

Pervasive® DataRush™  
A Description of the  
Highly Parallel Dataflow Framework

---

A Pervasive Software Architecture White Paper  
July 2008

## TABLE OF CONTENTS

<b>AUDIENCE</b> .....	3
<b>THE NEED FOR A DATAFLOW FRAMEWORK</b> .....	<b>3</b>
THE BUSINESS OF IT IS CHANGING .....	3
HARDWARE VENDORS ANSWER THE CHALLENGE .....	3
THE ENTERPRISE JAVA COMMUNITY LACKS AN ANSWER .....	3
<b>HOW DATAFLOW FRAMEWORKS FIT TO INDUSTRY TRENDS</b> .....	<b>4</b>
<b>DATAFLOW FRAMEWORKS IN THE COMPUTING AND DATA INTEGRATION INFRASTRUCTURE</b> .....	<b>5</b>
<b>PERVASIVE'S DATAFLOW APPROACH—PERVASIVE DATA RUSH</b> .....	<b>5</b>
<b>PERVASIVE DATA RUSH: THE NEXT LEVEL DOWN</b> .....	<b>6</b>
<b>PERVASIVE'S DESIGN-TIME EXPERIENCE</b> .....	<b>6</b>
<b>PERVASIVE DATA RUSH AND PERVASIVE DATA PROFILER™</b> .....	<b>8</b>
<b>SUMMARY</b> .....	<b>8</b>
<b>APPENDIX: J2EE VS. DATAFLOW FRAMEWORKS</b> .....	<b>9</b>
CONTACT/TRADENAME INFORMATION .....	10

## AUDIENCE

This architecture paper is written with the systems architect or Java developer in mind. It is intended to provide a high-level overview of the highly parallel data-processing framework from Pervasive Software.

## THE NEED FOR A DATAFLOW FRAMEWORK

### THE BUSINESS OF IT IS CHANGING

The data requirements of many industries are quickly expanding and growing more complex. Several emerging industry trends point to this. First, CRM, ERP and BI data warehouses and enterprise records management (ERM) data storage devices face data proliferation as companies and organizations comply with regulations such as Sarbanes Oxley, Basel II and others. A second trend is a growing data management problem inside the IT operations of business process outsourcers (BPO) and business service processors (BSP) – these “data hubs” of multinational corporations are under tremendous pressures to process and synthesize more and more information less and less time.

Another equally important trend is the requirement of business managers to receive real-time, or near real-time, business intelligence on their customers, supply chain and operations – management is no longer satisfied with monthly or weekly reports. This emerging trend is known as Complex Event Processing (CEP). It includes the collection and analysis of real-time or near-real-time events. These events span the spectrum from stock market ticks to network device logs. They are also known as “events in motion.” But, CEP encompasses not only real-time event processing but also the historical analysis of the events. The ever-increasing capacity and ever-decreasing cost of storage devices has led to an explosion in the storage of these events, whereby they become “events at rest.” Once stored, the events can be analyzed in a more detailed and iterative fashion for higher quality correlations and deeper learning.

### HARDWARE VENDORS ANSWER THE CHALLENGE

The hardware industry is addressing these data trends, particularly the slowing acceleration of processor speeds due to heat dissipation issues, the demand for lower-cost, high-powered systems and the need for parallelism. CPUs are moving from “single core” to “multi-core” processors as witnessed by the use of AMD Opteron™, Sun Niagara and Intel Xeon® processors, and the wide-spread development of commodity multi-core SMP servers.

The reality is this: the “Big Iron” once only found in mainframe shops is being pushed down to departmental servers and even lower. Server consolidation and virtualization (e.g., VMWare) also are taking hold – but not on mainframes. Increasingly, data centers are using lower-cost, mid- and high-end UNIX/Linux and Windows® systems based on 8, 16, and 32-core SMP servers.

### THE ENTERPRISE JAVA COMMUNITY LACKS AN ANSWER

The software industry, meanwhile, must help facilitate the efficient use of multi-core machines. New processor advances are wasted if applications are written in an unstructured fashion leaving the OS or Java™ Virtual Machine (JVM) to “guess” on how to use parallelism to increase performance. For example, in the case of Java-based server-side technologies, any new concurrent programming framework must provide the ability to solve the large data set problem (“large” here ranges from many gigabytes to many terabytes) and provide for the scalability of data-intensive business applications. A dataflow framework provides the ability to solve these types of data processing challenges.

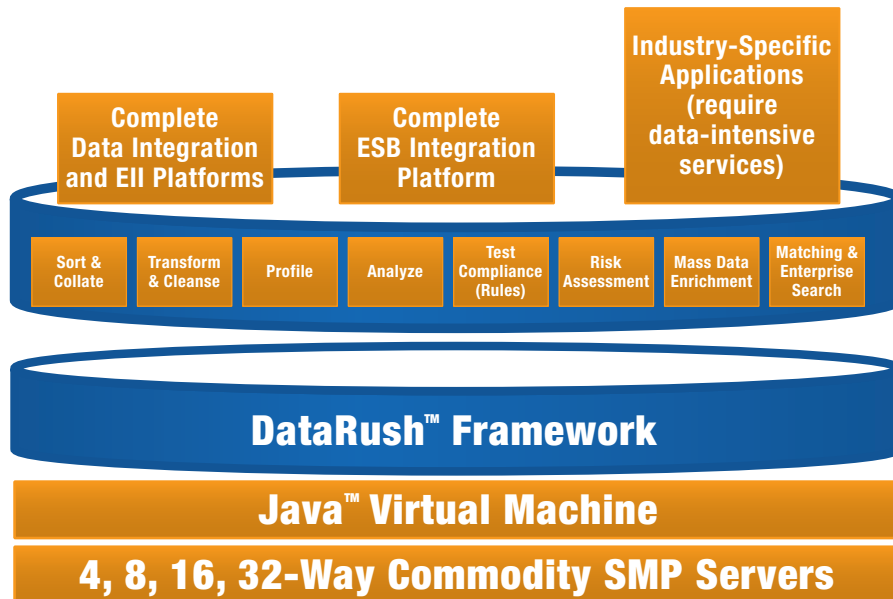
## HOW DATAFLOW FRAMEWORKS FIT TO INDUSTRY TRENDS

The use of J2EE™ application servers to power Web-centric OLTP (on-line transaction processing) applications has been immensely successful. However, the entire architecture of an application server is made to serve short-lived transactions, not data-intensive, computationally intensive, long-running back-office applications.

A dataflow framework can address such a scenario. The application types suited to a dataflow framework rather than J2EE architecture are seen in Figure 1 and include risk analysis, matching, sorting, compliance checking and general data transformation services for SOA. The study of dataflow engine technology has been going on since the 1960s, but has never been productized in a 100% Java server-side framework until now. In order for the dataflow framework to succeed, a dataflow engine must be scalable in several dimensions and the solution design must be simple and reusable. Additionally, the ability to leverage computing resources must be highly refined and advanced, including the ability to take advantage of parallelization points within a dataflow graph on a multi-core system. And the dataflow framework must handle very large data sets resulting from the previously mentioned industry trends. In order to work effectively, a dataflow framework also must be extensible with a lightweight, POJO-like (“Plain Old Java Object”) component framework and must be 100% Java to provide hardware and OS platform portability.

Of equal importance, a dataflow engine should be embeddable into both business and scientific research applications. It must have a small footprint and have a relatively small but powerful API to reduce complexity and increase developer efficiencies. See the Appendix for a discussion about J2EE vs. dataflow frameworks.

Figure 1. Application types suited for the dataflow framework



## DATAFLOW FRAMEWORKS IN THE COMPUTING AND DATA INTEGRATION INFRASTRUCTURE

Dataflow engine technology will power data- and computationally intensive enterprise applications, ranging from ETL to profiling/analyzing enterprise data to rules compliance. It is a highly dynamic and robust technology that seamlessly handles large data feeds with records requiring parsing, processing and transformation to disparate data stores.

The following types of infrastructure software benefit from the concept of a dataflow engine:

- **ETL:** Dataflow engines should be the underpinnings of any highly scalable Extract, Transform and Load (ETL) toolset. Many of the operations in ETL tools lend themselves to parallelism such as sorting, grouping and data transforms at the record and field level.
- **Data Integration:** Both real-time and batch data integration suites can be complemented by dataflow engine technology to ensure complex transformation and auditing processes can execute in the most efficient manner.
- **EII:** The dataflow engine aids Enterprise Information Integration (EII), or the ability to pull large, disparate datasets from across the enterprise in real time to build composite information views. This type of operation requires highly scalable SMP server horsepower and software that can take advantage of multi-core architectures. EII leaves the source data in its original form and location, but must centralize query results in a real-time cache and process the data for final delivery of the composite result set. In contrast, ETL technology physically moves the data and permanently writes it to a new storage location.
- **ESB:** Enterprise Service Buses (ESBs) integrate business processes and data in real time. The problem is that most data from legacy platforms comes in binary or unstructured formats and most ESBs require the data in XML form. So real-time, binary to XML transformations are a computationally heavy task that would bring most ESB nodes to their knees. The dataflow engine can be delegated “Binary to XML Transform” tasks on larger SMP servers to offload this burden.
- **Enterprise Search:** Gathering word processing, spreadsheet and presentation documents and indexing them is one part of enterprise search. But what about the terabytes of structured data in ERP, CRM, data warehouses, data marts and customer data hubs? A dataflow engine can perform the massive data integration and processing tasks necessary to load enterprise search indexing systems.
- **Complex Event Processing:** Streaming events are being captured in real-time and stored. Analyzing massive quantities of these historical events for usage patterns, fraud detection and other data attributes reveals previously hidden information.

## PERVASIVE’S DATAFLOW APPROACH – PERVASIVE DATA RUSH

Pervasive’s approach to developing a dataflow framework hides the complexity of parallel programming from the Java developer/architect, and uses a POJO-like, component-based approach that “keeps things simple.” So dataflow, at its essence, is the high-speed flow of data through the input and output “ports” of a collection of computational operators. Pervasive DataRush handles locking, threading and dynamic disk caching when memory boundaries are reached – keep in mind that these all-Java operators require no locking or threading code.

Pervasive DataRush offers dynamic engine scalability. At startup, the dataflow engine will take advantage of CPU resources at that point in time, dynamically modifying its “plan of attack” as resource availability changes (such as which dataflow operators to make parallel and how many concurrent threads to spawn). This enables the engine to not only scale up, but also scale down to low-end servers such as small department-level, Windows-based systems.

Composition of dataflow graphs (applications) within Pervasive DataRush is accomplished using a simple Java-based API as seen in figure 2 (page 7). Once a dataflow graph is composed, it can be executed using the Pervasive DataRush execution environment. The execution of a dataflow graph is done via a Java API. The execution environment utilizes the Java Management Extensions (JMX) capability of Java to expose information useful for profiling and debugging of DataRush-based applications. A plug-in for JConsole, the JMX event viewer shipped with Java, is provided, allowing the end user to utilize JConsole to view extensive DataRush run-time information in a standard way. JConsole also includes system and JVM-level information integrated into one easy-to-use console.

## **PERVASIVE DATA RUSH: THE NEXT LEVEL DOWN**

Pervasive DataRush is a 100% Java SDK and supports the Java 6 environment. It also offers wide platform availability (including 32-bit or 64-bit JVMs). A sophisticated design time framework is part of Pervasive's framework, with Java-based dataflow composition, reuse of POJO-like operators and flexible "port linking" which is the ability to link the output fields (for example, Customer Name) to input fields of the next component. The run-time engine is lightweight. Pervasive DataRush comes with a simple Java API for controlling engine-level invocation. In addition, a Windows or UNIX-style command line operation is supported.

During composition, the operators support the dynamic creation of additional dataflow graph elements to increase scalability on the run-time platform. For instance, this allows a composition element to decide at run-time how many horizontal partitions to create to increase scalability. This allows the component to be very flexible as far as the compute environments it supports. The same graph can be run on a small machine for testing and then be deployed on a larger machine for production work. The Pervasive DataRush application does not need to be modified or rebuilt before deployment. The application will adjust to its run-time environment due to the built-in composition flexibility.

At execution time, Pervasive DataRush handles instantiation of a dataflow graph by creating the underlying threads, memory queues and other control structures. Queues are linked as defined by the dataflow graph composition, and the graph is started by executing the threads.

The execution environment performs thread monitoring and dataflow deadlock detection so that the Java developer does not need to undertake complex, specialized Java parallelizing development tasks. Again, the components are POJO-like with few interfaces that must be implemented for the component to blend into the framework.

## **PERVASIVE'S DESIGN-TIME EXPERIENCE**

Pervasive DataRush, being all-Java, is built to enable Java developers to quickly create scalable, data-intensive applications. The composition interface is simple to use and very flexible. It allows the developer to use the Pervasive DataRush component library and extend the library as needed with additional operators. Several example applications are included to demonstrate the ease with which applications can be created using Pervasive DataRush.

Being Java-based, DataRush integrates well into popular Java IDEs, including Eclipse and NetBeans. The developer can take advantage of all of the useful IDE capabilities they have come to know and love, such as code completion, integration with Javadoc, execution, debugging and profiling.

Pervasive DataRush's component library is extensive, containing Pervasive-developed dataflow operators that are customer extensible. There is a foundational library including I/O, sort, merge and join, among other components. At runtime, each operator becomes inherently parallel through something Pervasive terms "dynamic composition."

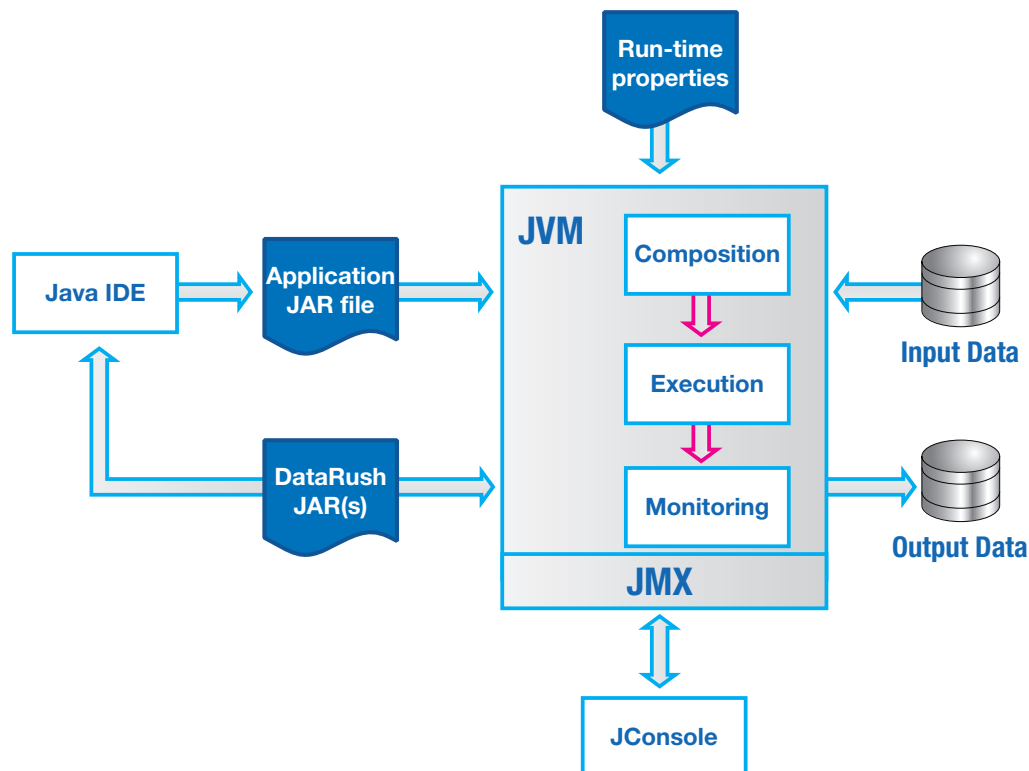
Composition is the phase during which a dataflow graph can be constructed. The construction takes place by creating an application graph and adding components to the graph. Each component can have multiple inputs and outputs. Each component is represented as a Java class whose constructor defines the composition time information needed to instantiate the component. Components that require an input will take an input dataflow as one of their constructor parameters. Components that produce outputs will expose “getter” methods that return their output dataflows. Using these composition standards, a graph can be produced by handing the output flow of one component to the input of another. Once a component is instantiated, it is then added to the application graph and given a name. The name is useful for human consumption within the monitoring application, a DataRush JConsole plug-in.

When a component is instantiated by creating a new Java class instance, the component dynamically composes itself. The component creates a sub-graph that implements its functionality and will be wired into the higher-level application graph. During component creation, the component code can dynamically decide how to construct itself, based on the configuration of the machine on which it is running. Resource availability such as memory and CPU cores can affect the component graph creation. For example, a component may support horizontal partitioning of data to increase scalability. At composition time, the component decides dynamically how many partitions to create. This can be based on run-time properties passed to the component or on the number of configured CPU cores.

Once an application graph is complete it may be executed by simply invoking the run() method on the graph. Several run-time configuration options are available, such as whether or not to monitor the application (using JMX), logging configuration and default queue and buffer sizes. An application can be run synchronously or it may be run in the background using a future construct to obtain status.

The Pervasive DataRush framework is available for download at <http://www.pervasivedatarush.com>. The download includes full Javadoc of all APIs and sample applications with source code.

**Figure 2. Pervasive DataRush Architecture**



## PERVASIVE DATA RUSH AND PERVASIVE DATA PROFILER

The Pervasive DataRush engine is incorporated in Pervasive Data Profiler. Pervasive DataRush serves as the starting point to build the engine that processes data rapidly, meeting the real-time needs of business management and scientific researchers. In addition, Pervasive DataRush handles all the source data connectivity needs of Pervasive Data Profiler.

### ABOUT PERVASIVE DATA PROFILER

A shipping product for over a year; currently at version 4.2

- Profiles customer data set; provides profiler report
- Utilizes Pervasive DataRush as data processing engine
- Provides a UI construct on top of Pervasive DataRush
- Generates a profiler descriptor according to user preferences
- Converts profiler descriptor to a dataflow graph
- Executes the dataflow graph using the Pervasive DataRush engine
- Pervasive Data Profiler uses DMS, a Java wrapper over Pervasive Data Integrator™ C++ 32-bit connectors for a wide variety of connectivity options
- Provides huge functionality boost in breadth of data reach

### SUMMARY

Pervasive DataRush fills the need for the data-intensive, multi-threading business logic required to make full use of multi-core SMP hardware. The framework seamlessly provides this logic through a 100% Java solution and meets the requirements for large data feeds. It can be easily embedded and allows wide-ranging source and target connector development, as well as component development providing “custom” data transformation with general processing logic.

## APPENDIX: J2EE vs. DATAFLOW FRAMEWORKS

Let's start first with what an application server provides:

- A container with aspect-oriented capabilities providing highly configurable security, transaction support, database interaction support and communications support ... basically all of those things needed by most client/server types of projects
- Distributed transactions (clustering capability for scalability)
- A framework for defining object life cycles, threading models, multiple interaction policies (session beans, stateless session beans, entity beans, message-driven beans) and transaction dependencies

The types of applications J2EE is targeted at:

- High-level business transactions
- Web applications
- Web services

This inherently gives J2EE the following nature:

- Heavyweight, high-granularity business transactions
- A thread is allocated per business transaction
- A DB connection may be allocated per transaction
- At least one or more objects controlled by the framework per transaction
- Single-threaded transactions
- Heavy limitations on what can be done within a transaction (container contract requires the bean to live within a "box")

Application servers are targeted at business transactions that may be distributed since they commonly use a central database repository for OLTP-type data. A business transaction is typically a unit of work that requires accessing and possibly modifying multiple tables within a database or even across databases. The transactions are usually discrete events that are independent of each other. This helps scalability as the transactions can be distributed.

However, when a developer wants to run a data-intensive application with an input of billions of rows, an application server is not the engine of choice. The overhead per record or transaction is much too high, the "events" are not discrete because they probably have many dependencies (think of aggregation) and each "transaction" may or may not involve a database. In this case, a highly scalable data processing engine is needed: one that enables parallel processing of the data, managing data dependencies and enabling a wide variety of connectability.

One of Pervasive's key BSP customers got it right. They use an application server farm for "control" processing. When a new input file arrives from a customer, a control event hits their network through a Web service, goes through an application server (IBM WebSphere®) that writes a database transaction to denote the new file and its state. Then a data processing server (the Pervasive DataRush engine) is invoked to do the heavy data lifting. This data processing may involve updating a database, but it usually is more of a bulk extract/load or bulk update, not OLTP. Once the heavy data lifting is done, the application server gets involved again to "push" the business transaction on its way (which may include more heavy data processing down the line). This is a good mix of "control" and "data" processing using the right tools for the job.

So both server frameworks, an application server and a data processing server, are needed as they provide very different capabilities. They can be used in concert to implement business processes that enable the handling of millions of records of business data in a highly scalable fashion.

## Contact Information

Pervasive Software Inc.  
12365 Riata Trace Parkway, Building B  
Austin, Texas 78727

### United States

800.287.4383  
512.231.6000  
Fax: 512.231.6010  
info@pervasive.com

### EMEA

+800.1212.3434  
cic@pervasive.com

<http://www.pervasive.com>  
<http://www.pervasivedatarush.com>

---

©2008 Pervasive Software Inc. All rights reserved. All Pervasive brand and product names are trademarks or registered trademarks of Pervasive Software Inc. in the United States and other countries. All other marks are the property of their respective owners.