

WHITE  
PAPER  
SERIES

---

*The ObjectSpark® API for COM*  
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 Datatern, Inc.

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

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

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

The Software is provided under a license agreement containing restrictions on use and disclosure. The software contains trade secrets of Datatern, 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 Datatern, Inc.

## Trademarks

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

DataTern, Inc.

330 Madison Ave 31<sup>st</sup> Floor

New York, NY 10017

(212) 210-6221

[www.datatern.com](http://www.datatern.com)

# ObjectSpark API for COM

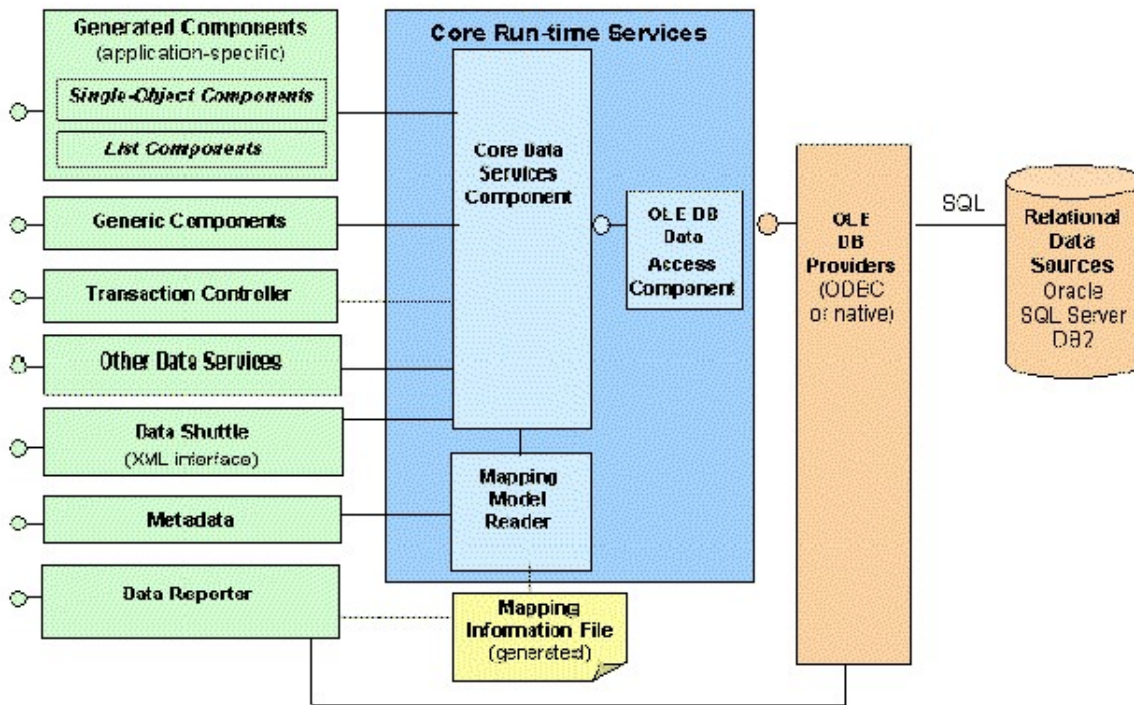
Once you have your ObjectSpark data services layer implemented and deployed, you will need to develop the application components that are the clients of the data services layer. This document provides a brief overview of the API used by those client components.

The ObjectSpark data services interface provides support for clients to store and retrieve data passed in the form of standard COM data types, arrays, ADO recordsets, and XML documents. Within a COM-based architecture, you can implement data services clients using any programming language that supports COM, such as Visual Basic, Visual C++, Visual J++, and scripting languages, such as VB Script, that are used to code ASP pages. In most cases, the data services clients are business components that use data components to access corporate data in the execution of business processes. In other cases, the client may be an ASP page.

The ObjectSpark data services architecture consists two major component groups:

A combination of *generated* and *standard* libraries 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.



The ObjectSpark Data Services API is provided by the following components:

Generated Data Components – provide an application-specific interface for storing and retrieving persistent application entities. Includes both single-object and list components.

Generic Data Components – provide a generalized implementation of the interface provided by the application-specific data components that are generated from your map.

ObjectSpark Transaction Controller – coordinates transactions using MTS services.

Other Base Data Services Components – provides error components, and support for object and attribute collections.

Data Shuttle – provides support for storing and retrieving data in the form of an XML document.

Metadata Components – provide support for dynamically reading application data component metadata from a binary map at run time.

Data Reporter – provides an OLE DB provider interface that uses a map to read complex data structures. Available for COM components only.

Security Session Component – provides support for using pass-through security.

Of all the components that make up the ObjectSpark Data Services Interface, the most commonly used part of the data services API – the DSL Object Interface – is provided by either the generated data components or the generic data components, depending on whether you want to work with application-specific components or a more generalized interface.

## The DSL Object Interface

You will most likely use the DSL Object Interface to implement most of the data persistence functions in your application. The DSL Object Interface is provided by either the generated data components or the generic data components. The interface presents two types of data component:

Single Object – used to access a *single* persistent entity within your application.

List – used to access homogenous *groups* of persistent entities.

The ObjectSpark Development Tool generates a single object and a list component for each of the mapped classes in your application object model. The generated data components provide an application-specific programming interface. The ObjectSpark Generic Data Component library also provides a generalized implementation of a single object and a list, which you can use to access the same data.

Lists may be instantiated as a standalone lists or in the context of a single object as a related list. For a standalone list, the default membership of the list is determined by the class definition and mapping. For a related list, membership is determined by the relationship definition held by the object that owns the list.

---

### **Single Object Components**

Your generated single-object components and the generic single-object component (DslObj) both provide support for instantiating single objects.

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 properties 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.

### Properties and Methods of Single-Object Components

Properties	Methods
value	Find
<attribute>	Search
isNull	Refresh
isModified	Save
isRetrieved	Remove
classInfo	ClearState()
txnController	GetAsArray
entityType	GetAsXml
underlyingValue	GetAsAdo
isNew	SetFromArray
isDirty	SetFromXml
forceUpdate	SetFromAdo
	GetAttrValues
	GetErrors

---

### List Components

Your generated list components and the generic list component (DslLst) both provide support for instantiating list objects.

List components provide a convenient interface for manipulating groups of objects of the same type. A list 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.

List components retrieve and manipulate data using a recordset-like structure in which each row holds the property 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 properties and methods that let you:

- Populate a list from the data source
- Read property 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
- Use the data in the list to populate a collection of objects

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. However, if you need a collection, all list components (in the COM API only) provide a method that makes it easy to turn any list into a collection. ObjectSpark provides the **DslCollection** component in the Base Data Services library for manipulating object collections.

## Properties and Methods of List Components

Properties	Methods
value	Find
<attribute>	Search
isNull	Refresh
isModified	ClearState()
isRetrieved	GetAsArray
classInfo	GetAsXml
txnController	GetAsAdo
entityType	SetFromArray
currentObject	SetFromXml
maxCount	SetFromAdo
count	GetAttrValues
listCount	GetErrors
position	GetQuery
isEmpty	SetQuery

**Properties**

isValid  
sortOrder  
filter  
relType

**Methods**

MoveFirst  
MoveLast  
MoveNext  
MovePrev  
RetrieveData  
GetAsCollection  
GetQueryCount  
Add  
Remove()  
Modify

## The Metadata Components

The ObjectSpark Metadata Components provide support for reading metadata from a generated binary 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.

## Transaction Management and Security

The ObjectSparkTransaction Controller component provides support for defining transactions in your business services layer. The ObjectSparkTransaction Controller provides the same methods as the Microsoft Transaction Server transaction interface (ITransactionContext). However, the ObjectSpark component, **DslTxnController**, provides additional support for attaching to an existing transaction. This feature allows you to create and manipulate objects outside the context of a transaction and only initiate a transaction when a transaction is needed to save changes to the data source.

The **DslTxnController** component provides an alternative interface to the ITransactionContext interface provided by the MTS API for implementing transactions in the business services layer. The **DslTxnController** component implements the IDslTxnController Interface interface.

## Data Shuttle - XML Document Support

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

The Data Shuttle interface currently provides support for data to be exchanged as well-formed XML documents using tags defined by the ObjectSpark data type definition.

## Base Data Services and Other Core Components

In addition to the Data Shuttle component, the ObjectSpark Base Data Services library includes a number of components that provide other base data services.

**Error Components** – provide support for manipulating any collection of errors that is returned by any COM method called on ObjectSpark data components.

**Object Collections** – provides support for manipulating single objects as an object collection.

**Attribute Collections** – provide support for manipulating attributes of a single object in a structure that includes the attribute metadata.

## Generic Data Components and Metadata Interface

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. 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 as Strings. The required metadata can be read dynamically from a binary map using the Metadata Component interface.

**DslObj** is a standard component of the DslObjs library that provides a general interface for manipulating single objects by passing application specific metadata into the interface as Strings.

**DslLst** is a standard component of the DslObjs library that provides a general interface for manipulating list objects.

## Data Reporter

The ObjectSpark Data Reporter is an OLE DB provider interface that presents an alternative to using the Data Component API for reading data from corporate data stores. In cases that require reports using complex data structures (e.g. a list of objects and related object data), you may find it both easier and faster to use this recordset-based query interface for data retrieval.

The Data Reporter:

- Provides you with a standard read-only interface that conforms to the Microsoft OLE DB provider specification. The Data Reporter query interface supports a simple subset of the standard SQL language.

- Allows you to construct queries based on the data structures defined in your application interface. The Data Reporter uses the binary map generated by the Development Tool to convert the query to use the data structures defined in your data source. The result is an ADO recordset that is structured based on the application data structures defined in your original query.

- Provides support for using standard OLE DB consumers to access corporate data stores by constructing queries based on the classes and relationships defined in your application object model. There are a number of development tools on the market today, such as Seagate Crystal Reports™, which require an OLE DB provider interface on the back end to provide an easy-to-use data

reporting interface on the front end. The ObjectSpark Data Reporter allows you to take advantage of such tools *and* take advantage of the mapping interface that is one of the key benefits of ObjectSpark. ObjectSpark software allows developers to access data without having detailed knowledge of the physical structure of the database.

## 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. To use the security session component, the security type defined in the mapping model must be **Pass-through** or **Pass-through and Default**.

For COM components, ObjectSpark also supports other security implementation models, which include support for role-based and user-based security.