Appendice C: IBNDBTestingKit.

Nel codice che segue ho oggiunto alcuni commenti per renderlo più comprensible anche a chi non conosce il linguaggio Objective-C e le librerie EOF.

IBNDBTestCase.h

```
#import <Foundation/Foundation.h>
#import <SenTestingKit/SenTestingKit.h>
#import <EOControl/EOControl.h>
#import <EOAccess/EOAccess.h>

@interface IBNDBTestCase: SenTestCase  // Dichiarazione dell'oggetto IBNDBTestCase.

+ (EOEditingContext *)editingContext;          // Dichiarazione di un metodo di classe.
+ (EOEditingContext *)administrativeEditingContext;
+ (EOAdaptorChannel *)channel;
+ (EOAdaptor *)adaptor;
+ (void)enableLogSQL:(BOOL)isEnabled;

- (EOEditingContext *)editingContext;          // Dichiarazione di un metodo d'istanza.
- (EOEditingContext *)administrativeEditingContext;
- (EOAdaptorChannel *)channel;
- (EOAdaptor *)adaptor;
- (void)enableLogSQL:(BOOL)isEnabled;

- (id)DBTestManager;

- (NSArray *)objectsContainingValue:(NSString *)aValue forKey:(NSString *)aKey
entityNamed:(NSString *)anEntityName;      // Dichiarazione di un metodo con 3 parametri.
- (void)items:(NSArray *)someItems valueForKey:(NSString *)aKey shouldContain:(NSString
*)aValue;
- (NSArray *)valuesFromObjects:(NSArray *)searchObject key:(NSString *)aKey;

@end
```

IBNDBTestCase.m

```
// È la classe da cui derivare le classi contenenti I metodi di test.

#import "IBNDBTestCase.h"
#import "IBNDBTestManager.h"
#import "IBNDBTestManager_DatabaseLogic.h"

@implementation IBNDBTestCase     // Implementazione dell'oggetto IBNDBTestCase.

+ (IBNDBTestManager *)DBTestManager {  return [IBNDBTestManager currentManager]; }
- (IBNDBTestManager *)DBTestManager {   return [[self class] DBTestManager];  }
//=========================================================
+ (EOEditingContext *)editingContext {         return [[self DBTestManager] editingContext]; }
// EOEditingContext è una classe EOF ad alto livello per accedere al database.
+ (EOEditingContext *)administrativeEditingContext {
        return [[self DBTestManager] administrativeEditingContext];
}
+ (EOAdaptorChannel *)channel {       return [[self DBTestManager] channel];       }
// EOAdaptorChannel è una classe EOF a basso livello per accedere al database.
+ (void)enableLogSQL:(BOOL)isEnabled {
```

```
        [EOAdaptorContext setDebugEnabledDefault:isEnabled];
}
+ (EOAdaptor *)adaptor {          return [[self DBTestManager] adaptor];}
// EOAdaptor è una classe EOF a basso livello per accedere al database.
//=========================================================
- (EOEditingContext *)editingContext {          return [[self class] editingContext];     }
- (EOEditingContext *)administrativeEditingContext {
        return [[self class] administrativeEditingContext];
}
- (EOAdaptorChannel *)channel {       return [[self class] channel];       }
- (void)enableLogSQL:(BOOL)isEnabled {
        [EOAdaptorContext setDebugEnabledDefault:isEnabled];
}
- (EOAdaptor *)adaptor { return [[self class] adaptor];       }
//=========================================================
- (NSArray *)objectsContainingValue:(NSString *)aValue forKey:(NSString *)aKey
entityNamed:(NSString *)anEntityName {
        EOFetchSpecification     *fs;
        EOQualifier              *q; // Filtro che astrae codice SQL.
        NSString                 *valueWithWildChars;
        NSArray                  *result;

        valueWithWildChars = [NSString stringWithFormat:@"*%@*", aValue];
        q = [[EOKeyValueQualifier alloc] initWithKey:aKey
                                        operatorSelector:EOQualifierOperatorLike
                                        value:valueWithWildChars];
        fs = [EOFetchSpecification fetchSpecificationWithEntityName:anEntityName qualifier:q
sortOrderings:nil];
        result = [[self editingContext] objectsWithFetchSpecification:fs];

        [q release];

        return result;
}
//---------------------------------------------------------
- (void)items:(NSArray *)someItems valueForKey:(NSString *)aKey shouldContain:(NSString
*)aValue {
        int      i, c;

        c = [someItems count];
        for (i=0; i<c; i++) {
                should ([[[someItems objectAtIndex:i] valueForKey:aKey]
rangeOfString:aValue].length != 0);
        }
}
//---------------------------------------------------------
- (NSArray *)valuesFromObjects:(NSArray *)searchObject key:(NSString *)aKey {
        NSMutableArray    *result;
        int      i, c;

        result = [NSMutableArray array];
        c = [searchObject count];
        for (i=0; i<c; i++) {
                [result addObject:[[searchObject objectAtIndex:i] valueForKey:aKey]];
        }

        return result;
}
//=========================================================
// Metodi per configurare i test..
+ (void)setUp {
        [IBNDBTestManager createWithTestCase:self];
        [[self DBTestManager] doClassSetUp:self];
```

```
}
- (void)setUp {       [[self DBTestManager] doInstanceSetUpForTest:self];     }
- (void)tearDown { [[self DBTestManager] doInstanceTearDownForTest:self];       }
+ (void)tearDown {
        [[self DBTestManager] doClassTearDown:self];
        [IBNDBTestManager destroy];
}
@end
```

IBNDBTestManager.h

```
#import <Foundation/Foundation.h>
#import <EOAccess/EOAccess.h>

#import "IBNDBTestCase.h"
#import "AdaptorStrategy.h"

@interface IBNDBTestManager : NSObject {
        NSBundle                        *_bundle;               // Variabile istanza
        NSDictionary                    *_testConfiguration;
        EOAdaptor                       *_adaptor;
        Class                           sqlExpressionAdaptor;
        NSMutableDictionary             *_connectionDictionary;
        NSMutableDictionary             *_administrativeConnectionDictionary;
        EOEditingContext                *_editingContext;
        EOEditingContext                *_administrativeEditingContext;
        EOAdaptorChannel                *_channel;
        NSMutableArray                  *neededModels;
        NSString                        *user;
        BOOL                            testAllFromScratch;
        BOOL                            keepDatabases;
        BOOL                            logTest;
}

+ (void)createWithTestCase:(Class)aTestCase;
+ (void)destroy;
+ (IBNDBTestManager *)currentManager;

- (id)initWithTestCase:(Class)aTestCase;

- (void)doClassSetUp:(Class)aTestCase;
- (void)doInstanceSetUpForTest:(IBNDBTestCase *)aTestInstance;
- (void)doInstanceTearDownForTest:(IBNDBTestCase *)aTestInstance;
- (void)doClassTearDown:(Class)aTestCase;

- (NSBundle *)bundle;
- (EOAdaptor *)adaptor;
- (EOAdaptorChannel *)channel;

- (id)configurationElement:(NSString *)anElementName;
- (NSDictionary *)connectionDictionary;
- (NSDictionary *)administrativeConnectionDictionary;

- (EOModel *)modelNamed:(NSString *)aModelName;
- (NSMutableArray *)neededModels;

- (BOOL)logTest;

- (NSDictionary *)testConfiguration;
- (void)setTestConfiguration:(NSDictionary *)testConfiguration;

- (BOOL)deletable;
```

```
- (BOOL)doDestroy;
- (BOOL)modalityIsReadOnly;
- (BOOL)cleanDatabaseOnStart;
- (BOOL)cleanDatabaseOnEnd;

@end
```

IBNDBTestManager.m

// Classe principale che gestisce tutte le principali funzioni o le delega ad altre classi.

```
#import "IBNDBTestManager.h"
#import "IBNDBTestManager_Initializer.h"
#import "IBNDBTestManager_DatabaseLogic.h"
#import "IBNDBTestManager_PopulateDatabase.h"

#import <IBNExtensions/IBNExtensions.h>
#import "FrameworkDefines.h"

@implementation IBNDBTestManager

static IBNDBTestManager        *_IBNDBTestManager;
+ (void)createWithTestCase:(Class)aTestCase {
        if (_IBNDBTestManager != nil) {         [_IBNDBTestManager release];        }
        _IBNDBTestManager = [[IBNDBTestManager alloc] initWithTestCase:aTestCase];
}
+ (void)destroy {
        [_IBNDBTestManager release];
        _IBNDBTestManager = nil;
}
+ (IBNDBTestManager *)currentManager {   return _IBNDBTestManager;    }
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
- (id)initWithTestCase:(Class)aTestCase {    // Inizializzatore
        if (self = [super init]) {
                [self listenForCooperatingObjectStoreAdded];
                [self readEnvironment];
                [self initializeConfigurationAndAdaptorForTestCase:aTestCase];
                [self initializeConnectionDictionary];
                [self initializeLogs];
                [self initializeModelsAndEditingContext];
        }
        return self;
}
- (void)dealloc {              // Invocata automaticamente quando l'oggetto viene distrutto
        [_bundle release];          // Rilascia l'oggetto _bundle
        [_channel release];
        [_testConfiguration release];
        [_adaptor release];
        [_connectionDictionary release];
        [_administrativeConnectionDictionary release];
        [_editingContext release];
        [_administrativeEditingContext release];
        [super dealloc];
}
- (NSBundle *)bundle {    return _bundle;     } // Per accedere ad alcune variabili del progetto
- (NSDictionary *)connectionDictionary {      return _connectionDictionary;   }
// Contiene le informazioni per collegarsi al database
- (NSDictionary *)administrativeConnectionDictionary {
        return _administrativeConnectionDictionary;
}
- (EOAdaptor *)adaptor { return _adaptor;    }
- (EOAdaptorChannel *)channel {
        EOAdaptorContext*context;
```

```
        if (_channel == nil) {
                context = [[self adaptor] createAdaptorContext];
                _channel = [[context createAdaptorChannel] retain];
        }
        return _channel;
}
// EOModel è un modello che contiene la struttura di un database
- (EOModel *)modelNamed:(NSString *)aModelName {
        EOModel     *result;

        result = [[EOModelGroup globalModelGroup] modelNamed:[aModelName
stringByDeletingPathExtension]];
        if (result == nil) {
                NSLog(@"#### IBNDBTestManager_PopulateDatabase, modelNamed: Cannot
find model named %@ taken from plist.",aModelName);
        }
        return result;
}
- (NSMutableArray *)neededModels {
        NSDictionary        *testsWithModelNames;
        NSEnumerator        *allTestsWithModelNames;
        NSString            *aTest;

        if (neededModels == nil) {
                neededModels = [NSMutableArray array];
                testsWithModelNames = [_testConfiguration objectForKey:@"models"];
                allTestsWithModelNames = [testsWithModelNames keyEnumerator];
                while (aTest = [allTestsWithModelNames nextObject]) {
                        [neededModels addObject:[self modelNamed:[testsWithModelNames
objectForKey:aTest]]];
                }
        }
        return neededModels;
}
- (id)configurationElement:(NSString *)anElementName {
        return [_testConfiguration objectForKey:anElementName];
}
- (NSDictionary *)testConfiguration {   return _testConfiguration;          }
- (void)setTestConfiguration:(NSDictionary *)testConfiguration {
        _testConfiguration = testConfiguration;
}
- (BOOL)logTest {  return logTest;       }
- (NSDictionary *)modality {       return [self configurationElement:@"modality"];     }
- (BOOL)deletable {
        NSString      *deletable;

        deletable = [[self modality] objectForKey:@"deletable"];
        if (deletable == nil) {
                [NSException raise:@"deletable missing" format:@"#### IBNDBTestManager:
'deletable' missing in DBTestConfiguration.plist"];
        }
        return([deletable isEqualToString:@"YES"]);
}
- (BOOL)modalityIsReadOnly {
        return([[[self modality] objectForKey:@"isReadOnly"] isEqualToString:@"YES"]);
}
- (BOOL)cleanDatabaseOnStart {
        return testAllFromScratch || [[[self modality] objectForKey:@"cleanDatabaseOnStart"]
isEqualToString:@"YES"];
}
- (BOOL)cleanDatabaseOnEnd {
        return testAllFromScratch || [[[self modality] objectForKey:@"cleanDatabaseOnEnd"]
isEqualToString:@"YES"];
```

```
}
-(BOOL)doDestroy {
	BOOLresult;

	result = ([self cleanDatabaseOnEnd] && [self deletable]) || ![self modalityIsReadOnly];
	if ((keepDatabases && [self modalityIsReadOnly]) || (logTest && !testAllFromScratch)) {
		result = NO;
	}
	return result;
}
//#############################################################
- (void)doClassSetUp:(Class)aTestCase {
	if (logTest) {
		NSLog(@"++++++++++++++++++++++ doClassSetUp: %@", aTestCase);
	}
	if ([self deletable]) {
		if ([self cleanDatabaseOnStart]) {
			[(id <AdaptorStrategy>)[self adaptor]
createDatabaseWithTestManager:self];
			if ([self modalityIsReadOnly]) {
				[(id <AdaptorStrategy>)[self adaptor]
setUpDatabaseWithTestManager:self];
			}
		} else {
			if ([self modalityIsReadOnly]) {
				if ([(id <AdaptorStrategy>)[self adaptor]
createDatabaseIfNotExistWithTestManager:self]) {
					[(id <AdaptorStrategy>)[self adaptor]
setUpDatabaseWithTestManager:self];
				}
			} else {
				[(id <AdaptorStrategy>)[self adaptor]
createDatabaseWithTestManager:self];
			}
		}
	}
}
- (void)doInstanceSetUpForTest:(IBNDBTestCase *)aTestInstance {
	if (logTest) {
		NSLog(@"---------------------- doInstanceSetUpForTest: %@", aTestInstance);
	}
	if ([self deletable] && ([self modalityIsReadOnly] == NO)) {
		[(id <AdaptorStrategy>)[self adaptor] setUpDatabaseWithTestManager:self];
	}
}
- (void)doInstanceTearDownForTest:(IBNDBTestCase *)aTestInstance {
	if (logTest) {
		NSLog(@"---------------------- doInstanceTearDownForTest: %@", aTestInstance);
	}
	[self invalidateObjectsAndCloseChannels];
	if ([self deletable] && ([self modalityIsReadOnly] == NO)) {
		[(id <AdaptorStrategy>)[self adaptor] tearDownDatabaseWithTestManager:self];
	}
}
- (void)doClassTearDown:(Class)aTestCase {
	if (logTest) {
		NSLog(@"++++++++++++++++++++++ doClassTearDown: %@", aTestCase);
	}

	if ([self doDestroy]) {
		[(id <AdaptorStrategy>)[self adaptor] destroyDatabaseWithTestManager:self];
	}
	[EOAdaptorContext setDebugEnabledDefault:[EOAdaptorContext
```

```
debugEnabledDefault]];
	[[NSNotificationCenter defaultCenter] removeObserver:self];
}

@end
```

AdaptorStrategy.h

```
// Definisce una interfaccia astratta per personalizzare l'accesso ad un determinato database.

#import <Foundation/Foundation.h>
#import <EOAccess/EOSQLExpression.h>

@class IBNDBTestManager;

@protocol AdaptorStrategy
- (NSDictionary *)personalServerInformationsForDatabase:(NSDictionary *)database;
- (NSString *)usernameForConnectionDictionaryWithTestManager:(IBNDBTestManager
*)testManager;

- (NSString *)cutLongName:(NSString *)name;
- (void)cutLongNamesForForeignKeysInExpression:(EOSQLExpression *)expression;

- (BOOL)createDatabaseIfNotExistWithTestManager:(IBNDBTestManager *)testManager;
- (void)createDatabaseWithTestManager:(IBNDBTestManager *)testManager;
- (void)setUpDatabaseWithTestManager:(IBNDBTestManager *)testManager;
- (void)tearDownDatabaseWithTestManager:(IBNDBTestManager *)testManager;
- (void)destroyDatabaseWithTestManager:(IBNDBTestManager *)testManager;
@end
```

OracleAdaptor.h

```
#import <OracleEOAdaptor/OracleAdaptor.h>
#import "AdaptorStrategy.h"

@interface OracleAdaptor (Strategy) <AdaptorStrategy>
// Dichiarazione della categoria denominata Strategy che estende la classe OracleAdaptor
// appartenente alla libreria EOF. Strategy implementa il protocollo AdaptorStrategy.
@end
```

OracleAdaptor.m

```
// Definisce una strategia di accesso al database Oracle

#import "OracleStrategy.h"
#import "IBNDBTestManager.h"
#import "IBNDBTestManager_PopulateDatabase.h"
#import "IBNDBTestManager_DatabaseLogic.h"

@implementation OracleAdaptor (Strategy)

- (NSDictionary *)personalServerInformationsForDatabase:(NSDictionary *)database {
	return [NSDictionary dictionaryWithObject:[database objectForKey:@"server"]
forKey:@"serverId"];
}
//------------------------------------------------------------
- (NSString *)usernameForConnectionDictionaryWithTestManager:(IBNDBTestManager
*)testManager {
	NSDictionary		*databaseConfiguration;
	NSString		*key;

	databaseConfiguration = [testManager
configurationElement:@"databaseConfiguration"];
```

```
        if (![[databaseConfiguration objectForKey:@"database"]
                                isEqualToString:[databaseConfiguration
objectForKey:@"userName"]])
        {
                NSLog(@"# warning: 'database' and 'userName' should be the same for Oracle in
DBTestConfiguration.plist");
        }

        if ([testManager modalityIsReadOnly]) {
                key = @"userName";
        } else {
                key = @"database";
        }
        return [databaseConfiguration objectForKey:key];
}
//---------------------------------------------------------
- (NSString *)cutLongName:(NSString *)name {
        NSString     *result;
        if ([name length] > 30) {
                result = [name substringToIndex:30];
        } else {
                result = name;
        }
        return result;
}
//---------------------------------------------------------
- (void)cutLongNamesForForeignKeysInExpression:(EOSQLExpression *)expression {
        NSRange              range;
        NSString     *expressionStatement;
        NSString     *beginOfStatement;
        NSString     *endOfStatement;
        int                  index, stringLength;

        expressionStatement = [expression statement];
        range = [expressionStatement rangeOfString:@"_FK FOREIGN KEY"];
        if (range.length > 0) {
                index = range.location;
                while ([expressionStatement characterAtIndex:index] != ' ') {
                        index--;
                }
                stringLength = range.location - index + 2;
                if (stringLength > 30) {
                        beginOfStatement = [expressionStatement substringToIndex:range.location
- (stringLength - 30)];
                        endOfStatement = [expressionStatement
substringFromIndex:range.location];
                        [expression setStatement:[beginOfStatement
stringByAppendingString:endOfStatement]];
                }
        }
}
//=========================================================
- (void)dropUserOnException:(NSException *)exception
withTestManager:(IBNDBTestManager *)testManager {
        NSLog(@"#### OracleStrategy.dropUserOnException:%@",exception);
        if ([testManager doDestroy]) {
                NSLog(@"User dropped on error.");
                NS_DURING
                        [self destroyDatabaseWithTestManager:testManager];
                NS_HANDLER
                NS_ENDHANDLER
        }
        [NSException raise:@"dropUserOnException" format:@"Error caught in
```

```
OracleStrategy"];
}
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
- (BOOL)createDatabaseIfNotExistWithTestManager:(IBNDBTestManager *)testManager {
	BOOLresult;

	if ([testManager logTest]) {
		NSLog(@"OracleStrategy, connectionDictionary:%@",[self
connectionDictionary]);
	}

	NS_DURING
		[self createDatabaseWithAdministrativeConnectionDictionary:
					[testManager administrativeConnectionDictionary]];
		result = YES;
	NS_HANDLER
		result = NO;
	NS_ENDHANDLER
	return result;
}
//---------------------------------------------------------------------------
- (void)createDatabaseWithTestManager:(IBNDBTestManager *)testManager {
	int times;

	if ([testManager logTest]) {
		NSLog(@"OracleStrategy, connectionDictionary:%@",[self
connectionDictionary]);
	}

	for (times = 0; times < 2; times++) {
		NS_DURING
			[self createDatabaseWithAdministrativeConnectionDictionary:
						[testManager
administrativeConnectionDictionary]];
			break;
		NS_HANDLER
			if (times == 0) {
				[self destroyDatabaseWithTestManager:testManager];
			} else {
				[self dropUserOnException:localException
withTestManager:testManager];
			}
		NS_ENDHANDLER
	}
}
//---------------------------------------------------------------------------
- (void)setUpDatabaseWithTestManager:(IBNDBTestManager *)testManager {
	NS_DURING
		[testManager createTables];
	NS_HANDLER
		[self dropUserOnException:localException withTestManager:testManager];
	NS_ENDHANDLER
}
//---------------------------------------------------------------------------
- (void)tearDownDatabaseWithTestManager:(IBNDBTestManager *)testManager {
	NS_DURING
		[testManager invalidateObjectsAndCloseChannels];
		[testManager dropTables];
	NS_HANDLER
		[self dropUserOnException:localException withTestManager:testManager];
	NS_ENDHANDLER
}
//---------------------------------------------------------------------------
```

```
- (void)destroyDatabaseWithTestManager:(IBNDBTestManager *)testManager {
	NS_DURING
		[testManager invalidateObjectsAndCloseChannels];
		[self dropDatabaseWithAdministrativeConnectionDictionary:[testManager
administrativeConnectionDictionary]];
	NS_HANDLER
		NSLog(@"#### IBNDBTestingKit.OracleStrategy:user not
dropped:%@",localException);
		if ([[testManager channel] isOpen]) {
			NSLog(@"%@ destroyDatabaseWithTestManager channel is open",[self
class]);
		}
	NS_ENDHANDLER
}

@end
```

FrontBaseAdaptor.h e FrontBaseAdaptor.m

// Non le ho trascritte perché sono abbastanza simili ad OracleAdaptor

IBNDBTestManager_Initializer.h

```
#import <Foundation/Foundation.h>
#import "IBNDBTestManager.h"

@interface IBNDBTestManager (Initializer)

- (void)readEnvironment;
- (void)initializeConfigurationAndAdaptorForTestCase:(Class)aTestCase;
- (void)initializeConnectionDictionary;
- (void)initializeLogs;
- (void)initializeModelsAndEditingContext;

@end
```

IBNDBTestManager_Initializer.m

// Categoria di IBNDBTestManager utilizzata durante la fase di inizializzazione della classe

```
#import "IBNDBTestManager_Initializer.h"
#import "IBNDBTestManager_DatabaseLogic.h"
#import "ConfigurationNormalizer.h"

#import <IBNExtensions/IBNExtensions.h>

@implementation IBNDBTestManager (Initializer)

- (void)readEnvironment {
	NSDictionary	*environment;

	environment = [[NSProcessInfo processInfo] environment];
	testAllFromScratch = [[environment objectForKey:@"TEST_ALL_FROM_SCRATCH"]
isEqualToString:@"testAllFromScratch"];
	keepDatabases = [[environment objectForKey:@"KEEP_DATABASES"]
isEqualToString:@"keepDatabases"];
	user = [environment objectForKey:@"USER"];
}
//-------------------------------------------------------
- (void)initializeConfigurationAndAdaptorForTestCase:(Class)aTestCase {
	NSString			*path;
	NSDictionary			*configurationDictionary;
	ConfigurationNormalizer	*normalizer;
```

```
        _bundle = [[NSBundle bundleForClass:aTestCase] retain];

        path = [_bundle pathForResource:@"DBTestConfiguration" ofType:@"plist"];
        if (path == nil) {
                [NSException raise:@"No file" format:@"##### File DBTestConfiguration.plist
does not exist"];
        }
        configurationDictionary = [[NSDictionary alloc] initAndCheckWithContentsOfFile:path];

        _adaptor = [[EOAdaptor adaptorWithName:[configurationDictionary
objectForKey:@"selectedAdaptor"]] retain];
        sqlExpressionAdaptor = [_adaptor expressionClass];

        normalizer = [[ConfigurationNormalizer alloc] init];
        _testConfiguration = [[normalizer
normalizedConfigurationDictionary:configurationDictionary forTestCase:aTestCase] retain];
        [normalizer release];
}
//-------------------------------------------------------
- (void)initializeConnectionDictionary {
        NSDictionary                *serverDictionary;
        NSDictionary                *databaseDictionary;

        databaseDictionary = [self configurationElement:@"databaseConfiguration"];

        _administrativeConnectionDictionary = [[NSMutableDictionary alloc]
initWithObjectsAndKeys:
                                [databaseDictionary objectForKey:@"administrativeUserName"],
        @"userName",
                                [databaseDictionary objectForKey:@"administrativePassword"],
        @"password", nil];

        _connectionDictionary = [[NSMutableDictionary alloc] initWithObjectsAndKeys:
                [(id <AdaptorStrategy>)_adaptor
usernameForConnectionDictionaryWithTestManager:self],        @"userName",
                [databaseDictionary objectForKey:@"password"],                @"password", nil];


        if (testAllFromScratch && [self deletable]) {
                NSString     *newString;
                newString = [(id <AdaptorStrategy>)_adaptor cutLongName:
                        [NSString stringWithFormat:@"%@%@",user,[_connectionDictionary
objectForKey:@"database"]]];
                [_connectionDictionary setObject:newString forKey:@"database"];
                newString = [(id <AdaptorStrategy>)_adaptor cutLongName:
                        [NSString stringWithFormat:@"%@%@",user,[_connectionDictionary
objectForKey:@"userName"]]];
                [_connectionDictionary setObject:newString forKey:@"userName"];
        }

        serverDictionary = [(id <AdaptorStrategy>)_adaptor

        personalServerInformationsForDatabase:databaseDictionary];
        [_administrativeConnectionDictionary addEntriesFromDictionary:serverDictionary];
        [_connectionDictionary addEntriesFromDictionary:serverDictionary];
        [_connectionDictionary addEntriesFromDictionary:[databaseDictionary
objectForKey:@"addictionalInfo"]];

        [_adaptor setConnectionDictionary:_connectionDictionary];
}
//-------------------------------------------------------
- (void)initializeLogs {
```

```
        NSDictionary                *logsDictionary;

        logsDictionary = [self configurationElement:@"logs"];

        logTest = [[logsDictionary objectForKey:@"logTest"] isEqualToString:@"YES"];
        [EOAdaptorContext setDebugEnabledDefault:[[logsDictionary objectForKey:@"logSQL"]
isEqualToString:@"YES"]];
}
//-------------------------------------------------------
- (void)initializeModelsAndEditingContext {
        NSArray             *models;
        int     i, c;

        _editingContext = [[EOEditingContext alloc] init];
        [[self class] addObjectStore:[_editingContext parentObjectStore]];

        models = [self neededModels];
        c = [models count];
        for (i=0; i<c; i++) {
                EOModel                         *model;
                NSMutableDictionary             *modelConnectionDictionary;
                EODatabaseContext               *databaseContext;

                model = [models objectAtIndex:i];
                [model setAdaptorName:[[self adaptor] name]];
                modelConnectionDictionary = [NSMutableDictionary
dictionaryWithDictionary:[model connectionDictionary]];
                [modelConnectionDictionary addEntriesFromDictionary:[[self adaptor]
connectionDictionary]];
                [model setConnectionDictionary:modelConnectionDictionary];

                databaseContext = [EODatabaseContext
registeredDatabaseContextForModel:model editingContext:_editingContext];
                [[[databaseContext adaptorContext] adaptor]
setConnectionDictionary:modelConnectionDictionary];
        }
}

@end
```

IBNDBTestManager_DatabaseLogic.h

```
#import <Foundation/Foundation.h>
#import "IBNDBTestManager.h"

@interface IBNDBTestManager (DatabaseLogic)

+ (void)addObjectStore:(EOObjectStore *)objectStore;
+ (void)classHandleForCooperatingObjectStoreAddedNotification:(NSNotification *)notification;
- (void)listenForCooperatingObjectStoreAdded;

- (EOEditingContext *)administrativeEditingContext;
- (void)invalidateObjectsAndCloseChannels;

@end
```

IBNDBTestManager_DatabaseLogic.m

```
// Categoria di IBNDBTestManager che si oggupa della gestione degli oggetti che
// accedono al database.
#import "IBNDBTestManager_DatabaseLogic.h"

@implementation IBNDBTestManager (DatabaseLogic)
```

```
static NSMutableArray    *_storeCoordinators;
+ (void)addObjectStore:(EOObjectStore *)objectStore {
        if (_storeCoordinators == nil) {
                _storeCoordinators = [[NSMutableArray alloc] init];
        }
        if (![_storeCoordinators containsObject:objectStore]) {
                [_storeCoordinators addObject:objectStore];
        }
}
//-------------------------------------------------------
+ (void)classHandleForCooperatingObjectStoreAddedNotification:(NSNotification *)notification
{
        [[self class] addObjectStore:[notification object]];
//              NSLog(@"%@", _storeCoordinators);
}
//+++++++++++++++++++++++++++++++++++++++++++++++++
- (void)handleForCooperatingObjectStoreAddedNotification:(NSNotification *)notification {
        [IBNDBTestManager
classHandleForCooperatingObjectStoreAddedNotification:notification];
//      NSLog(@"notification name:%@", [notification name]);
//      NSLog(@"added coordinator: %@",[notification object]);
}
//-------------------------------------------------------
- (void)listenForCooperatingObjectStoreAdded {
        [[NSNotificationCenter defaultCenter] addObserver:self

        selector:@selector(handleForCooperatingObjectStoreAddedNotification:)

        name:@"EOCooperatingObjectStoreWasAdded"
                                                                object:nil];
}
//===============================================
- (EOEditingContext *)editingContext {
        if (_editingContext == nil) {
                _editingContext = [[EOEditingContext alloc] init];
        }
        return _editingContext;
}
//-------------------------------------------------------
- (EOEditingContext *)administrativeEditingContext {
        if (_administrativeEditingContext == nil) {
                EOObjectStoreCoordinator      *objectStoreCoordinator;
                NSArray                                 *models;
                int             i, c;

                objectStoreCoordinator = [[EOObjectStoreCoordinator alloc] init];
                _administrativeEditingContext = [[EOEditingContext alloc]
initWithParentObjectStore:objectStoreCoordinator];
                [objectStoreCoordinator release];

                models = [self neededModels];
                c = [models count];
                for (i=0; i<c; i++) {
                        [EODatabaseContext      forceConnectionWithModel:[models
objectAtIndex:i]
                                                connectionDictionaryOverrides:[self
administrativeConnectionDictionary]
                                                editingContext:_administrativeEditingContext];
                }
        }
        return _administrativeEditingContext;
}
```

```
//-------------------------------------------------------
- (void)invalidateAndCloseStoreCoordinatorsChannels {
	int i,c;

	c = [_storeCoordinators count];
	for (i=0; i<c; i++) {
		NSArray			*stores;
		int j,d;

		stores = [(EOObjectStoreCoordinator *)[_storeCoordinators objectAtIndex:i]
cooperatingObjectStores];
		d = [stores count];
		for (j = 0;j < d;j++) {
			NSArray				*channels;
			EODatabaseContext	*databaseContext;
			int k,e;

			databaseContext = [stores objectAtIndex:j];
			NS_DURING
				[databaseContext invalidateAllObjects];
			NS_HANDLER
				NSLog(@"#### IBNDBTestManager.closeChannels: Error while
invalidatingAllObjects");
				NSLog(@"localException:%@", localException);
				[localException raise];
			NS_ENDHANDLER
			channels = [databaseContext registeredChannels];
			e = [channels count];
			for (k=0; k < e; k++) {
				[[[channels objectAtIndex:k] adaptorChannel] closeChannel];
			}
		}
	}
}
//-------------------------------------------------------
- (void)closeAdaptorChannels {
	NSArray	*adaptorContexts;
	int i,c;

	adaptorContexts = [_adaptor contexts];
	c = [adaptorContexts count];
	for (i=0; i<c; i++) {
		NSArray		*adaptorChannels;
		int j,d;

		adaptorChannels = [[adaptorContexts objectAtIndex:i] channels];
		d = [adaptorChannels count];
		for (j = 0;j < d;j++) {
			[[adaptorChannels objectAtIndex:j] closeChannel];
		}
	}
}
//-------------------------------------------------------
- (void)invalidateObjectsAndCloseChannels {
	[self invalidateAndCloseStoreCoordinatorsChannels];
	[self closeAdaptorChannels];
}

@end
```

IBNDBTestManager_PopulateDatabase.h

```
#import <Foundation/Foundation.h>
#import "IBNDBTestManager.h"

@interface IBNDBTestManager (PopulateDatabase)
- (void)createTables;
- (void)dropTables;
@end
```

IBNDBTestManager_PopulateDatabase.h

```
// Categoria di IBNDBTestManager per creare e popolare il database
#import "IBNDBTestManager_PopulateDatabase.h"
#import <IBNExtensions/IBNExtensions.h>

@implementation IBNDBTestManager (PopulateDatabase)

- (void)createStatementsWithOptionDictionary:optionDictionary
                                        fromModel:(EOModel *)model
                                        cutLongForeignKeyNames:(BOOL)
cutLongNames;
{
        NSArray                 *entities;
        EOEntity                *entity;
        NSMutableArray          *usableEntities;
        NSArray                 *expressions;
        EOSQLExpression         *expression;
        EOAdaptorChannel        *channel;
        int i,c;

        entities = [model entities];
        usableEntities = [[NSMutableArray alloc] init];
        c = [entities count];
        for (i = 0; i<c; i++) {
                entity = [entities objectAtIndex:i];
                if (![[entity name] hasPrefix:@"OLD_"]) {
                        [usableEntities addObject:entity];
                }
        }
        expressions = [sqlExpressionAdaptor
schemaCreationStatementsForEntities:usableEntities

                        options:optionDictionary];
        [usableEntities release];

        if ([[[self configurationElement:@"logs"] objectForKey:@"logExpressions"]
isEqualToString:@"YES"]) {
                NSLog(@"expressions:%@",expressions);
        }
        channel = [self channel];
        [channel openChannel];
        c = [expressions count];
        for (i=0; i<c; i++) {
                expression = [expressions objectAtIndex:i];
                if (cutLongNames) {
                        [(id <AdaptorStrategy>)_adaptor
cutLongNamesForForeignKeysInExpression:expression];
                }
                NS_DURING
                        [channel evaluateExpression:expression];
                NS_HANDLER
                        NSLog(@"expressions:%@",expressions);
                        NSLog(@"expression:%@",expression);
                        [channel closeChannel];
```

```
                        NSLog(@"############Exception: %@",localException);
                        [NSException raise:@"evaluateExpression"
                                format:@"#### IBNDBTestManager_PopulateDatabase.
createStatementsWithOptionDictionaryFromModel:%@, errorOnEvaluateExpression",[model
name]];
                NS_ENDHANDLER
        }
        [channel closeChannel];
}
//---------------------------------------------------------------------------
- (void)createSchema {
        NSMutableDictionary             *optionDictionary;
        NSArray                                 *models;
        int i,c;

        if (logTest) {
                NSLog(@"%@_Populatedatabase createSchema",[self class]);
        }

        optionDictionary = [NSMutableDictionary dictionaryWithObjectsAndKeys:
                //@"YES",   @"createTables",
                @"NO",      @"dropTables",
                @"NO",      @"createPrimaryKeySupport",
                @"NO",      @"dropPrimaryKeySupport",
                //@"YES",   @"primaryKeyConstraints",
                //@"NO",    @"foreignKeyConstraints",
                //@"NO",    @"createDatabase",
                //@"NO",    @"dropDatabase",
                nil];
        models = [self neededModels];
        c = [models count];
        for (i=0; i<c; i++) {
                [self createStatementsWithOptionDictionary:optionDictionary

        fromModel:[models objectAtIndex:i]

        cutLongForeignKeyNames:NO];
        }

        if (logTest) {
                NSLog(@"%@_Populatedatabase createSchema end",[self class]);
        }
}
//---------------------------------------------------------------------------
- (NSDictionary *)loadFromPlist:(NSString *)plistName {
        NSString                *path;

        path = [NSString stringWithFormat:@"%@/%@",[_bundle resourcePath],plistName];
        if (path == nil) {
                [NSException raise:@"No file"
                                format:@"####
IBNDBTestManager_PopulateDatabase.importDataFromList: File %@ not found",

                                                                plistName];
        }
        return [NSDictionary dictionaryAndCheckWithContentsOfFile:path];
}
//---------------------------------------------------------------------------
- (NSDictionary *)loadDataDictionaries {
        NSMutableDictionary             *result;
        NSDictionary                    *testsWithArrayOfPlistNames;
        NSDictionary                    *testsWithModelNames;
        NSEnumerator                    *allTests;
```

```
	NSString			*aTest;
	NSString			*modelName;
	int i,c;

	result = [NSMutableDictionary dictionary];

	testsWithArrayOfPlistNames = [_testConfiguration objectForKey:@"data"];
	testsWithModelNames = [_testConfiguration objectForKey:@"models"];
	//NSLog(@"populateDatabase, data: %@",data);   //{test = (test.plist); }
	//NSLog(@"populateDatabase, models: %@",models); //{test = TestOracle.eomodeld; }

	allTests = [testsWithModelNames keyEnumerator];
	while (aTest = [allTests nextObject]) {
		NSMutableDictionary		*dictionaries;
		NSArray				*plistNamesArray;

		dictionaries = [NSMutableDictionary dictionary];
		plistNamesArray = [testsWithArrayOfPlistNames objectForKey:aTest];
		modelName = [testsWithModelNames objectForKey:aTest];
		c = [plistNamesArray count];
		for (i=0; i<c; i++) {
			NSString	*plistName;

			plistName = [plistNamesArray objectAtIndex:i];
			[dictionaries setObject:[self loadFromPlist:plistName] forKey:plistName];
		}
		[result setObject:dictionaries forKey:modelName];
	}
	return result;
}
//--------------------------------------------------------------------------
- (void)convertStringsInRow:(NSMutableDictionary *)row forEntity:(EOEntity *) entity {
	NSEnumerator		*allAttributeNames;
	NSString			*anAttributeName;
	NSString			*valueClassName;
	NSString			*valueType;
	NSString			*oldValue;
	id				newValue;

	allAttributeNames = [row keyEnumerator];
	while (anAttributeName = [allAttributeNames nextObject]) {
		valueClassName = [[entity attributeNamed:anAttributeName] valueClassName];
		valueType = [[entity attributeNamed:anAttributeName] valueType];

		oldValue = [row objectForKey:anAttributeName];
		newValue = oldValue;
		if ([oldValue isEqualToString:@"__NULL__"]) {
			newValue = [EONull null];
		} else if ([valueClassName isEqualToString:@"NSNumber"]) {
			if ([valueType isEqualToString:@"i"]) {
				newValue = [NSNumber numberWithInt:[oldValue intValue]];
//			} else if ([valueType isEqualToString:@"f"]) {
//				newValue = [NSNumber numberWithFloat:[oldValue floatValue]];
			} else if ([valueType isEqualToString:@"d"]) {
				newValue = [NSNumber numberWithDouble:[oldValue doubleValue]];
			} else {
				[NSException raise:@"NSNumber unknown valueType"
format:@"#### IBNDBTestManager.convertStringsInRow:%@ unknown valueType for
NSNumber in model",row];
			}
		} else if ([valueClassName isEqualToString:@"NSCalendarDate"]) {
			newValue = [NSCalendarDate dateWithString:oldValue
calendarFormat:@"%b %d %Y %H:%M"];
```

```
        }

                if (newValue == nil) {
                        [[self channel] closeChannel];
                        [NSException raise:@"Bad format value" format:@"####
IBNDBTestManager.convertStringsInRow: bad format value for key '%@' in row
%@",anAttributeName,row];
                } else {
                        [row setObject:newValue forKey:anAttributeName];
                }
        }
}
//-------------------------------------------------------------------------
- (void)importDataFromDictionary:(NSDictionary *)dataDictionary
                                ofPlistName:(NSString *)aPlistName
                                forModel:(EOModel *)model
{
        EOEntity                *entity;
        EOAdaptorChannel        *channel;
        NSArray                         *attributeNames;
        NSArray                         *rows;
        NSMutableDictionary     *row;
        EOSQLExpression         *expression;
        int                             i,k,c,d;

        if (logTest) {
                NSLog(@"%@_Populatedatabase importDataFromList:%@ forModel:%@",
                                [self class],                                   aPlistName,
        [model name]);
        }

        channel = [self channel];
        [channel openChannel];
        c = [[model entities] count];
        for (i=0; i<c; i++) {
                entity = [[model entities] objectAtIndex:i];
                attributeNames = [[dataDictionary objectForKey:[entity name]]
objectForKey:@"attributeNames"];
                rows = [[dataDictionary objectForKey:[entity name]] objectForKey:@"rows"];

                d = [rows count];
                for (k=0; k<d; k++) {
                        row = [NSMutableDictionary dictionaryWithObjects:[rows objectAtIndex:k]
forKeys:attributeNames];
                        [self convertStringsInRow:row forEntity:entity];
                        expression = [sqlExpressionAdaptor insertStatementForRow:row
entity:entity];
                        //d//NSLog(@"expression:%@",expression); //dd//
                        NS_DURING
                                [channel evaluateExpression:expression];
                        NS_HANDLER
                                NSLog(@"expression:%@",expression);
                                [channel closeChannel];
                                [NSException raise:@"evaluateExpression"
                                        format:@"####
IBNDBTestManager_PopulateDatabase.importDataFromList:%@ exception:%@",

                                                aPlistName, localException];
                        NS_ENDHANDLER
                }
        }
        [channel closeChannel];
```

```
	if (logTest) {
		NSLog(@"%@_Populatedatabase importDataFromList end",[self class]);
	}
}
//---------------------------------------------------------------------------
- (void)createForeignKeyConstraintsAndPrimaryKeySupportFromModel:model {
	NSMutableDictionary	*optionDictionary;

	optionDictionary = [NSMutableDictionary dictionaryWithObjectsAndKeys:
		@"NO",	@"createTables",
		@"NO",	@"dropTables",
		//@"YES",	@"createPrimaryKeySupport",
		@"NO",	@"dropPrimaryKeySupport",
		@"NO",	@"primaryKeyConstraints",
		//@"NO",	@"foreignKeyConstraints",
		//@"NO",	@"createDatabase",
		//@"NO",	@"dropDatabase",
		nil];
	if (logTest) {
		NSLog(@"%@_Populatedatabase
createStatementsWithOptionDictionary:createPrimaryKeySupport fromModel:%@",
			[self class],
							[model name]);
	}
	[self createStatementsWithOptionDictionary:optionDictionary fromModel:model
cutLongForeignKeyNames:NO];
	[optionDictionary setObject:@"NO" forKey:@"createPrimaryKeySupport"];
	[optionDictionary setObject:@"YES" forKey:@"foreignKeyConstraints"];
	if (logTest) {
		NSLog(@"%@_Populatedatabase
createStatementsWithOptionDictionary:foreignKeyConstraints fromModel:%@",
			[self class],
							[model name]);
	}
	[self createStatementsWithOptionDictionary:optionDictionary fromModel:model
cutLongForeignKeyNames:YES];
	if (logTest) {
		NSLog(@"%@_Populatedatabase createStatementsWithOptionDictionary
end",[self class]);
	}
}
//---------------------------------------------------------------------------
- (void)populateDatabase {
	NSDictionary		*dictionariesForModels;
	NSEnumerator		*allModelNames;
	NSString		*aModelName;
	NSArray			*models;
	int i,c;

	if (logTest) {
		NSLog(@"%@_Populatedatabase populateDatabase",[self class]);
	}

	dictionariesForModels = [self loadDataDictionaries];

	allModelNames = [dictionariesForModels keyEnumerator];
	while (aModelName = [allModelNames nextObject]) {
		NSDictionary	*dictionaries;
		NSEnumerator	*allPlistNames;
		NSString	*aPlistName;

		dictionaries = [dictionariesForModels objectForKey:aModelName];
		allPlistNames = [dictionaries keyEnumerator];
```

```
            while (aPlistName = [allPlistNames nextObject]) {
                [self importDataFromDictionary:[dictionaries objectForKey:aPlistName]
                                                ofPlistName:aPlistName
                                                forModel:[self modelNamed:aModelName]];
            }
        }

        models = [self neededModels];
        c = [models count];
        for (i=0; i<c; i++) {
            [self createForeignKeyConstraintsAndPrimaryKeySupportFromModel:[models
objectAtIndex:i]];
        }

        if (logTest) {
            NSLog(@"%@_Populatedatabase populateDatabase end",[self class]);
        }
}
//---------------------------------------------------------------------------
- (void)createTables {
        [self createSchema];
        [self populateDatabase];
}
//---------------------------------------------------------------------------
- (void)dropTables {
        NSMutableDictionary             *optionDictionary;
        NSArray                                 *models;
        int i,c;

        if (logTest) {
            NSLog(@"%@_Populatedatabase dropTables",[self class]);
        }

        optionDictionary = [NSMutableDictionary dictionaryWithObjectsAndKeys:
            @"NO",      @"createTables",
            //@"YES",   @"dropTables",
            @"NO",      @"createPrimaryKeySupport",
            //@"YES",   @"dropPrimaryKeySupport",
            @"NO",      @"primaryKeyConstraints",
            //@"NO",    @"foreignKeyConstraints",
            //@"NO",    @"createDatabase",
            //@"NO",    @"dropDatabase",
            nil];

        models = [self neededModels];
        c = [models count];
        for (i=0; i<c; i++) {
            if (logTest) {
                NSLog(@"%@_Populatedatabase
createStatementsWithOptionDictionary:dropTables fromModel:%@",
                                    [self class],
            [[models objectAtIndex:i] name]);
            }
            [self createStatementsWithOptionDictionary:optionDictionary
                                                                fromModel:[models
objectAtIndex:i]

        cutLongForeignKeyNames:NO];
            if (logTest) {
                NSLog(@"%@_Populatedatabase createStatementsWithOptionDictionary
end",[self class]);
            }
        }
```

```
	if (logTest) {
		NSLog(@"%@_Populatedatabase dropTables end",[self class]);
	}
}

@end
```

ConfigurationNormalizer.h

```
#import <Foundation/Foundation.h>

@interface ConfigurationNormalizer : NSObject
{
	NSString		*_adaptorName;
	NSDictionary	*defaultConfiguration;
}

- (NSMutableDictionary *)normalizeKey:(NSString *)key forTestConfiguration:(NSDictionary
*)configuration;
- (NSDictionary *)normalizeModels:(NSDictionary *)configuration;
- (NSDictionary *)normalizeTestConfiguration:(NSDictionary *)configuration;
- (NSDictionary *)defaultConfigurationForDictionary:(NSDictionary *)configuration ;
- (NSDictionary *)normalizedConfigurationDictionary:(NSDictionary *)configuration
forTestCase:(Class)testCase;

@end
```

ConfigurationNormalizer.m

```
// Legge ed interpreta le opzioni di configurazione presi da una property list

#import "ConfigurationNormalizer.h"

@implementation ConfigurationNormalizer

- (void)dealloc {
	[_adaptorName release];
	[super dealloc];
}
//-------------------------------------------------------------
- (NSMutableDictionary *)normalizeKey:(NSString *)key forTestConfiguration:(NSDictionary
*)configuration {
	NSMutableDictionary	*partialResult;
	NSDictionary		*selectedKeyConfiguration;
	NSDictionary		*localDefaultConfiguration;
	NSDictionary		*selectedAdaptorConfiguration;

	selectedKeyConfiguration = [configuration objectForKey:key];
	localDefaultConfiguration = [selectedKeyConfiguration objectForKey:@"default"];
	selectedAdaptorConfiguration = [selectedKeyConfiguration
objectForKey:_adaptorName];

	partialResult = [NSMutableDictionary dictionary];
	if (defaultConfiguration != nil) {
		[partialResult addEntriesFromDictionary:[defaultConfiguration objectForKey:key]];
	}
	[partialResult addEntriesFromDictionary:localDefaultConfiguration];
	[partialResult addEntriesFromDictionary:selectedAdaptorConfiguration];

	return partialResult;
}
//-------------------------------------------------------------
```

```
- (NSDictionary *)normalizeModels:(NSDictionary *)configuration {
	NSMutableDictionary	*result;
	NSEnumerator		*allModels;
	NSString			*model;

	result = [NSMutableDictionary dictionary];

	[result addEntriesFromDictionary:[defaultConfiguration objectForKey:@"models"]];

	allModels = [configuration keyEnumerator];
	while (model = [allModels nextObject]) {
		NSDictionary	*modelConfiguration;
		NSString		*selectedModel;

		modelConfiguration = [configuration objectForKey:model];
		selectedModel = [modelConfiguration objectForKey:_adaptorName];
		if (selectedModel == nil) {
			selectedModel = [modelConfiguration objectForKey:@"default"];
		}

		if (selectedModel == nil) {
			NSLog(@"##### Plist error on models. No model defined for: %@", model);
		} else {
			[result setObject:selectedModel forKey:model];
		}
	}

	return result;
}
//-------------------------------------------------------
- (NSDictionary *)takeDataForTestConfiguration:(NSDictionary *)configuration {
	NSDictionary	*result;

	result = [configuration objectForKey:@"data"];
	if (result == nil) {
		result = [defaultConfiguration objectForKey:@"data"];
	}
	if (result == nil) {
		result = [NSDictionary dictionary];
	}
	return result;
}
//-------------------------------------------------------
- (NSDictionary *)normalizeTestConfiguration:(NSDictionary *)configuration {
	NSMutableDictionary	*result;

	result = [NSMutableDictionary dictionary];

	[result setObject:[self normalizeKey:@"databaseConfiguration"
forTestConfiguration:configuration]
			forKey:@"databaseConfiguration"];
	[result setObject:[self normalizeKey:@"modality" forTestConfiguration:configuration]
forKey:@"modality"];
	[result setObject:[self normalizeModels:[configuration objectForKey:@"models"]]
forKey:@"models"];
	[result setObject:[self takeDataForTestConfiguration:configuration] forKey:@"data"];
	[result setObject:[self normalizeKey:@"logs" forTestConfiguration:configuration]
forKey:@"logs"];

	return result;
}
//-------------------------------------------------------
- (NSDictionary *)defaultConfigurationForDictionary:(NSDictionary *)configuration {
```

```
        _adaptorName = [[NSString alloc] initWithString:[configuration
objectForKey:@"selectedAdaptor"]];
        return [self normalizeTestConfiguration:[configuration
objectForKey:@"defaultTestConfiguration"]];
}

//------------------------------------------------------------
- (NSDictionary *)normalizedConfigurationDictionary:(NSDictionary *)configuration
forTestCase:(Class)testCase {
        NSDictionary                *result;
        NSDictionary                *testConfiguration;

        defaultConfiguration = [self defaultConfigurationForDictionary:configuration];

        testConfiguration = [[configuration objectForKey:@"tests"] objectForKey:[testCase
description]];
        if (testConfiguration == nil) {
                NSLog(@"# warning testConfiguration is taken from default configuration section
in DBTestConfiguration.plist");
                result = defaultConfiguration;
        } else {
                result = [self normalizeTestConfiguration:testConfiguration];
        }

        return result;
}

@end
```

DBTestConfiguration.plist

```
// Property list di configurazione dei test. Viene letta come dictionary.
{
//---------------------------------
        "selectedAdaptor" = "Oracle";
//---------------------------------
        "defaultTestConfiguration" = {
                "databaseConfiguration" = {
                        "default" = {
                                "password" = "defaultPassword";
                                "addictionalInfo" = {
                                        "databaseEncoding" = "ISO Latin-1";
                                        "NLS_LANG" = ".WE8ISO8859P1";
                                };
                        };
                        "Oracle" = {
                                "database" = "defaultUserName";
                                "userName" = "defaultUserName";
                                "server" = "otto";
                                "administrativeUserName" = "system";
                                "administrativePassword" = "manager";
                        };
                };
                "models" = {
                        "test" = {
                                "default" = "Test.eomodeld";
                                "Oracle" = "TestTestOracle.eomodeld";
                        };
                };
                "data" = {      "test" = ("TestTest.plist");          };
                "modality" = {
                        "default" = {
                                "deletable" = "YES";
```

```
                        "isReadOnly" = "NO";
                        "cleanDatabaseOnStart" = "NO";
                        "cleanDatabaseOnEnd" = "NO";
                    };
                };
                "logs" = {
                    "default" = {
                        "logSQL" = "NO";
                        "logTest" = "NO";
                        "logExpressions" = "NO";
                    };
                };
        };
//-------------------------------- parte personalizzata per gruppi di test
        "tests" = {
                "TestTest1" = {
                    "models" = {
                        "test" = {
                            "Oracle" = "TestTestOracle.eomodeld";
                            "FrontBase" = "TestTestFrontBase.eomodeld";
                        };
                    };
                    "data" = {     "test" = ("TestTest.plist");};
                    "modality" = {
                        "default" = {
                            "isReadOnly" = "NO";
                        };
                        "FrontBase" = {
                            "cleanDatabaseOnEnd" = "NO";
                        };
                        "Oracle" = {
                            "cleanDatabaseOnStart" = "YES";
                            "cleanDatabaseOnEnd" = "YES";
                        };
                    };
                };
        };
}
```

TestTest1.m

```
#import <Foundation/Foundation.h>
#import "../IBNDBTestCase.h"
@interface TestTest1: IBNDBTestCase
@end
```

TestTest1.m

```
// Esempio di  test.

#import "TestTest1.h"
#import <EOAccess/EOAccess.h>
#import <../IBNDBTestManager.h>

@implementation TestTest1

- (void)test_ownEditingContext {
        NSArray             *allObjects;

        allObjects = [[self editingContext] objectsForEntityNamed:@"TabellaTestOracle"];
        should([allObjects count] == 3);
}
//---------------------------------------------------------------
- (void)test_EditingContext {
```

```
	EOEditingContext  *editingContext;
	NSArray 			*allObjects;

	editingContext = [[EOEditingContext alloc] init];
	allObjects = [editingContext objectsForEntityNamed:@"TabellaTestOracle"];
	should([allObjects count] == 3);
}
//---------------------------------------------------------------
- (void)test_EditingContextWithPersonalObjectStore {
	EOObjectStoreCoordinator	*objectStoreCoordinator;
	EOEditingContext		*editingContext;
	IBNDBTestManager			*testManager;
	NSArray					*models;
	NSArray					*allObjects;
	int		i, c;

	objectStoreCoordinator = [[EOObjectStoreCoordinator alloc] init];
	editingContext = [[EOEditingContext alloc]
initWithParentObjectStore:objectStoreCoordinator];
	[objectStoreCoordinator release];

	testManager = (IBNDBTestManager *)[self DBTestManager];
	models = [testManager neededModels];
	c = [models count];
	for (i=0; i<c; i++) {
		[EODatabaseContext	forceConnectionWithModel:[models objectAtIndex:i]
			connectionDictionaryOverrides:[testManager connectionDictionary]
			editingContext:editingContext];
	}
	allObjects = [editingContext objectsForEntityNamed:@"TabellaTestOracle"];
	should([allObjects count] == 3);
}
//---------------------------------------------------------------
- (void)test_administrativeEditingContext {
	[self administrativeEditingContext];
}
//---------------------------------------------------------------
- (void)test_channelLeftOpen {
	[[[[self adaptor] createAdaptorContext] createAdaptorChannel] openChannel];
}
//---------------------------------------------------------------
- (void)test_LogSql {
	[self enableLogSQL:YES];
	[self enableLogSQL:NO];
}
@end
```

Appendice D: Test interfacce web.

SenTestCase_IBNExtra.h

```
#import <Foundation/Foundation.h>
#import <SenTestingKit/SenTestingKit.h>

@interface SenTestCase(SenTestCase_IBNExtra)

- (NSString *)resourcePath;
- (NSString *)javaPath;
- (NSString *)classPath;
```

```
- (void)compileAndRunJavaFile:(NSString *)fileName withArgs:(NSArray *)args;

@end
```

SenTestCase_IBNExtra.m

```
// Categoria che estende SenTestCase

#import "SenTestCase_IBNExtra.h"

@implementation SenTestCase(SenTestCase_IBNExtra)

- (NSString *)resourcePath {
	return [[NSBundle bundleForClass:[self class]] resourcePath];
}
- (NSString *)javaPath {
	return @"/System/Library/Frameworks/JavaVM.framework/Commands";
}
- (NSString *)classPath {
	return [NSString stringWithFormat:@"%@:%@:%@:%@:%@:%@",
//		@"/Network/Users/gabriele/Developer/java",
		@"/System/Library/Java",
		@"/System/Library/Frameworks/JavaVM.framework/Classes/classes.jar",
		@"/System/Library/Frameworks/JavaVM.framework/Classes/awt.jar",
		@"/System/Library/Frameworks/JavaVM.framework/Classes/swingall.jar",
		@"/Library/Java/xml4j.jar",
		[self resourcePath]];
}
//------------------------------------------------------------------------------
- (NSDictionary *)environment {
	NSMutableDictionary	*environment;

	environment = [NSMutableDictionary dictionaryWithDictionary:[[NSProcessInfo
processInfo] environment]];
	[environment setObject:[self classPath] forKey:@"CLASSPATH"];
	return environment;
}
//------------------------------------------------------------------------------
- (void)compileJavaFile:(NSString *)fileName {
	NSMutableArray	*args;
	NSTask			*task;

	task = [[NSTask alloc] init];
	args = [[NSMutableArray alloc] init];
	[args addObject:[NSString stringWithFormat:@"%@/%@", [self resourcePath],
fileName]];
	[task setCurrentDirectoryPath:[self resourcePath]];
	[task setArguments:args];
	[task setEnvironment:[self environment]];
	[task setLaunchPath:[NSString stringWithFormat:@"%@/javac", [self javaPath]]];
	[task launch];
	[task waitUntilExit];

	if ([task terminationStatus]) {
		[NSException raise:@"Error compiling Java file" format:@"An error occured while
compiling %@. Exit code: %d", fileName, [task terminationStatus]];
	}

	[args release];
	[task release];
}
//------------------------------------------------------------------------------
- (void)runJavaClass:(NSString *)className withArgs:(NSArray *)someArgs {
```

```
        NSMutableArray    *args;
        NSTask                  *task;

        task = [[NSTask alloc] init];
        args = [NSMutableArray arrayWithObject:className];
        [args addObjectsFromArray:someArgs];
        [task setCurrentDirectoryPath:[self resourcePath]];
        [task setArguments:args];
        [task setEnvironment:[self environment]];
        [task setLaunchPath:[NSString stringWithFormat:@"%@/java", [self javaPath]]];
        [task launch];
        [task waitUntilExit];

        if ([task terminationStatus]) {
                [NSException raise:@"Error running Java class" format:@"An error occured while
running %@.class. Exit code: %d", className, [task terminationStatus]];
        }

        [task release];
}
//---------------------------------------------------------------------------
- (void)compileAndRunJavaFile:(NSString *)fileName withArgs:(NSArray *)args {
        [self compileJavaFile:fileName];
        [self runJavaClass:[fileName stringByDeletingPathExtension] withArgs:args];
}

@end
```

Test_WebApplication.h

```
#import <WebObjects/WebObjects.h>
#import <IBNDBTestingKit/IBNDBTestingKit.h>

@interface WebDB : IBNDBTestCase
@end
```

Test_WebApplication.h

```
// Esempio di test completo con accesso a database e ad interfacce web. Il database
// viene creato e popolato automaticamente a partire dai dati nel file di configurazione.
#import <IBNExtensions/IBNExtensions.h>
#import "../Session.h"
#import "WebDB.h"

@implementation WebDB

- (void)test__1_stateOfSaveCommand {
        [self compileAndRunJavaFile:@"Test_WebApplication.java" withArgs:[NSArray
arrayWithObject:@"http://localhost:7777"]];
}

@end
```

Test_WebApplication.java

```
// File che viene compilato ed interpretato automaticamente

import com.meterware.httpunit.*;

import java.io.IOException;
import java.net.MalformedURLException;

import org.xml.sax.*;

public class Test_WebApplication {
```

```
   static void checkLinkState(WebLink webLink, boolean active, String comment) {
      if ((active && (webLink == null)) || (!active && (webLink != null))) {
         System.out.println("### Error in evaluating link state: " + comment);
         System.exit(1);
         }
      }

   public static void main( String[] params ) {
      try {
         WebRequest          request;
         WebResponse         response;
         WebConversation     conversation = new WebConversation();
         WebLink             webLink;
         String              nomeTextArea;

               // Apertura pagina view mode
         response = conversation.getResponse( params[0] );
         checkLinkState(response.getLinkWith("Salva"), false, "Salva link in view page");

               // Apertura pagina edit mode
         webLink = response.getLinkWith("Modifica");
         response = conversation.getResponse( webLink.getRequest() );
         checkLinkState(response.getLinkWith("Salva"), false, "Salva link in edit page");

               // Ricerca del parametro TextArea all'interno della form.
         WebForm form = response.getForms()[0];
         String[] parameterNames = form.getParameterNames();
         nomeTextArea = null;
         for (int i=0;i<parameterNames.length;i++) {
               if (parameterNames[i].endsWith("TextArea")) {
                     nomeTextArea = parameterNames[i];
               }
         }
               //       Settaggio del nuovo valore della TextArea
         request = form.getRequest();
         request.setParameter(nomeTextArea, "New text, programmatically setted");
         response = conversation.getResponse( request );
               //       Check dello stato del comando salva
         checkLinkState(response.getLinkWith("Salva"), true, "Salva link in modified View
page");

      } catch (Exception e) { System.err.println( "Exception: " + e ); }
   }
}
```