

Progress® DataXtend™ CE for C++

Executive Summary

Enterprise application architects and developers face challenges that include tight schedules, reduced budgets, and the complexities involved in supporting multiple platforms. Demanding requirements for performance, scalability, and high availability of business-critical data increase complexity and risk.

Progress DataXtend CE for C++ helps organizations meet these challenges and reduce risks during application design, development, and deployment by:

- Boosting developer productivity with sophisticated development tools and runtime capabilities.
- Improving application performance using sophisticated in-memory caching.
- Increasing system scalability and resiliency by synchronizing cached data amongst clustered systems.

In addition to its software offerings, the highly skilled Progress Real Time Division Professional Services organization can provide assistance at any stage of development. With extensive experience building highly demanding trading, telco, and logistics systems, they can offer valuable insight during the architecture and design process. When engaged early, Progress Real Time Division Professional Services can help ensure that a project is delivered on time and meets business requirements by fully exploiting the software’s capabilities.

The Benefits and Challenges of Using C++

While Java offers ease of use and cross-platform capabilities, many organizations choose C++ for enterprise application development. Some of the benefits include:

- A mature, full-featured, and efficient API for applications with stringent scalability and performance requirements. In many systems, C++ back-end systems support front-office programs written in VisualBasic, .NET, or Java.
- Requirements for interfacing with existing code, such as third-party libraries or legacy applications.
- Requirements to run on platforms not supported by Java or .NET.
- Existing development expertise and other investments in C++.

However, C++ development also poses challenges:

- Developers of Java and .NET applications have more options for tools, ranging from IDEs to application infrastructure (e.g., J2EE application servers).
- It is often hard to find skilled C++ developers. When you do find them, you want them to concentrate on your business concerns, not on application infrastructure.
- C++ applications can be challenging to integrate with Java or .NET systems.
- Standards used by C++ developers, such as CORBA, are general purpose. They solve broad problems that many applications have in common, but do not address critical data access problems.

Progress Real Time Division Credentials

Without tools, persistent data management consumes a large portion of development effort and accounts for the most expensive operations at runtime. The Progress Real Time Division has 12 years of experience in helping customers solve data access problems and implement demanding applications in industries such as financial trading, transportation logistics, supply chain management, telephony, and manufacturing. From working with our customers and observing their challenges and successes, we have gained extensive knowledge and experience in the development of enterprise persistent data tiers. Some of the most important lessons that we have learned include:

- Model-driven development is a critical discipline for addressing the complexity of enterprise data. Think data first and think of data as objects. Doing this early allows architects and developers to focus on the application instead of the database infrastructure. Once a data model “contract” is defined, it can be efficiently reused across a suite of applications or among development teams. This contract is modifiable when driven by changes in business requirements, but does not require changes simply because the implementation changes.
- Stateful applications outperform and outscale stateless applications. If designed correctly, stateful applications relieve database bottlenecks by allowing active data to reside closer to where it is processed. Distributed data caching enables distributed stateful architectures.
- Architecture matters. Application design must take persistent data into consideration. The right data caching strategy can provide the foundation for performance, scalability and even robustness, but it must be built into the application data management framework. When the data caching mechanism is integrated with the data layer, it allows the greatest performance and flexibility.
- Standards offer valuable solutions for general problems, while best-of-breed solutions are designed to handle the really difficult problems. The most successful applications leverage the use of standards where possible and apply tailored solutions where necessary. If you want to win a car race, would you buy from a dealer who handles everything from compacts to off-road vehicles or would you look to a racing specialist?

The Progress DataXtend CE Solution

Progress® DataXtend™ CE offers a unique combination of Object-Relational mapping and sophisticated runtime services for persistent data. With products and services for C++, C# and Java application development, solutions built using DataXtend CE can leverage the strengths of J2EE and CORBA, while taking advantage of the Progress Real Time Division best-of-breed solution for application data access. All Progress DataXtend CE products provide model-driven development and code generation, along with dynamic distributed caching that is both transparent and tightly integrated with the persistent object model.

For organizations that must deal with multiple platforms and disparate databases, Progress Real Time Division products allow you to standardize on an enterprise-wide strategy that supports applications written in Java, C++, C# or .NET running on Windows, UNIX, or Linux operating systems – even integrating with applications running on IBM® WebSphere® or BEA® WebLogic®.

Developing with DataXtend CE for C++

Progress DataXtend CE for C++ includes development tools and runtime libraries for managing persistent data. You use DataXtend CE tools to define object-to-relational mapping and to generate database-independent code for persistent objects. The runtime libraries link into your application and provide services such as database connection pool management, transaction management, and transaction-safe in-memory caching.

How to Use DataXtend CE for C++ As good design practice, specify the data model and architecture first. Determine the inheritance and relationships between the persistent objects needed by your application. DataXtend CE includes tools to define this persistent object model and to specify how those persistent objects will map to a relational database schema. The tools can be used standalone or in conjunction with Rational Rose. DataXtend CE tools generate C++ classes for the persistent objects. You write business objects which make calls on the generated objects to create, access, and update persistent data. At runtime, the generated objects translate these calls to SQL statements that are highly optimized for your database. Your application may also call

management classes to configure database connections, handle transactions, and tune the in-memory cache.

Use your choice of distribution technologies to allow clients to access the business objects. In this way, a DataXtend CE application offers a clean separation and abstraction of the business logic from the persistency mechanism. Figure 1 shows a logical view of an application that uses an ORB for client communication.

The remainder of this paper outlines how your development team can gain benefits from Progress DataXtend CE for C++ by:

- Boosting developer productivity
- Improving application performance
- Increasing system scalability and resilience

Boosting Developer Productivity

Without DataXtend CE, designing and building the persistent portion of an enterprise application requires cooperation across two disparate fields: object-oriented programming and relational database theory and administration. Within large organizations, developers and DBAs usually belong to different departments. Their problem domains differ because of the object-relational mismatch, which frequently results in conflicting “best practices.”

Even without this separation, writing code to persist objects to a database is complex and error-prone. An R.B. Webber study found that data access accounts for 30 to 40 percent of the overall application code in enterprise systems. Experts estimate the amount of code required to persist and retrieve a single object ranges from hundreds to thousands of lines.

O/R Mapping and Model-Driven Development Object-Relational (O/R) Mapping is the bridge between application architects and database architects. With DataXtend CE tools, the object model restricts the relational schema as little as possible, and vice versa. The generated code reflects years of development, quality assurance, and testing – both internally by DataXtend CE and externally in real application environments. By working at the level of the application object model and using the code generated by the DataXtend CE tools for persistent objects, you save resources that would otherwise be required

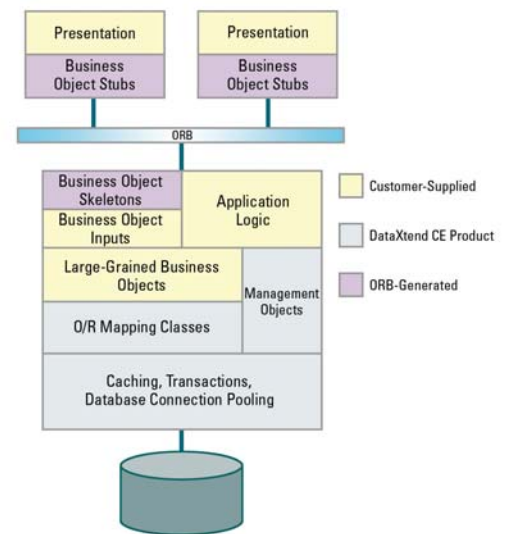


Figure 1: Logical view of a DataXtend CE-based application.

to write and test the object persistency layer of your application. Each object exposes a well-defined interface that shields business objects from implementation and database-specific details. Figure 2 illustrates how DataXtend CE for C++ fits into your development process. Shaded boxes identify application components that you will write.

This “model-driven development” allow application developers to work at a higher level of abstraction, the level of the business domain, rather than dealing with low-level details that don’t provide business value.

The Persistent Data Model Inheritance and abstraction are important tools for C++ programmers. All Progress Real Time Division products provide the following in the persistent data model:

- Single inheritance.
- Multiple classes can inherit from the same base class.
- Base classes are abstract by default, but can be defined as concrete so they can be mapped to a database table. Base classes can be used as usual in object-oriented programming, i.e., to specify common attributes (and queries, keys, and optimistic control attributes) for all subclasses in a single place.
- Application code can access all methods on the base class or an instance of its derived classes.

The DataXtend CE tools allow flexibility in the way you map persistent objects to the database by supporting both horizontal and union mapping:

- Horizontal: maps one object to a single database table. All rows in the table map to instances of the class.
- Union: maps objects from a contiguous subtree of an inheritance hierarchy to the same table. Rows in the table map to instances of concrete classes in the hierarchy.

Generated Code DataXtend CE generated classes include create, update, and delete methods, along with accessors to retrieve an object by its attribute values, and to obtain related instances. DataXtend CE can optionally generate hook methods and customized multi-attribute queries.

The example to the right shows a portion of the header file generated for a simple Account object with just a few attributes and one relationship:

It is quite simple to create an instance of this object and use DataXtend CE to persist it to the database. (The *_var classes perform smart memory management for you. The PS_TransactionMgr class, as its name implies, manages transactions.)

Improving Application Performance

While relational databases can be highly optimized and efficient, they simply were not designed to handle the increased volumes and data access patterns of modern enterprise systems. The (relatively) limited number of connections places a threshold on the number of requests that can be handled concurrently. When all connections are in use, incoming requests block. Locks required during updates can cause additional delays.

Caching in the application tier is widely accepted as the best way to reduce these bottlenecks. By satisfying a large portion of client requests from cached data, an application saves expensive trips to the database, and reduces the load on the database, allowing it to more efficiently handle operations which actually modify the data set.

With caching you have two choices, build or buy. Building a caching system has disadvantages similar to writing your own data access:

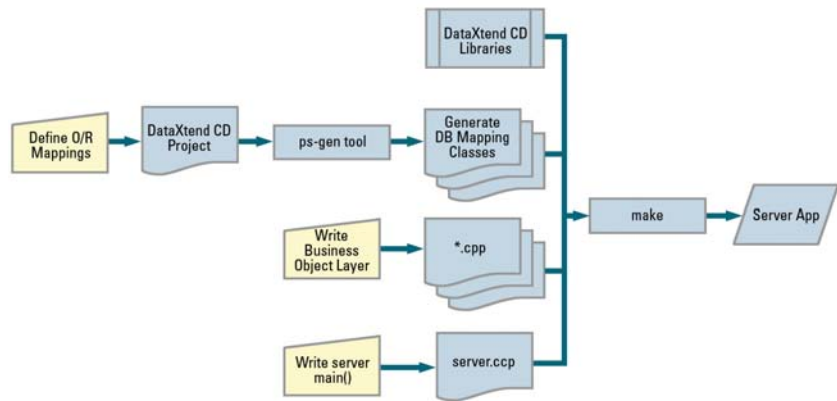


Figure 2: Application development process using Progress DataXtend CE for C++.

```

public:
    // BEGIN ----- PS(Trade,,fact_methods)
    // Custom Factory_Account methods.
    // END ----- PS(Trade,,fact_methods)

    static Account_Factory_var instance();

    // Instance creation and/or removal
    Account_var create(const char* accountId, double balance, long numTransactions) const;
    void remove(Account_var doomed) const;

    // Query methods
    Account_Iter_var allInstances(unsigned long qFlags) const;
    Account_Iter_var query(const PS_ClassId_Cltm& clslist,
        const char* from, const char* where,
        const PS_ArcId_Cltm& arclist) const;
    Account_Iter_var queryJoinSP(const char* spStatement,
        const PS_ClassId_Cltm& clslist,
        const PS_ArcId_Cltm& arclist) const;
    Account_var fromXML(const PS_String_var xmlDoc, PS_bool validateFlag=kps_false) const;
    Account_var queryKey(const char* accountId, PS_Defs::ObjectSource onlyInCache) const;
    Account_Iter_var querySP(const char* spStatement) const;
    Account_Iter_var querySQLwhere(const char* whereClause,
        unsigned long qFlags=(PS_Defs::k_databaseOnly | PS_Defs::k_includeSubclasses)) const;
    Account_Iter_var querySQLwhereWithHints(const char* whereClause,
        const char* hints, unsigned long qFlags=(PS_Defs::k_databaseOnly | PS_Defs::k_includeSubclasses)) const;
    PS_Defs::ClearingPolicy clearingPolicy();
    void clearingPolicy(PS_Defs::ClearingPolicy policy);

    // State Access Begin

    // State Access methods
    Account_var createFromState(const Account_Creation_State& info) const;
    Account_Cltm_var createCltmFromStateCltm(const Account_Creation_State_Cltm& infolist) const;
    void writeStateCltm(const Account_State_Cltm& statelist) const;
    void removeObjectCltm(const Account_Cltm& objlist) const;
    Account_State_Cltm_var associateObjectsWithStates(const Account_Cltm& objlist) const;
    Account_State_Cltm_var associateStatesWithObjects(const Account_State_Cltm& statelist) const;
    static void destroyObjectState(PS_Object_State*);

    // State Access End

    virtual PS_ArcId makeArcId(const char* arcname) const;
    virtual PS_ClassId_var makeClassId(const char* alias) const;

    // For constructing a leaf shell
    static PS_PObject* makeShell(PS_PObject_Rep*);

    const char* className() const;

private:
    Account_Factory();
    Account_Factory(const Account_Factory&);
    Account_Factory& operator=(const Account_Factory&);

    static Account_Factory m_inst;
}

```

```

//declaration of variables
const char* accountId = "1";
double balance = 500.00;
long numTransactions = 0;

// Initialize factory pointers
PS_TransactionMgr_var pTransMgr = PS_TransactionMgr::instance();
Account_Factory_var acctFactory = Account_Factory::instance();

// Start a transaction, create object, commit
pTransMgr->begin();
Account_var newAcct = acctFactory->create(accountId, balance,
numTransactions);
pTransMgr->commit();

```

- It siphons resources away from the development of your business-specific logic.
- It requires starting from scratch – incurring design, development, and testing overhead.
- High-volume, transactional, multithreaded caching is exceptionally complex and difficult to get right.

Additionally, home-grown and most commercial caching services are usually designed to handle read-only or predictably changing data, such as static content and reference information. When data changes predictably or not at all, an application can easily refresh the cache periodically to ensure that all clients have accurate information. But, when critical information changes unpredictably, the burden falls on the developer to synchronize data. This problem is too complex to address on an application-by-application basis. The difficulty of synchronizing data increases exponentially as servers are clustered to support more clients or higher availability.

Dynamic Caching Rather than restricting caching to read-only data or maintaining separate views of changing data, Progress DataXtend CE for C++ provides one multi-threaded shared cache per DataXtend CE server. When a user starts to update data, DataXtend CE creates an isolated transactional cache to capture the changes until the transaction commits. Upon commit, DataXtend CE updates the shared cache so that all users have the freshest information. Because of its tight integration with the data model, DataXtend CE caching is nearly transparent to the application developer. An application can control which objects stay in the cache. DataXtend CE creates an in-memory cache index for the results of certain queries. Requests for application data can specify whether the data should be retrieved from the cache or the database.

For instance, the following call attempts to find an object in the cache and only accesses the database if the object is not found:

```
// Account to retrieve
const char* acctID = "1";

// Retrieve an Account object
Account_var acct = acctFactory->querykey(acctID, PS_Defs::k_cacheThenDatabase);
```

You can define additional multi-attribute queries that provide the ability to access the cache or database. Optionally, DataXtend CE can create additional cache indexes that will optimize the results of cache queries for these generated methods.

Continuous Cache Coordination

Object-oriented applications built on the advanced caching capabilities of DataXtend CE operate in complex environments where other enterprise applications that make use of traditional SQL access methods may be modifying the data as well. It is critical that updates made by these other enterprise applications are reflected in the object cache in a timely and consistent way.

Typical O/R mapping and caching infrastructures follow a traditional “request-response” model for accessing data. In this model changes made by other enterprise applications would not be seen until an explicit request is made to reload the cached data. This can result in accessing stale or even inconsistent data.

To address the caching requirements of these complex environments, DataXtend CE features Continuous Cache Coordination to proactively “push” changes made to the database out to the distributed cache. This ensures that the enterprise data caching infrastructure maintains fresh and consistent data. The advanced data caching technology of Continuous Cache Coordination is only available with DataXtend CE.

Additional Performance-Enhancing Features

Progress Real Time Division realizes that every application is different. Rather than having a one-size-fits-all strategy, DataXtend CE is highly optimized to address specific application requirements and to further increase performance, including features such as:

- Multi-class object fetch retrieves a collection of objects and any objects to which they are related in one database call. This method preserves the relationships between objects in the cache, allowing for highly efficient in-cache queries.
- State access methods create an object or multiple objects and set all of their attributes in one call.
- Batch writes pack multiple inserts, updates, and deletes into a single SQL Statement.
- Deferred writes access the database at transaction commit and reduce database calls when multiple objects and attributes are updated in the same transaction.
- Sparse updates optimize changes to database tables with many columns where few columns change.
- Lazy-fetch manages large result sets by providing incremental access to objects and discarding unwanted results.

Increasing System Scalability and Resiliency

Enterprise applications provide high performance, scalability, and fault tolerance with replication or clustering and remote management technologies. This distribution of data exacerbates the difficulties involved with caching dynamic data. It increases the number of shared caches and concurrent users, which must all be notified when changes occur.

To handle this challenge, DataXtend CE automatically propagates changes to the shared caches of multiple servers within a cluster.

Interestingly, there is no technical reason why cache synchronization may only occur within a single data center. DataXtend CE’s multi-site Distributed Dynamic Caching architecture enables operational data to be synchronized across multiple data centers, requiring only one database in a central location (with the option of a second, for failover and disaster recovery). Providing application services to geographically distributed users can be achieved by deploying inexpensive “virtual” data centers to remote locations.

Virtual data centers use DataXtend CE servers only, and utilize synchronization to cache and access active data, as in Figure 3. Critical data becomes available across and beyond the enterprise, without compromising data integrity – or breaking the bank.

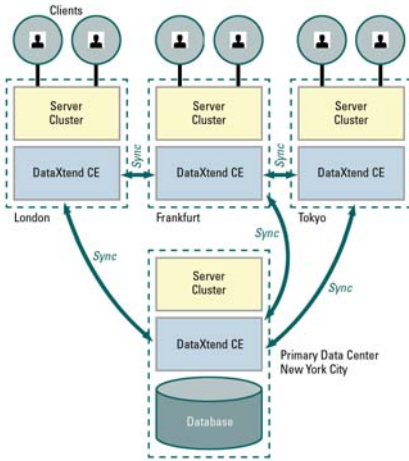


Figure 3: Geographically separated clusters effectively place data "closer" to their primary users.

Implementing Cache Both single and multi-cluster deployments of cache synchronization offer a number of options. Cache synchronization uses the following components: the PSAdmin_SyncMgr class, the PS_SyncMessenger abstract class, a PS_SyncMessenger implementation class, as well as an underlying messaging system to handle communication between servers. To enable cache synchronization, the application creates the appropriate messenger, sets any messaging-specific parameters, and defines which classes will participate in synchronization. Synchronization then occurs automatically whenever data changes. Express synchronization sends the synchronization message after a transaction completes. Guaranteed synchronization integrates the sending of the synchronization message with the transaction and also writes the message to a persistent store.

DataXtend CE provides an express messenger for the for the Sonic MQ and TIB/Rendezvous messaging system, and ships with a guaranteed messenger for OracleAQ. Provided interfaces can be customized to allow use of any messaging system. Progress Real Time Division Professional Services can supply messengers for 7 other messaging systems, including one that uses sockets and therefore does not require an additional messaging system.

Conclusion

Progress DataXtend CE for C++ provides object-to-relational mapping and sophisticated caching that boosts developer productivity and improves performance and scalability for mission critical applications. A data services layer built on Progress Real Time Division products can provide these benefits to applications written in C++, Java, or .NET, on the Windows, Linux, and UNIX platforms.

DataXtend CE offers a clear path for maintenance and growth. A data layer built with DataXtend CE has the ability to support multiple applications within a local area network, as well as geographically remote virtual data centers.

DataXtend CE enables businesses to unlock one of the most valuable resources of any organization – accurate, real-time information.

Progress Real Time Division is committed to providing the products and services that will make your application successful. Our Proof of Concept (POC) program assists organizations in their review, evaluation, and assimilation of DataXtend CE technology. A POC is an on-site implementation of a prototype application assisted by a Progress Real Time Division Professional Services consultant.

A POC delivers an initial architecture and working prototype of your business application, coupled with targeted education and on-site mentoring. It is a structured and detailed process with specific goals, deliverables, and success criteria, and emphasizes recommended architectures and best practices for utilizing DataXtend CE products to satisfy your business and technical requirements. Most POCs take a very short amount of time, typically 2-3 weeks, and concretely demonstrate the benefits of our software in your operational environment.

*Ask for a Progress Real Time Division Proof of Concept today.
Contact the Progress Real Time Division at 800-477-6473.*

Worldwide and North American Headquarters

Progress Real Time Division, 14 Oak Park, Bedford, MA 01730 USA Tel: +1 781 280 4000

UK and Northern Ireland

Progress Real Time Division, 210 Bath Road, Slough, Berkshire, SL1 3XE England Tel: +44 1753 216 300

Central Europe

Progress Real Time Division, Konrad-Adenauer-Str. 13, 50996 Köln, Germany Tel: +49 6171 981 127

France

Progress Real Time Division, 3 Place de Saverne, Les Renardières B, 92901 Paris la Défense Tel: +33 1 41 16 16 56

www.progress.com/realtime

PROGRESS
SOFTWARE

www.progress.com

Specifications subject to change without notice.

© 2005 Progress Software Corporation.

All Rights Reserved.

EXC/BC/0705-

Code 2482



0000105332

© 2005 Progress Software Corporation. Progress and DataXtend are trademarks or registered trademarks of Progress Software Corporation in the U.S. and other countries. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. Any other trademarks or service marks contained herein are the property of their respective owners.