Version 7.0 | Java and .NET

New in db4o Version 7.0

Users of object databases are innovators. Although the technology is not new, only the 2nd generation of object databases can claim to bring true value and unprecedented efficiency to its users. These users are breaking the mold of traditional thinking when they choose db4o as the database or the persistence solution for their application.

As these users have stated in numerous forum and blog posts – anyone developing an application in Java or .NET that requires a database should consider using an object database. The benefits speak for themselves: aside from significant development time and corresponding cost savings the software is easier to maintain, refactor, and reuse.

db4o 7.0 was released on November 13th, 2007 to a community of now over 26,000 users that understand the value proposition of an object database – a community of innovators.

db4o is the open source object database that enables Java and .NET developers to store and retrieve any application object with only one line of code, eliminating the need to predefine or maintain a separate, rigid data model.

World class innovators deploy the db4o database engine at the heart of next generation data-driven devices and applications to cache user-generated and client-side data, enabling compelling new features and achieving unprecedented performance and flexibility.

db4o 7.0 at a glance

Transparent Activation	Transparent Activation (TA) automatically detects which objects are required by the application, thereby only loading the absolute minimum from disk, which consumes less memory and boosts performance.
Performance	db4o 7.0 enhancements that improve performance include: ● A new IoAdapter cache ● The introduction of batching of asynchronous messages ● Multi-Transactional-ObjectContainer (MTOC) ● The use of a very efficient Freespace Management System based on BTrees
Exception Handling	db4o 7.0 includes a revised Exception Handling implementation that provides immediate feedback of problems by throwing Runtime/Unchecked Exceptions. This also enables new features such as Unique Constraints.
Unique Constraints	Unique Constraints allow a user to define a field to be unique across a particular Class. This means that an object cannot be saved where a previously committed object had the same field value for fields marked as unique.
Multi-Transactional-ObjectContainer (MTOC)	MTOC improves embedded Client/Server performance by passing objects directly between "client" and "server" instead of marshalling and unmarshalling them.
Committed Callbacks with Pushed Updates	Commit time callbacks allow to ensure that specific constraints are not violated. Pushed updates allow clients to listen to changes, to be able to update objects with the latest versions that other clients have committed.
OSGi-compliant service interface	The OSGi framework is a Java middle-ware that significantly extends the functionality of Java by allowing software to be structured as dynamic components

...and more.

Transparent Activation (TA)

TA provides users with an easy and hassle-free approach to object activation, a db4o-specific mechanism that controls object instantiation in a query result.

To explain activation it is best to look at an example of a database, that has a Tree structure with one Root object that has N nodes, each node in its turn has K subnodes and the whole structure has M levels. When running a query retrieving the root object, all the sub-objects will have to be created in the memory. If N,K and M are large numbers, chances are that an OutOfMemory exception will be triggered.

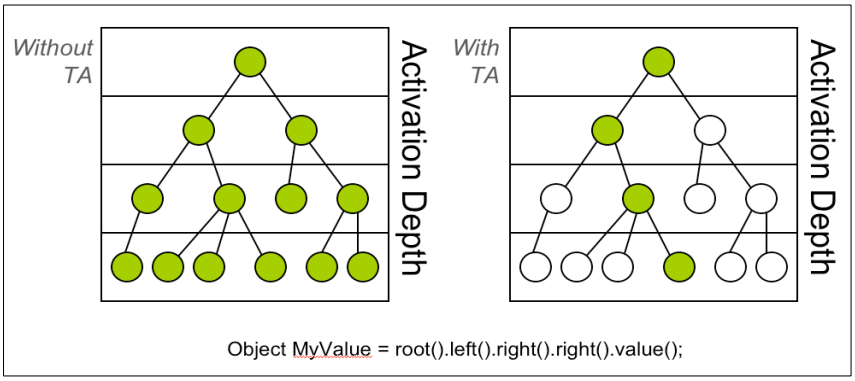
"Our application makes use of some large and complex object graphs. For Nexidia, Transparent Activation will reduce the active memory footprint of the application as well as reduce launch times and other graph-loading costs at runtime."

Joseph Duda of Nexidia, a leading vendor of speech recognition software based in Atlanta, GA.

Luckily db4o does not behave like this - when a query retrieves objects, their fields are loaded into memory (activated in db4o terms) only to a certain depth - activation depth. In this case depth means "number of member references away from the original object". All the fields at lower levels (below activation depth) are set to null (for classes) or to default values (for primitive types).

Activation works in several modes and is configurable on a database, object or field level, determining how many levels are actually instantiated when a query returns objects. In the past the activation level was defined manually but manual activation configuration can be a tricky task: instantiating too many levels results in very high memory consumption, whereas not instantiating an object, when it is needed produces a null result, which can be wrongly interpreted. This task can be especially difficult in projects with deep object graphs, when activating the whole structure results in a serious performance penalty.

This is where TA steps in: it automates the process of object activation. With TA, enabled objects are activated when they are requested. This approach makes development much easier, as the developer does not have to worry whether an object is activated or not. Another big advantage is minimum memory consumption, as any object is only loaded into memory when it is requested.



TA is implemented by using special interfaces for persistent classes, which makes them TA aware. This interface can be implemented when a class is coded, or it can be injected in the compiled bytecode. TA injection also comes in 2 options:

- Load-time instrumentation – TA awareness is injected with a special classloader, which in its turn starts the application.
- Build-time instrumentation – TA awareness in injected as part of the build process. In this case the resulted application is TA aware and the process is mainly transparent to the developer (only the build script has to be modified).

Transparent Activation Improvements

The first instantiation of TA was introduced in db4o 6.3. Important features added in db4o 7.0 are:

- Collection support (http://tracker.db4o.com/browse/COR-814), Java implementation
- Tests for all supported types and environments
- Build-time instrumentation using Ant script (http://tracker.db4o.com/browse/COR-839)
- Load-time instrumentation has improved (proper inheritance hierarchy handling COR-797 and COR-755, multiple assembly support COR-829, instrumentation of field values COR-826 and COR-827) and API extended COR-879
- Handling of mixed cases (TA + manual activation) added

Another important feature was added as a side effect of the TA implementation: db4o-tools. db4o-tools provides a simple interface to bytecode instrumentation tasks. db4o-tools is now used instead of the bloat library and is involved in Native Query optimization and Transparent Activation instrumentation. db4o-tools serves as a Java alternative to Db4oAdmin.exe on the .NET side, thus providing a convenient set up for people working on both platforms. You can read more in chapter 15 of the db4o 7.0 Java Tutorial, which is included in the download.

db4o 7.0 supports Transparent Activation for Java and db4o collections, a beta .NET version of the software is planned for December and a production ready release is expected in the first quarter of 2008 upon approval from the developer community.

More information and examples on TA can be found in the reference documentation.

Performance!

Performance always makes it into the top 3 when we ask what is important to users. To ensure product quality, specifically when it comes to performance, db4objects has integrated the open source database performance test suite PolePosition into the continuous build process. These tests will alert developers of check-ins that cause db4o performance to degrade either in terms of memory consumption, speed or database file size.

db4o 7.0 has seen numerous improvements that affect performance and memory consumption. Here are a few examples:

- A new IoAdapter cache that is now installed by default results in a 100% improvement over older versions of db4o.
- The introduction of batching of asynchronous messages significantly enhances the performance of the client/server version by a factor of 15-20x.
- Multi-Transactional-ObjectContainer (MTOC) boosts the speed of db4o in embedded Client/Server mode by an order of magnitude over previous versions
- The use of a very efficient Freespace Management System based on BTrees results in very low memory consumption at runtime and zero space-loss on abnormal terminations while still providing excellent performance.

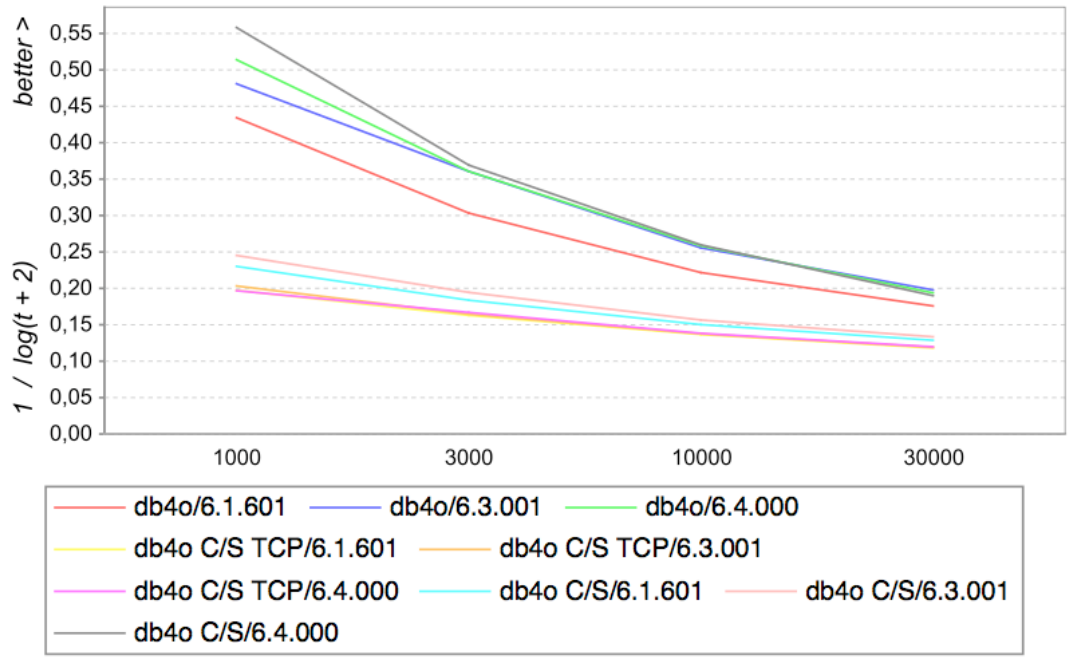
More concretely, the following PolePosition test show how db4o has improved over previous versions.

The Barcelona Circuit measures the performance of writes, reads, queries and deletes of objects with a 5 level inheritance structure.

The graph shows the improvement in read time over previous versions. The db4o core engine has improved by a factor of 1.7x for 100 selects in 30,000 objects. Thanks to MTOC, the embedded client server performance improved by factor 12x.

t [time in ms]	**selects:100 objects:1000**	**selects:100 objects:3000**	**selects:100 objects:10000**	**selects:100 objects:30000**
db4o/6.1.601	8	25	89	293
db4o/6.3.001	6	14	48	155
db4o/6.4.000	5	14	46	172
db4o C/S	157	460	1502	4748
db4o C/S	135	432	1433	4434
db4o C/S	159	397	1378	4224
db4o C/S/6.1.601	75	229	776	2370
db4o C/S/6.3.001	57	169	592	1791
db4o C/S/6.4.000	4	13	45	191

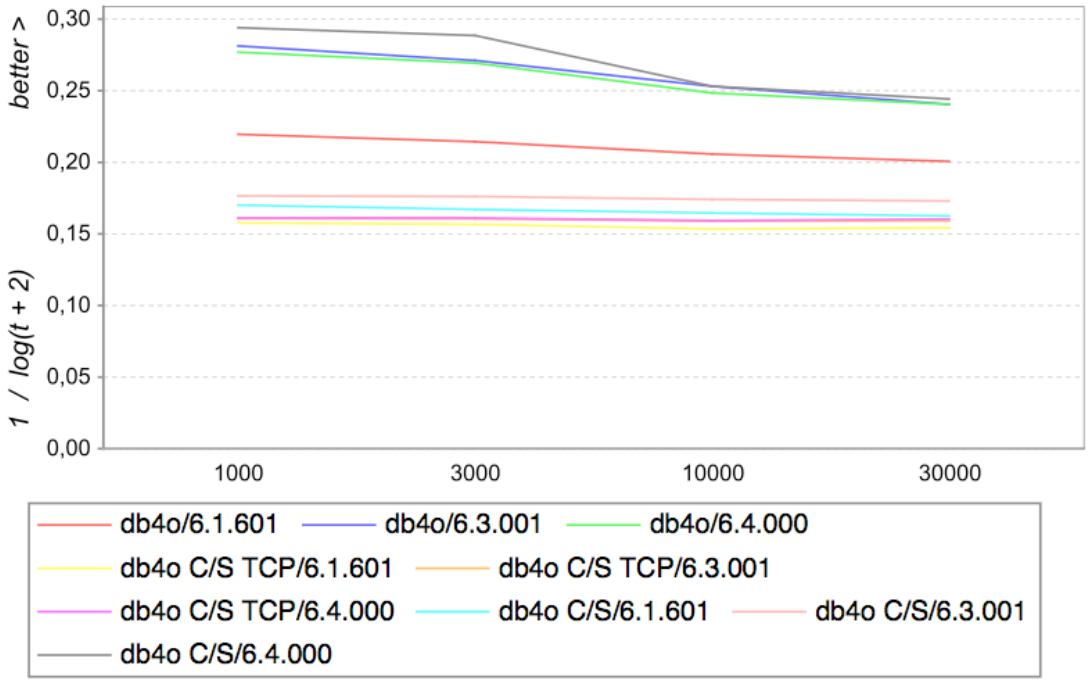
db4o C/S indicates times for Client Server connections via TCP/IP



db4o is known to perform best with complex structures but even with simple flat objects performance has improved as shown in the Bahrain Circuit. This test measures results of write, query, update and delete operations on simple flat objects individually.

The results for the indexed string query test show that db4o has has cut time in half across all tested amount of objects. For embedded client/server mode the speed has improved by more than factor 8x.

t [time in ms]	**selects:900 objects:1000 updates:100**	**selects:900 objects:3000 updates:100**	**selects:900 objects:10000 updates:100**	**selects:900 objects:30000 updates:100**
db4o/6.1.601	93	104	127	144
db4o/6.3.001	33	38	50	62
db4o/6.4.000	35	39	54	62
db4o C/S	564	589	673	651
db4o C/S	500	504	536	533
db4o C/S	494	494	531	509
db4o C/S/6.1.601	355	395	432	467
db4o C/S/6.3.001	286	290	310	322
db4o C/S/6.4.000	28	30	50	58



All performance improvements came without any trade-offs: memory consumption and database size have stayed constant across all tests and releases.

Exception Handling

db4o 7.0 includes a complete Exception Handling implementation. All expected exceptions have been added to the API documentation.

db4o was initially designed to "always work" so it did not throw runtime/unchecked exceptions, but simply absorbed them to keep the engine running at all times. This has proven to limit the ability to debug and to implement new features.
The need for more transparency lead to a dramatic change to how exceptions are handled within the db4o core. Now, db4o 7.0 includes a complete Exception Handling implementation.

This is a great step forward in terms of usability for db4o providing immediate feedback of problems and enabling new features such as Unique Constraints. Unique Constraints allow users to specify that a particular field must be unique across all instances of a Class. The reason this feature can now be implemented is because db4o can now throw a Unique Constraint Exception that will show exactly what constraint was violated along with the violating object and value.

Unique Constraints

Unique Constraints allow a user to define a field to be unique across a particular Class. This means that an object cannot be saved where a previously committed object had the same field value for fields marked as unique.

A Unique Constraint is checked at commit-time and a constraint violation will cause a UniqueFieldValueConstraintViolationException to be thrown. Multiple constraints can be defined on the same class if required. It is also possible to create specialized constraints using Commit-Time Callbacks (explained further down).

```
// First, index a field that should be unique:
config.objectClass(Item.class).objectField("_str").indexed(true);
// Second, add the constraint:
config.add(new UniqueFieldValueConstraint(Item.class, "_str"));
// open objectContainer
try {
       // do some work and save some objects
       objectContainer.commit();
} catch(UniqueFieldValueConstraintViolationException exc) {
        objectContainer.rollback();
}
```

Unique Constraints are based on commit-time callbacks which allow users to listen for commit-time events. The Unique Constraints use the "on committing" event to verify that the fields are unique before the final commit is actually executed.

Multi-Transactional-ObjectContainer (MTOC)

MTOC boosts the speed of db4o in embedded Client/Server mode by an order of magnitude. Embedded C/S is what we call a scenario where multiple client applications generate transactions concurrently against one ObjectContianer server.

MTOC improves embedded Client/Server performance by passing objects directly between "client" and "server" instead of marshalling and unmarshalling them. It keeps a local reference system for each transaction and makes sure the correct reference system is used for query processing. Metadata is shared between all transactions. Embedded Client/Server now works as if an ObjectContainer had multiple transactions – and that what defined the name: Multi-Transactional-ObjectContainer (MTOC).

The new functionality is compatible with the normal (old) embedded Client/Server API. It only requires opening an ObjectServer and connecting to it locally with ObjectServer#openClient().

The concrete improvements are:

- Opening a local client (a transaction) with ObjectServer#openClient() now takes less than 1 millisecond. Only a few objects need to be created: a wrapper around the ObjectContainer, a transaction and a reference system. No metadata needs to be created or transferred.
- Storing and loading objects is at least twice as fast for very simple objects. For bigger object graphs, the performance advantage can be 40x or more.
- Query processing can benefit from the new setup because the query processor can operate against objects that may already be in the cache.
- Loading of objects may be faster if parts of an object have already been loaded by the query processor.
- Less memory is used both on the "server" side and on the "client" side.
- Less threads are needed.

The results: embedded Client/Server is now just as fast as local mode.

Commit-Time Callbacks

Commit-Time Callbacks to external Callbacks allows users to be notified of commit-time events 'On Committing' - which is called before the commit actually happens - and 'On Commit' - which takes place after the commit is complete.

Commit time callbacks provides the ability to ensure that specific constraints are not violated whenever objects are to be committed to a database file. It also keeps the client-side object cache in sync with whatever changes are committed to a server in a client-server setting.

Two new events were added to the event interface:

- **Committing**: event subscribers are notified before the container starts any meaningful commit work and are allowed to cancel the entire operation by throwing an exception; the ObjectContainer instance is completely blocked while subscribers are being notified which is both a blessing - because subscribers can count on a stable and safe environment - and a curse because it prevents any parallelism with the container
- **Committed**: event subscribers are notified in a separate thread after the container has completely finished the commit operation

The following snippet shows how to subscribe to the Committing event and add some application specific validation:

```
private IObjectContainer _container;

public Program()
{
    _container = Db4oFactory.OpenFile("c:/test.db4o");
    EventRegistry().Committing +=
                      new CommitEventHandler(Container_Committing);
}

// CommitEventArgs includes information about
// the objects being added, updated and deleted in the
// current transaction
void Container_Committing(object sender, CommitEventArgs args)
{
    CheckSSN(args.Added);
    CheckSSN(args.Updated);
}

private void CheckSSN(IObjectInfoCollection collection)
{
    foreach (IObjectInfo info in collection)
    {
        Person p = info.GetObject() as Person;
        if (p == null) continue;

        IQuery query = _container.Query();
        query.Constrain(typeof(Person));
        query.Descend("_ssn").Constrain(p.SSN);
        IObjectSet found = query.Execute();
        if (1 != found.Count)
        {
            throw new Db4oException("SSN must be unique: " + p.SSN);
        }
    }
}

private IEventRegistry EventRegistry()
{
    return EventRegistryFactory.ForObjectContainer(_container);
}
```

Pushed Updates

A listener can now be registered for the committed event and be uses to keep all clients informed about changes committed by other clients.

The syntax to register a committed listener looks like this:

```
// Java

EventRegistry eventRegistry = EventRegistryFactory.forObjectContainer
(objectContainer);
eventRegistry.committed().addListener(new EventListener4() {
   public void onEvent(Event4 e, EventArgs args) {

   }
});


// .NET

IEventRegistry registry = EventRegistryFactory.ForObjectContainer(Db());
registry.Committed += new CommitEventHandler(OnEvent);

public void OnEvent(object sender, Db4objects.Db4o.Events.CommitEventArgs
args)
{
}
```

This is particularly interesting in scenarios were objects on all clients need to be kept in sync with committed changes, if objects are refreshed from within the listener. The local cache is always updated automatically.

When working with the new functionality in a Client/Server setup it is important to highlight that the listener will be called from a different thread than the normal application so synchronization will be required to guard the application from objects being modified behind it's back.

It also is good practice to remove listeners before shutting down clients.

OSGi Compliant Service Interface

The OSGi framework is a Java middleware that significantly extends the functionality of Java by allowing software to be structured as dynamic components which can be installed, started, stopped, updated, and uninstalled without affecting other components running in the framework. SD Times calls it "a quiet contender for the title of most important technology of the decade".

db4o includes an OSGi-compliant service interface.

db4objects partner ProSyst has adopted db4o as the standard Object Persistence Package for both their commercial and open source OSGi platforms. Under the partner agreement db4objects has adapted the native Java object database to OSGi specifications and has released "db4o for OSGi solutions", an OSGi specific distribution. ProSyst has adopted db4o as the standard Object Persistence Package of ProSyst's commercial product mBedded Server Professional Edition and has signed a reseller agreement with db4objects to provide a single point of contact for customers seeking the market's leading and most comprehensive OSGi framework with integrated persistence.

In addition ProSyst has embedded db4o as an optional package into its open source distribution mBedded Server Equinox Edition.

For more information please read the press release and visit our OSGi page.

Other improvements in db4o 7.0

Client/server improvements

Batching asynchronous messages on the client side:
When batching asynchronous messages for client/server mode the client will hold asynchronous messages until commit/rollback/or any other synchronized messages. It is user transparent, and easy to configure. To experience batched messages mode:

1. download db4o 7.0
2. configure batched mode – oc.configure().clientServer().batchMessages(true);
3. optionally configure maxBatchQueueSize.

Please note that batching messages will consume more memory than non-batched mode. To control memory consumption, db4o provides configuration for maximum queue size. If the queued messages is greater than maxBatchQueueSize, all batched messages are flushed to the server. The following commands are used to cofigure it: oc.configure().clientServer ().maxBatchQueueSize(maxSize)

Client/Server socket connection overhaul:
Requirements for a socket connection between client and server are straightforward:

- The connection should not be terminated when both client and server are still alive, even if either of the machines are running under heavy load.
- Whenever a client dies, peacefully or with a crash, the server should clean up all resources that were reserved for the client.
- Whenever a server goes offline, it should be possible for the client to detect that there is a problem.
- Since many clients may be connected at the same time, it makes sense to be careful with the resources the server reserves for each client.
- A client can be a very small machine, so it would be good if the client application can work with a single thread.

The solution that is in place now was selected for its simplicity:
A timer thread (a native Timer on .NET) on the client sends a message to the server on a regular basis and the server replies to the client immediately.

That way both the server-side message dispatcher and the client never run into timeouts as long as the system is responsive.

The elegance of this setup is shown in the source code: No exceptions need to be handled. When an exception occurs in the socket read or write messages, the sockets can be closed.

Nothing is perfect, so even this system does have two small downsides:

- We gave up on allowing completely single-threaded clients.
- If an action on the server takes longer than the socket timeout, the connection will be closed.

In the latter case the behavior of db4o can be adjusted with the following configuration settings:

```
Db4o.configure().clientServer().timeoutServerSocket()
Db4o.configure().clientServer().timeoutClientSocket()
```

An easy rule of thumb:
If clients are disconnecting, the timeout value should be raised.
If clients crash frequently or the network is very unstable, the values should be lowered, so resources for disconnected clients are freed faster.

db4o fully tested in multi-core environment
The entire db4o build, integration and test infrastructure has now been moved to multi-core machines. Our build process is now running on a quad-core server. What that means is that the current db4o continuous build and all future releases are tested for both single and multi-core environments.

Moving to that environment uncovered issues that were not apparent when using single core machines:

- Configuration calls were not threadsafe, due to 'KeySpecHashtable4'
- The central classes for String conversion, 'LatinStringIO' and 'UnicodeStringIO' were not reentrant for multiple threads because they held state between calls.

Thanks to our policy of running continuous regression tests and our goal to maintain a zero critical bug status, these bugs have now been removed.

Improving C# readability
A majority of the C# sources for db4o.net are generated from the Java version. This is possible because of the similarities of the two platforms, and of the two languages. To simplify the task we created a converter that takes care of translating the Java sources to plain C# sources. So far the code we generated was readable by the C# compiler but it was a little harder to read for humans. Now, with the new and improved converter, we organize the imports for every C# source file. It provides perfectly valid C# code that is much more readable.

Naming Convention Explained
We have implemented a new naming convention for the releases with the following structure:

[major].[minor].[iteration].[SVN revision]

[major]
Major releases generally indicate a major change in the code, for example a change in architecture or a major innovation such as Transparent Activation.

[minor]
A turn of the minor number means that a build contains a substantial piece of completely new functionality in comparison to a previous minor version.

[iteration]
This number shows in which week a build was produced. The db4objects development process uses weekly iterations. Weekly milestones are created and completion of tasks is tracked against these weekly milestones. The iteration number will help users find out which fixes are included, by looking at the change-log in Jira (http://tracker.db4o.com/).

[SVN revision]
This number is generated by our SVN source control system. It relates directly to a checkin.

Everything we produce goes online as the Continuos Build after all regression tests pass. Note: the "date added" date in our download center does not mean anything – it is a shortcoming of our forum system.

Registered users of db4o can download db4o 7.0 from the developer site at http://developer.db4o.com/files/folders/db4o_70/default.aspx.

Sales Contact
Submit inquiries for a commercial license of db4o 7.0:
http://www.db4o.com/commercial/purchase/enquiry.aspx
Email: sales@db4o.com
Corporate Headquarters:
db4objects, Inc., 1900 S Norfolk Street, Suite 350, San Mateo, CA 94403, United States
Phone: +1 (650) 577 2340 or **Fax** (+1) 650 240-0431
Website: www.db4o.com