BENEFITS OF DATASET CREATOR (dsC) VIS-À-VIS OTHER RO FRAMEWORKS

1)  The dsC components provide a full-featured but self-contained scientific-computing environment that is re-implemented for each data set.  According to the current RO protocol, code is published alongside data so as to transparently demonstrate how the author uses raw data to reach stated conclusions.  But such published code, taken out of context, is of limited value when it comes to reproducing or verifying research work.  To truly support transparency, RO Bundles should provide a self-contained scientific-computing environment which reconstructs the data models, coding assumptions, and analytic calculations germane to the original work.

Existing technology addresses this problem in several ways.  One approach -- represented by the ReproZip project -- is to identify (via runtime analysis) all library dependencies in a research environment and package them along with the RO code and data itself, to create a duplicate environment as if the end-readers were running software on the same computers as the original authors.  Another option which some publishers have favored is packaging all dependencies into a "virtual image", using technologies like Docker and Kubernetes, which act like a virtual disk partition.

Unfortunately, both of these solutions produce RO Bundles which can be very large and difficult to use.  With dsC, instead, a self-contained computing environment can be engineered in more compact and self-contained fashion, by deploying a customized "Dataset Application" based on Qt -- which enables cross-platform, native-compiled, fully-featured desktop software without extra dependencies -- and distributing analytic and modeling components as Qt/C++ source code.

With dsC, most projects will use the (freely available) Qt platform, providing the basic visual/application code. On that basis, with no extra dependencies, dsC data sets support many powerful features, such as scripting, networking, 2D charts and graphics, 3D/multimedia, embedded web browsers, embedded PDF viewers (for example, to read and link to the original article in the context of a Dataset Application), and Procedural Data Modeling.  Qt is widely used in scientific software -- as the GUI engine for many open-source technical applications in fields like medical imaging, chemical engineering, 3D modeling, physics (e.g. in CERN's ROOT system), etc. -- but Qt is not only useful in large main-frame applications like ROOT; Qt also facilitates self-contained, tightly-focused RO software (a potential that hitherto has not been appreciated by the RO community). This self-contained architecture enables greater transparency.  Low-level scripting and networking details (which would ordinarily be linked as external C libraries) become fully visible as published source code, accessible to debuggers and Integrated Development Environments.  Consequently, RO users never have to worry about third-party libraries presenting version incompatibilities, becoming obsolete, or complicating code certification for legal or institutional compliance requirements.

2)  With dsC, published data sets become visual, interactive supplements to academic books and research articles. Each data set includes a "Dataset Application" -- self-contained desktop software explicitly built for each research project, documenting the researcher’s methods and protocols. This is driven by several features possible in a native application development environment: context menus that can be linked to GUI elements (like buttons, notebook tabs, and column headers) identifying statistical or mathematical parameters within a data model (so readers can get more information about parameters' meaning, scales, dimensions, and locate where they are discussed in research text); coordinating between tabular, 2D, and 3D displays (for instance, showing the same highlighted sample via emphasized text, colored 2D regions, and marked 3D bars, tracking selections across multiple application windows); and organizing GUI layout to optimally explicate data models (leveraging native GUI features like layout managers, splitters, dock widgets, and dialog boxes).

3)  By linking code and data, dsC represents a more powerful kind of technology than solutions which merely disseminate raw data. Scientific data sharing should \*intrinsically\* link raw data to code libraries through which raw data can be processed and analyzed. This maxim influences how information should be described via metadata and semantic models. Unlike the Semantic Web, dsC assumes that formal data has no "intrinsic" semantics outside of the software used to share, visualize, and analyze it. As such, a data sharing platform is only complete if all code needed to process scientific information is automatically bundled with the data itself. In conjunction with dsC we are therefore standardizing a data sharing, representation, and metadata model that reconciles competing "semantics of science" approaches and absorbs structures and theories that have been proposed as alternatives to the Semantic Web: Conceptual Role Semantics; Conceptual Space Theory; Interface Theory of Meaning; Object Oriented Workflow and Simulation; Directed Hypergraphs; Graph-Lambda Calculii. Low-level code in dsC is explicitly designed around these meta-scientific paradigms. By eliminating external dependencies and transparently bundling all requirements as internal rather than external libraries, the code accompanying each data set has a rigor and openness befitting the scientific method.  The RO code becomes a scientific extension of the research itself, not just a tool for disseminating raw data.