

WHITE  
PAPER  
SERIES

---

# *The ObjectSpark® API for Java*

An Overview

DataTern, Inc.  
330 Madison Ave 31<sup>st</sup> Floor  
New York, NY 10017  
(212) 210-6221  
[www.datatern.com](http://www.datatern.com)



---

© 2009 Datatarn Inc.

The information in this Document is the intellectual property of Datatarn, Inc.

This Document may not be reproduced in whole or in part, by any means, without the written consent of Datatarn, Inc. All Rights Reserved.

The Software described in this Document and all copies of the Software are the property of Datatarn, Inc.

The Software is provided under a license agreement containing restrictions on use and disclosure. The software contains trade secrets of Datatarn, Inc. Reverse engineering of the Software is prohibited.

If provided to the U.S. Government, this Software and this Document are provided with Restricted Rights. Use, duplications, or disclosure by the Government is subject to restrictions set forth in FAR 52-227-19 (c) (2) (June 1987) or DOD FAR Supplement 252.227-7013 (c) (1) (ii) (June 1988) or the NASA FAR Supplement as applicable.

Contractor/Manufacturer is Datatarn, Inc.

### **Trademarks**

Datatarn is a registered trademark and ObjectSpark, the ObjectSpark logo, and the Datatarn logo are trademarks of Datatarn, Inc.

Microsoft and Visual Studio are registered trademarks and Windows, Windows NT, Visual Basic, Visual C++, SQL Server, and Microsoft Transaction Server are trademarks of Microsoft Corporation in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

IBM and DB2 are trademarks of International Business Machines Corporation.

Rational Rose is a trademark or registered trademark of Rational Software, Inc., in the United States and/or other countries.

Sun, Sun Microsystems, the Sun logo and JavaBeans are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

All other product and company names mentioned in this document are trademarks or registered trademarks of their respective companies or organizations.

DataTarn, Inc.  
330 Madison Ave 31st Floor  
New York, NY 10017  
(212) 210-6221  
[www.datatarn.com](http://www.datatarn.com)

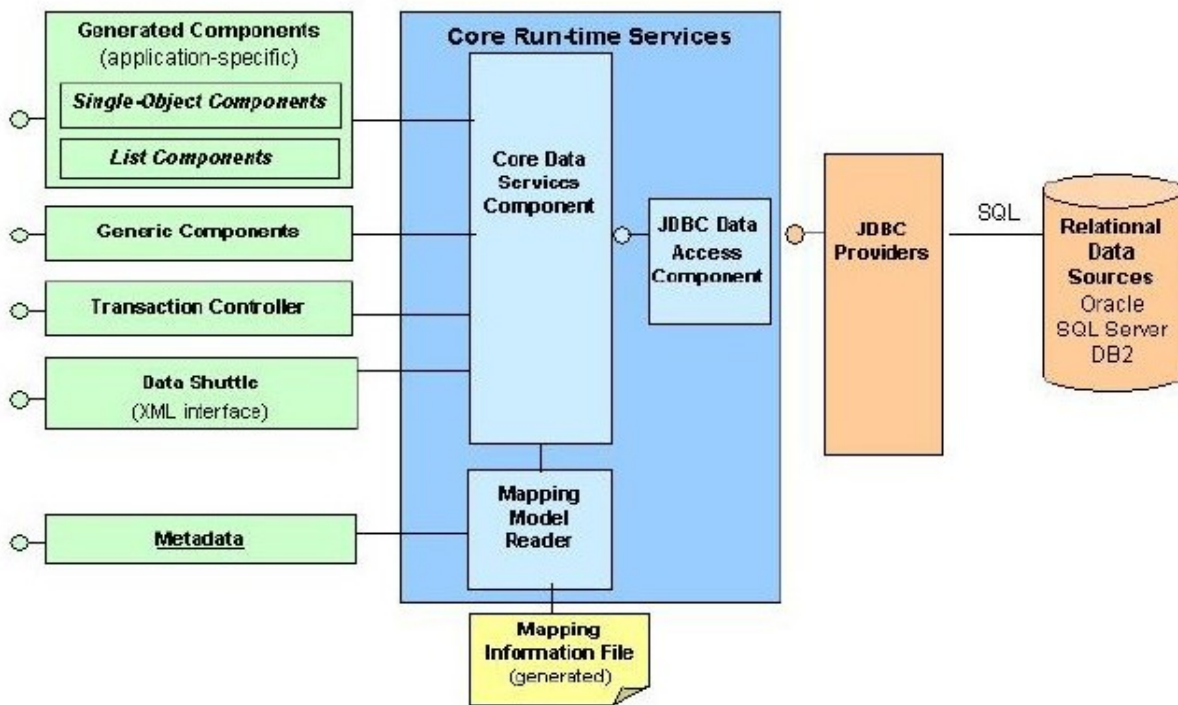
---

# ObjectSpark API for Java

The ObjectSpark data services architecture in Java consists of:

A combination of *generated* and *standard* JavaBeans components (packages) that provide various data services to the clients that call them.

A set of Core Run-time Services components, which use the data stored in a map file to dynamically generate the commands needed to provide transparent access to data stored in one or more of the supported data sources on the back end.



## Generated Data Components

The ObjectSpark Designer generates a Java package for each package defined by an object model in your map. Each Java package contains two generated data components for each mapped class in the package, one for manipulating single objects and one for manipulating lists.

The generated components are implemented according to the mapping you defined between your application object model and one or more data sources. The resulting component libraries provide a simple, application-specific interface that enables almost any developer to create, read, update and remove persistent objects in corporate data stores without having to know much about SQL syntax or the underlying database organization.

In the generated component libraries, single object components are named **Dsl<class>** and the list components are named **Dsl<class>List**, using a convention where **<class>** represents the name of a class in your object model. For example, if your object model defines **Product** as a persistent class, the ObjectSpark Designer generates a single-object component called **DslProduct** and a list component called **DslProductList**.

Every generated component typically provides two public methods used to get and set attributes. These are defined based on the name and data type of the attributes of the corresponding mapped class. For example, related class attributes are generated to hold references to other objects based on the relationship definitions in your object model. The related class attributes get their names from the role names associated with the relationships.

The generated components also implement all the methods that are defined by the standard ObjectSpark data component interfaces, which include the methods that handle all data access and persistence of data on the back end.

---

## Single-Object Components

Your generated single-object components (Dsl<class> -- for example, DslEmployee) inherit methods defined in **RtObject**, while the generic single-object component (DslObj) provides a generic implementation of those methods.

Single-object components let you retrieve and manipulate the persistent data for a single entity within your application. Single-object components provide support for both read and write operations on the data source.

The single-object component interface enables you to:

- Get an existing object from the data source based on either a key value or a query.
- Read and modify scalar attributes of an object
- Navigate relationships to access related objects
- Modify relationships held by the object
- Save the state of an object to the persistent data store, creating or updating records based on commands defined in the mapping model
- Remove an object from the persistent data store, deleting or updating records based on commands defined in the mapping model
- Synchronize the state of your local object with its persistent state in the data store
- Read various flags that tell you about the state of the local object relative to its persistent state in the data store.

---

## List Components

Your generated list components (Dsl<class>List – for example, DslEmployeeList) inherit methods defined in **RtList**, while the generic list component (DslList) provides a generic implementation of those methods.

List components provide a convenient interface for manipulating groups of objects of the same type. A lists component can be instantiated either as a *standalone* list, or as *related* lists in the context of the single object that owns the relationship that defines the membership list. (To learn more about the

difference between standalone lists and related lists, see "Standalone or Related List -- What's the Difference?".)

List components retrieve and manipulate data using a recordset-like structure in which each row holds the attribute values for a member of the list. The number of rows indicates the number of members in the list. The column structure is based on the attributes and relationships defined for the class that defines the list type. As with any recordset, a list has a pointer that points to the row from which data is currently being read. Since a row effectively represents an object, we often refer to the current row as the *current object*. The list structure provides support for iterating through the list and reading the data for each member by reading values from the current row.

All lists implement methods that let you:

- Populate a list from the data source

- Read attribute values from the current row

- Iterate over the currently-visible rows in the list to call methods on each row

- Set filter criteria that determines which rows are visible to other forms of list manipulation, such as iteration

- Determine the number of currently-visible rows in the list

- Set the list position to a particular row based on an index or search criteria

- Instantiate the single object referenced by the current row of the list

List components do not allow you to directly modify the state of any objects represented by the list. You can only modify objects through the single-object interface. To modify the state of a list member, you can use the list component interface to instantiate any object represented by the list as a single object.

It is important to keep in mind that list components are *not* the same as object collections. They hold the persistent data for the objects represented by the list; not the objects themselves. For read operations, the list object structure is more efficient than an object collection.

## Other Components

---

### **Core Run-Time Services**

The ObjectSpark Core Run-time Services components are the internal "engine" within the ObjectSpark data services layer that:

- Reads the mapping information stored in the map file; a map is read into memory once at server start-up and is shared by all components that depend on that map.

- Uses the mapping information to dynamically construct and execute the commands required to access the appropriate data sources on the back end.

- Provides concurrent access to data by implementing support for synchronizing the disconnected state of an object with its persistent state under an optimistic locking scheme.

- Coordinates transaction processing across data sources using internal transaction management services.

- Establishes and manages data connections using data connection pooling and security services.

All front-end components of the ObjectSpark data services interface require the support of one or more of these core run-time services.

---

## ***The Metadata Component***

The ObjectSpark Metadata Components provide support for reading metadata from a generated deployment map, which contains application-specific information about your data component interfaces and their relationship to other data components in your application. This information can be used in combination with the Generic Data Components to dynamically create and manipulate data components at run time.

---

## ***The Generic Data Components***

As an alternative to using the generated data components, ObjectSpark provides two *generic* data components, **DslObj** and **DslLst**, which enable you to write generalized code that works with *any* mapping. These components are in the **com.objectspark.dsl.objs** package. The generic components provide methods for creating and manipulating application-specific single objects and lists based on metadata that is passed into the generic component methods. The required metadata can be read dynamically from a map using the Metadata Component interface.

---

## ***The Security Session Component***

ObjectSpark provides a security session component, **DslSession**, which provides support for using pass-through security to establish connections to the data sources to which your components are mapped. An implementation of the Security Session component is provided in every generated data component library.

The security session component lets you pass a database user ID and password straight through to the database server in the context of a security session object.

---

## ***The Data Shuttle***

The ObjectSpark Data Shuttle provides data services clients with faster access to the data for persisted objects. The Data Shuttle is a stateless component that offers both read and write access to the data store with support for transaction management.

## Support for Passing Components as XML Documents

Relative to other data formats, XML documents are extremely portable and offer significant advantages as a format for transferring data over a network. Several components of the ObjectSpark Data Services Interface provide support for handling data in the form of XML documents.

Both *generated* data components and *generic* data components provide methods for XML support:

**getAsXML** returns an XML document that holds the structure and state of the object

**setFromXML** sets the state of the object based on the data contained in the XML document that is passed into the method

**DataShuttle** component provides the **execXML** method, which takes and returns XML documents as arguments.

The documents must be structured using tags defined by the ObjectSpark XML Document Type Definition (DTD). The object structures defined by the XML documents can be nested. In other words, the document may not only hold data for an object, but also for any objects that are either referenced or contained by the object that is represented by the document. The XML documents also contain the information needed to determine whether data values have changed since the document was created.