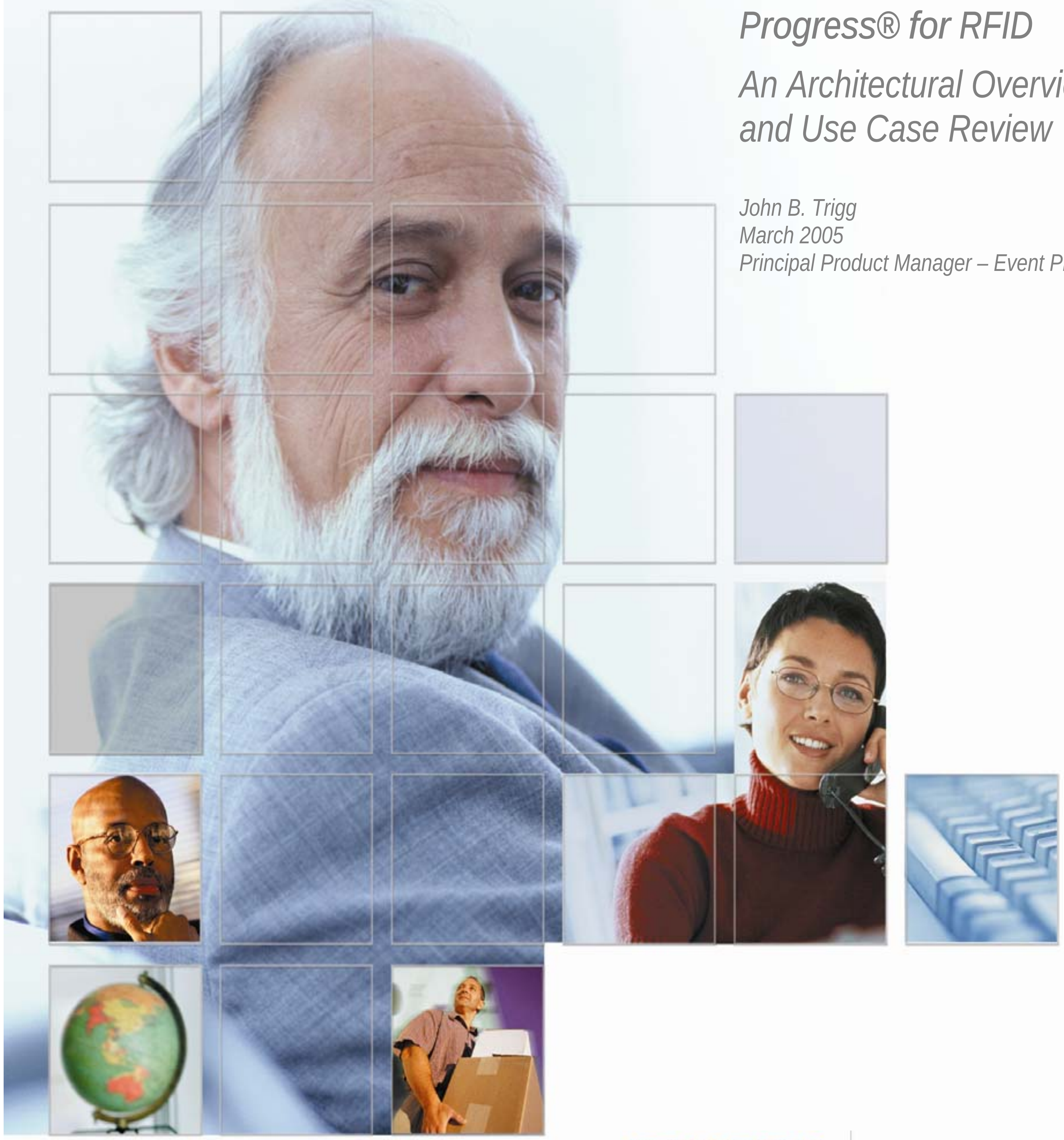


# Progress® for RFID

## An Architectural Overview and Use Case Review

*John B. Trigg*
*March 2005*
*Principal Product Manager – Event Processing*

PROGRESS SOFTWARE | Real Time Division

## *Table of Contents*

## Executive Summary

Traditional software applications used to track an inventory of assets, whether products, containers, vehicles, equipment or people, through the assets lifecycle, are being challenged to not only accept RFID as a new form of data capture, but also to track those assets at the most atomic level – the individual unit. As the vision of RFID becomes reality, as costs and environmental barriers to successful deployment of the physical architecture are eliminated or obviated, so the ability to truly get to an 'internet of things' will get closer. The ability to tag, track and interrogate each asset, via a simple scan of a unique serial number (EPC identifier) will require the need for enterprise information architectures to be able to store and retrieve everything pertinent to that serial number

Beyond the discrete transaction occurrence, we must look at how the continuous stream of RFID related EVENTS can provide meaningful insight, both real-time and historically, into performance, trends and exceptions. Traditional software approaches fail due to the nature of treating each entry by itself and not having the capability, without the aid of expensive supercomputing power, to gather, apply context, analyze and/or aggregate and act upon granular level data, such as that RFID promises.

This paper proposes an evolution in the thinking and processing of business transactions to match this new challenge in the management of RFID data. Providing an ability to look at EVENTS both as discrete occurrences and as related bodies can trigger new means of managing the data generated from RFID readers. This is known as Complex Event Processing.

One solution to these challenges is the introduction of Progress® for RFID. As a solution built around the framework of Complex Event Processing, Progress for RFID brings a scalable, rapidly deployable infrastructure that extends existing applications into an RFID-enabled enterprise.

## The RFID Data Management Challenge

Traditional software applications were designed to track asset inventories throughout the asset lifecycle. However, regardless as to whether those assets represent products, containers, vehicles, or equipment, these applications are increasingly challenged to accept RFID as a new form of data capture and to track these assets at their most atomic level – the individual unit. With costs trending downward and environmental barriers eliminated or obviated, the ability to reach an 'internet of things' continues to get closer. But the ability to tag, track and interrogate each asset via a simple scan of a unique serial number (e.g. an Electronic Product Code [EPC] ) will require adaptation of enterprise information architectures to store, retrieve and make decisions on the data pertinent to each serial number.

Existing business applications understand and consume discrete transactions (e.g. a purchase order receipt of 30 cases of ABC). RFID changes the dynamic. Now those applications must digest a continuous stream of EVENTS that can provide meaningful insight, both real-time and historically, into performance, trends and exceptions. So is there a need for an evolution in the thinking and processing of business transactions to match this new challenge in the management of RFID data? Is there an ability to look at EVENTS both as discrete occurrences and as related bodies that can trigger new methods of managing the data generated from RFID readers? And can important contextual data behind each event be made available in real time to allow sophisticated rule processing that can introduce or improve critical business decision points?

One solution to these challenges is the introduction of Complex Event Processing.

## Complex Event Processing: A New Computing Style

Any current business application that is exposed to a network of RFID readers, each of which is capturing tag IDs from many products and containers, will be required to handle event streams. These event streams will contain some meaningful events, but also many events of little relevance to the core processing of the application. The event streams will also contain events that are reported in error or which contain errors. Managing this inbound data will require implementation of a processing tier that can handle such event streams, filter the meaningful from the irrelevant, and generate a more concise and pertinent set of events. That processing operation is one of the core principles of complex event processing. For, it is not the events in a complex event processing operation that are complex in nature. Rather it is the ability to derive meaning from an event by understanding the event – within the context of other events as well as other forms of business data – that provides the "complexity" of complex event processing.

Furthermore, while event streams are often rapid, that should not imply that they are necessarily transient. Complex event processing must also recognize that not just the current event stream determines current state or activity. Event processing engines must have access to both current event streams and a persisted history of events in order to measure, compare and contrast activity over time.

## An Event Processing Architecture for RFID

Above the standard RFID architecture of tags, readers and ALE middleware, a successful, scalable CEP architecture consists of:

1) Event Engine

    a) Once events move through the collection phase, they can be processed, according to type. The event engine executes event pipelines that can apply the various forms of filtering, aggregation and transformation according to event type.

    b) A pipeline is a programmatic structure made up of individual components. Each component performs some form of processing (gathering, filtering, grouping, transforming) on an event and streams its results to the next component. A pipeline can consist of a single process or multiple processes each performing a single step in a complex transformation. The result is a high performance, adaptable event processing construct called a pipeline.

2) Event Database

    a) A repository for events. Rather than allow events to be simple transitory entities, the event engine has the ability to record each event, either in its raw form or in a processed form, in a scalable, distributable event database. This database forms a historical repository for event processing and for external applications looking to monitor and measure event processing over time and against standard metrics.

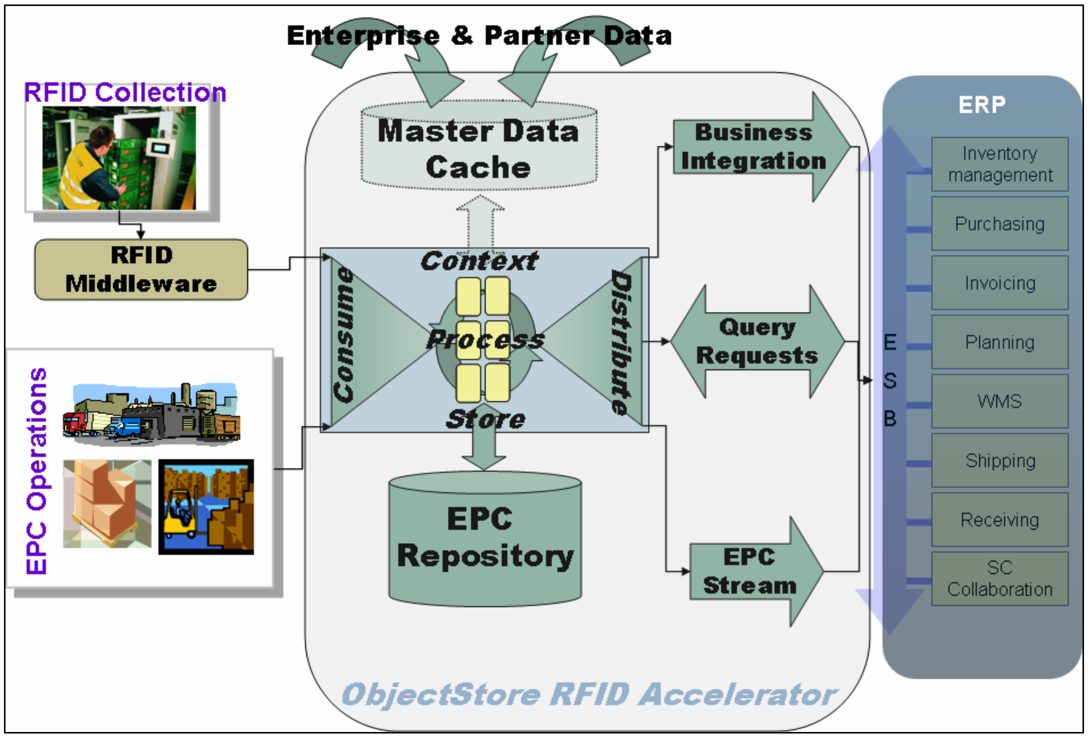
3) Event Server

    a) Provides a flexible interface to the event engine, allowing integration of different asset coding standards (GRAI vs. SGTIN vs. UCC-128), and event query tools. Through the event server, the event engine can be used to process not only RFID feeds and serve a single set of RFID-driven applications, but also feeds from standard bar code readers, database file/table-driven inputs and provide data that can be syndicated across multiple applications via a JMS subscription model.

4) Event Caching
   a) The event cache provides the real-time access to a host of client applications seeking to query the event database. The cache architecture allows for many models of deployment (point-to-point via queues or topic-based publish and subscribe) depending on the required information systems integrations.
5) Event Query Language (EQL)
   a) The EQL provided with the architecture will allow external applications to retrieve data from the event database. It must provide a standard interface for all calling applications to ensure consistent use of the event history.
   b) For RFID, the typical queries must support lists of tags passing a named reader in a given time period, a list of readers through which a particular tag has passed within a given time period, and counts of events pertaining to a particular tag, reader, organization or time period.

Progress® for RFID is an implementation of the event processing architecture that delivers a scalable, adaptable framework with which to manage the challenges of an RFID-enabled enterprise. The Progress for RFID is built on the Progress® Event Engine. The power of Event Engine allows Progress for RFID to deliver a scalable, adaptable framework with which to manage the challenges of an RFID-enabled enterprise.

Figure 1 – Progress for RFID



## Progress for RFID

### Progress Event Engine Architecture

The Progress® Event Engine provides a high-performance, scalable solution for collecting and processing large volumes of real-time event data. Event Engine enables you to collect events from a data feed and perform real-time analysis on the events as soon as they have been collected. Because of the critical nature of collecting events from a live feed, Event Engine has been optimized to collect tens of thousands of events per second and to ensure that the collection process is uninterrupted. All collected events—including the most current—are immediately available for processing. Event Engine handles data management transparently to the user, allowing the developer to focus on the event-processing logic. Included with Event Engine is a framework that enables the application developer to:

⇒ Define the events and the processing logic required by the application using XML.

⇒ Set up the environment that enables the Event Engine framework to execute and manage event processing.

The first of these definitions is the EVENT definition. As well as giving a name to the event type, the event definition also instructs the event engine if these events are of a transient nature – that is the actual event does not need to be persisted. Transient events are used where some extrapolated value relating to the event type is required (e.g. the total number of tag reads) rather than the actual event itself. If not transient, an event is considered persistent and can be stored in the Event Engine database.

When persisted to the Event Engine database, there are three important characteristics that each event carries.

1. A **category** that enables an event to be grouped with other, similarly related events. Categories allow queries to select groups of events for analysis and processing. Examples of categories include the item number of a product being tracked or the ID of an RFID reader that is recording receipts in a warehouse.
2. An **ordering value** that enables Event Engine to position an event relative to other events within the same category so that its position is known. Ordering allows you to select a set of events within a range of ordering values. In general, events are ordered by time, forming a time-series.
3. **User-defined data** of interest to the application, such as a GPS co-ordinate or temperature of the article that is tagged.

*Event Definition*

```
<?xml version='1.0' ?>
<EventDefinitions namespace="SensorBundle">
      <Event name="SensorEvent" type="TimeCategorizedEvent">
            <Member name="sensorID" type="Int32">
                  <FieldAction type="usedForEqual"/>
            </Member>
            <Member name="temperature" type="Int32">
            </Member>
            <Member name="humidity" type="Int32">
            </Member>
      </Event>
</EventDefinitions>
```

To process events, the Event Engine framework provides a means of stringing small, well defined processing modules together to form a single, scalable execution framework. The EVENT PIPELINE is a run-time construct that drives all work in Event Engine. As a programming tool, the pipeline is well suited for performing a series of operations on a large volume of homogeneous data items. In UNIX shell programming, for example, pipes provide a simple, flexible, effective way to connect a series of commands for processing text streams. Similarly, the Event Engine pipeline can capture stream events, pipeline them through different stages of processing, and output the results.

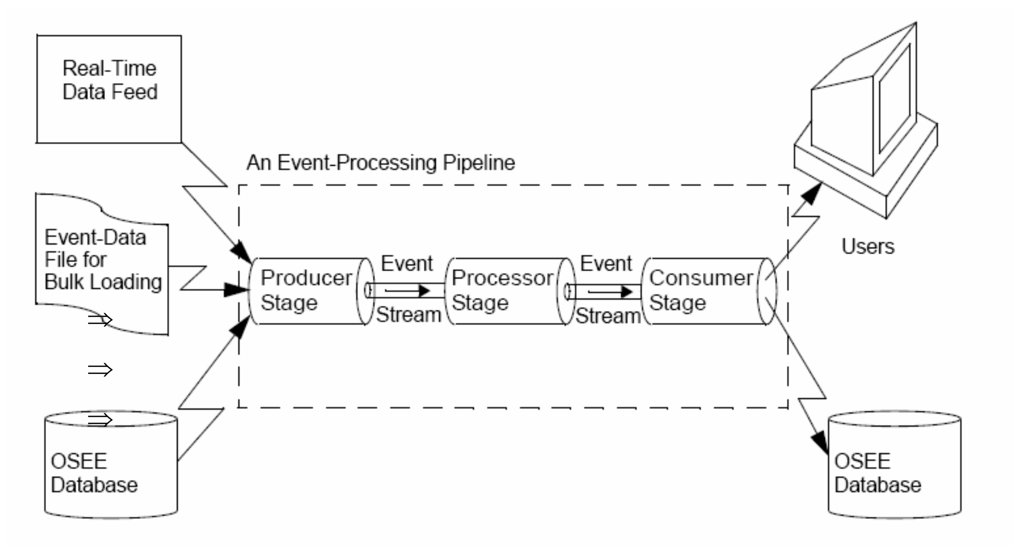


Figure 2: Baseline Event Engine Architecture

In Progress for RFID this model is used in two modes:

1) To capture events from an ALE middleware layer, refining the RFID reads with filtering, aggregation, grouping and staging to business applications that will execute updates on the refined event data.
2) To process query requests from external applications against the events stored within the Event Engine database.

The event processing pipeline is made of stages that are designed to be small units of process and control. The results of each stage flow from one to another as an event stream. The table below describes each of the pipeline stages.

| Stage | Description |
|---|---|
| Producer | In the producer stage, the pipeline produces events and places them on its output stream. At this stage, the pipeline captures external events, (e.g. those that originate from an external data feed—for example, a NYSE market feed or RFID reader or ALE middleware). It can also retrieve existing events from a database, file, or some other container. When retrieving events from a database, the producer stage can apply filtering so that only events of a specified category or within a specified interval are placed on the output stream. |
| Processor | The processor stage has both an input stream, consisting of events that have been output by the producer stage, and an output stream, consisting of any events resulting from the processor stage that must be transferred to the consumer stage for additional processing. The kind of processing that can occur at the processor stage is mainly determined by what the application wants to do with the events. Multiple processor modules can be chained together in the processing stage for more advanced processing without compromising modularity. A processor stage is optional in a pipeline definition. |
| Consumer | The consumer stage consumes the events produced by the previous stage (either producer or processor) and transfers them out of the pipeline or |

| | |
|---|---|
| | persists them in a database. Any processing that is required to prepare the event for transfer to the external world occurs at this stage. For example, if a pipeline were part of a subscription service application, the consumer stage would propagate result events to a set of subscribers, perhaps by preparing the results so that they could be displayed in a spreadsheet. The consumer stage terminates the pipeline. |

Each stage of the pipeline is constructed from C++ building blocks called event processors. An event processor (EP) executes a well-defined, clearly bound set of logic against an event stream. Event Engine comes with pre-defined event processors for counting, filtering, sorting, and storing to and retrieving from a database. User defined event processors can be created and included in pipelines to extend the processing power of the event engine. For instance:

1) A customer collector EP can be included to source event data from a machine-mounted or handheld reader.
2) A series of custom processor EP's can be included to append purchase order and product data to the event stream to provide context, and aggregate read events for the same product into a single 'receipt' event.
3) A custom consumer EP channels event data relating to priority baggage to a high priority alert queue.

```
Pipeline Definition
<?xml version='1.0' ?>
<Pipeline name="SensorCollector">
       <Functor name="SensorBundle::SensorFeed">
              <Parameter argumentname="portname="port"/>
       </Functor>
       <Functor name="EventEngine::StoreRealTime">
              <Argument name="seriesName"
       value="SensorBundle::SensorSeries"/>
       </Functor>
</Pipeline>
```

With the definitions of pipelines (an XML description of the pipeline name and series of event processors to be invoked), and event processors, events can be processed through Event Engine. Event Engine instantiates pipelines and event schemas through an engine class that provides pipeline lifetime management and component access facilities. The engine uses a dynamic registry to manage the creation and destruction of pipelines that are defined within the pipeline schemas.

The Event Engine architecture, with its modular definitions and ability to chain complex processing through simple pipeline definitions, can scale to support hundreds of pipelines executing simultaneously. The ability to scale to match the complexity of different data sets and sources within a single environment and manage to interpret and process the event data from an RFID-enabled enterprise, are the bedrock of the Progress for RFID.

## Integrating Event Engine with Your Environment – Progress Event Server

The Progress® Event Server is a framework for developing Java applications that use the Event Engine. Event Server provides a set of services for managing an Event Engine database and a pluggable framework into which custom collection and query adapters can be added. It is the set of adapters loaded into an instance of Event Server that give it specific application functionality and instructions on how event data is to be integrated within the application infrastructure. Progress for RFID is an example of a series of collection and query adapters that provide an RFID-aware set of functionality to an instance of Event Server.  That RFID functionality, from capture of tag read events to providing a refined event output, and a historical store of event streams, insulates traditional applications from being exposed to either the granular event data or the volume of the granular event data.  Instead traditional applications can look at the output of Progress for RFID as another simpler integration with little or no amending of their standard interfaces or supporting structures and functionality.

Any applications that the user can develop for use with Event Server consist of event and pipeline definitions and adapters that can be loaded into Event Server. Event Server does not expose the C++ Event Engine API. Instead, it exposes a simplified Java API for the rapid development of Event Engine applications. The configuration of Event Server augments the Event Engine configuration with adapter definitions.  Each adapter definition is an XML-formatted file that specifies the name, Java class file and property name/value pairs of the adapter.  The Event Server framework provides a programmer with all of the lifecycle tools to manage an adapter – from load through to shut down.

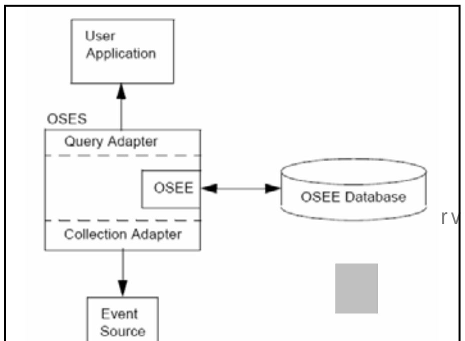
Adapters provide one of three types of functionality when interacting with Event Engine. They can be:

⇒ Collection Adapters – coded to import event data from a source into Event Engine. For instance, an adapter written to an environment sensor or a generic ALE interface. The collection adapter interacts with the producer stage Event Processor within a pipeline to move the incoming data into the Event Engine processing.

⇒ Query Adapters – provide access to external applications to the event data held within Event Engine.  The query is accepted by a query adapter which runs a QUERY PIPELINE to process the query.  The pipeline invokes the corresponding process EP logic and returns the result of its interrogation and transformation, via the CONSUMER EP back to the Query adapter.  The query adapter can then disseminate the data to its client or clients.

⇒ Service Adapters – provide control, scheduling, or other services for the management of collection and query adapters.

To allow for scalable deployments where workloads can be shared, the Event Server architecture allows for any one of the following models.



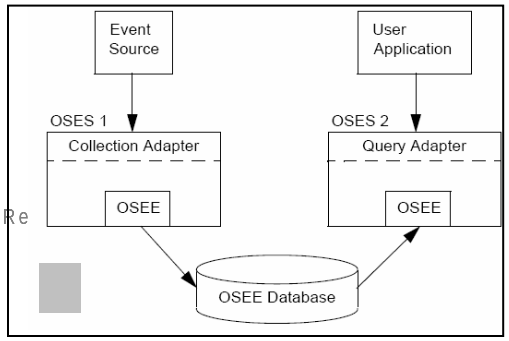


Figure 3: Progress for RFID Deployment Models

1) In its simplest deployment model a single instance of Event Server loads a single collection adapter and a single query adapter.

2) Alternatively the collection and query adapters can be loaded into separate Event Server instances where each shares the same Event Engine database. As these instances can be distributed across machines, RFID reads can be read and processed on a dedicated machine while end-user applications retrieve tag data through their own dedicated processors. This can be expanded where multiple Event Server are hosting their own query adapters.

3) An Event Server application is not limited to a single Event Engine database. It is valid to use multiple databases with different schemas to store different event data or to store the same event that is categorized differently in each database. Progress for RFID is a good example. It uses three separate Event Engine databases for storing RFID event data.

4) An Event Server application is not limited to a single type of collection or query adapter. An instance of Event Server can have multiple collection adapters, where each adapter is receiving events from different event sources. Also, one or more instances of Event Server can have multiple query adapters where each services queries in a different format or for different protocols. The only restriction on this deployment model is that all collection adapters storing events in the same Event Engine database must share the same Event Engine collector pipeline.

## Progress Event Database

The Progress for RFID's data cache is based on a real-time in-memory event database (RIED). The RIED manages data much like virtual memory systems in modern operating systems. Virtual memory systems support uniform memory references to data, whether that data is located in primary or secondary memory. Data is held in memory on a most recently used basis and paged out when the space is required by another program. Using this virtual memory mapping architecture (VMMA), the database moves unused data to a secondary storage area where it can be persistent or transient, depending upon the needs of the RFID application. When the application needs to access an object that is not in primary memory, the RIED transfers the page containing the referenced data automatically — possibly together with adjacent pages — to primary memory that serves as the application's cache. This cache acts as a pool of event objects that have been recently used in the virtual memory of the application's host. The database's direct mapping mechanism uses lock caching (as well as data caching) and call back locking to ensure cache coherency with no run-time overhead after the initial access.

### RIED Performance: Data at the Speed of Smart

An Progress for RFID located near the edge of the network in a large distribution center might have to log tens of thousands of events a second. A typical RFID reader can process hundreds of tags per second. A large distribution center may have hundreds of readers. As shown in table 1, tests performed early last year at the Auto-ID Center show the impact of primarily disk versus primarily in-memory database systems. For each test, a DELL PowerEdge Server 2500, with an 1133MHz Intel Pentium III processor, 512KB cache, and 512MB RAM was used. In these tests, a million events were passed to a logging process and then to a storage system. In scenario 1, the events were written to a traditional relational database. In scenario 2, a prototype RIED was used to capture the events. As expected, the RIED more closely matched the needs of the real-time simulation.

**Table 1: Measuring data capture performance**

| | **Scenario 1**<br><br>**Traditional DBMS** | **Scenario 2**<br><br>**Real-time In-memory Database** |
|---|---|---|
| **Time for an EPC event** | **10m/EPC Event** | **64µ/EPC Event** |
| **Number of events per second** | **100 events/ sec** | **15,000 events/ sec** |

The net effect of this design is that all data is kept in memory as much as is practical, thus providing the kind of in-memory performance that is required by RFID applications. The advantages of this approach over simplistic in-memory data management schemes include:

⇒ Size of the database - A simplistic in-memory data manager is constrained by the size of addressable memory for that process. With Progress for RFID's VMMA, there is no problem if the amount of event data requires more memory than is available in its process address space. The RIED storage manager transparently pages data from secondary to primary memory as soon as it is required.

⇒ Resilience to system interrupts – Since a copy of all EPC data is held in secondary memory in a transactionally consistent way, the state of the application can be automatically recovered when it comes back up.

⇒ Data Sharing – To avoid concurrency conflicts and reduce memory contention, simplistic in-memory data managers will keep one or more snapshots of the event data in memory so that the database can be queried without interfering with the real-time streaming. This may be appropriate for some applications, but because it reduces the size of available primary memory, it is often not ideal. The RIED provides many options for sharing in-memory data across multiple processes and machines. This allows external applications access to data without interfering with the real-time data streaming in from the readers.

⇒ Data Distribution – The database's Cache-Forward® Architecture provides a simple, transparent, transactionally consistent data distribution mechanism so that the Progress for RFID can efficiently collect data in local caches, but share it across several physical machines.

## *Implementing RFID Collection*

In an RFID implementation, the majority of reads will be passed from the various reader sites through local ALE middleware to Progress for RFID. Some reads may be stored and passed en masse from handheld devices or remote locations without the requisite network connectivity to enable real-time reader integration.

Progress for RFID includes a purposed adapter to collect events from ALE Middleware. This adapter adheres to the ECReport (Event Cycle Report) specification created by EPCglobal. ECReport is a standard specification for any ALE middleware to publish tag read events. It identifies the tags read at a particular reader. ECReport messages can be the result of a request from a user application (a call using the ECSpec message) or a direct communication to a user application or Progress for RFID based on subscription rules.

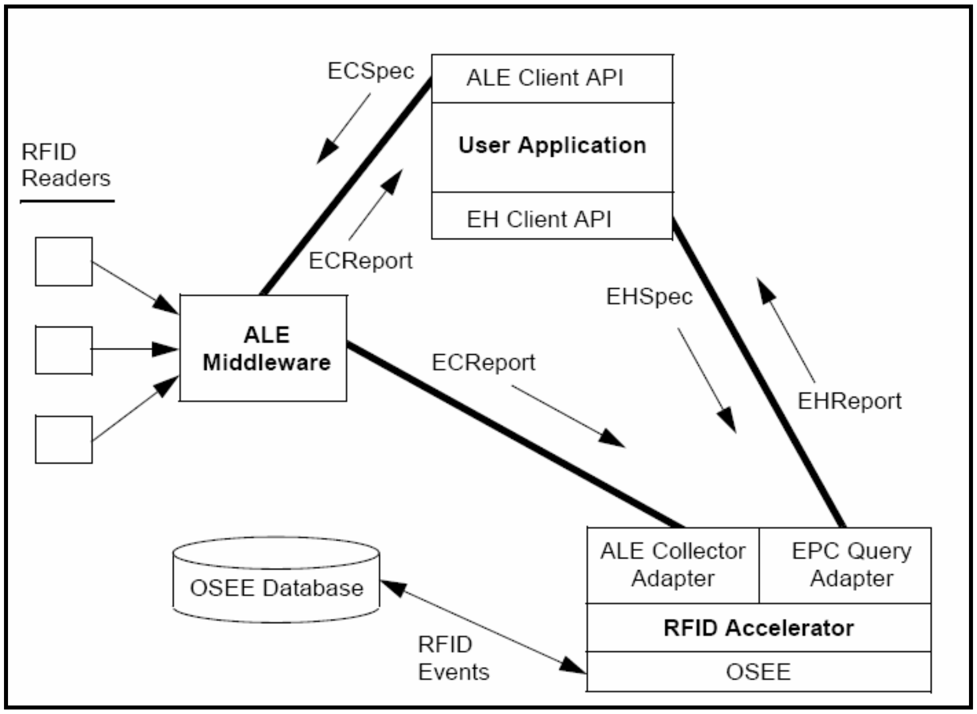


Figure 4: Connecting ALE with Back End Architecture

In the diagram above, note the ALE middleware communicates with the ALE Collector adapter. If the implementation is using either:

⇒ Different event/middleware standards (e.g. non-EPC compliant tags or non-EPC compliant middleware)

⇒ A direct link between the readers and Progress for RFID

Then the standard ALE collector adapter would be replaced with an adapter that adheres to the event/sensor/middleware environment in use. Using the Event Server framework this adapter can be included with the implementation of Progress for RFID in the place of the standard ALE collector.

The communication between the ALE middleware and Progress for RFID is established by a subscription established during configuration. The subscription identifies to the ALE middleware the readers under its jurisdiction that will be handled by an identified instance of

Progress for RFID. Any series of RFID events that is being collected by an implementation will reside in a single Event Engine database. The collection adapters for that Event Engine database must reside in a single Progress for RFID, but there can be multiple collectors. In this configuration, which allows different reader sites to be directed to dedicated collection adapters, an implementation can scale the Progress for RFID by separating the collection and query functions, as described in the above Event Server Deployment discussion.

Processing the events from the collection adapters is now a function of the pipelines and event processors implemented in Event Engine to handle the RFID event stream. Based on event type, pipeline processing may persist the raw ALE read, assimilate or aggregate its content with similar events (reads of individual units making up a single product movement through a reader) or any other scenario-specific function.

When persisting RFID event data in Event Engine, Progress for RFID stores the following values:

| **Attribute** | **Description** |
|---|---|
| Tag ID | This is the unique serial number of the entity that has been read. Tag formats vary by industry and purpose. EPC defines a set of standard encodings that Progress for RFID understands. The basic purpose of the tag is to identify the origin of the entity, the item identifier and a unique serial number for the instance of the item. The supported EPC 64 bit and 96 bit tag encodings are listed in Appendix A. |
| Reader ID | The assigned identifier of a reader in a network. |
| Event Type | Event Type refers to the movement of a tag through a reader's field of interest. 'Add' is the value used to record the entry of a tag into the field of a reader. 'Delete' is the value used to record the exit of a tag from the field of a reader. |
| Time | Date and Time of the event. |

## Serving RFID Data to External Applications

Revisiting the diagram depicting ALE, Progress for RFID and end user applications above, it can be seen that queries against the RFID event data are managed via a query adapter. The query adapter masks the complexity of the actual retrieval of RFID data from the calling application by accepting queries in the form of a standardized message – an EHSpec (Event History specification) and result set (EHReport – Event History Report).

Progress for RFID has modeled these messages after the existing EPCglobal specifications for end user applications to request current tag reads from an ALE application – ECSpec and ECReport. These interactions deal with the tag reads that the ALE currently has available to it. So ECSpec and ECReport can answer the question 'Which tags are currently at reader x'?

As the concept of event history is not completely evolved, and that event history offers a greater scope of query request than can a request against transient data, the event history queries use the corresponding "EC" message as a basis and extend the capabilities using the EQL functions of Event Engine.

The EQL offers tools to support filtering, both inclusion and exclusion filter patterns and group functions on the event data set. Hence an application can request from Progress for RFID any of the following:

⇒ List the tag events at a reader (or readers) during a specified time period.

⇒ A count of all tag events at a reader (or readers) during a specified time period.

⇒ A list of all readers that have ‘read’ a tag during a specified time period.

### Sample EQL – Looking for All Products from 2 Suppliers Moving Through Dock Door 24 within a Given Time Period

```
<EHSpec includeSpecInReports="true">

 <boundarySpec>
   <startTime>2004-10-01T15:55:47.622Z</startTime>
   <endTime>2004-10-17T09:30:47-05:00</endTime>
  </boundarySpec>

  <reportSpecs>

     <readerReportSpecs>

        <logicalReaders>
          <logicalReader>DockDoor24</logicalReader>
        </logicalReaders>

        <readerReportSpec reportName="ReaderReport1">
          <reportSet set="ALL"/>
          <output includeList="true" includeCount="false"/>
          <filterSpec>
             <includepatterns>
                  <includePattern>urn:epc:pat:sgtin-64:3.0037000.*.*</includePattern>
                  <includePattern>urn:epc:pat:sgtin-64:3.0029000.*.*</includePattern>
            </includePatterns>
        </filterSpec>
     </readerReportSpec>

     </readerReportSpecs>

 </reportSpecs>

</EHSpec>
```

As we will explore in the later section ‘Harnessing Events for Critical Business Processes,’ these query types allow Progress for RFID to serve RFID event data to many different applications.

In addition to the point-to-point query model supported by the standard Progress for RFID, the application can be deployed with an enterprise scale information bus, such as the SonicMQ product. This specialized query adapter, also built on the Event Server framework, allows the query/response to work in a publish and subscribe model whereby a single enterprise client can request a set of event data and have the result set served to a series of subscribing clients.

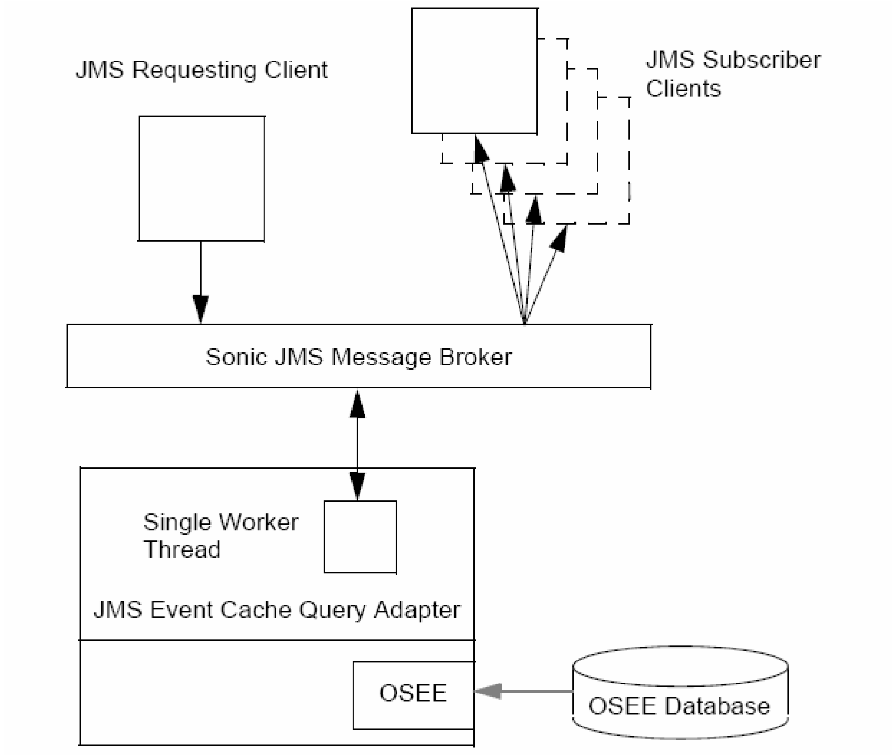


Figure 5: Progress for RFID Serving EQL Requests through Sonic JMS

The above provides a means of insulating an enterprise during a connectivity failure, allowing Progress for RFID to continue to receive RFID event reads from the supply-chain. On the resumption of connectivity with enterprise applications, a single 'Synchronize' client can have Progress for RFID replay events captured since the connectivity break. The replayed event data can be disseminated to the appropriate applications using an intelligent message broker, such as the Enterprise Service Bus supplied by Sonic Software.

### Context on an RFID Event

To apply some meaning to an RFID event – which contains the EPC code, the data and time of the event and the reader ID (to provide location), event processing must be able to retrieve master data that gives the tag read some context. As examples, this context can come from product master files, vendor or customer master files, purchase order or contract files. The ability to retrieve this data in real time, apply it to the raw event and make a business decision is critical to the success of Complex Event Processing.

As the Event Engine processing evolves, so does the ability to cache large volumes of this data in memory, ensuring the data stays synched with the resident master files or other cache instances. The caching of master data close to the processing location will become another major thread in the implementation of CEP. The EdgeXtend product from Progress is a leading technology in the areas of caching and cache synchronization that will be leveraged in the Event Engine implementations of Complex Event Processing.

## *Harnessing Events for Critical Business Processes*

Under 'Serving RFID Data to External Applications' we looked at the use of the Event Query Adapter within Progress for RFID to request subsets of the RFID event data within the Event Engine database. To provide some context, in this section we look at 5 distinct use cases that retrieve RFID data for typical supply-chain needs.

#### Example 1: Monitoring Distribution Center Activity

Current workload monitoring in a warehouse or distribution center revolves around the management of individual workloads (tasks) being completed on-time. With RFID sensor data being collected from various activities around a physical location, users can trend performance of tasks by station. For instance:

⇒ Number of tags arriving at all RECEIPT door readers

⇒ Number of tags exiting all SHIPPING door readers

⇒ Average length of time receipts stay in STAGING (difference between tag arrival and tag departure times at readers in STAGING areas)

### Example 2: Tracing Case History

A user wishes to discover how a case of products have ended up in the wrong location in the Distribution Center (DC). Scanning the tag on the case starts a process that queries for all readers that have seen this case. Progress for RFID returns the reader history for the case showing point of receipt, all movement and all reader ID's, in time sequence giving a history of how the case moved around the DC.

This differs from a traditional case history in an ERP or WMS application which will only record those movements entered by a user in a particular function (e.g. picking). The possibility of a mispick or a putaway maybe hidden in this application if the user overrides or does not enter the correct data (or any data at all).

### Example 3: Product Recall

In the event of a product recall due to a quality check failure post-distribution, an implementation will want to identify all lots that are affected, and all cases in those lots. To track the individual cases, tagged with an EPC code, the implementation queries Progress for RFID for the list of tags, requesting all readers for each tag and date/time. This shows the:

a) latest distribution/warehouse location of any case remaining in the facility;

b) last shipment history for any case that has been shipped.

The advantage of this over a traditional ERP application is that the RFID picture is real-time vs. an ERP/WMS solution that is dependent on manual scanning of the cases as they are shipped or moved around the facility.

### Example 4: Replaying Warehouse Activity

After finding cycle count errors for a particular product in the warehouse, the user wishes to see how all cases moved in and out of the warehouse during the last 72 hours. The query to Progress for RFID requests all events relating to tags on the identified product (using its EPC code) in a date/time range of (Cycle Count Date/Time – 72 hours) to (Cycle Count Date/Time). The returned list of tag events show all movement against the starting inventory point and where the discrepancy in the actual inventory vs. expected inventory resulted.

### Example 5: Integration Requirements

To integrate their legacy application with the new RFID infrastructure, an organization is faced with a long and costly project if it is to be able to meet mandated timeframes. Using Progress for RFID, the organization is able to buffer the legacy supply-chain application from immediate overhaul. Progress for RFID is integrated into the RFID infrastructure, capturing and storing RFID events as they stream from the various reader locations. The legacy application integration is established using query adapters that retrieve 'batches' of RFID events pertinent to the existing API and functionality. The query adapters retrieve the data since the last function call and format the event data into the API signature currently supported by the enterprise application. To demonstrate some immediate RFID connectivity, one legacy function is focused on and enabled to receive 'streamed' events from Progress for RFID without using the 'store/forward' technique.

## *Progress for RFID in Action*

So how can Progress for RFID enable and improve on existing RFID data initiatives? In this section we look at the advantages an implementation can get by leveraging Progress for RFID through 2 case studies where Progress for RFID is making an impact on the use of RFID event data.

### RFID in Supply Chain

The German forestry industry is one of the largest in the world. With almost one-third of the country's total land area forested, Germany produces 50 million cubic meters of timber every year, satisfying two-thirds of domestic demand. However, transporting timber from Germany's forests to sawmills can be a disruptive, multi-party process with many steps, companies and regulations involved. As a result, the leading German forestry company found that not only was it suffering from inventory shrinkage, but the payment process was inefficient and time consuming.

To solve these problems, the company worked with its solution provider to integrate Progress for RFID with its forestry application so that individual trees can be tagged and the RFID data collected from those logs as they are transported to its sawmills can be stored.

When deployed, a unique nail-shaped RFID tag is inserted into each tree that will be tracked as it moves through the supply chain, from the skidder that removes the tree from the forest, to the truck transportation to the rail car, etc. Using a handheld device, the information will be uploaded through GSM (Global System for Mobile communication) technology to a central location where Progress for RFID takes over, aggregating, normalizing and propagating the data into the central forestry application.

By creating this new system, the company expects much greater visibility into the raw materials pipeline and efficient management of its suppliers and customers. Further, the company anticipates that Progress for RFID will reduce the current inventory shrinkage by up to 70 percent and reduce the payment reconciliation process by more than 50 percent. This will drive efficiencies throughout the supply-chain: for the independent forest owners, the sawmills and for the company itself.

### RFID for Baggage Handling

The business case for airlines is simple: no one likes to lose their bags.  Current processes are typically driven by barcodes, but still involve a lot of manual loading and sorting, which leaves plenty of margin for error. With the average cost of a misdirected bag in the US estimated to be $150, the drag on the bottom line is serious for a high-volume business, perhaps as much as $US100 million a year, according to travel industry analysts.

The advantages of RFID over barcodes in this application are that it is contactless and unidirectional.

Progress for RFID can assist in real-time tracking of baggage through the numerous legs of origin to destination.  Capturing the bags that have been loaded on a manifest and matching them against the expected manifest in real time (as bags are loaded) is enabled through the processing within an Event Engine pipeline checking for misplaced bags. Due to its high volume of data, generated from the number of units moving at any one time and the number of events each bag generates, Progress for RFID also provides the scalability for airline deployment on a global scale.  The types of events include:

- ⇒ Bags arriving at check-in
- ⇒ Passing through security
- ⇒ Being loaded into a tagged container
- ⇒ The container being loaded onto an aircraft
- ⇒ Being unloaded at its destination or intermediate stop

⇒ Bags for the current destination removed from container, new bags loaded into the container

⇒ Final arrival

⇒ Final unload of the containers

⇒ Final bag unloading

⇒ Bag routing and passing into arrival hall

## *Conclusion*

Progress for RFID is designed for managing the immediate and future challenges of RFID event data management. Its architecture and performance capabilities lend themselves to many advantages in an implementation looking to adapt to the world of RFID.

### Application Integration Efforts

Many applications used today to manage transactions in a particular field (supply-chain, healthcare, transportation, etc.) are facing the challenge of how to adapt to the changes RFID brings. This covers both the type of data and the types of transactions. Progress for RFID mitigates the immediate need for change to these applications by:

1) Acting as a buffer from the raw RFID event stream and passing along event data that has been transformed, through pipeline event processors, into a standard transaction message that the target application can handle with its existing interfaces.

2) Acting as a repository for the wealth of new RFID event data. Today, the end user business application may not have processes or transactions that can take advantage of the additional depth that Progress for RFID can serve up from its event repository. Over time as these new or improved processes emerge, Progress for RFID can provide the data and visibility without burdening the business application with new data stores.

3) Providing a single source for event data to all applications. Using different query pipelines for different application requests, RFID event data can be retrieved from Progress for RFID in the form, and according to the specific needs of, the different applications in an implementation (inventory vs. logistics vs. billing).

### For Data Mapping Requirements

RFID will not be an immediate replacement for existing identification encoding schemes or labeling technologies. Further, while the goal of EPCglobal and ISO is to define standard encoding schemes, in a truly global supply chain it is likely that many different numbering schemes will co-exist. Progress for RFID, with its pipeline and adapter architecture inherited from the Event Engine and Event Server, is a framework that can support implementations that will be processing products and assets labeled in different ways. This capability to define and differ the core processing based on encoding and labeling allows an implementation to use Progress for RFID as a means of integrating all automatic identification technologies and events into a single layer. The central role of Progress for RFID to then serve the data to any consuming application in a consolidated form reduces the point-to-point integration efforts of application to technology/scheme.

### Scalability & Distributed Environments

As RFID implementations expand from pilots, often single site, targeted product set and operation set, to true enterprise scale, the size of the required infrastructure (labeling, reading) and volume of tag event data will expand dramatically. Progress for RFID and the architecture of Event Server are built to manage the scale of an enterprise deployment by supporting:

⇒ Distributed deployment of collectors and query adapters.

⇒ Processing of event collection data in the form of small, atomic event processors that are invoked only under scenarios that the event data (event type, product, type of tag) requires.

⇒ High scale caching of contextual information.

⇒ Event Query Language optimized for tag event search, transformation and retrieval vs. use of standard relational queries.

⇒ An underlying object database that can store and retrieve event data, either as a raw event or enhanced with content from its supporting context, as a single object rather than a series of relational database tables.

## Appendix A: Supported EPC Encoding Standards

GTIN-64 (Global Trade Identification Number)

SSCC-64 (Serial Shipping Container Code)

GLN-64 (Global Location Number)

GRAI-64 (Global Returnable Asset Identifier)

GIAI-64(Global Individual Asset Identifier)

GTIN-96

SSC-96

GLN-96

GRAI-96

GIAI-96

## Appendix B: References

### Articles

The 7 Principles of RFID Data Management (Mark Palmer, August 2004, Enterprise Systems Journal)

Build an Effective RFID Architecture (Mark Palmer, February 2004, RFID Journal)

### Books

The Power of Events (David Luckham, 2002, Addison Wesley)

### Organizations

EPC Global – www.epcglobalinc.org

## *The Company*

### Who is Progress Real Time Division?

Progress Real Time Division is a global provider of real-time data management products. Its products enable corporate data caching and complex event processing, and its leading object database is renowned for performance and scalability. Progress products are supporting RFID implementations, and are deployed in industries such as finance, telecommunications and travel, where companies rely on them to complement their corporate data management infrastructure.

Progress Real Time Division is an operating company of Progress Software Corporation (Nasdaq: PRGS), a $300+ million global software industry leader. Headquartered in Bedford, Mass., Progress can be reached on the Web at http://www.progress.com/realtime or by phone at +1-781-280-4000.

### Progress Software

Progress Software Corporation (NASDAQ: PRGS) is a $300+ million global software industry leader. PSC supplies technologies for all aspects of the development, deployment, integration and management of business applications through its operating companies: Progress Software, Sonic Software Corporation, DataDirect Technologies and Progress. Headquartered in Bedford, Mass., PSC can be reached on the Web at http://www.progress.com or by phone at +1-781-280-4000.

PROGRESS SOFTWARE

www.progress.com/realtime

Worldwide and North American Headquarters
Progress Software Corporation, 14 Oak Park, Bedford, MA 01730 USA Tel: +1 781 280 4000 Fax: +1 781 280 4095

EMEA Headquarters
Progress Software Europe B.V., Schorpioenstraat 67, 3067 GG Rotterdam, The Netherlands Tel: +31 10 286 5700 Fax: +31 10 286 5777

UK
Progress Software Limited, 210 Bath Road, Slough, Berkshire England SL1 3XE Tel: +44 1753 216 300 Fax: +44 1753 216 301

Germany
Progress Software GmbH, Konrad-Adenauer-Str. 13, 50996 Köln, Germany Tel: +49-221-93-57-90 Fax: +49-221-93-57-978

France

Progress Software S.A.R.L., 3 Place de Saverne, Les Renardières B, 92901 Paris la Défense Tel: +33 1 41 16 16 56