scanning mock shells for potential errors such as missing information, logical inconsistencies, or deviations from standard structural conventions. These automated validations help teams catch and correct mistakes early in the process, reducing the risk of errors making it to later stages and supporting overall data integrity.
- **Version comparison to highlight changes between iterations:** The system automatically tracks versions of each mock shell and provides tools to compare different iterations side by side. This allows users to quickly identify what has been added, removed, or modified, supporting transparent review cycles and making it easier to audit changes and maintain a clear history of document evolution.

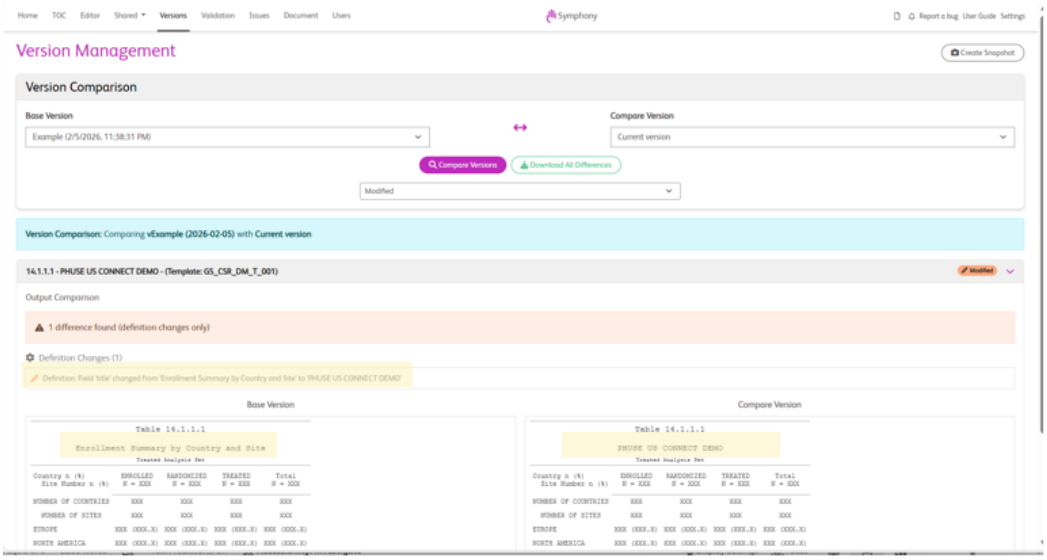

Figure 3: Version comparison tool

- **Metric tracking for TLFs and distributions:** Symphony tracks the total number of Tables, Listings, and Figures (TLFs) in a study, breaking down counts by type and highlighting repeated outputs with both counts and percentage of the total. It also shows which TLFs follow standard templates versus ad hoc formats, making it easy to monitor adherence to standards and identify non-standard deliverables. These metrics support resource planning, promote consistency, and guide standardization efforts.

By enabling these activities directly in the application, teams eliminate the inefficiencies of email-based feedback and reduce the risk of miscommunication.

ARTIFACTS AND DOWNSTREAM LEVERAGING OF WORK

One of Symphony's most valuable contributions is ensuring that upstream work is not lost. Once mock shells are defined, Symphony automatically generates:

- Structured metadata that feeds directly into SAS macros used in current production environments: Symphony captures all critical structural information for each mock shell. This metadata is formatted in a way that enables seamless integration with existing SAS macro frameworks. As a result, teams can automate the creation of production-ready tables and listings, ensuring that the outputs are consistent with approved specifications and reducing the need for manual data entry or rework. This direct pipeline from specification to production supports regulatory compliance, auditability, and reproducibility in statistical programming workflows.
- R code for all outputs (initially using mock data), reflecting the structure defined in the specification: For every mock shell, Symphony, leveraging its Composer engine, automatically generates R scripts that reproduce the exact table structure as defined in the specification. Initially, these scripts use mock data to visualize layouts and validate the design. This approach allows reviewers and programmers to confirm that the output will match expectations before real data is introduced, streamlining the review process and minimizing discrepancies between planned and actual outputs.
- Data Presentation Plans (DPPs): A comprehensive package of all mockups preserved within a qualified, version controlled system to support traceability and serve as the authoritative record of the approved requirements.-ups-controlled system to support traceability and serve as the authoritative record of the approved requirements.

Because all artifacts originate from a single source of truth, downstream programming is fully aligned with specifications, reducing rework and accelerating development.

DESIGN PRINCIPLES AND EXTENDED LEVERAGING OF SPECIFICATIONS

A core design principle of Symphony is that analyses are defined by their expected end products. By treating tables, listings, and figures as the primary analytical contract, Symphony captures dataset selection, variable definitions, population filters, grouping logic, and display intent at the time of mock shell creation. This output-driven approach shifts analytical decision-making earlier in the lifecycle and ensures that downstream artifacts remain aligned with approved scientific intent.

Centralized codelists and output metadata are treated as structured, exportable assets rather than UI-only constructs. Outputs defined with their associated datasets, variables, where-clause logic, codelists, and display specifications can be persisted and reused across organizational levels, including global, therapeutic-area, and study-specific standards. By storing specifications within this hierarchy, Symphony functions as both a creator and an enforcer of standards, enabling approved outputs to be reused without redefinition while still allowing controlled, study-specific overrides. Individual studies can discover existing standards and prior implementations and selectively copy and extend approved metadata, reducing variability and accelerating study startup.

By explicitly capturing dataset, variable, and filtering logic before data are available, Symphony enables the automated generation of annotated shells that remain synchronized with approved mock designs. The same structured metadata supports downstream reuse for Analysis Results Specifications (ARS) and Analysis Results Data (ARD), establishing a direct lineage from planned output to executable code and delivered result.

To further support traceability, global ADaM derivation specifications are modeled as a directed acyclic graph (DAG), where each target dataset is defined as a transformation of one or more source datasets (*target* <- *source*). This representation makes dependencies explicit and machine-interpretable, allowing derivation logic to be selectively subsetted to include only the final and intermediate datasets relevant to a given output, reducing reviewer burden and improving clarity.

Future development will extend this framework to Statistical Analysis Plan (SAP) automation. By leveraging agentic AI, structured metadata captured during mock shell creation can be used to draft SAP content and propagate it forward into Data Presentation Plans (DPPs), ARS, ARD, and executable code, maintaining consistency across documentation and execution within a single, traceable ecosystem.

COMPOSER: THE ENGINE BEHIND OUTPUT GENERATION

Composer is the R package that powers Symphony's output creation capabilities. While Symphony provides the UI for building and reviewing mock shells, Composer converts the underlying metadata into executable R code.
For every mock shell created, Symphony captures its structural metadata, grouping variables, denominators, layout sections, column definitions, footnotes, and ordering logic. Composer then uses this metadata to:

1. Generate mock R code that creates a table layout using mock data
2. Accept user-uploaded ADaM datasets
3. Produce final table code aligned to the exact structure defined in the mock shell

4. Ensure fidelity between the approved mock shell and the delivered output

This closes the loop between specification and execution. Where historically programmers manually transcribed mock shells into SAS or R code, Composer automates this translation, reducing risk of human error and saving substantial programming time. As Symphony and Composer mature, both packages will be open-sourced to encourage broader community collaboration, increased transparency, and harmonized practices across organizations.

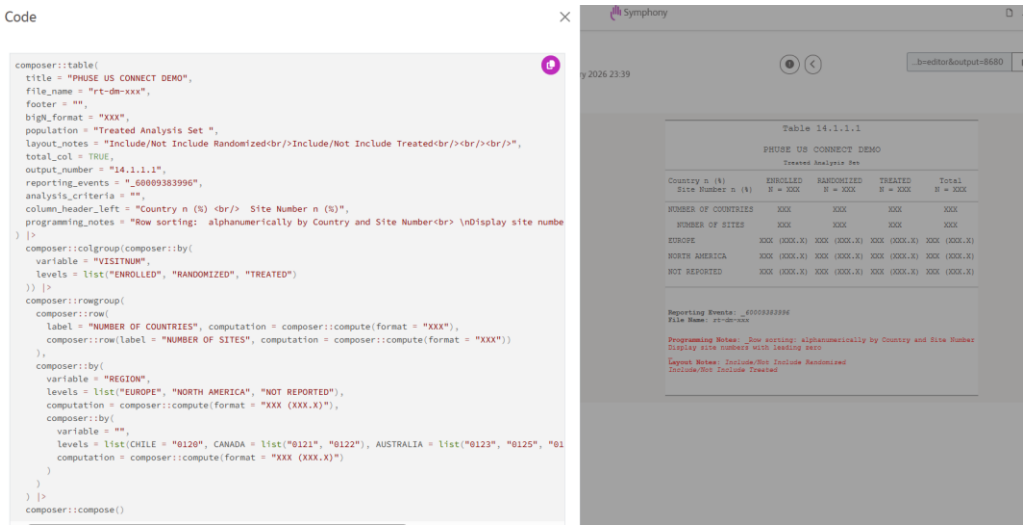

Figure 4: R code driving mock shells

IMPACT

The introduction of Symphony and Composer represents a meaningful shift in how clinical reporting teams design, review, and operationalize mock shells. By replacing historically fragmented workflows with a centralized, metadata-driven ecosystem, Symphony addresses long-standing inefficiencies embedded in mock shell creation and downstream programming.

First, Symphony significantly reduces the manual burden on statisticians and programmers. Tasks that previously required hours of formatting, re-entering specifications, or reconciling feedback across multiple file versions are now automated or consolidated within a single application. Early adoption across studies has demonstrated measurable reductions in turnaround time for mock shell development, enabling teams to initiate programming cycles earlier and with higher confidence in the accuracy of requirements.

Second, Symphony enhances cross-functional alignment by serving as a single source of truth throughout the entire specification lifecycle. Reviewers interact with the same digital artifact, eliminating discrepancies that arise when Word, Excel, and email threads diverge. Built-in validation rules and version comparison tools further reduce interpretational inconsistencies and enable more rigorous, transparent review cycles. This fosters stronger alignment across Biostatistics, Statistical Programming, Medical Writing, and Clinical Operations.

Third, Symphony ensures that the value of upstream effort is carried forward. Structured metadata, once trapped in Word documents, is now usable by downstream systems, including SAS macros, R scripts, and automation frameworks. Composer amplifies this benefit by converting mock shell metadata directly into executable code. This closed-loop integration shortens the development cycle from mock shell to final output and reduces the risk of human translation errors that traditionally occur during programming.

Finally, Symphony supports organizational modernization goals. It aligns with the broader movement toward digital documentation, reproducible workflows, and open-source integration. By unifying standards, metadata, and executable logic, Symphony positions teams to scale more efficiently and adapt to evolving regulatory expectations that increasingly favor traceability, automation, and structured data.

Together, Symphony and Composer reshape not only the efficiency of clinical reporting, but the quality, consistency, and reproducibility of the entire TLF lifecycle.

CONCLUSION

Symphony and Composer demonstrate how intentional digital design can modernize one of the most foundational steps in clinical reporting: the creation and execution of mock shells. By transforming unstructured documents into structured, metadata-driven specifications, the framework enables a level of consistency, transparency, and reuse that legacy tools could never support.-driven specifications, the framework enables a level of consistency, transparency, and reuse that legacy tools could never support.

More importantly, Symphony and Composer redefine how teams collaborate. With a unified environment, shared standards, and traceable metadata, cross-functional groups can align earlier and operate with greater confidence throughout the reporting lifecycle. The ability to automatically translate approved specifications into executable code further ensures that outputs remain faithful to scientific intent while reducing manual effort and programming risk.-functional groups can align earlier and operate with greater confidence throughout the reporting lifecycle. The ability to automatically translate approved specifications into executable code further ensures that outputs remain faithful to scientific intent while reducing manual effort and programming risk.

As both tools evolve and progress toward opensource release, they will help shape a more interoperable and community driven ecosystem for clinical reporting. Their architecture positions organizations to meet rising expectations for auditability, reproducibility, and automation, ultimately allowing teams to focus more on scientific insight and less on manual translation of requirements. Symphony and Composer mark an important step toward a more scalable, modern, and efficient future for clinical data science.-source release, they will help shape a more interoperable and community-driven ecosystem for clinical reporting. Their architecture positions organizations to meet rising expectations for auditability, reproducibility, and automation

ACKNOWLEDGMENTS

We gratefully acknowledge the contributions of John Coene from Opifex, his partnership was essential to the development of Symphony and Composer.

CONTACT INFORMATION

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

Margaret Wishart
Bristol Myers Squibb
3401 Princeton Pike, Lawrence, NJ 08648
Email: Margaret.wishart@bms.com

Karma Tarap
Bristol Myers Squibb
Route de Perreux 1, 2017 Boudry, Neuchâtel, Switzerland
Email: karma.tarap@bms.com