The VersatileUX Dataset Creator

Part I Dataset Creator

1.1 *Product Description*

The VersatileUX Dataset Creator is a framework for building standalone self-contained but fully-featured interactive data sets that conform to contemporary publishing standards, such as the Research Object protocol (a format for publishing data sets along with code and metadata) and the FAIR (Findable, Accessible, Interoperable, Reusable) initiative. The Dataset Creator (dsC) framework uses the VersatileUX application development toolkit, which helps developers implement flexible, native-driven desktop-style applications incorporating the valuable features of web applications — like cloud backup, user personalization, and multi-application networking — while avoiding the limitations of web applications, such as relying on a web browser and an always-online network connection. Part I describes dsC, while Part II describes VersatileUX.

1.2 *Benefits of* dsC

dsC will help researchers, scientists, authors, and editors convert raw data to self-contained FAIR Research Object Bundles. In those instances where the raw data was generated with specialized software, dsC will likewise help researchers, scientists, authors, and editors convert data sets into autonomous, cross-platform packages without external dependencies. In so doing, other researchers can access and reuse the published data without requiring the software used to create it. Features of that software will be recreated, as much as possible, within code published alongside the data as part of a Research Object Bundle. Therefore, future users will be able to explore and visualize the published data without needing any additional resources not included with the original Research Object. Data sets that provide their own customized and standalone applications will be exceptionally Accessible and Reusable.

1.3 *What is Unique about* dsC

Whereas all data sets published according to Research Object standards feature a combination of code and data, dsC takes this to a higher level, by providing sophisticated code customized to each data set, for deserializing, analyzing, and visualizing the downloaded data. Data will be delivered in optimized binary files, along with requisite code libraries to interpret them; or, if publishers prefer non-binary formats like XML, requisite domain-specific parsers will be included in the Research Object Bundle. As a result, other researchers will not have to write parsers to reconstruct data in such formats as XML, JSON, CSV, or XLS: these formats are popular for raw published data but require extra programmming whenever researchrs want to extend, rather than just look at, the original research work. Nor will researchers have to install external tools and libraries — components such as NumPy, R, SQL/RDF engines, and spreadsheet applications are common prerequisites for accessing Research Objects, but they violate the objectives of Flexibility and Reusability. Instead, with dsC each code-and-data package will include source code that implements a specialized processing and analysis platform targeted to that specific data set, creating a sophisticated starting point for subsequent exploration and reuse of the data.

Because truly "FAIR" data sets should not rely on external software, Research Object Bundles need to provide their own code ensuring that future researchers will be able to access their data. Scientists may generate tabular data from a spreadsheet application, for instance, but they cannot assume that those who download the data set can simply open the spreadsheets in the same application, or use a different application with similar features. Even when (as with spreadsheets) most users have free software that enable basic access to the data, generic applications will usually offer researchers only a primitive starting point for subsequent programming — formats like XLS are not particularly conducive to implementing new code, compared to shared-object libraries working with native memory.

For these reasons, Research Objects should be bundled with their own code to "deserialize" data files — that is, to read them into computer memory so they can be accessed as native data by C or C++ libraries (or other languages through a scripting runtime). This has the residual benefit that researchers will possess a complete environment for subsequent analysis, meta-analysis, replication, and re-evaluation of the provided data, or new data obtained via similar methods. Compared to generic software — such as spreadsheets or statistical programs — custom-built software will more precisely embody the Research Object's scientific background: the software itself becomes a demonstrative complement and interdisciplinary extension to the published research.

1.4 *Features of dsC Data Sets*

- Conformant to professional standards: Research Objects and "FAIR".
- Almost entirely self-contained: Minimal dependencies on external software or libraries.
- A full-featured platform: Embedded code can include Scripting, Querying, Analysis, Workflow, and Application Integration capabilities, all driven purely by the included code.
- A scientific architecture: The embedded code, via strong typing and code annotations, provides a technical overview and domain specification illustrating the research and methodology shaping the raw data; the code-and-data package thereby makes the code into a logical extension and formalization of the underlying data and the science behind it.
- Text, code, and data integration: Annotations can link code locations and data-points to textual anchors, allowing readers to explore data sets and published articles side-by-side. Although the publications will be compatible with any PDF viewer, data sets can embed code for custom eReaders (PDF and/or eBook viewers) which have built-in support for navigating between data and published text, and have visual tools — crafted explicitly for the data set — embedded directly in the eReader.

1.5 *Unique Features of the dsC Computing Environments*

In order to make data sets truly Reusable, publishers and data editors must anticipate the needs of researchers who will acquire data sets and then use these as a basis for new research. A full-featured computing environment for managing downloaded data sets will access the data in multiple ways, including:

- Interactive Visualization: Visualization in this context does not primarily mean diagrams and graphics, but rather apps and GUI windows which present the textual, mathematical, and graphical or multimedia content associated with each element in the data set in a structured, interactive fashion. Depending on how the data is organized, these GUI windows could have the layout of a key-value form, a tabular display, a collapsible list view (such as those often used for browsing file-folders) or a graphics or textual display with a central focus area. Using QT, typical dsC data sets will include custom designed GUI windows "out of the box" — a feature rarely found in other dataset publishing environments.

- Querying and Scripting: Researchers need convenient tools to unpack data sets in the context of their own computers and file systems, and to filter and analyze different parts of the data so as to explore it in new ways. dsC helps editors create explorable data sets featuring a script-level interface to underlying native (C or C++) code, where researchers can use a scripting environment that can be bundled with the data set as a self-contained package (like Embeddable Common Lisp) or their own customized scripting and query language.

- Formal Data Modeling: The importance of computer tools to work with data sets goes beyond just acquiring and manipulating data. Computer tools express formal models of data structure and the protocols for how data is accessed, explored, and modified. Whether via formal type systems, Semantic Web Ontologies, or protocol specifications, data models explain and codify the technical, dimensional, scientific, and logicomathematical properties of published data and the research processes from which it is derived. dsC addresses these modeling priorities with particular care, ensuring that data sets will be a formal and conceptual extension/summarization of their associated research.

- Application Integration: As researchers expand or further analyze a data set, their work may include integrating the data into other computing environments. dsC ensures that published data-management code is conducive to being integrated with other scientific applications and/or computing workflows — whether via rigorously organized project-specific data types, modular GUI and application components that can be bundled into external applications, or embodying workflow designs so that dataset-specific applications can serve as endpoints for multi-site sharing of data and procedural capabilities.

dsC is designed to balance the technical requirements relevant to all four of these feature-areas. Ordinarily this balance would require some trade-offs: for example, choosing most single programming languages for dataset code can make either fluid GUI design, rigorous data modeling, or script-like querying difficult or impossible, depending on the selected language. dsC, by contrast, provides a code environment where all of these features are rigorously supported.

1.6 *Using VersatileUX Throughout R&D*

dsC provides a scripting interface, GUI visualization, and formal modeling for data sets specifically curated toward publication. However, publishing research is often just the beginning of a project's commercial and scientific life span. The data models formulated when preparing research data for publication will remain useful when Research and Development transitions to managing commercial and/or large-scale deployed products or services.

Preparing Data Sets with ds$\mathcal{C}$ will give researchers a head start toward the computational infrastructure needed to manage large-scale and commercial projects. These data sets will acquire rigorous data models, clearly documented procedural protocols that can guide the development of subsequent software, and a library of GUI and visualization tools that can be the foundation for customized applications. Furthermore, one single platform will combine GUI design, Requirements Engineering, and interface to scientific computing environments — these disparate requirements will no longer be necessarily factored out into separate, non-interoperable frameworks.

Consider the transition from biomedical research to biomedical products such as pharmaceuticals, treatment therapies, medical devices, and bio-cyberphysical sensors and networks. Research describing prototypes or evaluations of these products will typically encompass multiple forms of data: clinical narratives, patient reports, patient histories subject to different time-index schemes, structured forms for data collections (sometimes called data "instruments" in fields like Epidemiology), chemical/molecular information, 3D and/or CAD models, and more. Managing this variety of information demands an integrated infrastructure that goes beyond a generic database or archive. For example, the variety of data forms yields a parallel variety in the kinds of GUI components matched to each kind of data: free-text reports, clinical time-series diagrams/tables for patient histories, multidimensional statistical distributions, biomedical/diagnostic imaging, and molecular visualization all require distinct GUI presentations and user-interaction models. Rigorously preparing data sets for publication ensures that the formal correspondence between data and GUI models is established throughout the lifetime of a project.

Rigorous data modeling also helps researchers document their commitment to responsible use of data and to justify claims made on its warrant. For example, biomedical researchers may need to demonstrate that personal information has been properly redacted from data sets; that sample and parameter construction for statistical analyses is unbiased; or that functional and material designs (of physical products, clinical treatment strategies, etc.) are suitable for deployment at scale. Researchers can garner trust in their material and logistical designs by providing data-management code that represents and implements the logistical structures that must be operationalized for a project to be deployed successfully. Well-documented code can demonstrate both the robustness of software systems and the correctness of physical and/or operational structures that a software component may analyze or simulate as a virtual system (for example, a "Digital Twin" simulating individual or networked appliances and devices).

Computer code that acts on a published data set is, by definition, working with a static body of information that has been curated and recorded for demonstration. The software governing deployed systems, by constrast, has to function properly in real time. Nevertheless, the code within a Research Object Bundle can serve as an initial prototype for real-time management software. For example, CyberPhysical Systems need to be governed by software whose network of functions and algorithms is orchestated to properly interpret and oversee the operations of sensors and activators. Code on the software side must check that the numerical ranges, scales of measurement, and functional coordination of devices on the physical side fall within the space of behaviors estalished, prior to deployment, as safe and proper for the system as a whole. These requirements should be met from the ground up; that is, the functional requirements of low-level segments of code need to be both double-checked via code review/testing and, if the quality of low-level code is assured, the requirements associated with low-level code then become units in the larger functional assessment of the system as an integral whole. Coding requirements and documentation established while preparing Research Object data sets during initial research can then form a scaffolding for subsequent unit and integration testing as the system moves closer to commercial and/or operational deployment.

Data Sets built with dsC will help research projects garner trust overall because their code components will be implemented via paradigms that prioritize code documentation and trustworthiness — in particular, patterns which help individual segments of code serve as an explicit warrant for claims made about code performance and conformance to technical and legal standards. This priority permeates dsC code bases both on a narrow scale — such as, guaranteeing code-conformance via narrowly-constructed data types rather than via procedural code-branching — and on a larger scale, such as a broad use of inter-type associations to link data types representing common information in different contexts (GUI classes, serialization/ deserialization, querying and analytics, etc.).

1.7 *Licensing for* dsC

Scientists, authors, researchers, editors, and publishers can acquire a commercial license to embed novel dsC code in published data sets. Any published code is dual-licensed, with an open-source license for those using published data, and a narrower open-source license for reusing published code in new data sets. Commercial licensees would also acquire from Linguistic Technology Systems, Inc. behind-the-scenes components that are not included in published data sets, such as Cloud Hosting code and Software Development tools. Alternatively, Linguistic Technology Systems, Inc. can provide dataset preparation as a service for individual books and articles or larger-scale journals and book series. Linguistic Technology Systems, Inc. will work with authors to ensure that published data sets are FAIR and professional, and in addition that the embedded code is scientifically sound and expository, presenting data models and processing protocols that concretize each data set's scientific, mathematical, and conceptual background.

Part II VersatileUX

2.1 *Product Description*

VersatileUX is an application-development toolkit that offers new techniques for implementing native-compiled software with the performance of desktop applications and the flexibility and personalization of web applications — using cloud services or multi-application workflows to enhance and fine-tune applications to better serve each particular user's needs.

VersatileUX can help make applications more dynamic and customizable than typical desktop-style applications and also more responsive and visually sophisticated than typical web applications.

2.2 *Benefits of* VersatileUX

In its simplest form, VersatileUX provides GUI component development tools; it then supplements GUI design with Requirements Engineering, protocol modeling, and code verification features.

VersatileUX's philosophy is summarized by our moniker "DyNaMo" — "Dynamic, Native, Modular". To elaborate: GUI Components are *dynamic* insofar as they embed their own scripting capabilities so that they can be flexibly modified to accommodate user preferences while the application is running. Components are *native* insofar as they can be embedded in standalone, desktop-style applications (i.e., they do not need to be "hosted" inside a web browser). Finally, components are *modular* insofar as each component combines GUI windows, data management (deserialization and filtering), and data/protocol modeling; the resulting datatype aggregate can be included in multiple applications as an integral unit.

In addition to providing flexible, responsive front-end development, VersatileUX combines GUI design with the type and protocol specifications suitable to regulate code which must transparently adhere to coding and domain standards in effect for a project. This produces a code base which is optimized for code review and for demonstrations of standards-compliance. By integrating this verification dimension with a GUI library, VersatileUX offers a multi-faceted application-development toolkit that can take the place of several disconnected and non-interoperable frameworks, where for example GUI components use different paradigms and coding practices than standards-conformant data management components.

In the context of dsC, these features of VersatileUX overall translate to sophisticated but self-contained code libraries bundled with dsC data sets, where desktop-style GUI front-ends, data and protocol models, and scripting/querying capabilities are unified into a holistic data-management environment.

2.3 *Components of* VersatileUX

VersatileUX encompasses two areas of code, one server-side and one application-side:

1 NDP Cloud (Native-Driven Platform for the Cloud): NDP Cloud provides server-side code optimized for cloud-hosting and "Container-as-a-Service" environments, such as Docker. NDP Cloud provides a cloud-computing environment adopting libraries and concepts from native application environments, to minimize the paradigm gap between cloud and desktop components. NDP Cloud prioritizes the implementation of services that enable cloud backup, personalization, and network communications between desktop-style applications.

With NDP Cloud, implementing binary data sharing between cloud (server) and application (client) endpoints is straightforward. Based on desktop-style libraries like QT, NDP Cloud encourages code-reuse across server and client components. Written primarily in C++, NDP Cloud ensures that the same programming language and development environment can be employed for both server-side and client-side code. The eliminates the need for two-track engineering where server-side and client-side projects are separate and incompatible. It also eliminates the need to map data back and forth into a "neutral" exchange format like XML or JSON, which improves development time. Furthermore, NDP Cloud services can be developed and tested as an ordinary executable, so that application-specific logic may be developed in isolation from both HTTP request management and from work on server-side deployment.

Data sent between NDP Cloud servers and applications can be used to personalize applications for each user. Users can "log in" to applications to load their personal preferences related to application layout, appearance, and plugins as well as to edit and save files from the cloud. Users can also share files and information via cloud transit points. Moreover, new application features might be deployed via plugins that users would dynamically install from the cloud. These cloud features help applications achieve the goals of “DyNaMo” application-design and “Versatile” User-Experience.

2 **PhaonIR**: *PhaonLib* and *DynamoIR*. These two components represent client-side libraries that can be bundled into native application code.

PhaonLib (Preconstructor/Hypergraph Augmented Pointer Library) is based on memory-manipulation schemes that encode additional information alongside C/C++ pointers. These enhancements serve several purposes depending on the kind of pointer involved:

- Object Pointers: Here pointers to typical C/C++ data structures are augmented with features asserting relations between C/C++ values. This allows pointers to be aggregated into graphs or hypergraphs for projects such as data modeling, knowledge engineering, compiler pipelines, and Semantic Web programming.
- Function Pointers: Here pointers-to-functions are augmented with information that allows functions to be invoked in a scripting or Workflow environment. By initializing function-pointers along with a series of details about how the target function acts upon its arguments, a runtime engine can invoke native-compiled functions from code or instructions that originate outside the compiled C/C++ environment. This makes applications more dynamic, personalized, and versatile because their capabilities can be exercised on cues from web resources (particularly NDP Cloud services) or from other applications (enabling multi-application workflows).
- Preconstructors: An innovative feature of PhaonIR, preconstructors are a form of augmented function pointer that can initialize values, typically indirectly through wrapped constructors. Depending on how they are used, preconstructors support several coding patterns. Used explicitly, preconstructors embody a “factory” pattern that allows value-initialization and/or allocation to be delayed until the values are needed or until contextual data that can affect those values is known. Used obliquely, preconstructors exhibit a “passkey” pattern where they may be queried to ensure that values, prior to their being used, satisfy engineering requirements. Preconstructors may also be used to classify type-specific data constructors (what C++ would call “constructors”) to allow for a kind of functional-style “pattern matching” rendered suitable for procedural programming, where code would branch based on the algebraic structure of type instances. In combination, preconsturctors are a tool for achieving certain functional-programming designs in otherwise procedural or Object-Oriented contexts.

Meanwhile, **DynamoIR** (DyNaMo Intermediate Representation) is a hybrid QT/Lisp code format. Specifically, DynamoIR code is Common Lisp code that works with a collection of QT classes and methods exposed to a Lisp runtime via the QT Meta-Object Protocol and via a foreign-function interface, established, for instance, by initializing the Lisp runtime within a QT application via Embeddable Common Lisp. DynamoIR is primarily used to build descriptions of function calls whose conditions and parameters are derived from dynamic contexts (like script, query, markup, or source-templated code, networked resources, or application plugins). DynamoIR enables rigorous description of all “Channels of Communication” employed by target functions, including functional side-effects, to both identify the proper target function and to check that the proposed function call meets all operational and security requirements.

DynamoIR is tightly integrated with PhaonLib insofar as DynamoIR code constructs representations of function-calls that are then matched to augmented function pointers. Moreover, both the DynamoIR format and PhaonLib function pointers are tightly integrated with NDP Cloud insofar as NDP Cloud services provide resources that translate or compile to DynamoIR code, allowing native application functions to be remotely targeted by scripts, plugins, or workflows represented in the cloud. These interrelationships provide the key architecture that can make native applications using VersatileUX more responsive and collaborative.

2.4 *Generalizing VersatileUX to Different Commercial Environments*

The goals and design of VersatileUX are influenced by projects such as Clasp (a Lisp dialect prioritizing scripting access to scientific computing environments, originally developed in the context of chemical analysis), Idris and Ivory (type-enhanced dialects of Haskell prioritizing specifications verification and Requirements Engineering), and QJSEngine (for QT scripting). The long-term goal of VersatileUX is to combine the benefits of these various platforms so that GUI Development, Requirements Engineering, and Workflow Management (or interfacing for scientific and High Performance Computing environments) can be unified into one development platform.

As explained above, one current project based on VersatileUX is the VersatileUX Dataset Creator, which is geared toward academic and technical publishing. dsC functions with deliberately curated data sets; as such, it allows VersatileUX components to be developed in a controlled environment. The business strategy of Linguistic Technology Systems, Inc. is to use dsC to prototype and stress-test VersatileUX components, leading toward more general solutions that provide specifications, analysis, and visualization for data sets and code-base management in many other contexts — such as biomedical, industrial, social sciences/linguistics/digital humanities or CyberPhysical/CyberSecurity — rather than focusing solely on data sets that have been (or are being) prepared to accommodate publishing standards and goals.

2.5 *Licensing for VersatileUX*

Commercial licenses for VersatileUX center on NDP Cloud. Licensees can work with Linguistic Technology Systems, Inc. to develop custom cloud solutions based on NDP Cloud — or can license and then adopt for their own purposes a core server-side "backbone" that could be used as a Docker container or adopted for other server environments. The NDP Cloud code base is currently smaller than the other two main components (PhaonLib and DynamoIR), but it allows those components to be used in a multi-application networking context and/or a client-server architecture rather than just within a single executable.

The PhaonIR components themselves can be used subject to various licenses. At least some PhaonIR related code in any given project will typically take the form of web resources which users can examine, if desired, so that particular code inherits licensing similar to other web formats like CSS and JAVASCRIPT. For similar reasons, the kernel code (such as low-level pointer-manipulation algorithms) that is common to all applications using PhaonIR will in some cases (particularly in the context of dsC) be included in applications distributed in source-code-only fashion. However, a commercial VersatileUX license would allow developers to close-source application-specific PhaonIR code (such as routines that construct tables of augmented function pointers for a DynamoIR runtime).

Linguistic Technology Systems, Inc. can also set up and (if desired) maintain a containerized NDP Cloud server instance and/or can design a scripting and workflow interface, compatible with PhaonIR, layered on top of an existing C/C++ code base.