

CACORE OBJECT CART VERSION 1.0

Programmer's Guide



**NATIONAL[®]
CANCER
INSTITUTE**

Center for Biomedical Informatics
and Information Technology

This is a U.S. Government work.

November 11, 2008

Revision History

The most current version of this document is located on the MDR GForge website: https://gforge.nci.nih.gov/docman/?group_id=369. Other related documents can also be found at this location on the GForge

Revision History

<i>Revision Date</i>	<i>Author</i>	<i>Summary of Changes</i>
05/07/2008	Denis Avdic	Initial Draft of document
06/04/2008	Denis Avdic	Final Draft of document
11/10/2008	Bronwyn Gagne	Conversion of document to CBIIT standard.
11/11/2008	Bronwyn Gagne	Final Release of document

Table of Contents

About This Guide	1
Purpose.....	1
Scope.....	1
Topics Covered.....	1
Text Conventions Used	2
Credits and Resources	2
Chapter 1 Object Cart Overview	3
Object Cart Features	3
Object Cart Service	4
Serializer/Deserializer	4
Classification/Typing	4
Cart Manager	4
Architecture Overview.....	4
Persistence and Application Service Tiers	5
Chapter 2 Workflow for Object Cart Integration	7
Integrating the Object Cart Client	7
CartManager	8
ObjectCartClient.....	9
Using the Cart	9
Using Object Cart with Java Objects.....	9
Using Object Cart with XML data	10
Expiration Date and Deleting Carts.....	10
Chapter 3 Deployment Model	11
Thin Client Deployment	11
Configuration	11
Service Deployment.....	11
Configuration	12
Thick Client Deployment.....	12

About This Guide

This preface introduces you to the *caCORE Object Cart v1.0 Programmer's Guide*.

Topics in this section include:

- [Purpose](#) on this page.
- [Scope](#) on this page.
- [Topics Covered](#) on this page.
- [Text Conventions Used](#) on page 2
- [Credits and Resources](#) on page 2

Purpose

This document provides all the information application developers need to successfully integrate their application with the caCORE Object Cart.

The Object Cart was developed to provide data caching and sharing capability across applications. Since many applications and services rely on other services, the Object Cart will improve workflow and data sharing among applications in a seamless way.

Scope

This document shows how to deploy and integrate the Object Cart client with your application. It covers basic configuration when deploying the thin (remote) client, the service itself, and the thick client.

Topics Covered

This brief overview explains what you will find in each chapter of this guide.

- [Chapter 1, Object Cart Overview](#) provides an overview of the Object Cart architecture and concepts and how they apply to your application.
- [Chapter 2, Workflow for Object Cart Integration](#) provides the necessary information for successfully integrating your application with the Object Cart.
- [Chapter 3, Deployment Model](#) describes how to deploy your own Object Cart service, including information on both Thin Client and Thick Client deployments as well as deployment of the service itself.

Text Conventions Used

This section explains conventions used in this guide. The various typefaces represent interface components, keyboard shortcuts, toolbar buttons, dialog box options, and text that you type.

Convention	Description	Example
Bold	Highlights names of option buttons, check boxes, drop-down menus, menu commands, command buttons, or icons.	Click Search .
<u>URL</u>	Indicates a Web address.	http://domain.com
text in SMALL CAPS	Indicates a keyboard shortcut.	Press ENTER.
text in SMALL CAPS + text in SMALL CAPS	Indicates keys that are pressed simultaneously.	Press SHIFT + CTRL.
<i>Italics</i>	Highlights references to other documents, sections, figures, and tables.	See <i>Figure 4.5</i> .
<i>Italic boldface monospace type</i>	Represents text that you type.	In the New Subset text box, enter <i>Proprietary Proteins</i> .
NOTE:	Highlights information of particular importance.	NOTE: This concept is used throughout this document.

Credits and Resources

caCORE MDR Development and Management Teams			
MDR Development Team	Other Development Teams	Documentation	Program Management
Denis Avdic ¹	Satish Patel ¹	Denis Avdic ¹	Bilal Elahi ³
	Steve Hunter ¹	Charles Griffin ¹	Charles Griffin ¹
		Bronwyn Gagne ⁴	
¹ Ekagra Software Technologies	² Science Applications International Corporation (SAIC)	³ National Cancer Institute Center for Bioinformatics	⁴ Lockheed Martin

Contacts and Support	
NCICB Application Support	http://ncicb.nci.nih.gov/NCICB/support Telephone: 301-451-4384 Toll free: 888-478-4423

Chapter 1 Object Cart Overview

The caCORE Object Cart is a convenient service that enables access to a shareable data cache. It provides a number of ways for this data to be stored, either as a serialized Java Object or an XML document qualified with an XML schema present in the [Global Model Exchange \(GME\)](#).

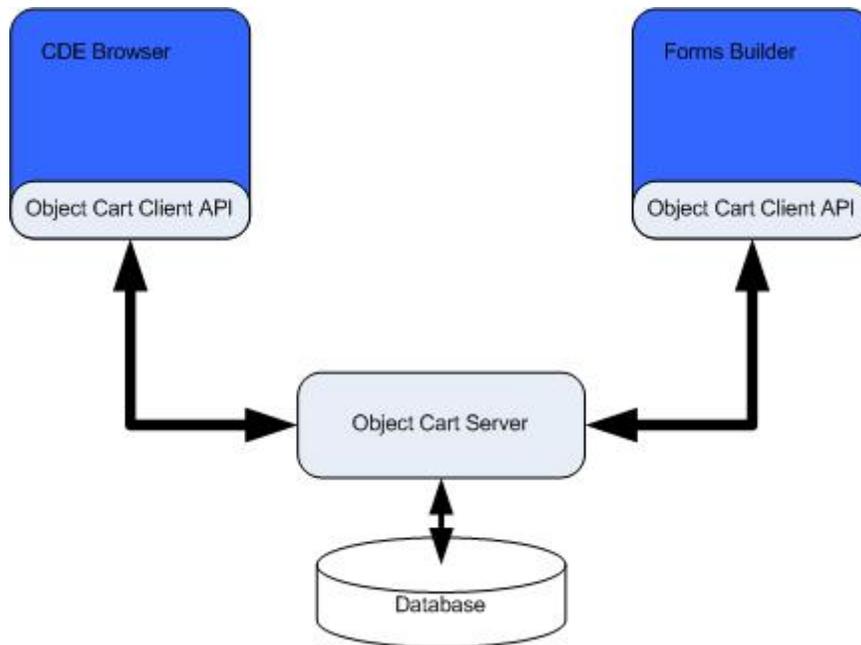


Figure 1-1 Overview Diagram of Object Cart functionality

Object Cart Features

The Object Cart provides the following capabilities:

1. Creation of a unique cart within a classification scheme, based on a user id and a cart name. The classification scheme, or type, is used to distinguish groups of carts associated with a particular set of applications.
2. Association of a pre-existing cart with a different user id.
3. A serializer and a deserializer for Java objects following the Java Bean pattern.
4. XML validation for XML data with a valid XML Schema present in the GME, and identified with a GME namespace name.
5. Data persistence.

The Object Cart APIs provide the following major components of the Object Cart feature set:

- Object Cart Service

- Serializer/Deserializer
- Classification/Typing
- Cart Manager.

Each of these components is described briefly below.

Object Cart Service

The service used for all cart operations is specific to the classification of the carts that are being manipulated. The operations include *create*, *delete*, and *retrieve* actions for a particular cart. There are no update operations for the cart, as the API encourages adding and removing objects from the cart using the Client methods.

Serializer/Deserializer

Data is stored in the carts in XML format and the Object Cart provides a simple interface for serialization of Java objects. Specifically, the Object Cart supports serialization of objects following the Java Bean specification.

The interface is positioned so that storing and retrieving a Java object is still only one operation, where the serialization happens before the object is sent to the service and after it is received from the service.

Classification/Typing

Along with the serialization support, the Object Cart provides classification of carts. This enables separation of object collections based on application, usage, or other criteria. In this first version of the Object Cart the classifications available are limited to those present on the server.

Cart Manager

Cart manager is a singleton holder of multiple client services. This reduces the number of open services to one per classification. While it is not necessary to use the Cart Manager, it simplifies access to and containment of the underlying services.

Architecture Overview

Object Cart is a caCORE SDK 4.0 generated system, extended to allow writing to as well as reading from the underlying service.

The infrastructure of an SDK generated system exhibits an n-tiered architecture with client interfaces, server components, backend objects, data sources, and additional backend systems. This allows the system to divide tasks or requests among different servers and data stores, isolating the client from the details of where and how data is retrieved from different data stores. The client receives information from backend objects. These back-end objects then communicate directly with data sources, in this case a relational databases (using Hibernate). SDK generated systems also perform common tasks such as logging.

For more information on SDK generated systems please refer to the *caCORE SDK 4.0 Developer's Guide*.

Persistence and Application Service Tiers

As mentioned before, the Object Cart API is an extension to an SDK generated system. In addition to the SDK provided query capabilities, the Object Cart provides write capabilities for its domain objects, namely the Cart and the CartObject.

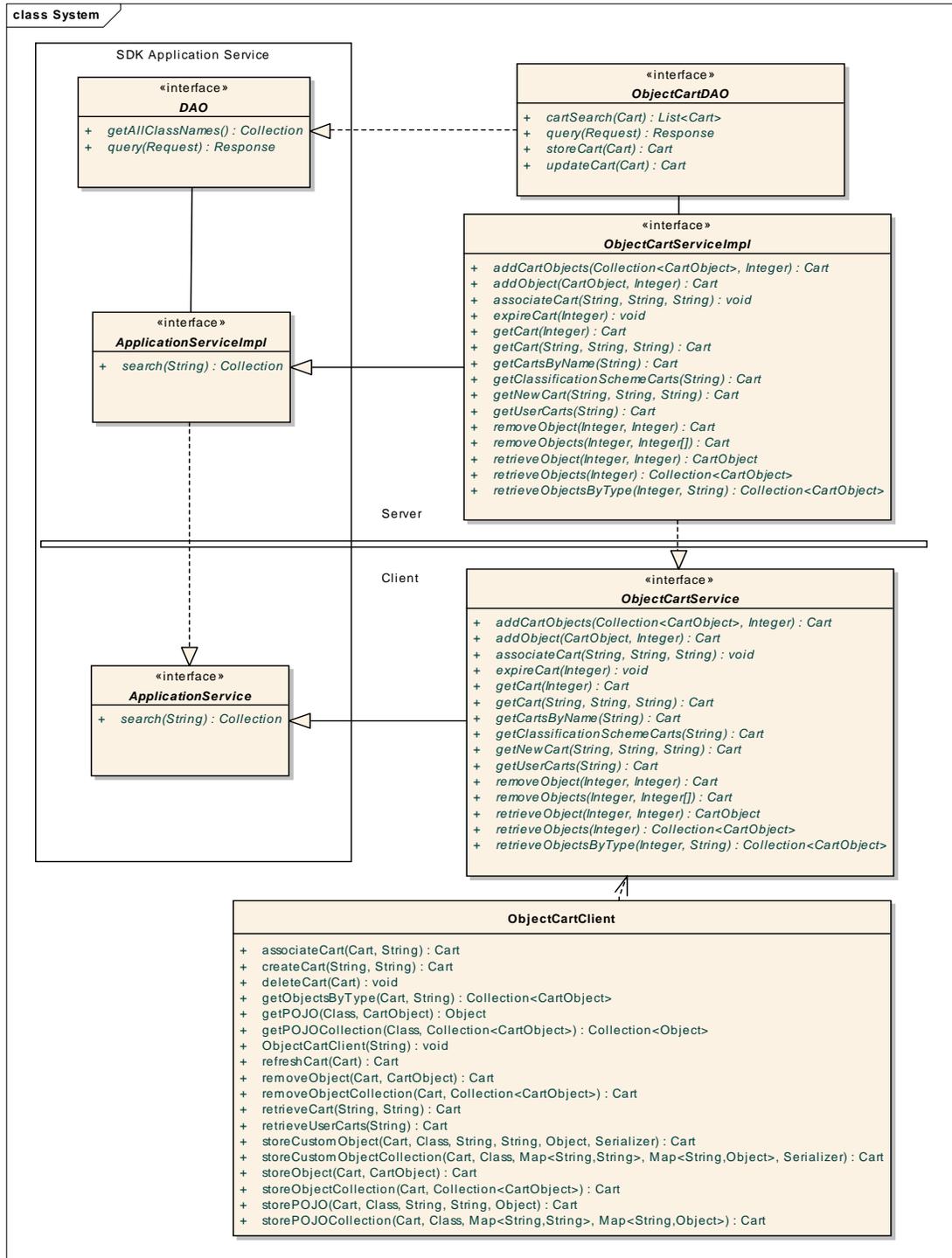


Figure 1-2 Object Cart Service and Client class diagram

The ObjectCartClient, shown in the figure above, is used as the main interface to the Object Cart Service and Server. The client contains all the methods related to the lifecycle of the Cart and Cart Object, as well as customized retrieval methods utilizing the SDK query system. Additionally the client has methods used for storing and retrieving Java Objects utilizing either a custom or provided serializer.

Chapter 2 Workflow for Object Cart Integration

This chapter outlines the fundamental steps, both strategic and technical, for successful Object Cart integration.

In order to integrate the Object Cart into your application:

1. **Decide which objects/data you would like to share with another application or make available for storage in general.**

If the data is already in XML format and has an XML schema present in the GME, the application can use the Cart with its CartObjects as wrappers for data. Otherwise you would be storing Java Objects, which would be serialized to XML. If the Java objects being stored are not serializable by the Castor API directly, you may need to write your own Serializer based on the interface provided.

2. **Determine an identification strategy.**

Since the carts are identified by user id and name, the Object Cart functions best within a Single Sign On environment. If the application is not part of such an environment, then a strategy must be devised as to how the user accessing the cart is being identified in all applications sharing the data. This could be accomplished by drawing on a common user directory, using a unique id sent between applications through URLs, or through another technique altogether.

3. **Create or adapt a user interface for browsing and interacting with the cart.**
4. **Integrate Object Cart Client code with your application and the cart user interface.**

Integrating the Object Cart Client

The Object Cart stores data in its XML representation. Your application might handle the data either as XML directly or represented in Java Objects. Thus there are two possible paradigms for utilizing the Object Cart Client and the Object Cart domain objects. With either approach it is recommended that you use the provided Client Manager to manage Object Cart Clients and, in turn, Object Cart Services. Additionally JavaDocs for Object Cart 1.0 are available on the GForge site at: https://gforge.nci.nih.gov/docman/index.php?group_id=369&selected_doc_group_id=4252&language_id=1.

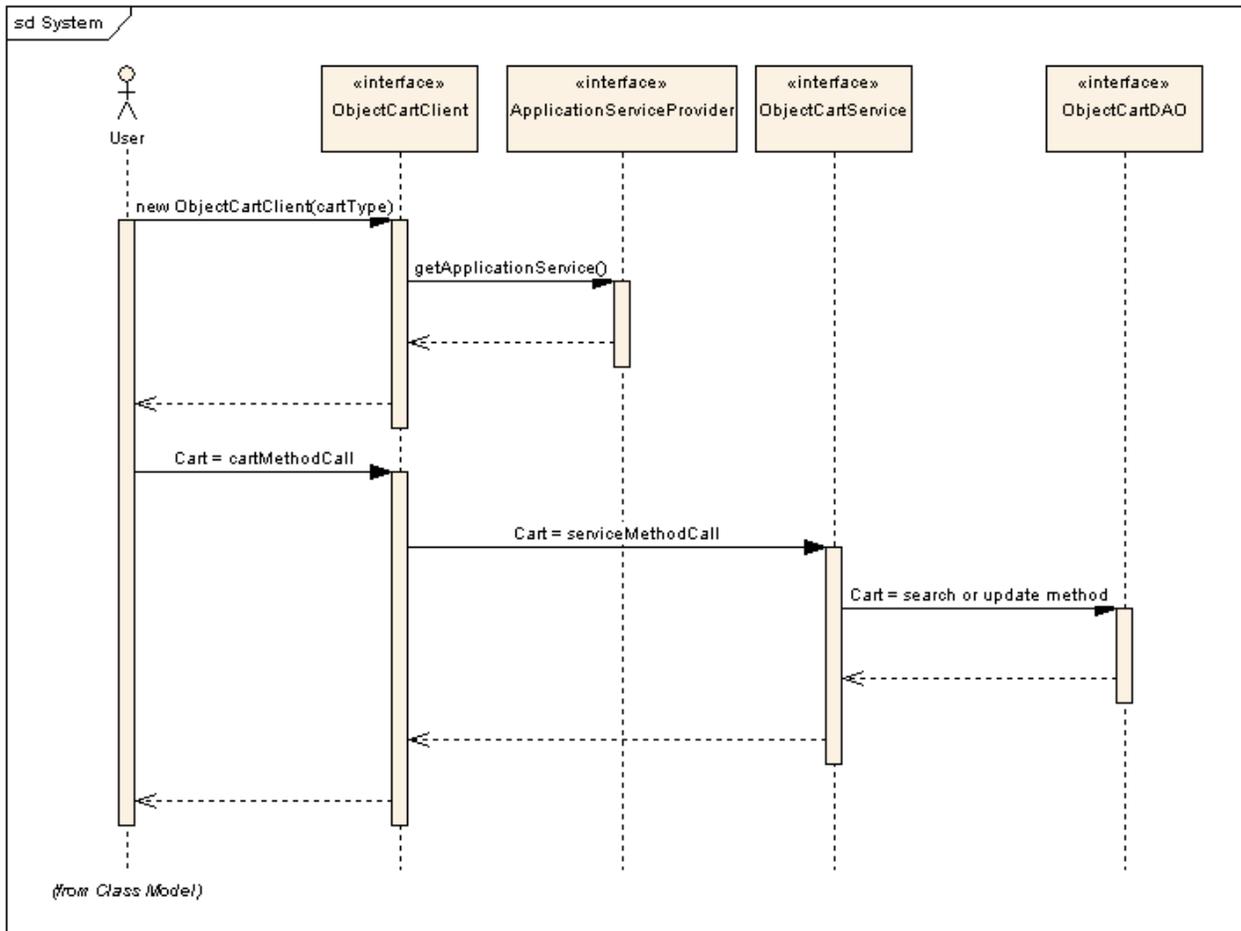


Figure 2-1 Java API communication with Object Cart Service

CartManager

CartManager is a singleton that needs to be initialized with all the classification schemes your application will be using in a String array. There is a minimum of one scheme necessary to access Object Cart Services successfully, although there might be many types already present in the system. Please contact the Object Cart team in order to determine if there is a type already present that might fit your needs.

The following is an example of initialization of the CartManager:

```

CartManager cartManager = CartManager.getInstance();
String[] classificationSchemes = {"First type", Constants.CARTSCHEME};
try{
    cartManager.initClients(classificationSchemes);
} catch(ObjectCartException oce){
    oce.printStackTrace();
}
    
```

ObjectCartClient

Once the `CartManager` is initialized, you can start utilizing the `ObjectCartClient`. In order to streamline the service calls, each client can interact with Carts classified with one `Classification Scheme`. To access a client simply call the `CartManager.getClient` method and supply it with the necessary classification scheme type as determined by you or the Object Cart team.

```
ObjectCartClient client = cartManager.getClient("First Type");
```

Once you have a client, you can perform all cart operations as described in the `JavaDocs`, located in `GForge`.

Using the Cart

Each cart within a particular classification scheme is uniquely identified by its name and the associated user id. Thus, in order to create a cart or retrieve one, you must simply supply the name/user id pair.

Depending on whether you are using the `CartManager` or the `Client` directly, your `createCart` method call would look like either of the following:

```
Cart myCart = client.createCart(userId, cartName);
```

```
Cart mySecondCart =
cartManager.getClient(Constants.CARTSCHEME).createCart(userId, cartName);
```

There are two major ways of retrieving a cart. The first one relies on the same cart name/user id pair, while the second one relies on the cart's unique identifier. The first method should be used if you do not have a reference to the cart needed. If the reference is present, you can use the client method `refreshCart`, which refreshes the cart based on the unique cart id.

```
Cart myCart = client.retrieveCart(userId, cartName);
```

```
myCart = client.refreshCart(myCart);
```

Using Object Cart with Java Objects

The process for serializing Java Objects into the Object Cart is integrated into the client by the way of the `Serializer` interface provided with the `ObjectCart Client API`.

If your objects follow the `Java Bean` specification you can use our `Castor`-based serializer. Use the provided `storePOJO` and `getPOJO` methods. Otherwise you should create your implementation of the `Serializer` interface and use the `storeCustomObject` and `getCustomObject` methods to store and retrieve your objects.

If you use the provided `POJO` methods, in addition to providing the actual object, you will need to provide the `Class` of the object being used in the serialization/deserialization process. Additionally you will need to provide the `nativeID` and `displayText` that will be exposed in the `CartObject` enveloping the

serialized data. The `nativeID` will assist in identifying the wrapped object and the `displayText` should contain text used when displaying the object within ObjectCart UI. This eliminates the need to deserialize the data for identification and display purposes.

```
String displayText =
item.getItem().getLongName()+" "+item.getItem().getDescription();

myCart = client.storePOJO(myCart, CDECartItem.class, displayText,
item.getId(), item);

Collection<CDECartItem> myCollection = (Collection<CDECartItem>)
getPOJOCollection(CDECartItem.class, myCart.getCartObjectCollection());
```

Using Object Cart with XML data

If you have data that is already in XML format, you can use the ObjectCart domain objects directly, without resorting to serialization. In this case use the `CartObject` directly where you can set all attributes to the desired values. Thus `setData` is used for actual XML payload whereas the `nativeID` is used to identify the wrapped object within the application, and `displayText` contains the text used when displaying the object within the ObjectCart UI. Then you can use the Client methods `setObject` and `getObject`, along with their collection analogues to store your data in a particular cart.

Expiration Date and Deleting Carts

The carts are not deleted directly by the client. Instead the client sets the expiration date for a cart at the current time, and thus makes it eligible for deletion by the maintenance thread that runs in parallel to the Object Cart service. The maintenance thread deletes all expired carts that have been inactive for a period of three days. By default, a cart does not have an expiration date. This allows the user to have a permanent storage until such time as the (or underlying application) decides to set an expiration date for that cart.

If using the cart for temporary storage, be sure to set the expiration date to either the default (three days) or to a specific date provided through the client.

Chapter 3 Deployment Model

The Object Cart APIs facilitate the transmission of data across a network to a central location. These APIs use Java, Spring, and Hibernate for this purpose.

Thin Client Deployment

The Object Cart APIs and domain objects are available as a JAR that needs to be placed in the class-path of the application. Along with this JAR, there are many supporting Jars on which the Object Cart API depends. These extra Jars are supplied as part of the Object Cart distribution and should be added into the folder `<application-web-root>\WEB-INF\lib`.

NOTE: The Object Cart service depends on having a correct version of `Hibernate3.jar` on the client side. This means there may be a need for you to update your code if you are using a different version of Hibernate. Similar constraints are present on versions of Castor and Spring distributions. Additional changes may be necessary to accommodate Object Cart if your application is using Castor and/or Spring.

Configuration

<i>File</i>	<i>Description</i>
<code>application-config-client.xml</code>	The XML file containing the configuration data for the client service. This configuration data may conflict with an existing configuration file if your application is using another SDK generated service. Should this conflict occur, merge the two files. There should be no need for any other adjustment.

Table 3-1 Thin-Client Configuration

Service Deployment

If you wish to deploy your own Object Cart service, once you have provided configuration details specific to your own production environment, you can build your own web archive file (WAR).

Once you've downloaded the ObjectCart source code, you will notice the `build.properties` file in the root project folder. In this properties file, change the database connection details for a particular environment to the one on which you will be deploying your service. Below is an example:

```
PROD_DB_CONNECTION_URL=jdbc:mysql://hostname:portnumber/databasename  
PROD_DB_USERNAME=databaseusername  
PROD_DB_PASSWORD=myspasswordiscomplex
```

These correspond to the local, development, QA, stage, and production environments, and have `ant` build targets to match (local-build, dev-build, qa-build, stage-build, and prod-build).

Additionally there is a property that identifies your service endpoint. Be sure to match it to the URL of your web server (such as JBoss) as this is used to generate configuration files for remote clients using your service.

```
PROD_SERVER_URL=http://hostname:portnumber/objectCart
```

In order to provide support for the persistence layer you will need to execute the `DBPrepMySQL.SQL` script on your database, found alongside the `build.properties` file. This will create all necessary tables in the correct configuration to support ObjectCart operations. Currently the Object Cart only officially supports MySQL, however that is expected to change with further testing of the current release or in a future release of the Object Cart.

Once you have provided the build properties and prepared the database, you can use `Ant` to package your ObjectCart application. Execute `ant` on the target `prod-build` (or other environment) and copy the `objectCart.war` file from the `/dist` directory to your web server deployment directory (`{server name}/default/deploy` in the case of JBoss).

The Object Cart service will be available after restart.

In the `/dist/remote` directory you will find the files necessary for the remote (thin) client deployment. After making the files available make sure to follow the instructions outlined in the [Thin Client Deployment](#) section above.

Configuration

<i>File</i>	<i>Description</i>
<code>build.properties</code>	The XML file containing the database and server properties information for the Object Cart service.

Table 3-2 Service Configuration

Thick Client Deployment

If you are planning to deploy your application utilizing the Object Cart service in the same web container as the Object Cart service itself, you can use the thick client deployment configuration for your application.

After deploying the Object Cart service as described in the Service Deployment section above, simply use the files from the `/dist/local` directory in your application instead of the files provided in the `/dist/remote` directory. There is no additional configuration necessary.

If deploying outside of the web container, make sure to implement the `ExpiredCartCleaner` and the `CleanerThread`, which take care of removing expired carts from the database. The `ExpiredCartCleaner` must be inside of the web container to function, so in its absence the developer must create a mechanism to start the `CleanerThread` on startup.