



ENOVIA SmarTeam | Dassault Systèmes

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

[www.3ds.com](http://www.3ds.com)

# **ENOVIA SmarTeam**

## ***SMARTTEAM I-PLATFORM SDK***

### ***Programmer's Guide***



# Copyright

© Dassault Systèmes, 2004, 2008. All rights reserved.

CATIA, ENOVIA, SMARTEAM and the 3DS logo are registered trademarks of Dassault Systèmes or its subsidiaries in the US and/or other countries.

**PROPRIETARY RIGHTS NOTICE:** This documentation is the property of Dassault Systèmes. This documentation shall be treated as confidential information and may only be used by employees or contractors of the Customer in accordance with the terms of the End-User License Agreement accepted by Customer.

Any use of the Licensed Program contained in this media or accompanying it, is subject to the terms of the End User License Agreement accepted by Customer. The Licensed Program is protected by international copyright laws and international treaties. Unauthorized use, reproduction and/or distribution of any of the Licensed Program, or any part thereof, may result in severe civil and/or criminal penalties, and will be prosecuted to the maximum extent possible under the law. Company names and product names mentioned herein are the property of their respective owners and certain portions of the Licensed Program contain elements subject to copyright owned by these entities. See the Documentation CD provided with the Licensed Program for details and/or additional terms and conditions relating to these entities.  
Part Number: DVS-A3-180007



## About this Guide

This document describes the SmarTeam i-Platform SDK and how to use it. It is primarily intended for developers of client applications that use the services of the SmarTeam i-Platform Server.

This manual is intended to be used with the “SmarTeam i-Platform Client Library Reference Guide” provided with the product.

For more information, see the SmarTeam Object Model Programmer’s Guide and SmarTeam Corporation’s web site (www.smarteam.com).

It is assumed that the reader is familiar with Java™ and JavaScript (ECMAScript 262 Edition 3).

JavaScript is documented by Microsoft at:

[<http://msdn.microsoft.com/scripting>](http://msdn.microsoft.com/scripting).

The ECMAScript documentation is at:

<http://www.ecma-international.org/publications/standards/Ecma-262.htm>.

Familiarity with XML is helpful but not essential (due to the encapsulation provided by the client library.)

# Table of Contents

Copyright	i
About this Guide	i
<b>1 INTRODUCTION</b>	<b>1</b>
Overview of SDK Components	1
Embedded-Scripts Client Library	2
SmarTeam i-Platform Application Server	2
SmartInetUtils Library	3
Network Architecture	4
<b>2 EMBEDDED-SCRIPTS CONNECTION PACKAGE</b>	<b>5</b>
Overview of Objects and Interfaces	5
IConnectionModel Interface	5
Using the IConnectionModel	7
Uploading and Downloading Files	9
Uploading an Attached File	9
SmartInetUtils.SmDownloadManager	10
<b>3 WRITING A SCRIPT</b>	<b>12</b>
Scripts	12
The Context Object	13
Common Tasks	15
Using Parameters in a Script	16
Supported Data Types	17
<b>4 RECORD LIST PACKAGE</b>	<b>18</b>
Record List Objects	18
ImmutableRecordList	19
ImmutableRecord	21
IRecordList	22
Package Events	23

Common Tasks	25
<b>A EMBEDDED-SCRIPTS ENGINE</b>	<b>27</b>
Embedded-Scripts Engine	27
Examples	28
Context Expiry	35





# 1 Introduction

The SmarTeam i-Platform SDK provides client applications with the ability to access SmarTeam functionality provided by the SmarTeam i-Platform Server. This functionality includes database operations, data structures including record lists and objects, performing queries, and managing workflow, as well as most operations available through the SmarTeam API.

The SmarTeam functionality is accessed through short scripts (referred to in this guide as *embedded scripts*) that are embedded in requests sent by the client application to the SmarTeam i-Platform and executed on the server.

## Overview of SDK Components

The three major components provided by the SDK are:

- SmarTeam i-Platform Application Server – a server application providing SmarTeam services via a variety of protocols and communication methods. This package includes the Embedded-Scripts Engine – a Web Service for executing scripts sent by the client and returning results to the client.
- Embedded-Scripts Client Library – a Java-based client side library for generating, encoding and sending client requests to the server and receiving results. This library contains the Embedded-Scripts Connection Package and the Record List Package.
- SmartInetUtils Library – a server-side library providing support for operations related to Internet protocols, such as file uploading and downloading.

The primary function of this document is to describe how to use these three components.

This chapter provides a brief overview of the SDK components and the architecture used.

[Chapter 2](#) Embedded-Scripts Connection Package  
Embedded-Scripts Connection Package  
Embedded-Scripts Connection Package  
Embedded-Scripts Connection Package

Package provides a description of the Embedded-Scripts Connection Package of the Embedded-Scripts Client Library and also includes the SmartInetUtils Library.

[Chapter 3](#) describes how to write client scripts.

[Chapter 4](#) describes the Record List Package of the Embedded-Scripts Client Library.

[Appendix A](#) details objects of the SmarTeam i-Platform Application Server.

## **Embedded-Scripts Client Library**

The Embedded-Scripts Client Library is a collection of Java packages that includes the following main packages:

- [Embedded-Scripts Connection Package](#)
- [Record List Package](#)

### **Embedded-Scripts Connection Package**

The Embedded-Scripts Connection Package provides all client-server connection functionality, including encoding client requests using SOAP. The Embedded-Scripts Connection Package is described in Chapter 2.

### **Record List Package**

The Record List Package is a Java package, which allows the client to work with a record list data type similar to that of SmarTeam. The Record List Package is described in Chapter 4.

## **SmarTeam i-Platform Application Server**

The SmarTeam i-Platform Application Server is a server application that provides SmarTeam services via a variety of protocols and communication methods. This package includes the Embedded-Scripts Engine – a Web Service for executing scripts sent by the client and returning results to the client. The SmarTeam i-Platform Server is described in [Appendix A](#).

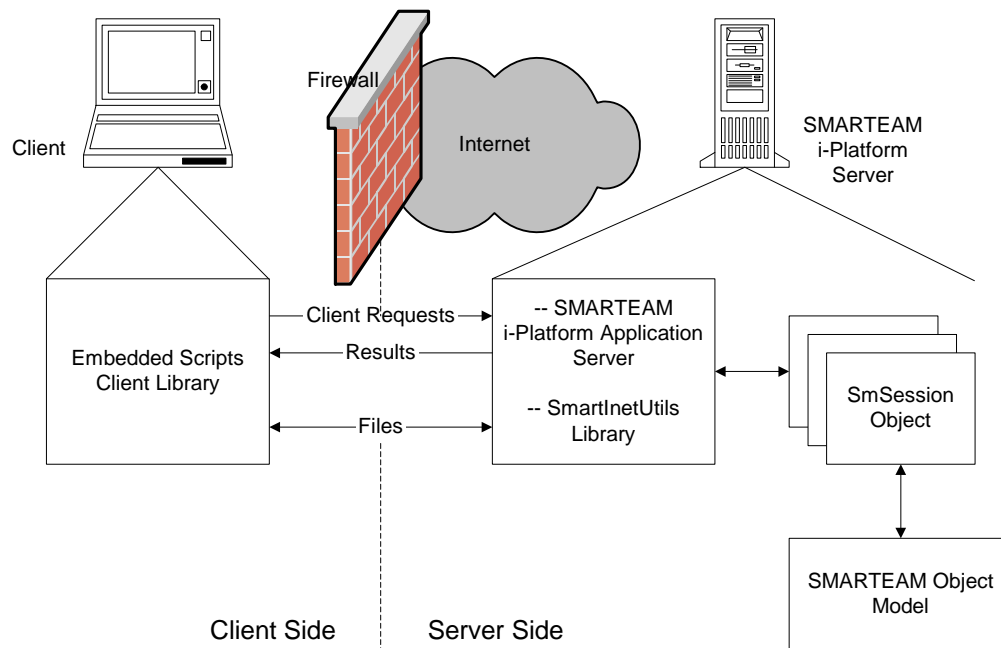
## SmartInetUtils Library

The SmartInetUtils Library on the server side includes uploading and downloading functionality, including:

- Managing the uploading of a file to the server. The section “[Uploading an Attached File](#)” on page [9](#) describes how to upload files using a script. The upload object in SmartInetUtils Library is not used directly by the user and therefore is not described in this document.
- Managing the downloading of a file from the server. The download object `SmartInetUtils.SmDownloadManager` is normally used by the SmarTeam Life Cycle mechanism, but can also be used directly by the user. This object is described on page [10](#).

## Network Architecture

The network architecture of the SmarTeam i-Platform SDK components is illustrated in [Figure 1](#).



*Figure 1 SmarTeam i-Platform SDK Components*

## 2 Embedded-Scripts Connection Package

This chapter describes the Connection Package of the Embedded-Scripts Client Library and how to use it to send an embedded script to the SmarTeam i-Platform Server.

### Overview of Objects and Interfaces

This section describes the objects and interfaces used to connect to the server.

#### IConnectionModel Interface

The IConnectionModel interface is the main interface used to connect to the server. This interface abstracts the actual protocol used to connect the client with the server.

The interface includes the methods:

Method	Description
createContext (connectionString)	Creates a context (session) for the client to work with SmarTeam
execute (scriptLang, script, params)	Encodes and sends a script to SmarTeam with parameters. A file can be uploaded as a parameter using the AttachedFile class.
releaseContext()	Releases the context  <b>Note:</b> If the Context is not released by the releaseContext method, it will terminate automatically after its expiry period (See <a href="#">Appendix A</a> ).

Method	Description
<code>setContextID(contextID)</code>	Set Smarteam Context ID. Use this method if you have already created a context, or have a context ID from a different source.

**Note:** These commands execute in synchronous mode: program execution waits for server response and method return before continuing. Invoking these commands from the main user interface thread may require using threads, or another mechanism for implementing an asynchronous mode, to prevent the main thread from blocking. For an example, see the Session class in the EmbeddedScriptsClientDemo package.

## **SOAP-Based Implementation**

An implementation of the `IConnectionModel` Interface using SOAP is provided in the class `ConnectionModel` in the package `com.smarteam.client.api.embeddedscripts.soap`.

The SOAP-based implementation complies with the SOAP Schema.

For information about the SOAP schema, see <http://schemas.xmlsoap.org/soap/envelope/>.

For the SOAP specification, see <http://schemas.xmlsoap.org/soap/encoding/>.

The SOAP-based implementation is useful when client and server communicate over the network using HTTP. Using HTTP allows client and server to communicate even over a firewall.

In addition to supporting the HTTP protocol, the SOAP implementation also supports the Secure Sockets Layer (SSL) using the HTTPS protocol for communicating with the server in a secure manner.

Implementations of the `IConnectionModel` Interface that cover additional scenarios will be provided in future releases of this SDK. In addition, you can create your own implementation of this interface.

## Using the IConnectionModel

This section describes how to use the SOAP-based implementation of IConnectionModel to send scripts to the SmarTeam i-Platform Server.

### Step 1 – Creating a Connection String

Using the class SmarTeamConnectionString:

```
//creating a new connection string object
SmarTeamConnectionString conString = new SmarTeamConnectionString();
//initiating the connection string object
conString.setUserName(userName);
conString.setUserPassword(userPassword);
// Database details can also be set using the method setDatabaseID or
// setDatabaseName
conString.setReplicaID(replicaID);
```

### Step 2 – Creating a Connection Model

Using the class ConnectionModel, the implementation of the IConnectionModel interface:

```
String targetURL = "http://www.myserver.com";
IConnectionModel connectionModel = new ConnectionModel(targetURL);
String context = connectionModel.CreateContext(conString.toString());
```

#### Remarks:

1. If you need to connect using HTTPS in a secure manner, change http in targetURL to https.
2. If you need to connect to a different port, add [PortNumber] to the end of targetURL , for example, http://www.myserver.com:667 connects to port 667 using HTTP.

### **Step 3 – Executing a Script**

```
String scriptLang = "javascript";
//A short script that retrieves the Database name
String script = "var DatabaseName = Context.SmSession.Database.Name;\n"+
               "Context.Result = DatabaseName;\n";
Parameter[] params = null;
Object result = connectionModel.execute(scriptLang, script, params);
System.out.println("Database name is: " + (String)result);
```

See [Uploading an Attached File](#) on page 9 for an example of how to execute a script with parameters.

### **Step 4 – Release Connection Model**

```
connectionModel.releaseContext();
```

In a typical client-server interaction, steps 1 and 2 are executed once to initiate the session, step 3 is executed as many times as necessary in the session and step 4 is executed once to release the session.

You must call `releaseContext` to release the session; otherwise it will continue to exist and successive sessions can accumulate on the server.

**Note:** If the Context is not released by the `releaseContext` method, it will terminate automatically after its expiry period, normally 30 minutes (See [Embedded-Scripts Engine](#)).



## Uploading and Downloading Files

This section discusses uploading and downloading files. The ability to upload files together with the request to the server, and to download files from the server, is important for many scenarios, such as performing life-cycle operations on documents.

### Uploading an Attached File

You upload a file attached to a SmarTeam object by using the client `AttachedFile` object as a parameter in a client script, as shown in the example below.

#### *Example – Checking in an Object with an Attached File at the Client:*

The following client Java program sends a script to instruct the life-cycle mechanism on the server to check in an object, where the object's attached file is to be uploaded from the client site and placed in the server vault. The program defines the attached file as a script parameter, using the `AttachedFile` object. The script invokes the server-side life-cycle sample object `CheckIn` function, using the attached file parameter as the file parameter in the `CheckIn` function.

The example uses the class `Parameter`, which represents a pair: (parameter name, parameter value). For more information, see [Using Parameters in a Script](#) on page 16.

```
String scriptLang = "javascript";
String fileToUpload = "c:\\mywork\\workfile.ext";
Parameter[] params = new Parameter[4];
params[0] = new Parameter("CLASS_ID", 356);
params[1] = new Parameter("OBJECT_ID", 8225);
// The following parameter uses AttachedFile to cause the file to
// uploaded to the server
params[2] = new Parameter("uploaded_file",
    new AttachedFile(fileToUpload));
params[3] = new Parameter("NOTE", "Any notes for the new revision");
String script = "var LCHelper =
    Context.CreateObject('SmartLifecycleDemo.SmLifecycleHelper');\n" +
    "LCHelper.Init(Context.SmSession);\n" +
    "LCHelper.CheckIn(Context.Params('CLASS_ID'),
    Context.Params('OBJECT_ID'), Context.Params('uploaded_file'),
```

```
Context.Params('NOTE')));\n";  
// The script does not return a result  
connectionModel.execute(scriptLang, script, params);
```

**Note:** To prevent malicious scripts from performing unauthorized operations on the server, a client script cannot take any direct actions on an uploaded file other than to retrieve its relative file name. However, the client script can invoke components installed on the server side, which are authorized to manipulate an uploaded file, for example, SmLifecycleHelper as in the above program. SmLifecycleHelper is installed in the SmartLifecycleDemo package.

## SmartInetUtils.SmDownloadManager

This object manages downloading a file from the server.

It has the following methods

Method	Description
Copy(FullPath, LifeTime)	<p>Copies the file specified by FullPath to a temporary location, and returns a valid URL for accessing the file.</p> <p>The temporary location and the file copy are automatically removed at the end of the session. To discard the file before the session end, use the Discard method.</p>
Discard(URL)	<p>Notifies the server that the file or folder tree indicated by the specified URL are no longer needed and may be deleted. The URL must have been created through a call to Copy or CreateLocation.</p>
CreateLocation(LifeTime)	<p>Creates a temporary location on the server and a URL pointing to that location.</p> <p>Files placed in the specified location are accessible through the URL by appending the file name at the end of the URL. The filename must be URL-encoded, using, for example, EncodeURL.</p>
EncodeURL(URL)	<p>Encodes URL</p>

## ISmDownloadInfo

This object provides destination download information:

It has the following properties

Properties	Description
URL	URL by which the client accesses the local server download file directory. The URL includes the final “/” character.
FullPath	Full path of the local server download file directory

## Examples

The following script example directs SmarTeam to copy a file from the vault to a local server download file directory. The URL of the download directory is passed to the client as a result. The client, who now has the URL, can download the file at his discretion.

```
// Note: WorkingObject is a file-managed ISmObject
var SmDownloadManager, SmDownloadInfo;
// create SmDownloadManager object
SmDownloadManager =
Context.SmSession.GetService('SmartINetUtils.SmDownloadManager');
// create local server download directory that has the session lifetime
SmDownloadInfo = SmDownloadManager.CreateLocation(0)
var Directory = SmDownloadInfo.FullPath;
//Retrieve the file name from the working object
FileName = WorkingObject.Data.ValueAsString('FILE_NAME');
//Copy file to download directory
WorkingObject.CopyFileFromVault(FileName, Directory);
var EncodedFileName = SmDownloadManager.EncodeURL(FileName);
//Return URL of download directory to client
Context.Result = SmDownloadInfo.URL + EncodedFileName;
```

## **3 Writing a Script**

This chapter describes the following topics:

- Writing a script
- Using Context properties and methods in a script
- Using Parameters in a script
- Data types returned from a script

### **Scripts**

A client application can access the functionality of SmarTeam on a server by submitting a script, which is run by the SmarTeam i-Platform Server in the SmarTeam Sandbox.

Scripts can be written in JavaScript version 3 (ECMAScript 262 Edition 3). JavaScript is recommended because it is standardized (by ECMA).

JavaScript is documented by Microsoft at:

[<http://msdn.microsoft.com/scripting>.](http://msdn.microsoft.com/scripting)

The ECMAScript documentation is at:

[< http://www.ecma-international.org/publications/standards/ECMA-262.HTM>.](http://www.ecma-international.org/publications/standards/ECMA-262.HTM)

## The Context Object

The Context object provides the script with access to SmarTeam functionality. The Context object is exposed to the script as a global object. You access the properties and methods of the Context object in the script in the form:

```
Context.[Context property or object]
```

For example:

```
Context.SmSession
```

```
Context.Params
```

### Context Properties

The following Context properties are available:

Property	Description
SmSession	Refers to the SmSession associated with the Context.  Example: Context.SmSession.ObjectStore.ObjectsFromData(query.QueryResult, true);
Params	This is a Windows Scripting Dictionary object. Therefore, the two member function obj.Keys() and obj.Items return arrays which VBArrays. To use them in JavaScript, you have to convert them to JavaScript arrays with the toArray() call. Example:  //get javascript array from VBArray var keys=Context.Params.Keys().toArray(); var items=Context.Params.Items().toArray();
Result	Used to return the script result to the client.  Example: Context.Result = objsData

NullObject	<p>Equivalent to a null interface. This is useful in languages, such as JavaScript, that do not provide an equivalent to “Nothing”.</p> <p>For such languages, the value of this property can be used as a parameter to methods that require it as a parameter.</p>
Constants	<p>Provides access to the SmarTeam API constant values for use by scripts. The Constants object provides property-based access to the different SmarTeam libraries, and from there to the actual constant value, in the following manner:</p> <p><code>Context.Constants.constant-name</code></p> <p>For example:</p> <p><code>Context.Result=Context.Constants.coNo;</code></p>

## Context Methods

The following Context methods are available:

Method	Description
Log	Writes a log entry to the server events log.
CreateObject	<p>Creates a COM Automation object.</p> <p>Example: <code>Context.CreateObject('SmRecList.SmRecordList');</code></p> <p><b>Note:</b> This CreateObject context method is the only way you can create a COM Automation object in the script. The built-in support in the scripting language for creating such objects (JavaScript's ActiveXObject) is disabled to prevent malicious scripts from performing unauthorized operations on the server.</p>

## Common Tasks

The following example shows the use of the Context properties and methods.

### Context:

#### Using Context properties and methods in a an embedded-script

The following is an example of a Javascript-based script sent to the server for execution

```
var query = Context.SmSession.ObjectStore.NewQuery();
var qDef = query.QueryDefinition;
var role = 'F';
qDef.Roles.Add(1, role);
qDef.Select.Add('CLASS_ID', role, false);
qDef.Select.Add('OBJECT_ID', role, false);
qDef.Select.Add('STATE', role, false);
query.Run();
//Creates an object within the context
var objsData = Context.CreateObject('SmRecList.SmRecordList');
//Refers to the SmSession created by the context
var objs =
Context.SmSession.ObjectStore.ObjectsFromData(query.QueryResult,true);

var i, n;
var o;
for (i = 0; i < objs.Count; i++) {
    o = objs.item(i).Clone();
    o.AddAllAttributes();
    o.RetrieveAttributes();
    n = objsData.AddRecord();
    objsData.CopySmRecord(o.data, n);
}
//Tells the system to return objsData to the client in the ExecuteResponse
method.
Context.Result = objsData;
```

## Using Parameters in a Script

A client can provide parameter values to a script running on the server, similar to using arguments in a function call.

The `IConnectionModel.execute` method packages the parameters specified by its `params` argument and sends them together with the script to the server.

The client creates the `params` argument using the class `Parameter`. The class `Parameter` encapsulates the actual parameter object and its name as a pair. The constructor of the `Parameter` class accepts the parameter name and the parameter value, which is of type `Object` (you can pass primitive types by casting them to an `Object` first.) The `params` argument is an array of `Parameter` objects, one item for each parameter required.

The script accesses a parameter value through the `Context` method `Params`, which takes a parameter name as an argument. For example:

```
Date = Context.Params('DateParam');
```

The method returns the value of the parameter corresponding to the parameter name `DateParam` and can be used anywhere in the script.

Parameters can have all supported data types except the URL, `RecordList` and XML types (see Table 1 for supported data types).

Example:

```
// simple script with parameters example
Parameter[] params = new Parameter[2];
// create parameter with name "A" and value 5
params[0] = new Parameter("A", new Integer(5));
params[1] = new Parameter("B", new Integer(5));

String script = "var sum;\n"+
    "sum = Context.Params('A') + Context.Params('B');\n"+
    "Context.Result = sum;\n";
```



## Supported Data Types

A script can return a value to the calling client by assigning the value to the Result property of the Context object.

For example:

```
Context.Result = 7
```

The client receives the value 7 as a response to the script execution request.

The following table lists the server-side value types that can be assigned to the Result property, and their equivalent Java types as received by the client.

*Table 1 Supported Return Data Types*

Server Type	Client Java Type
Integer	java.lang.Integer
Short	java.lang.Short
Byte	java.lang.Byte
Float	java.lang.Float
Double	java.lang.Double
Boolean	java.lang.Boolean
Date	java.sql.Date
Time	java.sql.Time
Datetime	java.sql.Timestamp
String	java.lang.String
URL	com.smartteam.client.api.URL (URL and caption)
Record list	com.smartteam.client.api.IMutableRecordList
XML	org.w3c.dom.NodeList

## 4 Record List Package

This chapter describes the Record List Package of the Embedded-Scripts Client Library.

The major objects are:

- RecordList
- Record
- Columns
- Column

Each of these objects is read-only, but each has a corresponding mutable (read/write) object, designated by the prefix Mutable.

### Record List Objects

The JavaClient package provides two record list types: IMutableRecordList and IRecordList.

The purpose of these Record List objects is to allow the client to work with record list data objects that are similar to those in the SmarTeam API (see SmarTeam Object Model Programmer's Guide for information about the RecordList data type in SmarTeam).

For example, a client can request that SmarTeam return information to him in a Record List (see the [Examples](#) in [Embedded-Scripts Engine](#).)

The IMutableRecordList is provided for working with and modifying the data in the record list. The IRecordList is provided when you want to read data while preserving it.

The two Record List objects are similar in structure. The main difference is that you can create read-only objects from the read-write objects but not vice versa.

## IMutableRecordList

The following figure shows the object diagram for the IMutableRecordList Object.

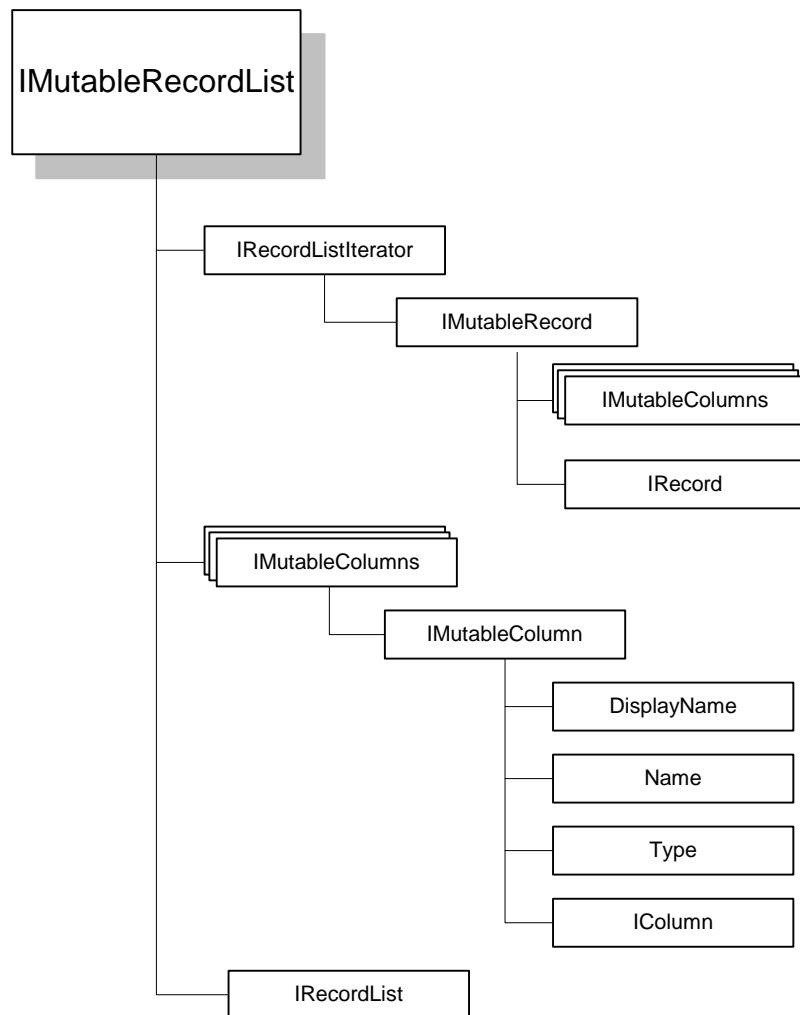


Figure 2 IMutableRecordList Object Diagram

The following objects can be obtained from the `ImmutableRecordList` object:

Object	Description
<code>IRecordListIterator</code>	You obtain a <code>IRecordListIterator</code> by one of the four methods: <ul style="list-style-type: none"><li>• <code>iterator</code></li><li>• <code>getIterator</code></li><li>• <code>getFilteredIterator</code></li><li>• <code>getSortedIterator</code></li></ul> The iterator returns <code>ImmutableRecord</code> objects.
<code>ImmutableColumns</code>	The <code>getColumns()</code> method returns the set of columns associated with the record list.
<code>IRecordList</code>	The <code>getRecordList()</code> method returns a copy of the current record list as a (read-only) <code>IRecordList</code> .

## Methods

The `ImmutableRecordList` object has the following methods:

Method	Description
<code>addRecord()</code>	Add an <code>ImmutableRecord</code> to the record list.
<code>Size()</code>	Number of records
<code>iterator()</code>	Returns an <code>Iterator</code>
<code>getIterator(boolean synchronize)</code>	Returns a simple <code>IRecordListIterator</code>
<code>getFilteredIterator(ICondition condition, boolean synchronize)</code>	Returns a filtered <code>IRecordListIterator</code> . The filtered iterator retrieves records from the <code>RecordList</code> that satisfy the condition specified by <code>ICondition</code> .
<code>getSortedIterator(Comparator comparator, boolean synchronize)</code>	Returns a sorted <code>IRecordListIterator</code> . The sorted iterator retrieves records from the <code>RecordList</code> sorted according to the <code>Comparator</code> object.

## **ImmutableRecord**

The following objects can be obtained from the ImmutableRecord object:

<b>Object</b>	<b>Description</b>
ImmutableColumns	The getColumns() method returns the set of columns associated with the record list.
IRecord	The getRecordList() method returns a copy of the current record as a (read-only) IRecord.

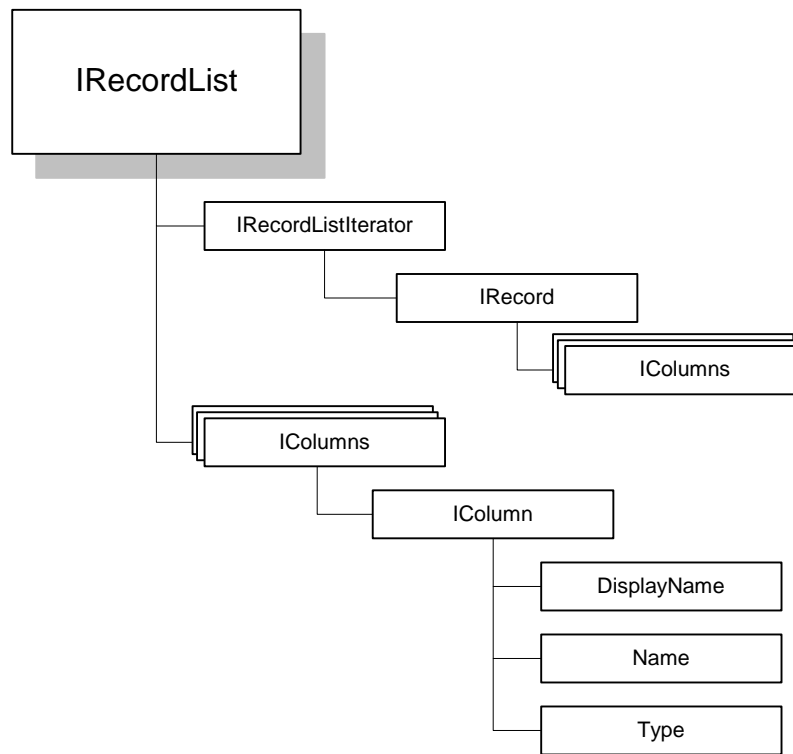
## **Setting and Getting Record Values**

The ImmutableRecord object has the following methods for setting and getting values of the record:

<b>Method</b>	<b>Description</b>
setValue(index, newValue)	Sets the value of a specific location in the record.
setValueByName(columnName, newValue)	Sets a value in the record according to the attribute's name
setValueAs[Boolean, Byte, Double, Float, Int, Long, Short](index, newValue)	Sets a value with casting of simple types to objects. (No need to cast int to Integer object)
getValue(index)	Returns the value of the attribute in this location
getValueByName(columnName)	Returns the value of the attribute with this name
getValueAs[Boolean, Byte, Double, Float, Int, Long, Short](index)	Returns the value, which casts back to simple values instead of objects (int instead of an Integer object)

## **IRecordList**

The following figure shows the object diagram for the IRecordList Object.



*Figure 3 IRecordList Object Diagram*

IRecordList and its associated interfaces are a read-only version of the IMutableRecordList object. They are used in the same way, with the following exceptions:

- All getValue methods work as in the IMutableRecord object; the setValue methods do not. You cannot set the values of an IRecord object; you can only read them.
- You cannot create an IMutableRecordList, IMutableRecord or IMutableColumn from an IRecordList, IRecord or IColumn object.

## Package Events

The package has four event types:

- **ColumnsChangeEvent** - Fires on change of Columns object
- **RecordChangeEvent** - Fires on change of Record object
- **RecordListChangeEvent** - Fires on change of RecordList object, such as adding a record to the RecordList
- **RecordListValueChangeEvent** - Fires on change of a record in a RecordList

All events have a `stop ( )` method that can be called by the listener to prevent the action from occurring. The event source checks the event stop flag before executing the action.

## ColumnsChangeEvent

### Creation

**ColumnsChangeEvent**(*Object* source, *Object* columnChanged)

### Listener methods:

Method	Description
columnBeforeAdd	Called before adding a column
columnAfterAdd	Called after adding a column
columnBeforeRemove	Called before removing a column
columnAfterRemove	Called after removing a column

## **RecordChangeEvent**

### **Creation:**

RecordChangeEvent(*Object* source, *int* index)

Listener methods:

<b>Method</b>	<b>Description</b>
valueBeforeChange	Called before changing a record value
valueAfterChange	Called after changing a record value

## **RecordListChangeEvent**

### **Creation:**

RecordListChangeEvent(*Object* source, *Object* recordChanged)

Listener methods:

<b>Method</b>	<b>Description</b>
recordBeforeAdd	Called before adding a record
recordAfterAdd	Called after adding a record
recordBeforeRemove	Called before removing a record
recordAfterRemove	Called after removing a record

## **RecordListValueChangeEvent**

### **Creation:**

RecordListValueChangeEvent(*Object* source, *Object* recordChanged, *int* index)



**Listener methods:**

Method	Description
valueBeforeChange	Called before changing a record value in the record list
valueAfterChange	Called after changing a record value in the record list

**Common Tasks**

The following sections describe methods and properties that are used to perform common tasks in client applications.

**ImmutableRecordList:  
Creating and setting values**

```
// Create new record list object
ImmutableRecordList recordList = new MutableRecordList();
// Set the columns for the new record list
ImmutableColumns mutableColumns = recordList.getColumns();
ImmutableColumn column;
column = mutableColumns.addColumn("ID", Integer.class);
column.setDisplayName("worker ID");
...
// Add record
ImmutableRecord record;
record = recordList.addRecord();
// Fill values to the newly added record
try {
    record.setValueByName("ID", integerObject);
} catch (MutableRecordException e) {
    // integerObject is not of type Integer ...
}
...
```

## **ImmutableRecordList:**

### **Usage of ICondition interface and filtered iterators**

```
// A condition class to find all records with Age column greater than  
two.  
public class GreaterThenTwo implements com.smarteam.client.util.ICondition  
{  
    public boolean evaluate(Object object) {  
        ImmutableRecord record = (ImmutableRecord)object;  
        return ((Integer)record.getValueByName("Age")).intValue() > 2;  
    }  
}  
  
...  
com.smarteam.client.util.ICondition condition = new GreaterThenTwo();  
// Obtain a disconnected filtered iterator from the RecordList  
IRecordListIterator iterator = recordList.getFilteredIterator(condition,  
false);  
ImmutableRecord tmpRecord = null;  
// Loop all records  
while (iterator.hasNext()) {  
    // Get record and change it's value  
    tmpRecord = (ImmutableRecord)iterator.next();  
    tmpRecord.setValueByName("someField", newValueForAllRecords);  
}  
  
...
```

## A Embedded-Scripts Engine

This appendix describes the SmarTeam i-Platform Application Server and covers the topics:

- Embedded-Scripts Engine
- Context Expiry

### Embedded-Scripts Engine

The Embedded-Scripts Engine object creates a client context and executes a client script in the context.

The Embedded-Scripts Engine Web Service has the following methods:

Method	Description
CreateContext (ConnectionString)	Opens a Context in SmarTeam in which the client can work. Returns a ContextHandle that specifies the context in method CreateContextResponse
Execute (ContextHandle, ScriptLanguage, Script, Params)	Executes a script in the Context specified by the ContextHandle. Parameter values are passed to the script through Params. Returns an execution result in method ExecuteResponse
ReleaseContext (ContextHandle)	Releases the Context specified by the ContextHandle.

**Note:** These methods are generally not accessed directly by the user. They are activated indirectly by client requests.

The Embedded-Scripts Engine is described by the WSDL (Web Service Description Language) document installed with the SDK. Access the document at: <http://localhost/smarteam/api/embeddedscripts/1.0/wsd/>

## Passing Parameters to a Script

The Params parameter in the Context object has the server-side data structure type IDictionary. The IDictionary structure holds data pairs, which represent parameters in the script passed in the Script parameter of the same Execute method call.

Each pair in the IDictionary data structure consists of a parameter key (name) represented by a string and a parameter value. The parameter value is accessed in the client script by using the parameter key in the Context method params, for example:

```
Context.Params( 'DateParam' );
```

See the example below of how parameters are passed in the Execute method.

For information, see the section [Using Parameters in a Script](#) on page 16.

## Examples

The following sections present examples of SOAP-encoded client requests to be executed by an Embedded-Scripts Engine and the corresponding results.

### Embedded-Scripts Engine: Creating a Context

The following is an example of a CreateContext SOAP request. The CreateContext method is sent together with the ConnectionString parameter.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <STES:CreateContext xmlns:STES="
      http://www.smarteam.com/dev/ns/iplatform/embeddedscripts">
      <ConnectionString
        xsi:type="xsd:string">Protocol=SmarTeam;DatabaseName=SmDem
        o;Username=joe;UserPassword=;</ConnectionString>
      </STES:CreateContext>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

The following is an example of a response to the CreateContext method request. This method returns the ContextHandle for the Context that was created.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <STES:CreateContextResponse xmlns:STES="
http://www.smarteam.com/dev/ns/iplatform/embeddedscripts"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <STES:return xsi:type="xsd:string">1025</STES:return>
    </STES:CreateContextResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## **Embedded-Scripts Engine: Executing a Script**

This section contains an example of the Execute method. The parameters ContextHandle, ScriptLanguage, and Script are sent.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <STES:Execute
xmlns:STES="http://www.smarteam.com/dev/ns/iplatform
/embeddedscripts"
xmlns:sof="http://www.smarteam.com/dev/ns/SOF/2.0">
      <ContextHandle>513</ContextHandle>
      <ScriptLanguage>JavaScript</ScriptLanguage>
      <Script>
        <![CDATA[var query =
Context.SmSession.ObjectStore.NewQuery();
var qDef = query.QueryDefinition;
var role = 'F';
qDef.Roles.Add(1, role);
qDef.Select.Add('CLASS_ID', role, false);
qDef.Select.Add('OBJECT_ID', role, false);
```

```
qDef.Select.Add('STATE', role, false);

query.Run();

var objsData =
Context.CreateObject('SmRecList.SmRecordList');

var objs =
Context.SmSession.ObjectStore.ObjectsFromData(query
.QueryResult, true);

var i, n;
var o;
for (i = 0; i < 2; i++) {
    o = objs.item(i).Clone();
    o.AddAllAttributes();
    o.RetrieveAttributes();
    n = objsData.AddRecord();
    objsData.CopySmRecord(o.data, n);
}

Context.Result = objsData;
]]>

</Script>
<Params SOAP-ENC:arrayType="sof:DictionaryItem[0]"/>
</STES:Execute>

</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The following is an example of the response to the Execute method request. This method returns a result, which was requested by the Context method Result in the previous example.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <STES:ExecuteResponse xmlns:STES="http://www.smarteam.com/dev/ns/
ipatform/embeddedscripts"
xsi:type="STES:ExecuteResponse"
xmlns:sof="http://www.smarteam.com/dev/ns/SOF/2.0"
sof:Version="1.0">
      <STES:return xsi:type="sof:RecordList">
```

```
<Headers>
  <Header>
    <Name>CLASS_ID</Name>
    <Type>xsd:short</Type>
  </Header>
  <Header>
    <Name>OBJECT_ID</Name>
    <Type>xsd:int</Type>
  </Header>
  <Header>
    <Name>CN_PROJECT_ID</Name>
    <Type>xsd:string</Type>
  </Header>
  <Header>
    <Name>STATE</Name>
    <Type>xsd:int</Type>
  </Header>
  <Header>
    <Name>CREATION_DATE</Name>
    <Type>xsd:dateTime</Type>
  </Header>
  <Header>
    <Name>CN_DESCRIPTION</Name>
    <Type>xsd:string</Type>
  </Header>
  <Header>
    <Name>USER_OBJECT_ID</Name>
    <Type>xsd:int</Type>
  </Header>
  <Header>
    <Name>USER_ID_MOD</Name>
    <Type>xsd:int</Type>
  </Header>
  <Header>
    <Name>MODIFICATION_DATE</Name>
    <Type>xsd:dateTime</Type>
  </Header>
  <Header>
    <Name>CN_TARGET_DATE</Name>
    <Type>xsd:date</Type>
  </Header>
  <Header>
    <Name>CN_TOTAL_BUDGET</Name>
```

```
<Type>xsd:int</Type>
</Header>
<Header><Name>CN_MANAGER</Name>
<Type>xsd:string</Type>
</Header>
<Header>
<Name>CN_COST</Name>
<Type>xsd:string</Type>
</Header>
<Header>
<Name>CN_START_DATE</Name>
<Type>xsd:date</Type>
</Header>
<Header>
<Name>CN_PRIORITY</Name>
<Type>xsd:int</Type>
</Header>
<Header>
<Name>TDM_SF_SECURE_LVL</Name>
<Type>xsd:int</Type>
</Header>
<Header>
<Name>TDM_SF_SERVICE</Name>
<Type>xsd:int</Type>
</Header>
</Headers>
<Records>
<Record>
<Value>459</Value>
<Value>2</Value>
<Value>Project-0001</Value>
<Value>0</Value>
<Value>1998-12-8T13:34:41.0Z</Value>
<Value>Snow Mobile Design - SolidWorks</Value>
<Value>1</Value>
<Value>1</Value>
<Value>2000-1-4T13:15:23.0Z</Value>
<Value>1999-6-11</Value>
<Value>800000</Value>
<Value>Natan</Value>
<Value>500000</Value>
<Value>1999-1-18</Value>
<Value>5</Value>
```



```
        <Value>0</Value><Value>-2147483647</Value>
    </Record>
    <Record>
        <Value>459</Value>
        <Value>3</Value>
        <Value>Project-0002</Value>
        <Value>0</Value>
        <Value>1998-12-8T13:43:37.0Z</Value>
        <Value>Shock Pivot Plate</Value>
        <Value>1</Value>
        <Value>1</Value>
        <Value>1999-2-16T15:25:28.0Z</Value>
        <Value>1999-3-18</Value>
        <Value>350000</Value>
        <Value>Avi</Value>
        <Value>300000</Value>
        <Value>1999-1-11</Value>
        <Value>5</Value>
        <Value>0</Value>
        <Value>-2147483647</Value>
    </Record>
</Records>
</STES:return>
</STES:ExecuteResponse>
</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

## **Embedded-Scripts Engine: Executing a Script with Parameters**

This section contains an example of the Execute method with the Params parameter. The Params parameter sets up the parameters for the Runtime methods in the script. When the Runtime method Params is invoked, it refers to the parameters set up in the Params parameter of the Execute function.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <STES:Execute xmlns:STES="http://www.smarteam.com/dev/ns/
iplatform/embeddedscripts"
xmlns:sf="http://www.smarteam.com/dev/ns/SOF/2.0">
            <ContextHandle>1537</ContextHandle>
```

```
<ScriptLanguage>JavaScript</ScriptLanguage>
<Script><![CDATA[Context.Result =
Context.Params('DateParam');]]></Script>
<Params SOAP-ENC:arrayType="sof:DictionaryItem[1]">
  <sof:DictionaryItem>
    <key xsi:type="xsd:string">DateParam</key>
    <value xsi:type="xsd:date">2001-09-20</value>
  </sof:DictionaryItem>
</Params>
</STES:Execute>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The following is an example of the response to the Execute method. This response returns a result which was requested by the Context methods in the script line: Context.Result = Context.Params('DateParam')

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <STES:ExecuteResponse xmlns:STES="http://www.smarteam.com/dev/ns/
    iplatform/embeddedscripts"
    xsi:type="STES:ExecuteResponse"
    xmlns:sof="http://www.smarteam.com/dev/ns/SOF/2.0"
    sof:Version="1.0">
      <STES:return xsi:type="xsd:date">2001-9-20</STES:return>
    </STES:ExecuteResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

### **Embedded-Scripts Engine: Releasing a Context**

The following is an example of the ReleaseContext method. The ReleaseContext method is sent after you have finished working with the Context.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
<STES:ReleaseContext xmlns:STES="http://www.smarteam.com/dev/ns/
iplatform/embeddedscripts">
  <ContextHandle xsi:type="xsd:string">1025</ContextHandle>
</STES:ReleaseContext>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The following is an example of the response to the ReleaseContext method request. This response is sent to verify that the Embedded-Scripts Engine released the Context.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <STES:ReleaseContextResponse
      xmlns:STES="http://www.smarteam.com/dev/ns/
      iPlatform/EmbeddedScripts">
    </STES:ReleaseContextResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## Context Expiry

A Context that has no activity – and has not been released by the ReleaseContext method – is automatically released by the server after it expires. This prevents Contexts accumulating on the server if the client does not send the ReleaseContext request at the end of each session.

The expiration time is set through the Registry value “ContextExpirationTime”, under the following Registry key:  
HKEY\_LOCAL\_MACHINE\Software\Smarteam\iPlatform\EmbeddedScripts\

where ContextExpirationTime is the time, in minutes, from when activity on the Context ceases until the Context expires and is released. The default value is 30 minutes.

When the value is set equal to 0 (zero) the expiry mechanism is disabled and, as a result, the Context does not expire automatically.