|  |  |  |  |
| --- | --- | --- | --- |
| Quarix ABL  Developers Manual | | 1 |  |
| Date | September 03, 2013 | |
| Version | 0.1 | |

Table of Contents

1 About Quarix 4

1.1 What is Quarix and myStrobe? 4

1.1.1 Introduction to web applications 4

1.2 Why Quarix? 4

1.2.1 Features 4

1.2.2 Scalable architecture 5

1.3 Who is using Quarix? 5

2 Using Quarix 6

2.1 Development environment 6

2.1.1 Progress OpenEdge 6

2.1.2 IDE 6

2.1.3 Database 6

2.2 Framework documentation 6

2.2.1 Architecture 6

2.2.1.1 Data integration 7

2.2.1.2 Low level architecture 8

2.2.1.3 General overview of a request 8

2.2.1.4 Packages 9

2.2.2 Business logic components 11

2.2.2.1 BaseObject 11

2.2.2.2 DataCollection 12

2.2.2.3 DataObject 21

2.2.2.4 DataTransaction 26

2.2.2.5 DataAccess 28

2.2.2.6 DataModel 33

2.2.2.7 DataContext 38

2.3 Development 42

2.3.1 Quarix by example 42

2.3.1.1 Getting Quarix 42

2.3.1.2 Configuring the framework 42

2.3.1.3 Setup the sample application 48

2.3.1.4 Defining temp-tables 52

2.3.1.5 How to define the dataset 53

2.3.1.6 Create a simple data model 53

2.3.1.7 DataAccess object 54

2.3.1.8 Getting value for calculated fields using lists of id’s 56

2.3.1.9 Data transactions 61

2.3.1.10 Setup parameters in the quarix database 63

2.3.1.11 Error handling using the try catch mechanism 65

2.3.2 Methods to override in DataModel 65

2.3.2.1 dataFetch 65

2.3.2.2 BeforeLoadData 65

2.3.2.3 AfterLoadData 66

2.3.2.4 BeforeSaveData 66

2.3.2.5 AfterSaveData 67

2.3.3 Methods to override in DataTransaction 67

2.3.3.1 dataFetch 67

2.3.3.2 BeforeDataFetch 68

2.3.3.3 AfterDataFetch 68

2.3.3.4 BeforeDataUpdate 69

2.3.3.5 AfterDataUpdate 69

2.3.4 Methods to override in DataAccess 69

2.3.4.1 AttachDataSource 69

2.3.4.2 AfterRowDelete 70

2.3.4.3 AfterRowFill 71

2.3.4.4 AfterRowSave 71

2.3.4.5 BeforePrepareQuery 71

2.3.4.6 BeforeRowDelete 72

2.3.4.7 BeforeRowFill 72

2.3.4.8 BeforeRowSave 73

2.3.5 Invoking a DataModel or DataTransaction from a procedure 73

2.3.6 DataRequest object 75

2.3.7 DataContext object 75

2.3.8 Logging and debugging! 79

2.3.8.1 Architecture of the logging mechanism 80

2.3.8.2 Error manager archirecture 81

2.3.9 Saving the data 82

2.3.10 Deployment Model 86

# About Quarix

## What is Quarix and myStrobe?

Quarix and MyStrobe are frameworks, based on a mixture of industry standard technologies, specially created to fulfill the need of rapid development and deployment of modern Web 2.0 applications. Applications based on these frameworks can be as rich as modern desktop applications without their drawbacks or as thin as regular web based application.

### Introduction to web applications

Imagine a developing environment in which your company creates multi-tier, distributed applications, in significantly less time and using far less resources than using traditional developing tools and technologies.

That is exactly what we set as target when we started developing the Quarix and myStroBe frameworks, they are specially created to fulfill the need of rapid development and deployment of modern web applications with the power of Progress OpenEdge in the backend.

MyStrobe together with Quarix are the perfect web development frameworks for Progress and not only. Build on a solid MVC (model-view-controller) pattern, completely separate the user interface from the business logic and data access.

AJAX is a web development technique used for creating interactive web applications, specifically to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user requests a change. This is intended to increase the web page's interactivity, speed, functionality, and usability.

All communication with the server side business logic is reduced to “pure data” (XML request and JSON response).

## Why Quarix?

### Features

Created to respond to a list of real business objectives among the most important ones were:

* Rich zero-footprint user interface to work on any client, any browser. The applications developed should run on any major operating system (Microsoft Windows, Linux, Mac OS-X, Sun Solaris …) in any modern web browser (Internet Explorer, Mozilla Firefox, Netscape Navigator, Safari, Konqueror, Opera). That means no installation or upgrades hustles, just point and run from any place that has an Internet connection, all this without sacrificing security
* SAAS model ready
* localization support (multi-language, multiple character-set, data field representation format)
* Great performances even over slow network connections by reducing the number of hits to the server and the data volume transferred between calls (most of the time it’s only pure data). Performance at runtime is a big issue for business application and a permanent goal for Quarix.
* scalability and availability support, based on Apache Tomcat and Progress Application Server coupled in a distributed architecture
* OS independence on both client and server side
* reduce the deployment time to minimum because of a “single-place deployment” and “zero-footprint client” which require no client updates
* greatly reduce the development time
* Customizable themes for interface modern look and feel (CSS based). Through CSS your designers can customize the applications anyway it is desired either globally or in particular, down to a specific window, giving your application a modern look and the branding it deserves
* complete separation between user interface and business logic
* straight 4GL on the server side, a diversity of common development API’s available: context, localization, security
* ease of customization by means of a series of already implemented developer hooks
* simple and straightforward, very easy to learn
* Model Driven Development (UML)
  + Separate Design from Architecture
  + Documented
* B2C – allow self-service consumers to directly access selected business functions
* B2B - expose existing business functions to partners or as services for other applications
* Remote Users – provide seamless and secure access to business functions for users on the field
* Scalable N-tier architecture
  + Progress Application Server – Running in Stateless Mode
  + Apache Tomcat – XSLT transformation
  + Apache/IIS – static content
* Compatbile with the following technologies and products:
  + Progress OpenEdge ABL;
  + Microsoft .NET Framework;
  + Microsoft SQL Server;
  + Oracle (and any other ODBC compatible DBMS);
* Handles generic data-access layer
* Uses a Repository Database
* Repository Driven Development or MDA
* Can act as a wrapper for existing backends

### Scalable architecture

The architecture building blocks consists of application servers with proved scalability and fail-over capabilities which allows one to scale from a single machine installation to a distributed n-tier environment with load-balancing and fail-over support.

Progress Application Server running in a stateless mode is the central part of the system holding the whole application business logic. Because of the stateless operation mode the Application Server can support a large number of clients with only a small spool of running agents.

## Who is using Quarix?

Typical applications that can benefit from using the frameworks are those implemented in distributed (even international) organizations, those having large user counts or those that deserves mobile workers, partners or even business to business integrations. Basically if it’s about the need of rich-user interface, scalability and performance coupled with minimal deployment costs then Quarix is the best solution.

# Using Quarix

## Development environment

### Progress OpenEdge

To be able to develop applications usin the framework it is required to use Progress OpenEdge version 10.2B or higher.

### IDE

The most suitable IDE to develop applications is OpenEdge Architect.

### Database

The framework comes together with a database called quarix, this database is used to store configurations of the projects like the application name, logging level, service configurations.

## Framework documentation

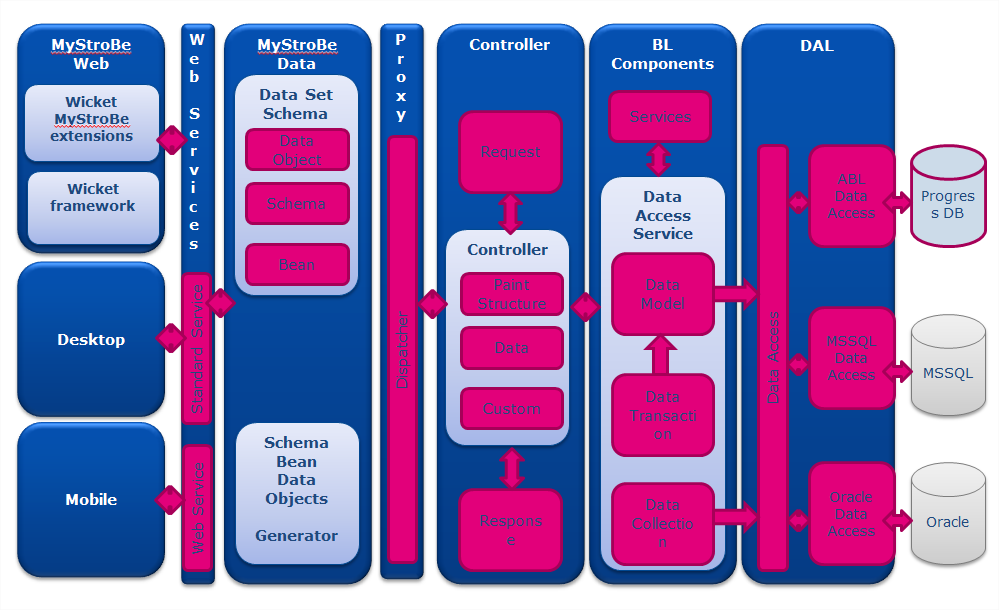
### Architecture

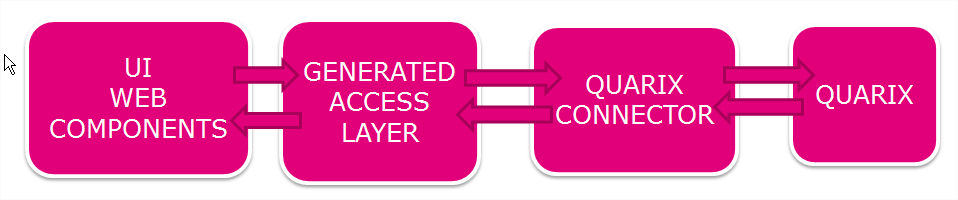
Quarix ABL Business Logic Framework is a comprehensive collection of business logic components written in Progress OO ABL which offers among others basic data access layer, common services like: localization, authentication, session and context management, error and logging management, etc.

The communication between the business logic layer and the Java middle tier is done through a single point of entry which acts as a request dispatcher while offering the standard processing for all requests (authentication, authorization, localization).

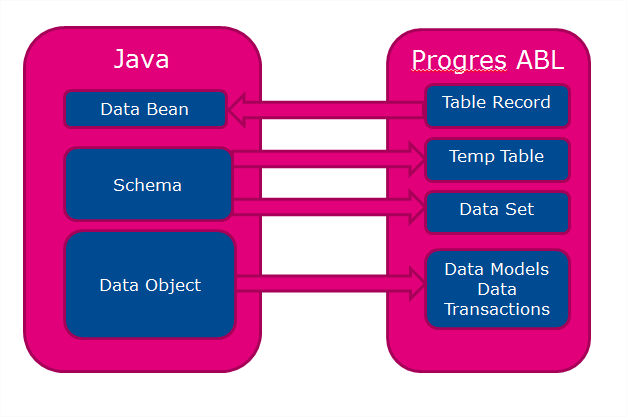
The business logic layer also provides support for accessing existing/legacy application business logic through façade components.

* OO ABL, Service Oriented Framework
  + Reusable Business Logic Components
  + Business Entities
  + Data Access Layer
  + Application Services: Localization, Session, Error Managers
  + Context management
  + Standard methods for Fetch/Store Data
  + Data serialization
  + Batching, Filter and Sorting
* Single Point of Entry (dispatcher/routing)

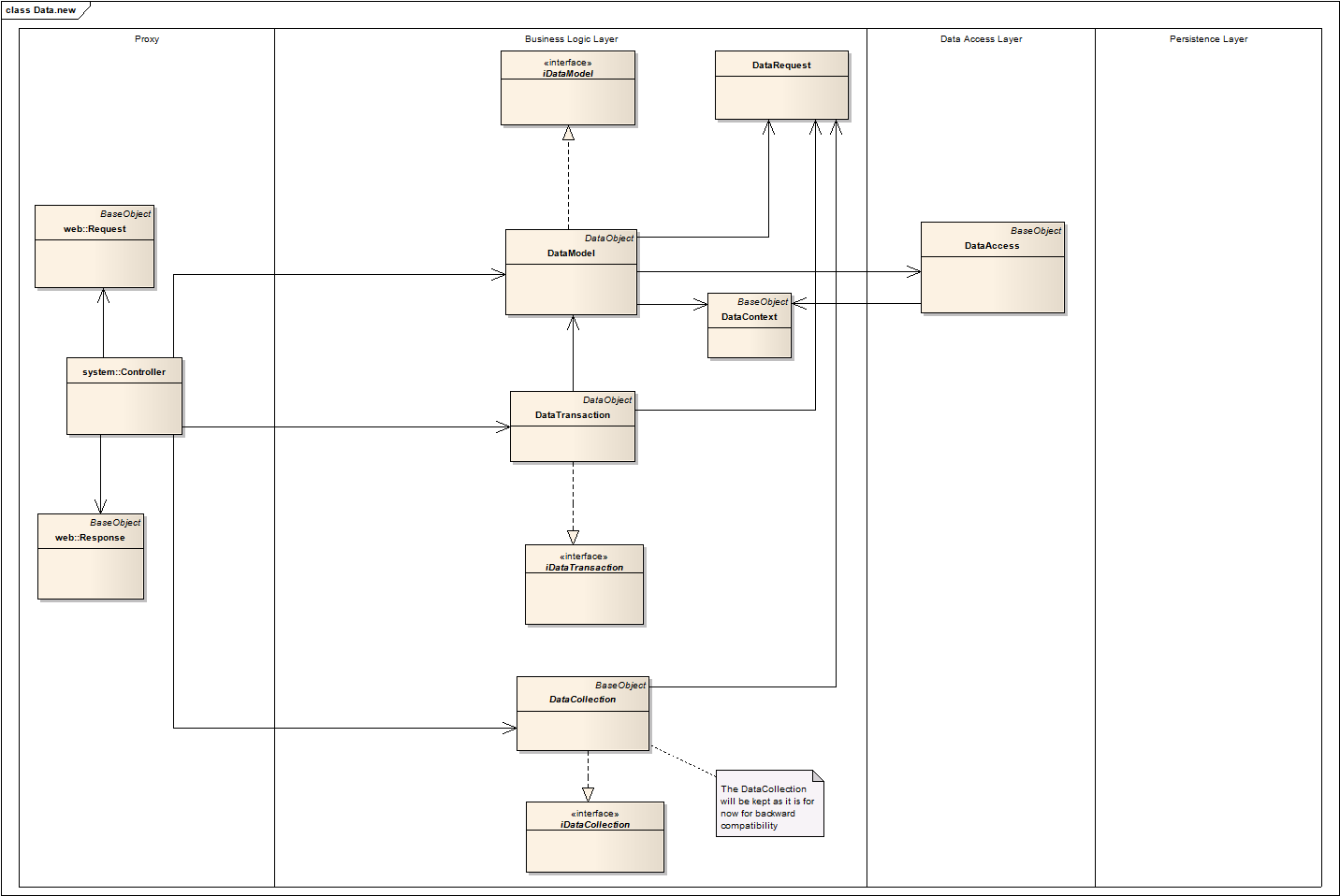




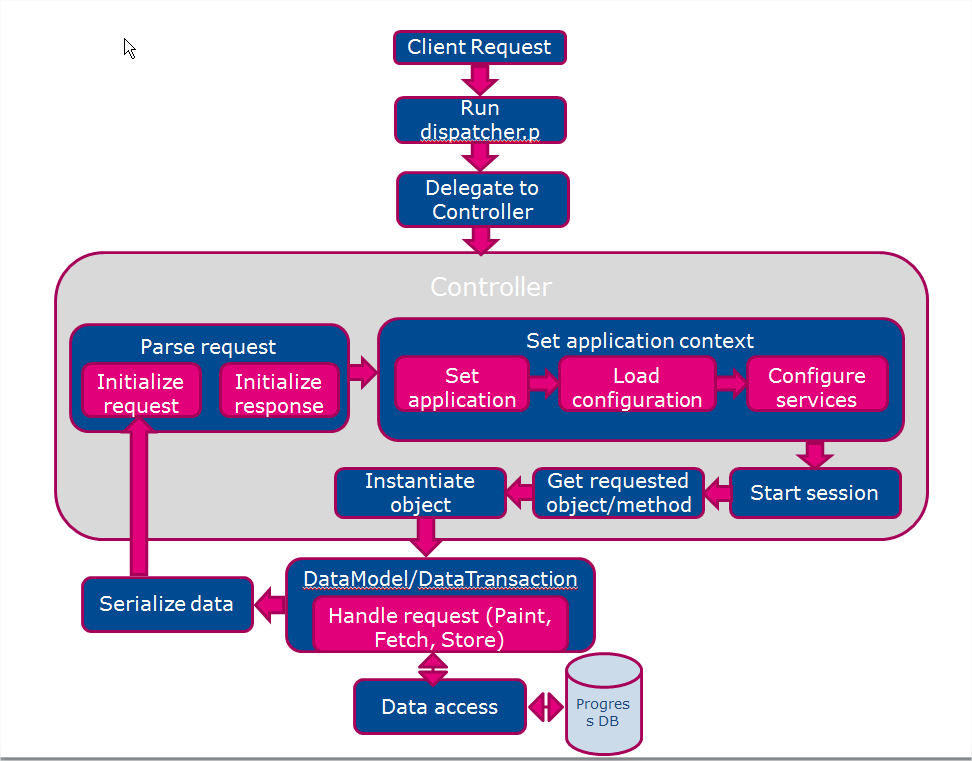
#### Data integration



#### Low level architecture



#### General overview of a request



#### Packages

The framework is grouped in packages:

* quarix.base
* quarix.bin
* quarix.data
* quarix.service
* quarix.system
* quarix.web

##### Base

Base is highest level where application classes should inherit from. The package contains:

* the base class for all application classes: BaseObject
* stack implementation classes: Collection, Array
* singleton interface: iSingleton
* singleton implementation: Factory
* procedure that checks that only one instance of a singleton exists: enforceSingleton.p
* procedure that helps instantiate classes: FactoryHelper.p
* interface iSerializable, implemented by objects that are able to serialize
* proxy class: Proxy
* utility class: Util

##### Bin

Two procedures are the most important:

* dispatcher.p - handle all calls from the interface layer
* startup.p - start persistent and hold a reference to the global Controller, start one instance of it if needed

##### Data

Contains 2 packages:

* quarix.data.parser
  + a class that serializes data into a json: JsonWriter
  + a class that implements a xml parser using SAX reader – XmlReader and a procedure that handles call-back methods of SAX Reader - XmlReaderHelper.p
* quarix.data.xsd
  + Contains XML schema used to validate xml request.

Classes used to send or receive parts of the table to/from the server:

* business logic component, class which implements interface iDataCollection and stores data as a dataset: DataCollection
* base class which holds the dataset and the methods that serialize the data in xml or json format: DataObject
* business logic components which inherit DataObject and handle client requests: DataModel and DataTransaction
* data access layer component: DataAccess
* class used to store request context: DataContext
* class used to parse the request sent by the client: DataRequest
* dataset definition used by class DataRequest to store information about client request

##### Service

Contains several packages with utility, configuration, service classes:

* quarix.service.authentication
* quarix.service.authorization
* quarix.service.configuration
* quarix.service.context
* quarix.service.error
* quarix.service.localization
* quarix.service.logging
* quarix.service.session

###### Authentication

Contains basic authentication service provider interface and it’s implementation: iAuthentication and AuthenticationCore.

###### Configuration

Configuration classes and interfaces:

* basic configuration service provider interface: iConfiguration
* abstract base class that can serve as skeleton for a configuration manager implementation: ConfigurationBase
* framework own configuration service provider: ConfigurationCore
* basic XML based configuration service provider: ConfigurationXml
* dataset and temp-table definitions used to store configuration information: dsConfig.i, ttConfig.i, ttEnvironment.i
* interface implemented by all objects that are configurable: iConfigurable

###### Context

Context service interface and classes:

* context service provider interface: iContext
* abstract base class that can serve as skeleton for a context manager implementation: ContextBase
* framework own context manager service implementation: ContextCore

###### Error

Error service manager interface and classes:

* error service manager interface: iError
* abstract implementation of the error service interface: ErrorBase
* framework own error service implementation: ErrorCore
* temp-table definition used to store error messages: ttError.i

###### Localization

Localization service manager interface and classes:

* localization service manager interface: iLocalization
* abstract base class that can serve as skeleton for a localization manager implementation: LocalizationBase
* framework own localization manager service implementation: LocalizationCore
* dataset and temp-table definitions used to store localization information: dsLocalization.i, ttLanguage.i, ttResource.i, ttResourceBundle.i

###### Logging

Logger service interface and classes:

* logger service interface: iLogger
* abstract implementation for logging service: BaseLogger
* framework own logger service records log messages in run-time repository: LoggingCore
* generic logger to record messages in a database table: DBMSLogger
* generic logger to record messages in flat files on file system: FileLogger
* generic logger to record messages using a JsonWriter object: JsonLogger
* generic logger to record messages using a SAX-WRITER object: SaxLogger

###### Session

Session service manager interface and classes:

* session service manager interface: iSession
* abstract base class that can serve as skeleton for a session manager implementation: SessionBase
* framework own session manager service implementation: SessionCore

##### Web

* interface Implemented by all objects that are to be called by the dispatcher: iWebObject
* abstract class Base Web Object: WebObject
* class that handles client request and temp-table definition used to store the data for it: Request, ttRequest.i
* class that handles server response and temp-table definition used to store the data for it: Response, ttResponse.i

### Business logic components

#### BaseObject

All application logic classes inherit from com.quarix.base.BaseObject. The methods in BaseObject are available to all classes and are all final.

##### BaseObject Attributes

* public property Application as com.quarix.system.Application - current application
* public public property ContextManager as com.quarix.service.context.iContext - application context service handler
* public property ErrorManager as com.quarix.service.error.iError - application error service handler
* public property Factory as com.quarix.base.Factory - factory handler
* public property Localization as com.quarix.service.localization.iLocalization - application localization service handler
* public property QuarixVersion as com.quarix.system.Version - framework version object
* public property Util as com.quarix.base.Util - commonly routines handler

##### BaseObject Methods

* Progress.Lang.Object GetInstance (className as character): proxy for the Factory method
* Progress.Lang.Object GetInstance (className as character, logError as logical): proxy for the Factory method, if not valid instance is returned the error is raised
* character GetString (input keyName as character): proxy for LocaleValue in localization service manager
* void ThrowDebug (input errorCode as integer, input errorText as character) and void ThrowDebug (input errorCode as integer, input errorText as character, input errorBuffer as character, input errorField as character): proxies to the AddDebug method in error service manager
* void ThrowError (input errorCode as integer, input errorText as character) and void ThrowError (input errorCode as integer, input errorText as character, input errorBuffer as character, input errorField as character): proxies to the AddError method in error service manager
* void ThrowInfo (input errorCode as integer, input errorText as character) and void ThrowInfo (input errorCode as integer, input errorText as character, input errorBuffer as character, input errorField as character): proxies to the AddInfo method in error service manager
* void ThrowWarning (input errorCode as integer, input errorText as character) and void ThrowWarning (input errorCode as integer, input errorText as character, input errorBuffer as character, input errorField as character): proxies to the AddWarning method in error service manager
* void UnloadInstance (instanceObj as Progress.Lang.Object): proxy for the Unload method in Factory
* void UnloadAll (collection as com.quarix.base.Collection): unloads all class instances from a given collection

#### DataCollection

##### Inheritance order

Inherits com.quarix.base.BaseObject and implements com.quarix.data.iDataCollection, com.quarix.web.iWebObject.

##### DataCollection Attributes

* public property AutoSync as logical – points if all the changes should be committed immediately (initial true)
* public property BatchMargin as integer – number of the records till the beginning/end of the batch and a fetch is requested (initial 5)
* public property BatchSize as integer – number of the rows requested at a time (initial 100)
* protected property DataRequest as com.quarix.data.DataRequest – request parser
* public property datasetHandle as handle – the handle of the dataset which stores the data
* public property OpenOnInit as logical – points if the dataCollection should load its initial batch of data
* public property ReadOnly as logical – shows whether any updates are allowed or not
* public property ReadOnlyFields as character – list of read-only fields
* public property RemoteChildFilter as logical – filter child records on the client, all records in child table need to be sent for all parent records sent in the batch
* public property SendChangesOnly as logical – shows whether unmodified rows are sent along the modified ones (initial true)
* public property SendFilterEveryTime as logical – send the filter conditions on each request, normaly it should stay on true as there is no state saved on the server side (initial true)
* public property ThreadTimeOut as integer – time out for server requests in miliseconds (initial 30000)
* public property UpdateMode as logical – set to true to put all visual data objects link to it in update mode automatically
* temp-table that stores data about the request: ttFilter (filters applied on tables), ttSort (sorts applied on tables), ttInfo (data about batch – first, last, current position)

##### DataCollection Methods

* void AddDataSource (pcTable as character, pcBuffer as character)
* void AddDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character)
* void AddDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character)
* void AddDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character)
* void AddDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character, excludeFields as character)
* void AddDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character, excludeFields as character, includeFields as character)
* void AddDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character, excludeFields as character, includeFields as character, joinClause as character)
* logical AddJoinClause (tableName as character, bufferName as character, joinClause as character)
* void AddPersistence (dataset-handle datasetHdl, variableName as character, includeData as logical)
* void AddPersistence (dataset-handle datasetHdl, variableName as character, includeData as logical, instanceGuid as character)
* logical AddWhereClause (tableName as character, bufferName as character, whereClause as character)
* void AfterDataService (actionName as character)
* void AfterLoadData ()
* void AfterRequest (actionName as character)
* logical AfterRowDelete (hBuffBi as handle)
* logical AfterRowSave (hBuff as handle, hDbBuff as handle)
* logical AfterSaveData ()
* void AfterService (actionName as character)
* logical AttachDataSource ()
* com.quarix.base.Collection AutocompleteKey (input tableName as character, input keyField as character, input keyValue as character)
* com.quarix.base.Collection AutocompleteKey (input tableName as character, input keyField as character, input keyValue as character, input filterXml as memptr)
* void BeforeDataOutput ()
* logical BeforeDataService (actionName as character)
* logical BeforeLoadData ()
* logical BeforeRemoveFilter (fieldName as character)
* logical BeforeRemoveSort (fieldName as character)
* logical BeforeRequest (actionName as character)
* logical BeforeRowDelete (hBuffBi as handle, hDbBuff as handle)
* logical BeforeRowSave (hBuff as handle, hBuffBi as handle, hDbBuff as handle)
* logical BeforeSaveData ()
* logical BeforeService (actionName as character)
* logical BeforeSetFilter (fieldName as character, operName as character, fieldValue as character)
* logical BeforeSetSort (fieldName as character, descendingSort as logical)
* void ClearFilters ()
* void ClearSort ()
* void ClearSort (input tableName as character)
* void DetachDataSource ()
* void DetachDataSource (pcTable as character)
* integer GetBatchSize (bufferName as character)
* character getDbRowid (bufName as character, bufRowId as character)
* character getDbRowid (hBuf as handle)
* memptr GetDescription (responseFormat as character, applicationPath as character)
* character getNewPosition (tableName as character)
* character GetPhysicalFieldName (logicalName as character)
* character getTableSort (tableName as character)
* logical HandleRequest (methodName as character, webRequest as Request, webResponse as Response)
* void HandleRequestDataService (actionName as character)
* void HandleRequestService (actionName as character)
* logical LoadData ()
* logical LoadPersistence (input-output dataset-handle datasetHdl, variableName as character, includeData as logical)
* logical LoadPersistence (input-output dataset-handle datasetHdl, variableName as character, includeData as logical, instanceGuid as character)
* character LookupKeyValue (input tableName as character, input keyField as character, input keyValue as character, input lookupField as character)
* character LookupKeyValue (input tableName as character, input keyField as character, input keyValue as character, input lookupField as character, input filterXml as memptr)
* void RemoveFilter (fieldName as character)
* void RemovePersistence (variableName as character)
* void RemovePersistence (variableName as character, onlyData as logical)
* void RemovePersistence (variableName as character, instanceGuid as character)
* void RemovePersistence (variableName as character, onlyData as logical, instanceGuid as character)
* void RemoveSort (fieldName as character)
* logical RepositionToRow (bufferName as character, repositionRowid as character)
* logical RepositionToRow (bufferName as character, repositionRowid as character, goBackward as logical)
* logical RepositionToRow (bufferName as character, repositionRowid as character, goBackward as logical, skipRecord as logical)
* logical RepositionToRow (bufferName as character, repositionRow as integer)
* logical RepositionToRow (bufferName as character, repositionRow as integer, goBackward as logical)
* logical RepositionToRow (bufferName as character, repositionRow as integer, goBackward as logical, skipRecord as logical)
* logical SaveData ()
* logical setBatchInfo (tableName as character, firstBatch as logical, lastBatch as logical)
* logical SetBatchSize (batchSize as integer)
* logical SetBatchSize (bufferName as character, batchSize as integer)
* void SetDataSource (pcTable as character, pcBuffer as character)
* void SetDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character)
* void SetDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character)
* void SetDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character)
* void SetDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character, excludeFields as character)
* void SetDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character, excludeFields as character)
* void SetDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character, excludeFields as character, includeFields as character)
* void SetDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character, excludeFields as character, includeFields as character, joinClause as character)
* void setDbRowid (hBuf as handle, dbRowid as character)
* logical SetFilter (fieldName as character, operName as character, fieldValue as character)
* logical setNewPosition (tableName as character, newRowid as character)
* logical SetSort (fieldName as character)
* logical SetSort (fieldName as character, descendingSort as logical)

###### AddDataSource

Public final method that adds a buffer to the query.

Parameters:

* pcTable as character – temp-table name
* pcBuffer as character – database buffer name
* pcKeyFields as character – comma-separated list of one or more database table fields that constitute a unique key
* pcWhere as character – query additional where clause
* pcFieldsMap as character – comma-separated list of field name pairs for all fields in a temp-table buffer that are mapped to corresponding fields Data-source object
* excludeFields as character – comma-separated list of fields omitted from query
* includeFields as character – comma-separated list of fields included in query
* joinClause as character – the join clause of the query

###### AddJoinClause

Public method that adds a join clause to data source query.

Parameters:

* tableName as character – temp-table name
* bufferName as character – database buffer name
* joinClause as character – the join clause

###### AddPersistence

Protected final method that makes a dataset persistent by saving it into context.

Parameters:

* dataset-handle datasetHdl – handle of the dataset to be persistent
* variableName as character – variable name in context in which dataset is saved
* includeData as logical – points whether the data should also be saved in context or only dataset schema
* instanceGuid as character – context section name

###### AddWhereClause

Public method that adds a where clause to data source query.

Parameters:

* tableName as character – temp-table name
* bufferName as character – database buffer name
* whereClause as character – where clause

###### AfterDataService

Developer hook run after calling a data service other than fetch or save.

Parameters:

* actionName as character – request action name

###### AfterLoadData

Developer hook run after fetching data from underlying database tables.

###### AfterRequest

Developer hook run after handling a request.

Parameters:

* actionName as character – request action name

###### AfterRowDelete

Developer hook run after deleting a record from buffer.

Parameters:

* hBuffBi as handle – before buffer handle

###### AfterRowSave

Developer hook run after creating or updating a record in database buffer.

Parameters:

* hBuff as handle – temp-table buffer
* hDbBuff as handle – database buffer

###### AfterSaveData

Developer hook run after saving data to underlying database tables.

###### AfterService

Developer hook run after handling a service request.

Parameters:

* actionName as character – request action name

###### AttachDataSource

Developer hook used to add data sources to dataset.

###### AutocompleteKey

Generates a list of strings with the values corresponding to Autocomplete request.

Parameters:

* tableName as character – requested table name
* keyField as character – requested field value
* keyValue as character – the value for which autocomplete is handled
* filterXml as memptr – requested lookup filters

###### BeforeDataOutput

Developer hook run before loading data from database tables.

###### BeforeDataService

Developer hook run before calling a data service other than fetch or save.

Parameters:

* actionName as character – request action name

###### BeforeLoadData

Developer hook run before fetching data from underlying database tables.

###### BeforeRemoveFilter

Developer hook run before removing a filter clause.

Parameters:

* fieldName as character – field name for which filter should be removed

###### BeforeRemoveSort

Developer hook run before removing a sort clause.

Parameters:

* fieldName as character – field for which sort condition should be removed

###### BeforeRequest

Developer hook run before handling a request.

Parameters:

* actionName as character – request action name

###### BeforeRowDelete

Developer hook run before deleting a record from buffer.

Parameters:

* hBuffBi as handle – before buffer handle
* hDbBuff as handle – database buffer handle

###### BeforeRowSave

Developer hook run before creating or updating a record in database buffer.

Parameters:

* hBuff as handle – temp-table buffer handle
* hBuffBi as handle – before buffer handle
* hDbBuff as handle – database buffer handle

###### BeforeSaveData

Developer hook run before saving data to underlying database tables.

###### BeforeService

Developer hook run before handling a service request.

Parameters:

* actionName as character – request action name

###### BeforeSetFilter

Developer hook run before setting a request filter.

Parameters:

* fieldName as character – filter field name
* operName as character – filter operator name
* fieldValue as character – filter value

###### BeforeSetSort

Developer hook run before setting a request sort.

Parameters:

* fieldName as character – sort condition field name
* descendingSort as logical – descending option for sort condition

###### ClearFilters

Remove all filters that were set before.

###### ClearSort

Remove all sort conditions for all the tables or only for a certain one.

Parameters:

* tableName as character – table name for which all the sort conditions are removed

###### DetachDataSource

Detaches a data source object from dataset.

Parameters:

* pcTable as character – table that should be detached

###### GetBatchSize

Returns the batch size of a given buffer.

Parameters:

* bufferName as character – buffer for which batch size is requested

###### getDbRowid

Returns full rowid of the data source.

Parameters:

* bufName as character – buffer name
* bufRowId as character – buffer rowid

###### GetDescription

Returns the response for the paint request.

Parameters:

* responseFormat as character – response format for a paint request
* applicationPath as character – path from where to get data and send changes

###### getNewPosition

Returns the current position in data source.

Parameters:

* tableName as character – table for which current position is requested

###### GetPhysicalFieldName

Returns database field name for a given temp-table field.

Parameters:

* logicalName as character – temp-table field

###### getTableSort

Returns all sort conditions for a certain table.

Parameters:

* tableName as character – buffer name

###### HandleRequest

Main method that handles all the requests.

Parameters:

* methodName as character – request name: paint, data, lookup, autocomplete, report or a custom one
* webRequest as Request – object that parses the request
* webResponse as Response – object that outputs the response

###### HandleRequestDataService

Handles a data service request other than fetch or save.

Parameters:

* actionName as character – request action name

###### HandleRequestService

Handles a request other than paint, data, lookup, autocomplete or report.

Parameters:

* actionName as character – request action name

###### LoadData

Fetch data from underlying database tables.

###### LoadPersistence

Loads in dataset a persistent dataset saved in context.

Parameters:

* dataset-handle datasetHdl – handle of persistent dataset
* variableName as character – context variable name
* includeData as logical – data load option
* instanceGuid as character – context section name

###### LookupKeyValue

Returns the description for a lookup field.

Parameters:

* tableName as character – lookup table
* keyField as character – lookup field
* keyValue as character – lookup value
* lookupField as character – description field
* filterXml as memptr – requested lookup filters

###### RemoveFilter

Remove the filter set on given buffer field.

Parameters:

* fieldName as character – filter field name

###### RemovePersistence

Removes a persistent value saved in context.

Parameters:

* variableName as character – context variable name
* onlyData as logical – option that shows if the dataset schema should be removed or only data
* instanceGuid as character – context section name

###### RemoveSort

Remove the sort condition set on given buffer field.

Parameters:

* fieldName as character – sort condition field name

###### RepositionToRow

Repositions buffer to a given row or rowid.

Parameters:

* bufferName as character – buffer name
* repositionRowid as character or repositionRow as integer – row/rowid where the buffer should reposition
* goBackward as logical – should the query reposition backward
* skipRecord as logical – should the current record be skipped or not

###### SaveData

Save changes made on the dataset, if errors should return false and set the number of errors.

###### setBatchInfo

Sets the info for the current batch.

Parameters:

* tableName as character –requested table name
* firstBatch as logical – contains first record
* lastBatch as logical – contains last record

###### SetBatchSize

Set the batch size for given buffer or the global batch size.

Parameters:

* bufferName as character – requested table name
* batchSize as integer – batch size

###### SetDataSource

Adds a data source to dataset.

Parameters:

* pcTable as character – temp-table name
* pcBuffer as character – database buffer name
* pcKeyFields as character – comma-separated list of one or more database table fields that constitute a unique key
* pcWhere as character – where clause
* pcFieldsMap as character – comma-separated list of field name pairs for all fields in a temp-table buffer that are mapped to corresponding fields Data-source object
* excludeFields as character – comma-separated list of fields omitted from query
* includeFields as character – comma-separated list of fields included in query
* joinClause as character – the join clause of the query

###### setDbRowid

Sets the full rowid of the data source.

Parameters:

* hBuf as handle – requested buffer handle
* dbRowid as character – full rowid

###### SetFilter

Set a filter on given buffer field.

Parameters:

* fieldName as character – filter field name
* operName as character – filter operator name
* fieldValue as character – filter value

###### setNewPosition

Sets the current position in data source.

Parameters:

* tableName as character – requested table
* newRowid as character – current position

###### SetSort

Set a sort condition on given buffer field.

Parameters:

* fieldName as character – sort condition field name
* descendingSort as logical – descending option for sort condition

#### DataObject

##### Inheritance order

Inherits com.quarix.base.BaseObject.

##### DataObject Attributes

* public property AutoSync as logical – points if all the changes should be committed immediately (initial true)
* public property BatchMargin as integer – number of the records till the beginning/end of the batch and a fetch is requested (initial 5)
* public property BatchSize as integer – number of the rows requested at a time
* public property DataContext as com.quarix.data.DataContext – data object’s context
* protected property DataRequest as com.quarix.data.DataRequest – request parser
* public property datasetHandle as handle – the handle of the dataset which stores the data
* public property LogResponse as logical –if true data object’s response is dumped to a file for debugging purposes (initial false)
* public property OpenOnInit as logical – points if the dataObject should load its initial batch of data
* public property ReadOnly as logical – shows whether any updates are allowed or not
* public property RemoteChildFilter as logical – filter child records on the client, all records in child table need to be sent for all parent records sent in the batch
* public property SendChangesOnly as logical – shows whether unmodified rows are sent along the modified ones (initial true)
* public property SendFilterEveryTime as logical – send the filter conditions on each request, normaly it should stay on true as there is no state saved on the server side (initial true)
* public property ThreadTimeOut as integer – time out for server requests in miliseconds (initial 30000)
* public property UpdateMode as logical – set to true to put all visual data objects link to it in update mode automatically

##### DataObject Methods

* void BeforeDataOutput ()
* void ClearContext()
* void ClearFilters ()
* void ClearProperties()
* void ClearProperty(input PropertyName as character)
* void ClearQueryData()
* void ClearSort ()
* void dataFetch ()
* void dataFetch (output dataset-handle dsHandle)
* void dataFetchIdList(input idlist as com.quarix.data.IdList, output dataset-handle hDsDataset)
* void dataFetchIdList(input idlist as com.quarix.data.IdList, output dataset-handle hDsDataset, input QueryName as character)
* void dataOutput()
* void dataUpdate ()
* void dataUpdate(input-output dataset-handle dsHandle)
* void DisableLogResponse()
* void DumpToXml()
* void DumpToXml(input pcName as character)
* void EnableLogResponse()
* handle getDefaultDatasetHandle()
* memptr GetDescription (responseFormat as character, applicationPath as character)
* character GetFilterValue (tableName as character, fieldname as character, operName as character)
* handle GetIdListTmpHandle()
* character GetProperty(input PropertyName as character)
* character GetQueryName()
* logical HandleRequest (methodName as character, webRequest as Request, webResponse as Response)
* logical IsFirstBatch()
* logical IsLastBatch()
* logical LoadRequestData ()
* void localizeDataset ()
* logical PropertyAvailable()
* logical PropertyAvailable(input PropertyName as character)
* void RemoveFilter (tableName as character, fieldName as character)
* void RemoveSort( input tableName as character, input fieldName as character )
* void RemoveTableFilters (tableName as character)
* void RemoveTableSort( input tableName as character )
* public logical SaveData ()
* logical serializeBufferToJson (table-handle tableHandle)
* void serializeErrors (bufferName as character)
* logical SetBatchSize ()
* void SetBatchSizeFromRequest()
* void SetDataContext (input dtContext as DataContext)
* logical SetFilter (tableName as character, fieldName as character, operName as character, fieldValue as character)
* void SetFiltersFromRequest()
* void SetIdListTmpHandle(input PropertyValue as handle)
* void SetParentBatchSize (parentBatchSize as integer)
* void SetProperty(input PropertyName as character, input PropertyValue as character)
* void SetQueryName(input PropertyValue as character)
* void SetRequestFilters (tableName as character, considerSearchFilter as logical)
* logical SetSort( input tableName as character, input fieldName as character, input descendingSort as logical )
* void SetSortFromRequest()

###### ClearContext

Clears data object’s context data (filters, sort, batch info, row id).

###### ClearFilters

Clears data object’s filter info.

###### ClearProperties

Clears data object’s request properties.

###### ClearProperty

Deletes a given property.

Parameters:

* PropertyName as character – property name

###### ClearQueryData

Cleats only context data necessary for query (filters, sort, properties).

###### ClearSort

Clears data object’s sort info.

###### dataFetch

User custom fetch method.

Parameters:

* output dataset-handle dsHandle – data object’s dataset

###### dataFetchIdList

Loads data for data object using a list of unique keys.

Parameters:

* idlist as com.quarix.data.IdList – object that represents the list of unique keys
* output dataset-handle hDsDataset – the dataset with the records to be returned
* QueryName as character – option that indicates if a data source is attached the default way or using a list

###### dataOutput

Serializes all dataset’s data into a JSON.

###### dataUpdate

User custom update method.

Parameters:

* input-output dataset-handle dsHandle – data object’s dataset

###### DisableLogResponse

Disables data object’s response dump into a file.

###### DumpToXml

Write data object’s dataset into a xml file.

Parameters:

* pcName as character – xml file name

###### EnableLogResponse

Enables data object’s response dump into a file.

###### GetDescription

Returns the response for the paint request.

Parameters:

* responseFormat as character – response format for a paint request
* applicationPath as character – path from where to get data and send changes

###### GetFilterValue

Returns filter value.

Parameters:

* tableName as character – table name
* fieldname as character – field name
* operName as character – filter operator

###### GetIdListTmpHandle

Returns the handle of the temp-table that represents the list with unique keys.

###### GetProperty

Returns property’s value.

Parameters:

* PropertyName as character – property name

###### GetQueryName

Returns the value of the property QueryName.

###### PropertyAvailable

Checks if there is any or a certain property defined for data object.

Parameters:

* PropertyName as character – property name

###### RemoveFilter

Removes a filter of the table.

Parameters:

* tableName as character – table name
* fieldName as character – field name

###### RemoveSort

Removes a sort condition of the table.

Parameters:

* tableName as character – table name
* fieldName as character – field name

###### RemoveTableFilters

Removes all the filters of the table.

Parameters:

* tableName as character – table name

###### RemoveTableSort

Removes all the sort conditions of the table.

Parameters:

* tableName as character – table name

###### serializeBufferToJson

Serializes the data from a table into a json string.

Parameters:

* table-handle tableHandle – table handle

###### SetBatchSize

Sets the global batch size.

###### SetBatchSizeFromRequest

Sets buffer batch size with the value from client request, if any is sent, and saves it to context.

###### SetDataContext

Sets a data context, instead of the default one.

Parameters:

* dtContext as DataContext

###### SetFilter

Sets a filter condition.

Parameters:

* tableName as character – table name
* fieldName as character – field name
* operName as character – filter operator
* fieldValue as character – filter value

###### SetFiltersFromRequest

Sets in data object’s context the filters and properties sent in the request.

###### SetIdListTmpHandle

Save the handle of the list with unique keys into context.

Parameters:

* PropertyValue as handle – the handle of the list

###### SetParentBatchSize

Save the given batch size to all dataset buffers info.

Parameters:

* parentBatchSize as integer – batch size

###### SetProperty

Sets a property for data object.

Parameters:

* input PropertyName as character – property name
* input PropertyValue as character – property value

###### SetQueryName

Sets property QueryName.

Parameters:

* PropertyValue as character – property value

###### SetRequestFilters

Sets the requested filters for a given table.

Parameters:

* tableName as character – requested table
* considerSearchFilter as logical – considers search filter or not

###### SetSort

Sets a sort condition.

Parameters:

* tableName as character – table name
* fieldName as character – field name
* descendingSort as logical – sort value

###### SetSortFromRequest

Sets in data object’s context the sort conditions sent in the request.

#### DataTransaction

##### Inheritance order

Inherits com.quarix.data.DataObject and implements com.quarix.data.iDataTransaction, com.quarix.web.iWebObject

##### DataTransaction Methods

* void AddDataModel (dataObject as com.quarix.data.DataObject, dsHandle as handle, isMain as logical)
* void AddDataModel (dataObject as com.quarix.data.DataObject, dsHandle as handle, isMain as logical, cRequestMapping as character)
* void AddDataModel (dataObject as com.quarix.data.DataObject, dsHandle as handle, isMain as logical, cRequestMapping as character, cMethodName as character)
* void AddDataModel (dataObject as com.quarix.data.DataObject, isMain as logical)
* void AddDataModel (dataObject as com.quarix.data.DataObject, isMain as logical, cRequestMapping as character)
* void AddDataModel (dataObject as com.quarix.data.DataObject, isMain as logical, cRequestMapping as character, cMethodName as character)
* void AfterDataFetch ()
* logical BeforeDataFetch ()
* void dataFetch (output dataset-handle dsHandle)
* void dataUpdate ()
* logical HandleRequest( input actionName as character, input webRequest as com.quarix.web.Request, input webResponse as com.quarix.web.Response )
* void HandleRequestDataService(actionName as character)
* void HandleRequestService (actionName as character)
* logical serializeBufferToJson (table-handle tableHandle)
* void setBatchInfo(input dtObject as com.quarix.data.DataObject)
* void SetMainDataModel(dataObject as com.quarix.data.DataObject, dsHandle as handle, cRequestMapping as character, cMethodName as character)
* void UpdateTtRowId(input hBufTarget as handle, input hBufSrc as handle)

###### AddDataModel

Adds or updates a valid data object to data transaction list of objects.

Parameters:

* dataObject as com.quarix.data.DataObject – valid data object
* dsHandle as handle – data object’s dataset
* isMain as logical – shows if it is the main data object
* cRequestMapping as character – represents the relation between a temp-table of the data transaction and a source temp-table of the component data object
* cMethodName as character – the method to be used instead of the default fetch

###### AfterDataFetch

Developer hook run after data fetch.

###### BeforeDataFetch

Developer hook run before data fetch.

###### dataFetch

Loads data in dataset and returns dataset handle. Calls the datafetch method or the user define method of the main data object and fills the default dataset, if a custom one is not defined.

Parameters:

* output dataset-handle dsHandle – quarix dataset handle

###### dataUpdate

Saves data in dataset and returns dataset handle.

Parameters:

* output dataset-handle dsHandle – quarix dataset handle

###### HandleRequest

Main method that handles all the requests.

Parameters:

* methodName as character – request name: paint, data, lookup, autocomplete, report or a custom one
* webRequest as Request – object that parses the request
* webResponse as Response – object that outputs the response

###### HandleRequestDataService

Handles a data service request other than fetch or save.

Parameters:

* actionName as character – request action name

###### HandleRequestService

Handles a request other than paint, data, lookup, autocomplete or report.

Parameters:

* actionName as character – request action name

###### serializeBufferToJson

Serializes the data from a table into a json string.

Parameters:

* table-handle tableHandle – table handle

###### setBatchInfo

Sets the batch information and the rowids for all the buffers from the dataset that have a request.

Parameters:

* dtObject as com.quarix.data.DataObject – the component data object from which batch info should be copied

###### SetMainDataModel

Sets a data transaction’s data object as main.

Parameters:

* dataObject as com.quarix.data.DataObject
* dsHandle as handle
* cRequestMapping as character
* cMethodName as character

###### UpdateTtRowId

Parameters:

* hBufTarget as handle
* hBufSrc as handle

#### DataAccess



##### Inheritance order

Inherits com.quarix.base.BaseObject and implements com.quarix.data.iDataAccess.

##### DataAccess Attributes

* public property DataContext as com.quarix.data.DataContext – class that stores all context data
* public property DataRequest as com.quarix.data.DataRequest – request parser
* protected property datasetHandle as handle – the handle of the dataset which stores the data
* public property LogQueryString as logical – property used for debugging, displays query string (initial false)
* public property ReadOnlyFields as character – list of read-only fields
* public property RemoteChildFilter as logical – filter child records on the client, all records in child table need to be sent for all parent records sent in the batch

##### DataAccess Methods

* void AddDataSource (pcTable as character, pcBuffer as character)
* void AddDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character)
* void AddDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character)
* void AddDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character)
* void AddDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character, excludeFields as character)
* void AddDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character, excludeFields as character, includeFields as character)
* void AddDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character, excludeFields as character, includeFields as character, joinClause as character)
* logical AddJoinClause (tableName as character, bufferName as character, joinClause as character)
* void AddPersistence (dataset-handle datasetHdl, variableName as character, includeData as logical)
* void AddPersistence (dataset-handle datasetHdl, variableName as character, includeData as logical, instanceGuid as character)
* logical AddWhereClause (tableName as character, bufferName as character, whereClause as character)
* logical AfterRowDelete (hBuffBi as handle)
* logical AfterRowFill (input hBuf as handle)
* logical AfterRowSave (hBuff as handle, hDbBuff as handle)
* logical AfterSaveData ()
* logical AttachDataSource ()
* com.quarix.base.Collection AutocompleteKey (input tableName as character, input keyField as character, input keyValue as character)
* com.quarix.base.Collection AutocompleteKey (input tableName as character, input keyField as character, input keyValue as character, input filterXml as memptr)
* logical BeforeRowDelete (hBuffBi as handle, hDbBuff as handle)
* logical BeforeRowFill(input hBuf as handle)
* logical BeforeRowSave (hBuff as handle, hBuffBi as handle, hDbBuff as handle)
* logical BeforeSaveData ()
* void DetachDataSource ()
* void DetachDataSource (pcTable as character)
* void DisableDebugQuery()
* void EnableDebugQuery()
* character getFullFieldName (fieldname as character)
* handle GetIdListTmpHandle()
* character GetKeyValue (input tableName as character, input keyField as character, input keyValue as character, input lookupField as character, input filterXml as memptr)
* character GetPhysicalFieldName (logicalName as character)
* character GetQueryName()
* logical LoadData ()
* void localizeDataset (dataset-handle dsHandle)
* logical RepositionToRow (bufferName as character, repositionRowid as character)
* logical RepositionToRow (bufferName as character, repositionRowid as character, goBackward as logical)
* logical RepositionToRow (bufferName as character, repositionRowid as character, goBackward as logical, skipRecord as logical)
* logical RepositionToRow (bufferName as character, repositionRow as integer)
* logical RepositionToRow (bufferName as character, repositionRow as integer, goBackward as logical)
* logical RepositionToRow (bufferName as character, repositionRow as integer, goBackward as logical, skipRecord as logical)
* logical SaveData ()
* void SetDataModel (dtModel as com.quarix.data.DataModel)
* void SetDataSource (pcTable as character, pcBuffer as character)
* void SetDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character)
* void SetDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character)
* void SetDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character)
* void SetDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character, excludeFields as character)
* void SetDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character, excludeFields as character)
* void SetDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character, excludeFields as character, includeFields as character)
* void SetDataSource (pcTable as character, pcBuffer as character, pcKeyFields as character, pcWhere as character, pcFieldsMap as character, excludeFields as character, includeFields as character, joinClause as character)

###### AddDataSource

Public final method that adds a buffer to the query.

Parameters:

* pcTable as character – temp-table name
* pcBuffer as character – database buffer name
* pcKeyFields as character – comma-separated list of one or more database table fields that constitute a unique key
* pcWhere as character – query additional where clause
* pcFieldsMap as character – comma-separated list of field name pairs for all fields in a temp-table buffer that are mapped to corresponding fields Data-source object
* excludeFields as character – comma-separated list of fields omitted from query
* includeFields as character – comma-separated list of fields included in query
* joinClause as character – the join clause of the query

###### AddJoinClause

Public method that adds a join clause to data source query.

Parameters:

* tableName as character – temp-table name
* bufferName as character – database buffer name
* joinClause as character – the join clause

###### AddPersistence

Protected final method that makes a dataset persistent by saving it into context.

Parameters:

* dataset-handle datasetHdl – handle of the dataset to be persistent
* variableName as character – variable name in context in which dataset is saved
* includeData as logical – points whether the data should also be saved in context or only dataset schema
* instanceGuid as character – context section name

###### AddWhereClause

Public method that adds a where clause to data source query.

Parameters:

* tableName as character – temp-table name
* bufferName as character – database buffer name
* whereClause as character – where clause

###### AfterRowDelete

Developer hook run after deleting a record from buffer.

Parameters:

* hBuffBi as handle – before buffer handle

###### AfterRowFill

Developer hook run after filling a record from buffer.

Parameters:

* hBuf as handle – buffer handle

###### AfterRowSave

Developer hook run after creating or updating a record in database buffer.

Parameters:

* hBuff as handle – temp-table buffer
* hDbBuff as handle – database buffer

###### AfterSaveData

Developer hook run after saving data to underlying database tables.

###### AttachDataSource

Developer hook used to add data sources to dataset.

###### AutocompleteKey

Generates a list of strings with the values corresponding to Autocomplete request.

Parameters:

* tableName as character – requested table name
* keyField as character – requested field value
* keyValue as character – the value for which autocomplete is handled
* filterXml as memptr – requested lookup filters

###### BeforeRowDelete

Developer hook run before deleting a record from buffer.

Parameters:

* hBuffBi as handle – before buffer handle
* hDbBuff as handle – database buffer handle

###### BeforeRowFill

Developer hook run before filling a record from buffer.

Parameters:

* hBuf as handle – buffer handle

###### BeforeRowSave

Developer hook run before creating or updating a record in database buffer.

Parameters:

* hBuff as handle – temp-table buffer handle
* hBuffBi as handle – before buffer handle
* hDbBuff as handle – database buffer handle

###### BeforeSaveData

Developer hook run before saving data to underlying database tables.

###### DetachDataSource

Detaches a data source object from dataset.

Parameters:

* pcTable as character – table that should be detached

###### DisableDebugQuery

Sets LogQueryString to false.

###### EnableDebugQuery

Sets LogQueryString to true.

###### getFullFieldName

Returns field’s database name.

Parameters:

* fieldname as character – field for which database name is requested

###### GetIdListTmpHandle

###### GetKeyValue

Returns the description for a lookup field.

Parameters:

* tableName as character – lookup table
* keyField as character – lookup field
* keyValue as character – lookup value
* lookupField as character – description field
* filterXml as memptr – requested lookup filters

###### GetPhysicalFieldName

Returns database field name for a given temp-table field.

Parameters:

* logicalName as character – temp-table field

###### GetQueryName

###### LoadData

Fetch data from underlying database tables.

###### localizeDataset

Sets dataset’s logical fields format using localization manager.

Parameters:

* dataset-handle datasetHdl – handle of the dataset

###### RepositionToRow

Repositions buffer to a given row or rowid.

Parameters:

* bufferName as character – buffer name
* repositionRowid as character or repositionRow as integer – row/rowid where the buffer should reposition
* goBackward as logical – should the query reposition backward
* skipRecord as logical – should the current record be skipped or not

###### SaveData

Save changes made on the dataset, if errors should return false and set the number of errors.

###### SetDataModel

Sets the data model.

Parameters:

* dtModel as DataModel – data model to be set

###### SetDataSource

Adds a data source to dataset.

Parameters:

* pcTable as character – temp-table name
* pcBuffer as character – database buffer name
* pcKeyFields as character – comma-separated list of one or more database table fields that constitute a unique key
* pcWhere as character – where clause
* pcFieldsMap as character – comma-separated list of field name pairs for all fields in a temp-table buffer that are mapped to corresponding fields Data-source object
* excludeFields as character – comma-separated list of fields omitted from query
* includeFields as character – comma-separated list of fields included in query
* joinClause as character – the join clause of the query

#### DataModel

##### Inheritance order

Inherits com.quarix.data.DataObject and implements com.quarix.data.iDataCollection, com.quarix.web.iWebObject.

##### DataModel Attributes

* protected property ActionName as character – request’s action name
* public property DataAccess as com.quarix.data.DataAccess – data model’s data access
* protected property EnableCache as logical – response’s enable cache option

##### DataModel Methods

* void AddPersistence (dataset-handle datasetHdl, variableName as character, includeData as logical)
* void AddPersistence (dataset-handle datasetHdl, variableName as character, includeData as logical, instanceGuid as character)
* void AfterDataService (actionName as character)
* void AfterLoadData ()
* void AfterRequest (actionName as character)
* logical AfterSaveData ()
* void AfterService (actionName as character)
* logical BeforeDataService (actionName as character)
* logical BeforeLoadData ()
* logical BeforeRemoveFilter (fieldName as character)
* logical BeforeRemoveSort (fieldName as character)
* logical BeforeRequest (actionName as character)
* logical BeforeSaveData ()
* logical BeforeService (actionName as character)
* logical BeforeSetFilter (fieldName as character, operName as character, fieldValue as character)
* logical BeforeSetSort (fieldName as character, descendingSort as logical)
* void dataFetch (output dataset-handle dsHandle)
* integer GetBatchSize (bufferName as character)
* character GetKeyValue (input tableName as character, input keyField as character, input keyValue as character, input lookupField as character)
* character GetKeyValue (input tableName as character, input keyField as character, input keyValue as character, input lookupField as character, input filterXml as memptr)
* logical HandleRequest (methodName as character, webRequest as Request, webResponse as Response)
* void HandleRequestDataService (actionName as character)
* void HandleRequestService (actionName as character)
* logical IsFirstBatch()
* logical IsLastBatch()
* logical LoadData ()
* logical LoadPersistence (input-output dataset-handle datasetHdl, variableName as character, includeData as logical)
* logical LoadPersistence (input-output dataset-handle datasetHdl, variableName as character, includeData as logical, instanceGuid as character)
* void localizeDataset ()
* void RemoveFilter (fieldName as character)
* void RemovePersistence (variableName as character)
* void RemovePersistence (variableName as character, onlyData as logical)
* void RemovePersistence (variableName as character, instanceGuid as character)
* void RemovePersistence (variableName as character, onlyData as logical, instanceGuid as character)
* void RemoveSort (fieldName as character)
* logical SaveData ()
* logical serializeBufferToJson (table-handle tableHandle)
* void SetDataContext (input dtContext as DataContext)
* logical SetFilter (fieldName as character, operName as character, fieldValue as character)
* logical SetFilter (tableName as character, fieldName as character, operName as character, fieldValue as character)
* logical SetSort (fieldName as character)
* logical SetSort( input tableName as character, input fieldName as character )
* logical SetSort( input tableName as character, input fieldName as character, input descendingSort as logical )
* logical SetSort (fieldName as character, descendingSort as logical)
* void setUseIndex (tableName as character, useIndex as character)

###### AddPersistence

Protected final method that makes a dataset persistent by saving it into context.

Parameters:

* dataset-handle datasetHdl – handle of the dataset to be persistent
* variableName as character – variable name in context in which dataset is saved
* includeData as logical – points whether the data should also be saved in context or only dataset schema
* instanceGuid as character – context section name

###### AfterDataService

Developer hook run after calling a data service other than fetch or save.

Parameters:

* actionName as character – request action name

###### AfterLoadData

Developer hook run after fetching data from underlying database tables.

###### AfterRequest

Developer hook run after handling a request.

Parameters:

* actionName as character – request action name

###### AfterSaveData

Developer hook run after saving data to underlying database tables.

###### AfterService

Developer hook run after handling a service request.

Parameters:

* actionName as character – request action name

###### BeforeDataService

Developer hook run before calling a data service other than fetch or save.

Parameters:

* actionName as character – request action name

###### BeforeLoadData

Developer hook run before fetching data from underlying database tables.

###### BeforeRemoveFilter

Developer hook run before removing a filter clause.

Parameters:

* fieldName as character – field name for which filter should be removed

###### BeforeRemoveSort

Developer hook run before removing a sort clause.

Parameters:

* fieldName as character – field for which sort condition should be removed

###### BeforeRequest

Developer hook run before handling a request.

Parameters:

* actionName as character – request action name

###### BeforeSaveData

Developer hook run before saving data to underlying database tables.

###### BeforeService

Developer hook run before handling a service request.

Parameters:

* actionName as character – request action name

###### BeforeSetFilter

Developer hook run before setting a request filter.

Parameters:

* fieldName as character – filter field name
* operName as character – filter operator name
* fieldValue as character – filter value

###### BeforeSetSort

Developer hook run before setting a request sort.

Parameters:

* fieldName as character – sort condition field name
* descendingSort as logical – descending option for sort condition

###### dataFetch

Loads data in dataset and returns dataset handle.

Parameters:

* output dataset-handle dsHandle – quarix dataset handle

###### GetBatchSize

Returns the batch size of a given buffer.

Parameters:

* bufferName as character – buffer for which batch size is requested

###### GetKeyValue

Returns the description for a lookup field.

Parameters:

* tableName as character – lookup table
* keyField as character – lookup field
* keyValue as character – lookup value
* lookupField as character – description field
* filterXml as memptr – requested lookup filters

###### HandleRequest

Main method that handles all the requests.

Parameters:

* methodName as character – request name: paint, data, lookup, autocomplete, report or a custom one
* webRequest as Request – object that parses the request
* webResponse as Response – object that outputs the response

###### HandleRequestDataService

Handles a data service request other than fetch or save.

Parameters:

* actionName as character – request action name

###### HandleRequestService

Handles a request other than paint, data, lookup, autocomplete or report.

Parameters:

* actionName as character – request action name

###### IsFirstBatch

Checks if it is the first batch of the data model.

###### IsLastBatch

Checks if it is the last batch of the data model.

###### LoadData

Fetch data from underlying database tables.

###### LoadPersistence

Loads in dataset a persistent dataset saved in context.

Parameters:

* dataset-handle datasetHdl – handle of persistent dataset
* variableName as character – context variable name
* includeData as logical – data load option
* instanceGuid as character – context section name

###### RemoveFilter

Remove the filter set on given buffer field.

Parameters:

* fieldName as character – filter field name

###### RemovePersistence

Removes a persistent value saved in context.

Parameters:

* variableName as character – context variable name
* onlyData as logical – option that shows if the dataset schema should be removed or only data
* instanceGuid as character – context section name

###### RemoveSort

Remove the sort condition set on given buffer field.

Parameters:

* fieldName as character – sort condition field name

###### SaveData

Save changes made on the dataset, if errors should return false and set the number of errors.

###### serializeBufferToJson

Serializes the data from a table into a json string.

Parameters:

* table-handle tableHandle – table handle

###### SetDataContext

Sets DataModel and DataAccess context.

Parameters:

* dtContext as DataContext - data context object

###### SetFilter

Set a filter on given buffer field.

Parameters:

* fieldName as character – filter field name
* operName as character – filter operator name
* fieldValue as character – filter value

###### SetSort

Set a sort condition on given buffer field.

Parameters:

* fieldName as character – sort condition field name
* descendingSort as logical – descending option for sort condition

###### setUseIndex

Sets in data model’s context the index to be used.

Parameters:

* tableName as character – table name
* useIndex as character – index name

#### DataContext

##### Inheritance order

Inherits com.quarix.base.BaseObject.

##### DataContext Attributes

* public property dsContextHandle as handle – context dataset handle

##### DataContext Methods

* logical AddWhereClause (tableName as character, bufferName as character, whereClause as character, isJoin as logical)
* void ClearContext()
* void ClearFilters ()
* void ClearProperties()
* void ClearProperty(PropertyName as character)
* void ClearQueryData()
* void ClearSort ()
* void CopyFilters( DataObject as com.quarix.data.DataObject, srctableName as character)
* void CopyFilters( DataObject as com.quarix.data.DataObject, srctableName as character, tableName as character)
* void CopyFilters( DataObject as com.quarix.data.DataObject, srctableName as character, tableName as character, srcfieldName as character)
* void CopyFilters( DataObject as com.quarix.data.DataObject, srctableName as character, tableName as character, srcfieldName as character, operName as character)
* void CopyFilters( DataObject as com.quarix.data.DataObject, srctableName as character, tableName as character, srcfieldName as character, fieldname as character, operName as character)
* void CopyProperties(input DataObject as com.quarix.data.DataObject, input propertyName as character)
* void CopyPropertyIdList(input DataObject as com.quarix.data.DataObject)
* void CopyPropertyIdList(input DataObject as com.quarix.data.DataObject, input QueryName as character)
* void CopyRowId(input targetTable as character, input tableName as character, input dataset-handle hdscontext)
* void CreateRowId(input tableName as character, input ttRowId as character, input dbRowId as character)
* void DumpToXml()
* void DumpToXml(input pcName as character)
* integer getBatchSize (tableName as character)
* character getDbRowid (hBuf as handle)
* character getFieldFilterCondition (fieldName as character, fieldOper as character, fieldVal as character)
* character getFilterArray(tableName as character)
* character GetFilterOperator (tableName as character, fieldName as character, fieldValue as character)
* character GetFilterValue (tableName as character, fieldname as character, operName as character )
* logical getFirstBatch (tableName as character)
* handle GetIdListTmpHandle()
* logical getLastBatch (tableName as character)
* character getLookupFilter (lookupTable as character, bufferName as character, xmlFilter as memptr)
* character getNewPosition (tableName as character)
* character GetPrivateData ()
* character GetProperty(input PropertyName as character)
* character GetQueryName()
* void getQueryTables (output table ttFilter bind, output table ttSort bind)
* character getSortArray(tableName as character)
* character getTableSort (tableName as character)
* character getUseIndex (tableName as character)
* logical PropertyAvailable()
* logical PropertyAvailable(input PropertyName as character)
* void RemoveFilter (fieldName as character)
* void RemoveSort (fieldName as character)
* logical RemoveTableFilters (tableName as character)
* logical RemoveTableSort (tableName as character)
* logical setBatchInfo (tableName as character, firstBatch as logical, lastBatch as logical)
* void setDbRowid (hBuf as handle, dbRowid as character)
* logical SetFilter (fieldName as character, operName as character, fieldValue as character)
* void SetIdListTmpHandle(input PropertyValue as handle)
* logical setNewPosition (tableName as character, newRowid as character)
* void SetProperty(input PropertyName as character, input PropertyValue as character)
* void SetQueryName(input PropertyValue as character)
* logical SetSort (tableName as character, fieldName as character, descendingSort as logical)
* logical setUseIndex (tableName as character, useIndex as character)
* void UpdateTtRowId(input tableName as character, input ttRowId as character, input newttRowId as character)

###### AddWhereClause

Public method that adds a where clause to data source query.

Parameters:

* tableName as character – temp-table name
* bufferName as character – database buffer name
* whereClause as character – where clause
* isJoin as logical - specifies whether it’s a join or not

###### ClearFilters

Remove all filters that were set before.

###### ClearSort

Remove all sort conditions for all the tables or only for a certain one.

Parameters:

* tableName as character – table name for which all the sort conditions are removed

###### getDbRowid

Returns full rowid of the data source.

Parameters:

* bufName as character – buffer name
* bufRowId as character – buffer rowid

###### getFieldFilterCondition

Returns the filter condition for a certain field .

Parameters:

* fieldname as character – field name
* fieldOper as character – filter operator
* fieldVal as character – filter value

###### getFilterArray

Returns a string with all the filters for a given table.

Parameters:

* tableName as character – table name

###### GetFilterOperator

Returns the filter operator .

Parameters:

* tableName as character – table name
* fieldname as character – field name
* fieldVal as character – filter value

###### getFirstBatch

Specifies if it is the first batch.

Parameters:

* tableName as character – table name

###### getLastBatch

Specifies if it is the last batch.

Parameters:

* tableName as character – table name

###### getLookupFilter

Returns the query string for a lookup table .

Parameters:

* lookupTable as character – lookup table
* bufferName as character – buffer name
* xmlFilter as memptr – filter request sent from client

###### getFilterArray

Returns a string with all the sort conditions for a given table.

Parameters:

* tableName as character – table name

###### getTableSort

Returns all sort conditions for a certain table.

Parameters:

* tableName as character – buffer name

###### RemoveFilter

Remove the filter set on given buffer field.

Parameters:

* tableName as character – table name
* fieldName as character – filter field name

###### RemoveSort

Remove the sort condition set on given buffer field.

Parameters:

* tableName as character – table name
* fieldName as character – sort condition field name

###### RemoveTableFilters

Removes the filters set on given buffer.

Parameters:

* tableName as character – table name

###### RemoveTableSort

Removes the sort conditions set on given buffer.

Parameters:

* tableName as character – table name

###### setBatchInfo

Sets the info for the current batch.

Parameters:

* tableName as character –requested table name
* firstBatch as logical – contains first record
* lastBatch as logical – contains last record

###### setDbRowid

Sets the full rowid of the data source.

Parameters:

* hBuf as handle – requested buffer handle
* dbRowid as character – full rowid

###### SetFilter

Set a filter on given buffer field.

Parameters:

* tableName as character – table name
* fieldName as character – filter field name
* operName as character – filter operator name
* fieldValue as character – filter value

###### setNewPosition

Sets the current position in data source.

Parameters:

* tableName as character – requested table
* newRowid as character – current position

###### SetSort

Set a sort condition on given buffer field.

Parameters:

* tableName as character – table name
* fieldName as character – sort condition field name
* descendingSort as logical – descending option for sort condition

## Development

### Quarix by example

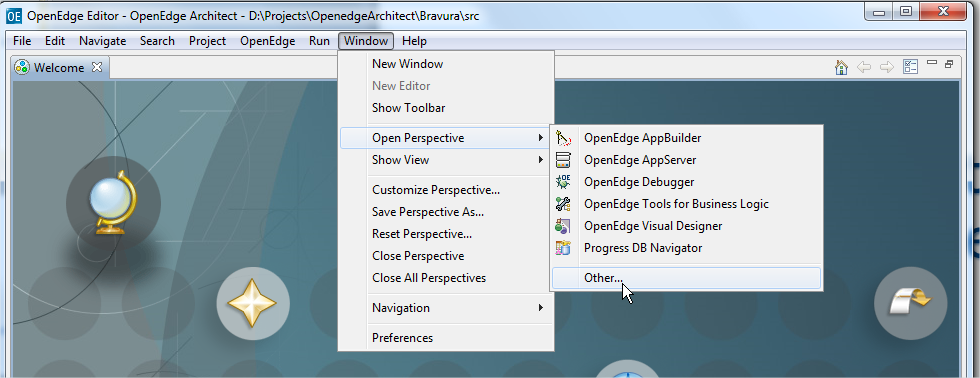
In the next section we will present to you a step by step guide about how the framework can be used with simple code samples. For the examples we will use the sports2000 database provided by Progress OpenEdge. For progress please use at least version 10.2B or above.

#### Getting Quarix

For getting the latest version of the backend framework and the sample application please use the following URL: <https://code.google.com/p/qrx-srv-oe/downloads/list> or you can use your favorite SVN client to get the project from Google repository: <https://qrx-srv-oe.googlecode.com/svn>

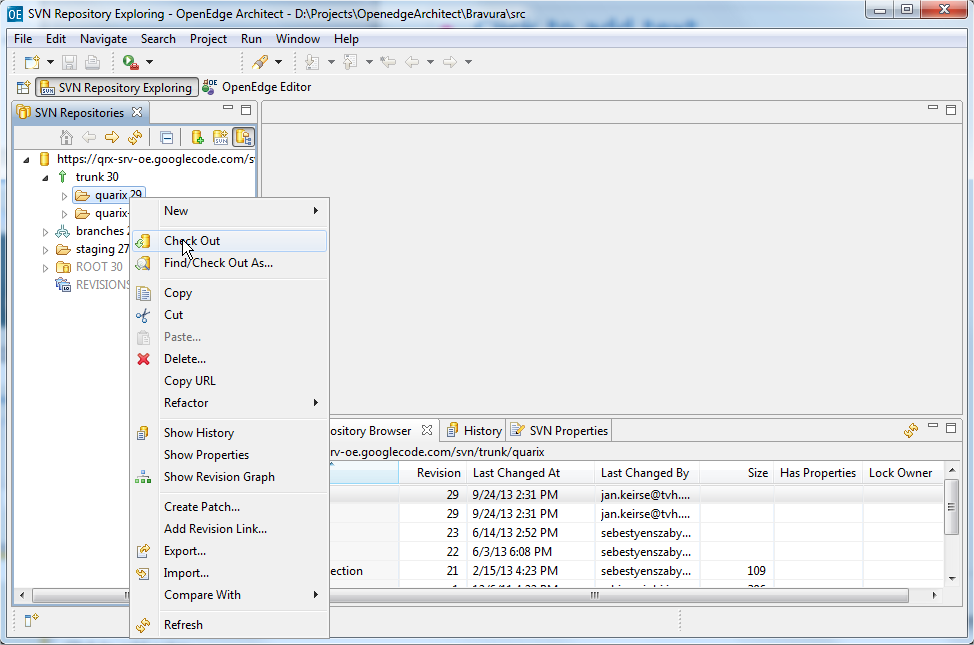
#### Configuring the framework

##### Checkout the project from the SVN repository

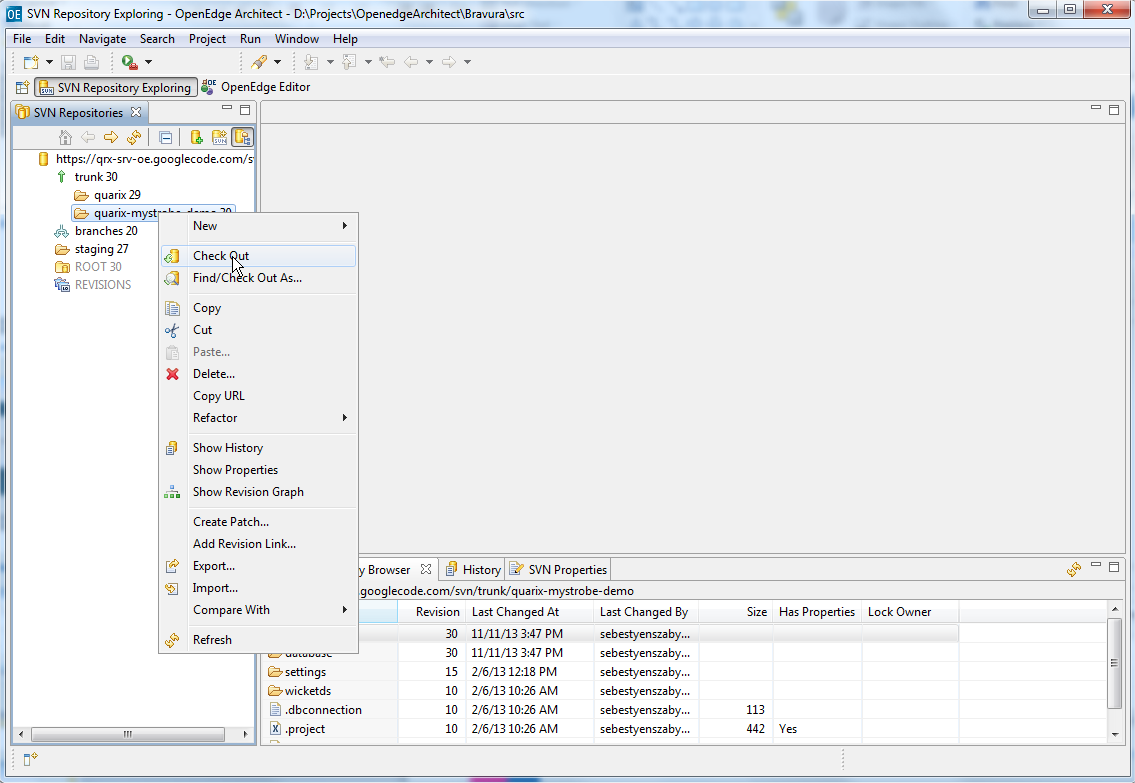


|  |  |
| --- | --- |
|  |  |

Checkout the Quarix framework from repository:

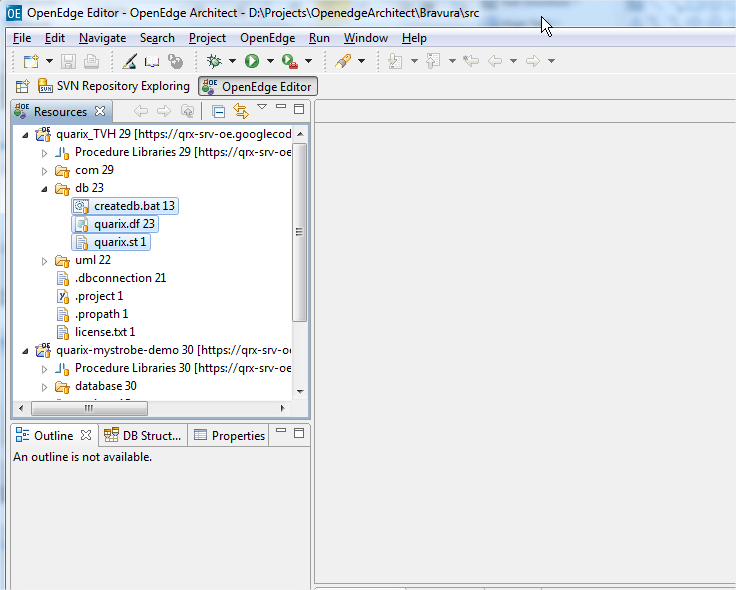


Checkout the demo application from SVN:

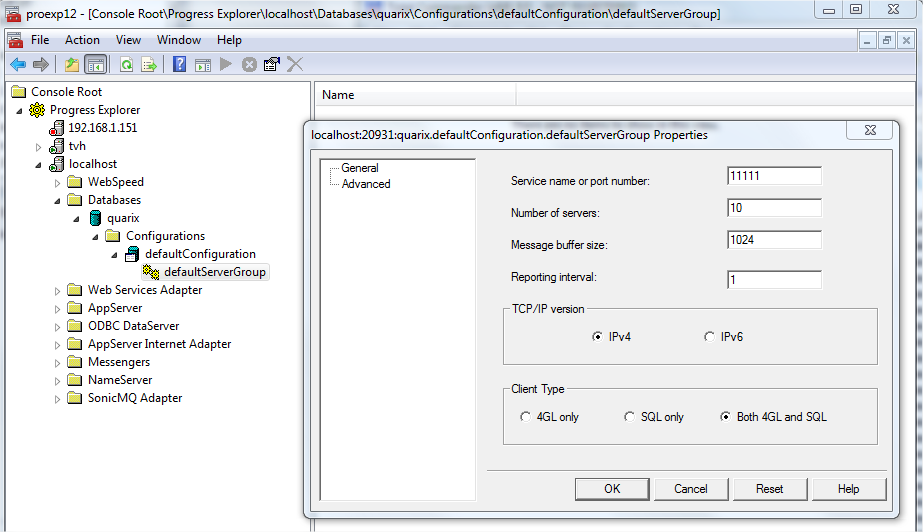


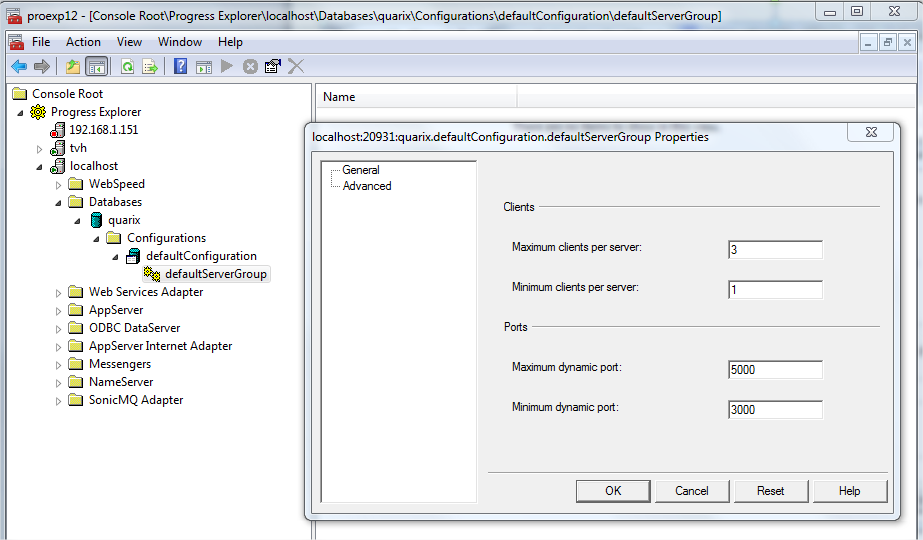
##### Setup database service for the quarix database

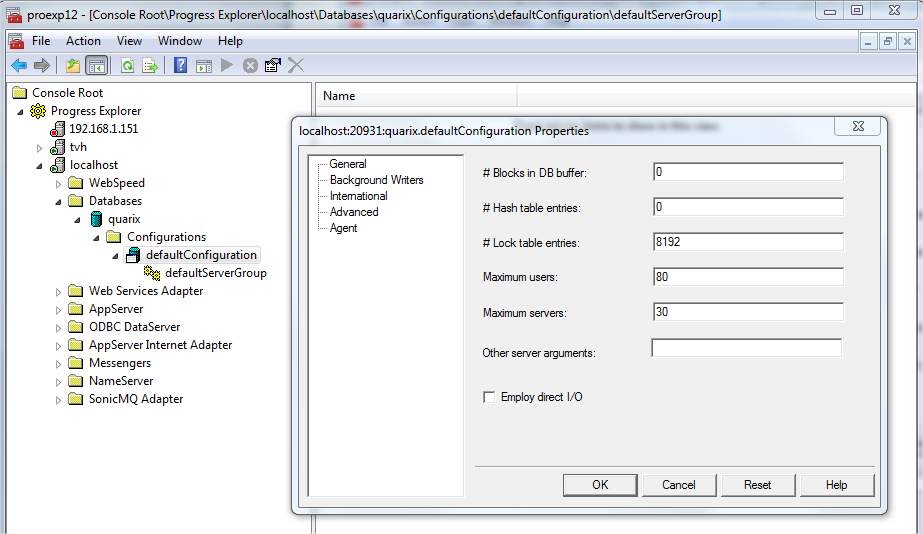
* Locate the quarix.st file inside the project and copy the below selected files into a location where the database will reside.
* Run the script to create the database: createdb.bat.
* Load the database definition file.

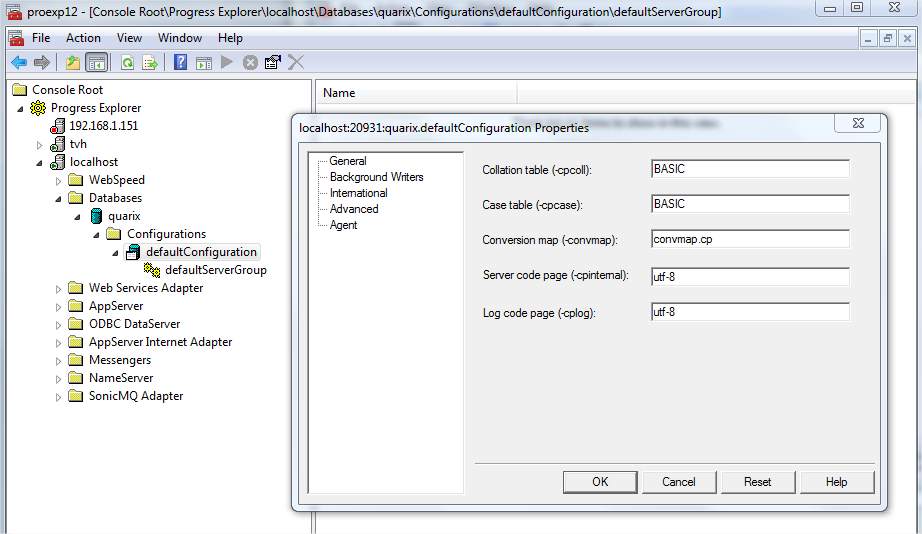


Setup a database server for quarix:



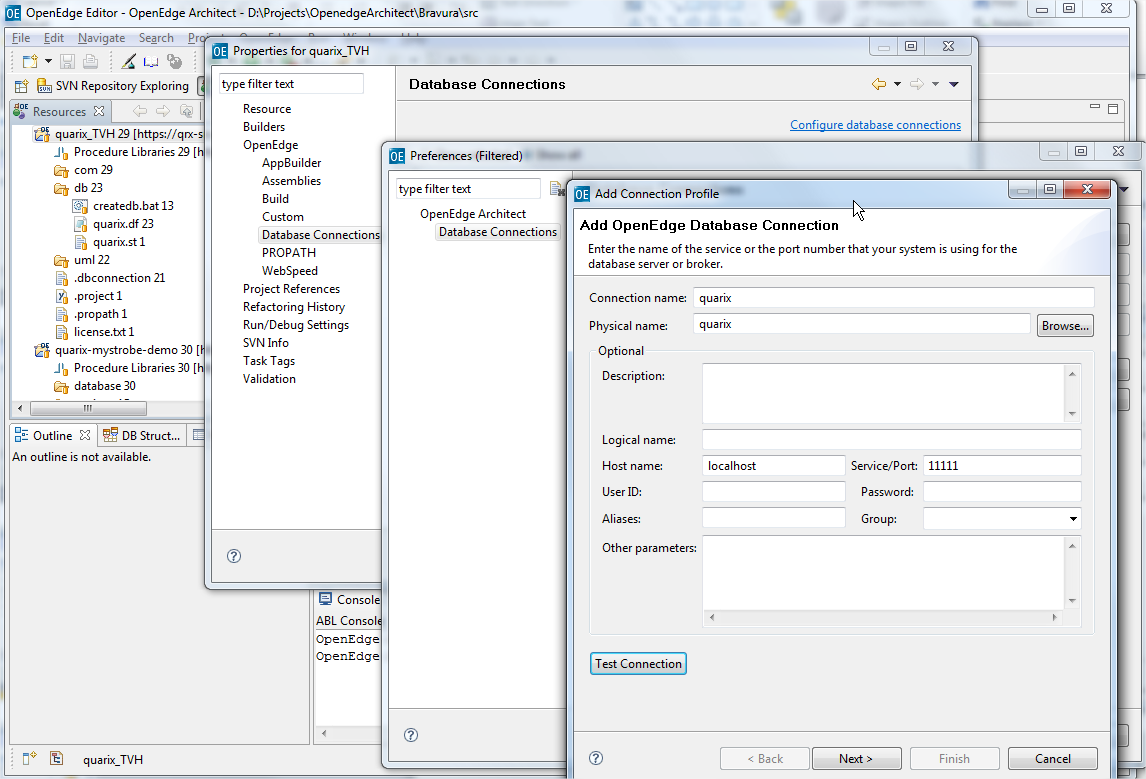


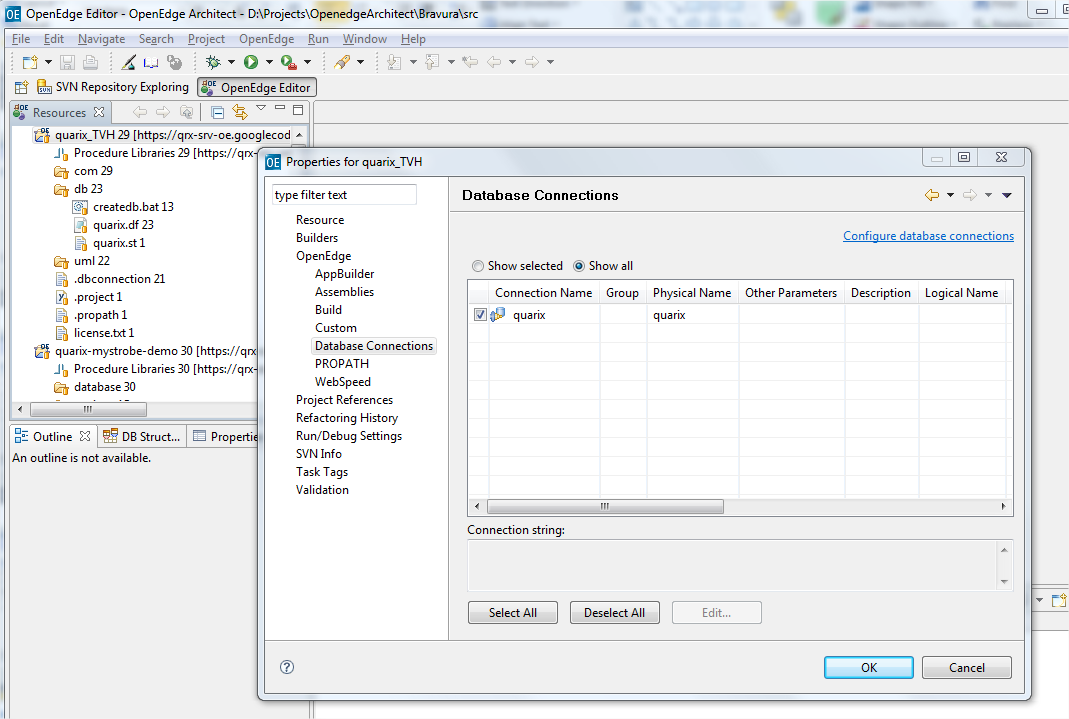




##### Setup the database connection for the project

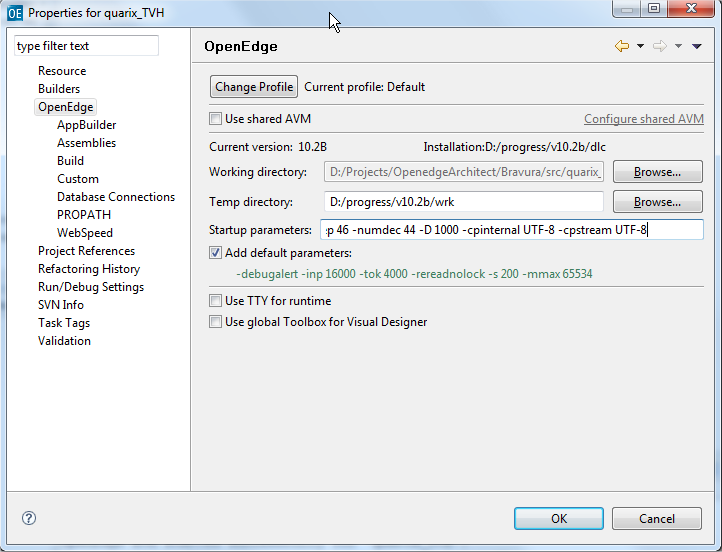
Create a new database connection:



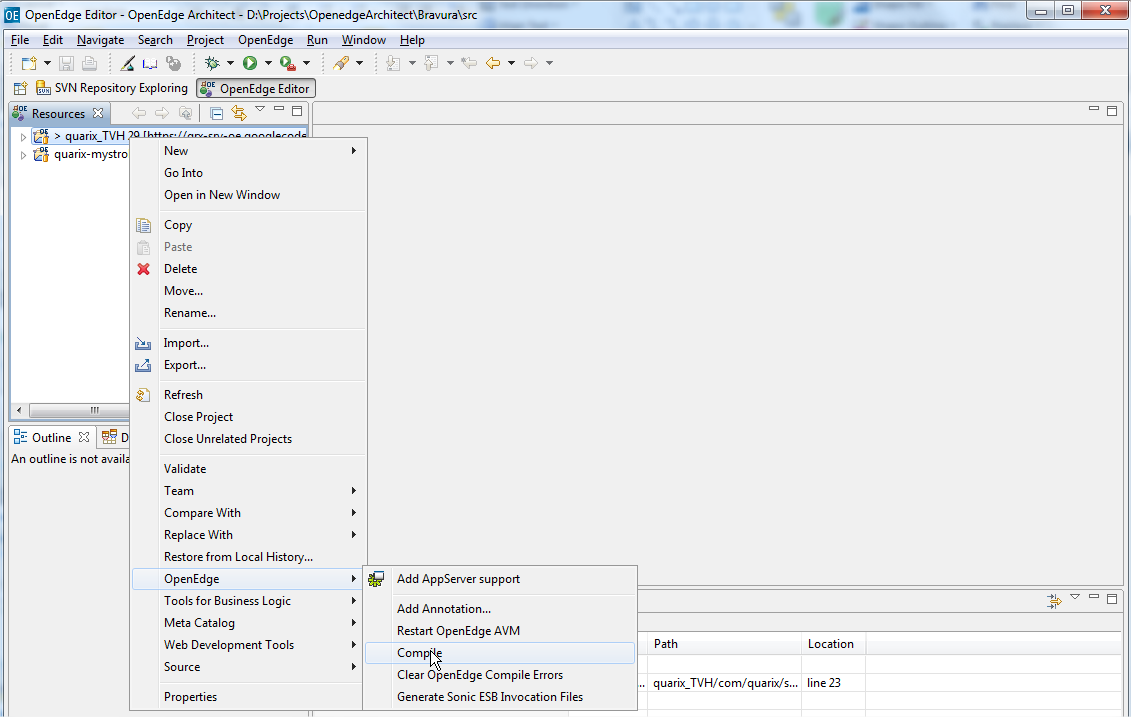


##### Settings for the project

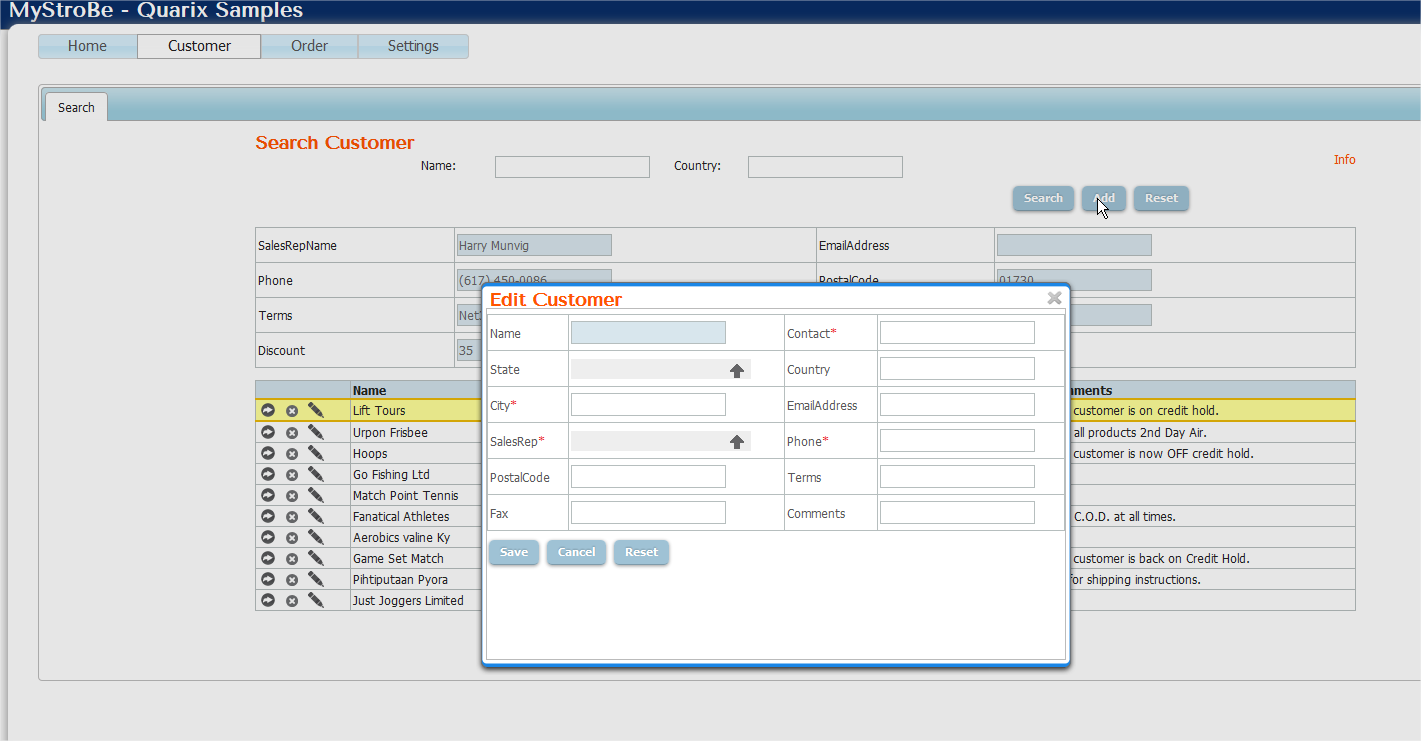
A separate build destination is also recommended to be specified.



##### Build the project



#### Setup the sample application



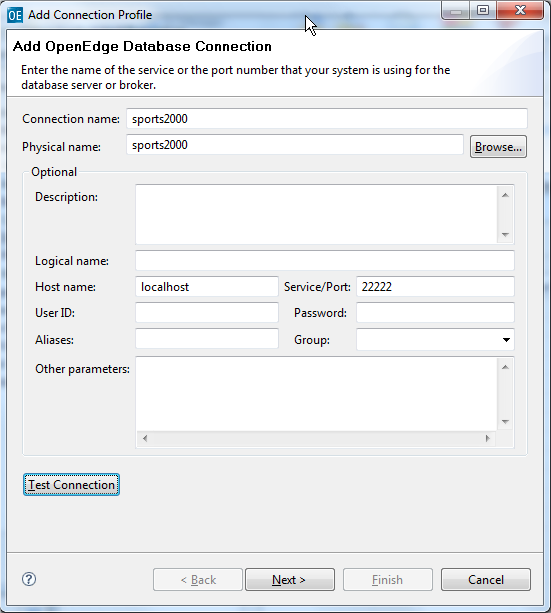
##### Setup database services

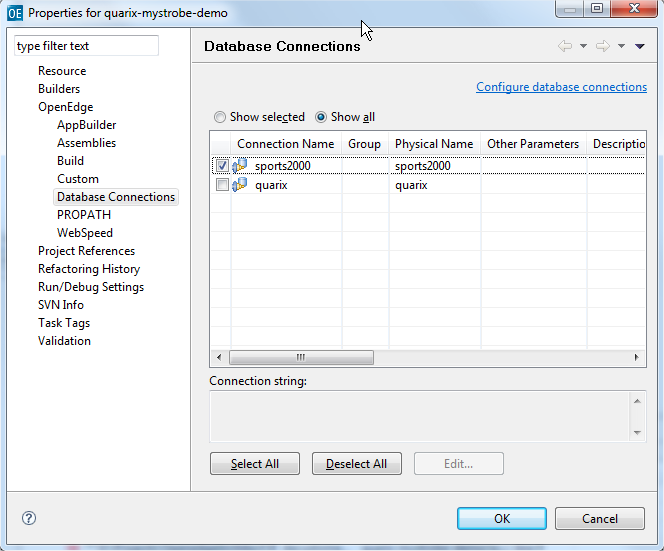
* Copy the sports2000 database provided with Progress into a separate location
* Create a database server for this

|  |  |
| --- | --- |
|  |  |
|  |  |

Please remove the triggers and the validations on the following tables: Customer, Item, Order and OrderLine. Without removing the validations on tables the delete operation will not work, by adding validation on table level the delete operation can be carried out only by the delete statement!

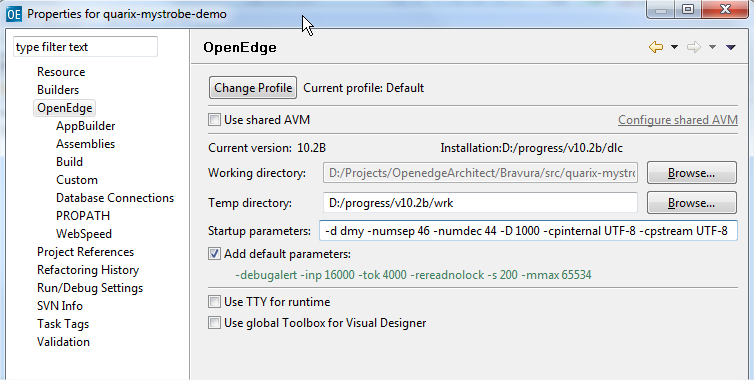
##### Configure database connection for sample



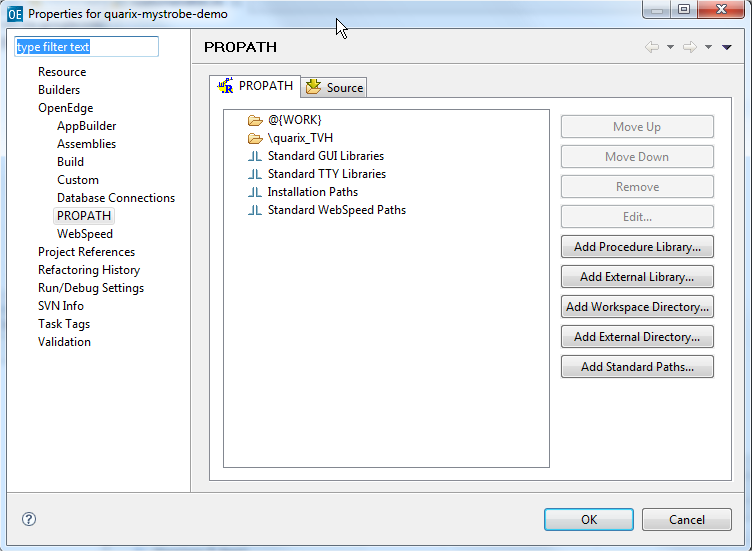


##### Initial configuration of the project

A separate build destination is also recommended to be specified.



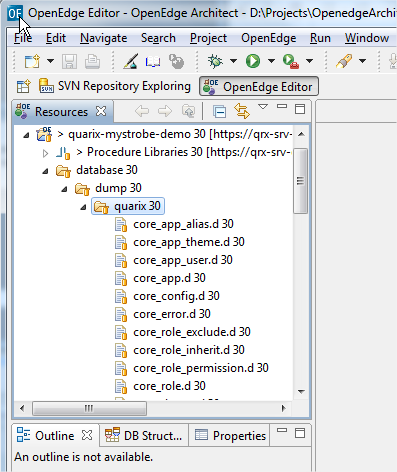
Check the PROPATH settings:



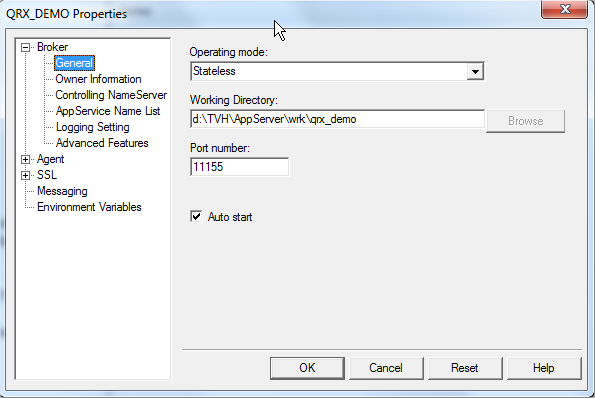
Please make sure that the PROPATH environment variable contains a reference to the Quarix framework.

##### Load the project settings into the quarix database

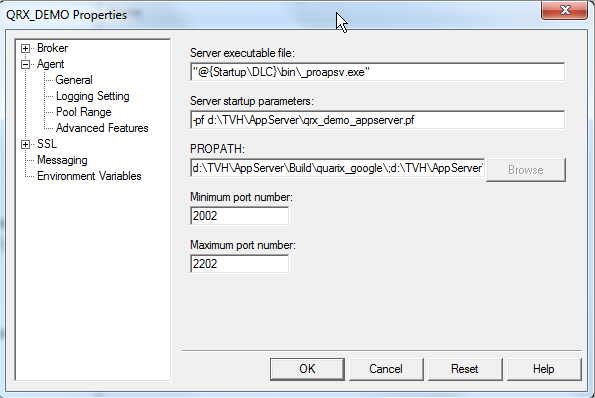
The settings regarding the sample application must be loaded into the quarix database, the dumped data is available in the project folder.



##### Application server broker settings



##### Application server agent settings



* Use the parameter file provided with the sample application and adjust according to your needs.
* The PROPATH must point to the folder set up as build destination containing the r-files.
* Do not forget to copy the xsd folder from the project to the folder containing the r-files.

#### Defining temp-tables

When defining a temp-table that will be involved in a dataset used for crud operations always use the following include with parameters: **com/quarix/data/sortorder.i**

This include contains some already defined fields that are used by the framework when the data is serialized. For example the SortOrder field is used to serialize the records in the temp-table in the same order as they were fetched from the database.

|  |
| --- |
| define temp-table ttCustomer no-undo  before-table btCustomer  {com/quarix/data/sortorder.i &fields="  field Address as character  field Address2 as character  field Balance as decimal  field City as character  field Comments as character  field Contact as character  field Country as character  field CreditLimit as decimal  field CustNum as integer  field Discount as integer  field EmailAddress as character  field Fax as character  field Name as character  field Phone as character  field PostalCode as character  field SalesRep as character  field State as character  field Terms as character  field StateName as character  field SalesRepName as character  "}  index PK\_Customer is unique CustNum. |

#### How to define the dataset

|  |
| --- |
| {wicketds/server/ttcustomer.i {&\*}}  define {&scope} dataset {&prefix}dscustomer {&REFERENCE-ONLY} for {&prefix}ttcustomer. |

The parameters of the include file can be used to define datasets with the same structure but with different name by setting the prefix.

#### Create a simple data model

The simplest data model for our Customer table will look in the following way:

|  |
| --- |
| routine-level on error undo, throw.  using Progress.Lang.\*.  using com.quarix.data.DataModel.  &GLOBAL-DEFINE DefaultDatasetHandle dataset dscustomer:handle  class wicketds.server.dmcustomer inherits DataModel:  {wicketds/server/dscustomer.i}  constructor public dmcustomer ( ):  DataAccess = cast(GetInstance('wicketds.server.dacustomer':u), 'wicketds.server.dacustomer':u).  datasetHandle = {&DefaultDatasetHandle}.  batchSize = 0.  id = 'daocustomer'.  OpenOnInit = false.  SendChangesOnly = false.  if not valid-object(DataAccess) or  not valid-handle(datasetHandle)  then return error.  end constructor.  method public override logical dataFetch ():  DataContext:setAllRecords(true).  return super:dataFetch().  end method.  end class. |

The piece of code from above supports the entire CRUD operations on the Customer table.

Let’s break the code and take a look at the settings:

First of all data models are responsible for holding the business logic, you are not allowed to access any table from the database and also you are not allowed to use other data access objects. Data models were built to hold business logic belonging to a single table, you are allowed to instantiate other data model or data transaction objects to get data for some calculated fields but data models were not meant for complex interactions between data models (business tasks) .

For business tasks data transactions must be defined.

When a data model is defined always must be prefixed with the “dm” for better understanding of the code.

* DefaultDatasetHandle – preprocessor defined for the handle of the dataset, used by the framework
* DataAccess – The instance of the data access object, this object is used to access the data from the database. Always use the GetInstance method to obtain an instance of an object!
* datasetHandle – property holding the handle of the dataset, this is available throught the entire data model.
* batchSize – when a value greater than zero is set than the amount of records fetched from the database will not be greater than this value. This is used for fetching the records using batches/pages. The batch size can also be set in the request sent by the UI, in this case the value set in the data model is not taken into consideration.
* Id – an id of the data object, this id is used on the UI.
* DataRequest – When a request is made from the UI the request is stored into an object called DataRequest, you can dump the content of this object by calling the DumpToXml method. The content of the request will dumped into an xml file located in the temporary folder set in the session or in the folder containing the application server log files.
* DataContext – Stores all the information found in the DataRequest object but you can add extra options/filters at runtime that influences the application logic.
* dataFetch – This method is called all the time when data is fetched. This is the entry point for the fetch operation.
* DataContext:setAllRecords(true) – the framework by default contains some limitations concerning the amount of data retrieved from the database, when no filters are defined the framework does not allow sending data back to the client, this was introduced for security reasons. But we can disable this behavior by setting an attribute in the DataContext object.

#### DataAccess object

As we mentioned in the previous chapter the DataModel object will make use of a DataAccess object to obtain the data from the database table.

The simplest DataAccess object for our Customer table will look like this:

|  |
| --- |
| routine-level on error undo, throw.  using Progress.Lang.\*.  using com.quarix.data.DataAccess.  &global-define tbl\_customer Customer  class wicketds.server.dacustomer inherits DataAccess:  {wicketds/server/dscustomer.i}  define buffer {&tbl\_customer} for {&tbl\_customer}.  constructor public dacustomer ( ):  ID = 'dacustomer':u.  end constructor.  method private character SetFieldMappings():  define variable cFieldMapping as character no-undo.  assign  cFieldMapping =  '{&tbl\_customer}.Address' + ',ttcustomer.Address' + ',' +  '{&tbl\_customer}.Address2' + ',ttcustomer.Address2' + ',' +  '{&tbl\_customer}.Balance' + ',ttcustomer.Balance' + ',' +  '{&tbl\_customer}.City' + ',ttcustomer.City' + ',' +  '{&tbl\_customer}.Comments' + ',ttcustomer.Comments' + ',' +  '{&tbl\_customer}.Contact' + ',ttcustomer.Contact' + ',' +  '{&tbl\_customer}.Country' + ',ttcustomer.Country' + ',' +  '{&tbl\_customer}.CreditLimit' + ',ttcustomer.CreditLimit' + ',' +  '{&tbl\_customer}.CustNum' + ',ttcustomer.CustNum' + ',' +  '{&tbl\_customer}.Discount' + ',ttcustomer.Discount' + ',' +  '{&tbl\_customer}.EmailAddress' + ',ttcustomer.EmailAddress' + ',' +  '{&tbl\_customer}.Fax' + ',ttcustomer.Fax' + ',' +  '{&tbl\_customer}.Name' + ',ttcustomer.Name' + ',' +  '{&tbl\_customer}.Phone' + ',ttcustomer.Phone' + ',' +  '{&tbl\_customer}.PostalCode' + ',ttcustomer.PostalCode' + ',' +  '{&tbl\_customer}.SalesRep' + ',ttcustomer.SalesRep' + ',' +  '{&tbl\_customer}.State' + ',ttcustomer.State' + ',' +  '{&tbl\_customer}.Terms' + ',ttcustomer.Terms'.  return cFieldMapping.  end method.  method public override logical AttachDataSource():  define variable cFieldMapping as character no-undo.  define variable cDbTables as character no-undo.  define variable cKeys as character no-undo.  cFieldMapping = SetFieldMappings().  assign  cDbTables = "{&tbl\_customer}"  cKeys = 'CustNum':U.  return AddDataSource ('ttcustomer':u, cDbTables, cKeys, ?, cFieldMapping).  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method.  end class. |

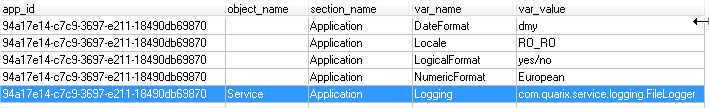
What needs to be set in order to be able to get the data:

* SetFieldMappings – this method maps the temp table fields with the database fields, in some cases the temp table is defined differently as the table from the database.
* AttachDataSource – method called to set the source of the data from where the temp-table is going to be filled, you have to specify at least the database table, primary keys and the field mappings.

##### How the generated query will look like

In case during the development the generated query must be visualized or index usage information is required this can be accomplished by setting the application log level in the core\_app table to 4 and also the logging service must be configured in the core\_config table.

For our sample application the core\_config table contains the following settings:



By setting this a file named quarix.log will be created in the session work folder or in the folder where the application server log files are located. This files contains information about the query that is executed in the DataAccess layer to fetch the date from the database.

Index information and the execution time of the query

|  |
| --- |
| Object: wicketds.server.dacustomer\_1051, Query executed: for each Customer no-lock Buffer: Customer, Used index: WHOLE-INDEX,CustNum  Query for buffer ttCustomer executed in 13 ms returned 11 records. |

Index information and the execution time of the query in case of id lists

|  |
| --- |
| Object: wicketds.server.dasalesrep\_1155, Query executed: for each ttsalesrepidlist no-lock , each Salesrep no-lock where Salesrep.SalesRep = ttsalesrepidlist.SalesRep  Buffer: ttsalesrepidlist, Used index: WHOLE-INDEX,PK\_IdListOrder  Buffer: Salesrep, Used index: SalesRep Query for buffer ttSalesrep executed in 26 ms returned 6 records. |

#### Getting value for calculated fields using lists of id’s

Suppose we would like to fill also some descriptions related with foreign keys in the already filled customer temp-table. As example, we would like to get the SalesRepName from SalesRep, for this we are going to gather all the SalesRep id’s into a list and based on that list get the records from the database in a single fetch.

##### Defining a temp table for the id’s

|  |
| --- |
| define {&scope} temp-table {&prefix}ttsalesrepidlist no-undo {&REFERENCE-ONLY}  {com/quarix/data/idlistorder.i &fields="  field SalesRep as character ~  "}  index idxSalesRep is unique SalesRep. |

This temp table will be loaded with the id’s, please notice that also in this case an include file is used for definition.

##### Define an object for the id’s

A new object salesrepidlist must be defined, this object is inherited from the framework provided class com.quarix.data.IdList developed for this purpose.

The code for this class:

|  |
| --- |
| routine-level on error undo, throw.  using Progress.Lang.\*.  class wicketds.server.salesrepidlist inherits com.quarix.data.IdList:  {wicketds/server/ttsalesrepidlist.i}  constructor salesrepidlist():  hbttHandle = buffer ttsalesrepidlist:handle.  end constructor.  method public void addSalesRep(input cSalesRep as character):  AddRecord('SalesRep':U, string(cSalesRep)).  end method.  end class. |

A new method is defined addSalesRep for adding values into the temporary table.

##### Create a new data model for salesrep

To fetch the data from SalesRep table using a list of id’s a special method created for this purpose from the data model is used.

|  |
| --- |
| routine-level on error undo, throw.  using Progress.Lang.\*.  using com.quarix.data.DataModel.  &GLOBAL-DEFINE DefaultDatasetHandle dataset dssalesrep:handle  class wicketds.server.dmsalesrep inherits DataModel:  {wicketds/server/dssalesrep.i}  constructor public dmsalesrep ( ):  DataAccess = cast(GetInstance('wicketds.server.dasalesrep':u), 'wicketds.server.dasalesrep':u).  datasetHandle = {&DefaultDatasetHandle}.  batchSize = 0.  id = 'daosalesrep'.  OpenOnInit = false.  SendChangesOnly = false.  if not valid-object(DataAccess) or  not valid-handle(datasetHandle)  then return error.  end constructor.  method public override logical dataFetch ():  DataContext:setAllRecords(true).  return super:dataFetch().  end method.  method public logical getbysalesrepidlist(input salesrepidlist as wicketds.server.salesrepidlist, output dataset dssalesrep):  return dataFetchIdList(input salesrepidlist, output dataset dssalesrep by-reference).  end method.  end class. |

As we can observe the method getbysalesrepidlist is used to fetch the data using id lists. This method accepts as parameter an object of type salesrepidlist and passed further to the DataAccess.

##### Implement the fetch by id lists in the data access layer

To be able to fetch the data using the passed id list object, in the data access layer we should create a hook and treat the case when we are dealing with list of id’s, this hook is handled when the data source is attached.

|  |
| --- |
| routine-level on error undo, throw.  using Progress.Lang.\*.  using com.quarix.data.DataAccess.  &global-define tbl\_salesrep Salesrep  class wicketds.server.dasalesrep inherits DataAccess:  {wicketds/server/dssalesrep.i}  define buffer {&tbl\_salesrep} for {&tbl\_salesrep}.  constructor public dasalesrep ( ):  ID = 'dasalesrep':u.  end constructor.  method private character SetFieldMappings():  define variable cFieldMapping as character no-undo.  assign  cFieldMapping =  '{&tbl\_salesrep}.Region' + ',ttsalesrep.Region' + ',' +  '{&tbl\_salesrep}.SalesRep' + ',ttsalesrep.SalesRep' + ',' +  '{&tbl\_salesrep}.RepName' + ',ttsalesrep.RepName'  .  return cFieldMapping.  end method.  method private logical AttachDataSourceDefault():  define variable cFieldMapping as character no-undo.  define variable cDbTables as character no-undo.  define variable cKeys as character no-undo.  cFieldMapping = SetFieldMappings().  assign  cDbTables = "{&tbl\_salesrep}"  cKeys = 'SalesRep':U.  return AddDataSource ('ttsalesrep':u, cDbTables, cKeys, ?, cFieldMapping).  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method.  method private logical AttachDataSourceIdList():  define variable hTmpList as handle no-undo.  define variable cJoinClause as character no-undo.  define variable cKeys as character no-undo.  define variable cFieldMapping as character no-undo.  define variable cDbTables as character no-undo.  define variable cExcludeFields as character no-undo.  hTmpList = GetIdListTmpHandle().  if not valid-handle(hTmpList)  then return false.  assign  cKeys = 'SalesRep':U  cExcludeFields = 'ttsalesrep.SalesRep':U.  if not AddDataSource ('ttsalesrep':u, hTmpList:default-buffer-handle, cKeys, ?, ?, cExcludeFields, ?, ?)  then return false.  cFieldMapping = SetFieldMappings().  assign  cDbTables = "{&tbl\_salesrep}"  cKeys = 'SalesRep':U.  cJoinClause = 'ttsalesrep.SalesRep = ttsalesrepidlist.SalesRep':U.  if not AddDataSource ('ttsalesrep':u, cDbTables, cKeys, ?, cFieldMapping, cExcludeFields, ?, cJoinClause)  then return false.  return true.  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method.  method public override logical AttachDataSource():  define variable cQueryName as character no-undo.  cQueryName = GetQueryName().  case cQueryName:  when 'FetchDataByIdList':U  then return AttachDataSourceIdList().  otherwise  return AttachDataSourceDefault().  end case. /\* case cQueryName \*/  return false.  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method.  end class. |

Handling of the data source is threated in the AttachDataSource method. The framework automatically sets a query name on the DataContext object, using this query name we can decide how to attach the data source. When a list of id’s are present the method AttachDataSourceIdList is called, actually in this method AddDataSource is called twice, once the temp table ttsalesrepidlist with id’s is added to the query and the second time the SalesRep table. The resulted query will be a join between the tables ttsalesrepidlist and SalesRep, in this way only the records mentioned in the id list are going to be fetched based on the id’s.

##### Fill the calculated field in the customer data model

Once we have in place the functionality for gettting the salesrep records using list of id’s the only thing we need to do is to populate such a list with values and get the data, this is realized in the customer data model.

First a property is needed as dmsalesrep:

|  |
| --- |
| def private property dmsalesrep as wicketds.server.dmsalesrep no-undo  get:  if not valid-object(dmsalesrep) then dmsalesrep = cast (GetInstance ('wicketds.server.dmsalesrep':u), wicketds.server.dmsalesrep).  return dmsalesrep.  end.  private set. |

#### Data transactions

Data transactions were created to be able to build complex business logic inside them, they contain interactions between data models or/and other data transactions. The dataset of the data transaction can contain multiple temporary tables. The dataset of the data transactions is always filled by using another data model or data transaction which is marked as main, this main data object is used to call the fetch mechanism and page the data. The request (filters, sorting options, ..) is passed to the main data object.

In the example below we will explain step by step how to build a data object.

##### Define the dataset dsorderinfo

The definition of the dataset/temporray tables is the same as in the case of data models, but in this case the dataset can contain more than one temporray table definition.

|  |
| --- |
| {sales/order/ttordlineinfo.i {&\*}}  {sales/salestext/ttsalestext.i {&\*}}  {sales/leadtime/ttleadtime.i {&\*}}  define {&scope} dataset {&prefix}dsorderinfo {&REFERENCE-ONLY} for {&prefix}ttordlineinfo, {&prefix}ttsalestext, {&prefix}ttleadtime. |

##### Creating dataset wrappers

By creating a dataset wrapper we reffer to the possibility to create a dynamic dataset definition that shares a temporray table used by another dataset, in this manner by filling the dynamic dataset using a data model actually the temporary table used by the original dataset is going to be filled, we gain performance by not having to copy the content of the temporary table.

We are going to use the dataset dsorderinfo defined above to create a wrapper for each temporary table from the dataset.

|  |
| --- |
| class sales.order.dsorderinfowrapper inherits com.quarix.data.dswrapper:  {sales/order/dsorderinfo.i}  constructor dsorderinfowrapper():  dsHandle = dataset dsorderinfo:handle.  end constructor.  method public static logical createOrdLineInfo(input dataset-handle hdsOrderInfo, input cSourceObject as character, output dataset-handle hdsOrdLineInfo):  return createDataset(input dataset-handle hdsOrderInfo by-reference, input 'ttordlineinfo|ttsalestext':U, input cSourceObject, output dataset-handle hdsOrdLineInfo by-reference).  finally:  delete object hdsOrderInfo no-error.  delete object hdsOrdLineInfo no-error.  end finally.  end method.  method public override handle getDsHandle():  return dataset dsorderinfo:handle.  end method.  end class. |

The method createOrdLineInfo will return dynamically created dataset containing the temporary tables ttordlineinfo and ttsalestext, these are the same temp tables that are defined in the dataset dsorderinfo. In the next steps we are going to make use of this wrapper to fill the dsorderinfo dataset by calling the fetch method of some data models.

##### Define a data transaction

|  |
| --- |
| class sales.order.dtorderinfo inherits com.quarix.data.DataTransaction:  {sales/order/dsorderinfo.i}  define private property dtordlineinfo as sales.order.dtordlineinfo no-undo  get:  if not valid-object(dtordlineinfo) then dtordlineinfo = cast (GetInstance('sales.order.dtordlineinfo':u), sales.order.dtordlineinfo).  return dtordlineinfo.  end.  private set.  constructor public dtorderinfo ( ):  datasetHandle = {&DefaultDatasetHandle}.  id = 'daoorderinfo'.  batchSize = 0.  OpenOnInit = false.  if not valid-handle(datasetHandle)  then return error.  end constructor.  method private logical BeforeDataFetch(input-output dataset dsorderinfo):  define variable hdsOrdLineInfo as handle no-undo.  dataset dsorderinfo:empty-dataset ().  sales.order.dsorderinfowrapper:createOrdLineInfo(  input dataset dsorderinfo by-reference,  this-object:ToString(),  output dataset-handle hdsOrdLineInfo by-reference  ).  if not valid-handle(hdsOrdLineInfo)  then return false.  hdsOrdLineInfo:empty-dataset ().  if not SetMainDataModel(dtordlineinfo, hdsOrdLineInfo, 'ttordlineinfo,ttordlineinfo':U, ?)  then do:  Util:DisposeDataset(hdsOrdLineInfo).  return false.  end.  dtordlineinfo:ClearQueryData().  dataContext:copyFilters(dtordlineinfo, "ttordlineinfo", "ttordlineinfo").  return true.  catch appError as Progress.Lang.Error :  Util:DisposeDataset(hdsOrdLineInfo).  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method.  method public override logical BeforeDataFetch ():  return BeforeDataFetch(input-output dataset-handle datasetHandle by-reference).  end method.  end class. |

At the first look a data transaction is not very different from a data model, almost the same methods are available as in the data model, but in this case we do not have a data access object available that provides the data. The data access object is replaced by a data model or a data transaction that is set as main, this will provide the data.

By taking a closer look at the BeforeDataFetch method we can observer that a dynamic dataset is created using the dataset wrapper, the handle of the dataset is returned in the variable hdsOrdLineInfo. The main data model (dtordlineinfo) is set and the request from temporary table ttordlineinfo (from dsorderinfo) is passed to temporary table ttordlineinfo (dynamic dataset).

Please note that the fetch method from dtordlineinfo is not called by programmer, this is handled by the framework, at runtime the framework will identify the main data model and call the appropiate fetch method with the dynamic dataset hdsOrdLineInfo as parameter.

The dynamically created dataset by the wrapper object must be deleted, in case of errors, this is handled by the catch block in the end of the method, if no errors occured than the dataset is deleted automaically by the framework.

#### Setup parameters in the quarix database

There are two tables available in the quarix database for storing the different application settings, these are: core\_app and core\_config. For the sample application the settings are dumped and the files with the settings are available in the application, so before using it please load the settings into the quarix database.

|  |  |  |
| --- | --- | --- |
| **Field name** | **Value** | **Description** |
| app\_id | 94a17e14-c7c9-3697-e211-18490db69870 | Unique generated value, guid can be used. |
| app\_name | qrxexmpl | Name of the application |
| app\_version | 1.0 | Version number |
| app\_desc | qrxexmpl | Long description |
| default\_lang\_code | RO | Language code |
| req\_authentication | no | Not used since authentication is not yet ready |
| req\_authorization | no | Not used since authorization is not yet ready |
| page\_default |  | Not used |
| page\_login |  | Not used |
| log\_level | 4 | Logging level, possible values: 1, 2, 3 or 4. Use level 4 only for debugging purpose during development. |
| theme\_code |  | Not used. |

|  |  |  |
| --- | --- | --- |
| **Field** | **Value** | **Description** |
| app\_id | 94a17e14-c7c9-3697-e211-18490db69870 | Unique generated value comming from core\_app. |
| section\_name | Application | Name of the section for which the setting is applied. |
| object\_name |  |  |
| var\_name | DateFormat |  |
| var\_value | dmy |  |

|  |  |  |
| --- | --- | --- |
| **Field** | **Value** | **Description** |
| app\_id | 94a17e14-c7c9-3697-e211-18490db69870 | Unique generated value comming from core\_app. |
| section\_name | Application | Name of the section for which the setting is applied. |
| object\_name |  |  |
| var\_name | Locale | Locale setting. |
| var\_value | RO\_RO |  |

|  |  |  |
| --- | --- | --- |
| **Field** | **Value** | **Description** |
| app\_id | 94a17e14-c7c9-3697-e211-18490db69870 | Unique generated value comming from core\_app. |
| section\_name | Application | Name of the section for which the setting is applied. |
| object\_name |  |  |
| var\_name | LogicalFormat | Logical values format. |
| var\_value | yes/no |  |

|  |  |  |
| --- | --- | --- |
| **Field** | **Value** | **Description** |
| app\_id | 94a17e14-c7c9-3697-e211-18490db69870 | Unique generated value comming from core\_app. |
| section\_name | Application | Name of the section for which the setting is applied. |
| object\_name |  |  |
| var\_name | NumericFormat | Numeric values format. |
| var\_value | European |  |

|  |  |  |
| --- | --- | --- |
| **Field** | **Value** | **Description** |
| app\_id | 94a17e14-c7c9-3697-e211-18490db69870 | Unique generated value comming from core\_app. |
| section\_name | Application | Name of the section for which the setting is applied. |
| object\_name | Service |  |
| var\_name | Logging | Logging service. |
| var\_value | com.quarix.service.logging.FileLogger | Different logging services can be used like: file, database, ...  In this case a log file will be created and if the log level is set to 4 than the debug messages and information about the query is logged into this file. |

#### Error handling using the try catch mechanism

As a best practice it is recommended to catch the errors at method level using the catch mechanism provided. Also pay attention to the objects created in the method, do not forget to delete them at the end of the method using the finally block.

For more information about error handling please consult the the next [document](Error_handling.pdf).

|  |
| --- |
| method private logical AfterLoadData (input-output dataset dscustomer):  define variable salesrepidlist as wicketds.server.salesrepidlist no-undo.  define variable stateidlist as wicketds.server.stateidlist no-undo.  Salesrepidlist = cast(GetInstance('wicketds.server.salesrepidlist':u), 'wicketds.server.salesrepidlist':u).  Stateidlist = cast(GetInstance('wicketds.server.stateidlist':u), 'wicketds.server.stateidlist':u).  return true.  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  finally:  UnloadInstance(stateidlist).  UnloadInstance(salesrepidlist).  end finally.  end method. |

### Methods to override in DataModel

#### dataFetch

This method can be overriden if custom fetch mechanism is needed or some extra filters are required in the DataContext. The best approach is to override the dataFetch method with dataset handle as parameter, in this way the dataset that is going to be filled is available in the parameter.

|  |
| --- |
| method public override logical dataFetch ():  DataContext:setAllRecords(true).  return super:dataFetch().  end method. |

#### BeforeLoadData

Extra filters can be inserted into the DataContext object or custom validations can be executed before loading the data, if this method fails the execution of the load mechanism is stopped and no data will be available into the dataset.

|  |
| --- |
| method public override logical BeforeLoadData():  return BeforeLoadData(input-output dataset-handle datasetHandle by-reference).  end method.    method private logical BeforeLoadData (input-output dataset dsorder):  SetFilter('ttcustomer', 'Name', '=', 'aaa').  end method. |

#### AfterLoadData

Once the data is available in the dataset it can be customized, for example calculated fields can be filled. This method is the suitable for filling calculated fields. As a best practice always define a custome AfterLoadData method that is called from the overriden method, in this way the dataset is accessible through his handle.

|  |
| --- |
| method public override logical AfterLoadData():  return AfterLoadData(input-output dataset-handle datasetHandle by-reference).  end method.  method private logical AfterLoadData (input-output dataset dscustomer):  end method. |

#### BeforeSaveData

Developer hook used mostly to validate the data before saving. If this method fails the save operation is cancelled.

|  |
| --- |
| method private logical BeforeSaveData(input-output dataset dsitem):  define variable itemidlist as wicketds.server.itemidlist no-undo.  itemidlist = cast(GetInstance('wicketds.server.itemidlist':u), 'wicketds.server.itemidlist':u).  for each btitem  where row-state(btitem) = row-deleted  no-lock:  itemidlist:addItemNum(btitem.itemnum).  end.  if itemidlist:IdListHasData()  then do:  dmorderline:ClearContext().  if not dmorderline:getbyitemidlist(itemidlist, output dataset dsorderline by-reference)  then return false.  find first ttorderline  no-lock no-error.  if available(ttorderline)  then do:  ThrowClientError(1000, 'Item cannot be deleted, first delete the order lines!').  return false.  end.  end.  return true.  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  finally:  UnloadInstance(itemidlist).  end finally.  end method.  method public override logical BeforeSaveData():  return BeforeSaveData(input-output dataset-handle datasetHandle by-reference).  end method. |

#### AfterSaveData

Developer hook, can be used for custom logic after the data is saved. If this method fails the save operation is cancelled and the transaction is undone.

|  |
| --- |
| method private logical AfterSaveData (input-output dataset dslistinfogroup):  return true.  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method.  method public override logical AfterSaveData ():  return AfterSaveData(input-output dataset-handle datasetHandle by-reference).  end method. |

### Methods to override in DataTransaction

#### dataFetch

This method can be overriden if custom fetch mechanism is needed or some extra filters are required in the DataContext. The best approach is to override the dataFetch method with dataset handle as parameter, in this way the dataset that is going to be filled is available in the parameter.

|  |
| --- |
| method public override logical dataFetch (output dataset-handle dsHandle):  return dataFetch(output DATASET-HANDLE dsHandle by-reference).  finally:  delete object dsHandle no-error.  end finally.  end method.  method private logical dataFetch (output DATASET dssurchargeinfo):  return true.  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method. |

#### BeforeDataFetch

Custom logic for adding filters or loading the data, it is recomended to define a new method BeforeDataFetch with dataset as parameter and call this new method from the overriden method.

|  |
| --- |
| method private logical BeforeDataFetch (input-output dataset dsmylist):  define variable cQueryName as character no-undo.  dataset dsmylist:empty-dataset().  cQueryName = DataContext:GetQueryName().  case cQueryName:  when 'FetchSharedModelHeaders'  then return BeforeFetchSharedModels(input-output dataset dsmylist by-reference).  when 'FetchAllModelList':u  then return BeforeFetchAllModelList(input-output dataset dsmylist by-reference).  end case. /\* case cQueryName \*/  return true.  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method.  method public override logical BeforeDataFetch ():  return BeforeDataFetch(input-output dataset-handle datasetHandle by-reference).  end method. |

#### AfterDataFetch

Custom logic for fetching calculated fields.

|  |
| --- |
| method private logical AfterDataFetch (input-output dataset dsmylistheader):  return true.  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method.  method public override logical AfterDataFetch ():  return AfterDataFetch(input-output dataset-handle datasetHandle by-reference).  end method. |

#### BeforeDataUpdate

Developer hook used mostly to validate the data before saving. If this method fails the save operation is cancelled.

#### AfterDataUpdate

Developer hook, can be used for custom logic after the data is saved. If this method fails the save operation is cancelled and the transaction is undone.

### Methods to override in DataAccess

#### AttachDataSource

This method should always be overriden, it is used to attach the data source of the dataset in the DataAccess object. If the data source depends on some logic, for example IdLists are used to get the data than the method GetQueryName can be used call different AddDataSource method with different parameters.

|  |
| --- |
| method private logical AttachDataSourceDefault():  define variable cFieldMapping as character no-undo.  define variable cDbTables as character no-undo.  define variable cKeys as character no-undo.  cFieldMapping = SetFieldMappings().  assign  cDbTables = "{&tbl\_customer}"  cKeys = 'CustNum':U.  return AddDataSource ('ttcustomer':u, cDbTables, cKeys, ?, cFieldMapping).  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method.  method private logical AttachDataSourceIdList():  define variable hTmpList as handle no-undo.  define variable cJoinClause as character no-undo.  define variable cKeys as character no-undo.  define variable cFieldMapping as character no-undo.  define variable cDbTables as character no-undo.  define variable cExcludeFields as character no-undo.  hTmpList = GetIdListTmpHandle().  if not valid-handle(hTmpList)  then return false.  assign  cKeys = 'CustNum':U  cExcludeFields = 'ttcustomer.CustNum':U.  if not AddDataSource ('ttcustomer':u, hTmpList:default-buffer-handle, cKeys, ?, ?, cExcludeFields, ?, ?)  then return false.  cFieldMapping = SetFieldMappings().  assign  cDbTables = "{&tbl\_customer}"  cKeys = 'CustNum':U.  cJoinClause = 'ttcustomer.CustNum = ttcustomeridlist.CustNum':U.  if not AddDataSource ('ttcustomer':u, cDbTables, cKeys, ?, cFieldMapping, cExcludeFields, ?, cJoinClause)  then return false.  return true.  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method.  method public override logical AttachDataSource():  define variable cQueryName as character no-undo.  cQueryName = GetQueryName().  case cQueryName:  when 'FetchDataByIdList':U  then return AttachDataSourceIdList().  otherwise  return AttachDataSourceDefault().  end case. /\* case cQueryName \*/  return false.  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method. |

#### AfterRowDelete

Developer hook, the before image of the row is available in the parameter.

#### AfterRowFill

Method called after each fetched row in the data access layer.

|  |
| --- |
| method override public logical AfterRowFill( input hBuf as handle ):  /\* Convert TVH country to Iso2 country \*/  assign hBuf::Country = ‘NL’  return super:AfterRowFill(input hBuf).  end method. |

#### AfterRowSave

Method called after each saved row, the temp table buffer and database table buffers are set and available in the parameters.

#### BeforePrepareQuery

This method is called before the query-prepare operation, this hook was created to be possible to adjust the generated query with custom where clause if needed and not possible throught the DataContext object.

|  |
| --- |
| method protected override character BeforePrepareQuery(input qryString as character):  define variable cQuery as character no-undo.  define variable cBy as character no-undo.  define variable cQueryName as character no-undo.  define variable cUseIndex as character no-undo.  cQueryName = GetQueryName().  if cQueryName = 'FetchOrdLineByHead'  then do:  if index(qryString, ' by ') > 0  then do:  assign  cQuery = trim(substring(qryString, 1, index(qryString, ' by ') - 1)).  cBy = trim(substring(qryString, index(qryString, ' by '))).  end.  else cQuery = qryString.  if DataContext:HasSortOnField('ttordhead':u, 'Orderdate':u)  then cQuery = substitute('&1 &2', cQuery, 'by {&tbl\_ordhead}.orddatum descending').  if DataContext:HasSortOnField('ttordhead':u, 'OrderNumber':u)  then cQuery = substitute('&1 &2', cQuery, 'by {&tbl\_ordhead}.ordnr descending').  if not Util:IsEmpty(cBy)  then cQuery = substitute('&1 &2', cQuery, cBy).  end. /\* if cQueryName = 'FetchInqLineByHead' \*/  return cQuery.  end method. |

#### BeforeRowDelete

Method called each time a row is deleted, the before image buffer and the database table buffer are set in the parameters.

|  |
| --- |
| method public override logical BeforeRowDelete (hBuffBi as handle, hDbBuff as handle):  rowtrans:  do transaction  on error undo, throw  on stop undo, leave:  if hBuffBi:row-state = row-deleted  then.  end.  return super:BeforeRowDelete(hBuffBi, hDbBuff).  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method. |

#### BeforeRowFill

This method is usefull if extra validations must be made on the record before adding in the dataset, if this method fails than the record is skipped.

|  |
| --- |
| method public override logical BeforeRowFill(input hBuf as handle):  define variable numBuf as integer no-undo.  define variable hSrcBuf as handle no-undo.  do numBuf = 1 to hBuf:data-source:num-source-buffers:  assign hSrcBuf = hBuf:data-source:get-source-buffer(numBuf).  if hSrcBuf:name = 'Order':u  then leave.  end. /\* do numBuf = 1 to hBuf:data-source:num-source-buffers\*/  return true.  catch appError as Progress.Lang.Error :  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method. |

#### BeforeRowSave

Extra checks on the row before saving the record.

|  |
| --- |
| method public override logical BeforeRowSave (phBuffer as handle, hBuffBi as handle, hDbBuff as handle):  rowtrans:  do transaction  on error undo, throw  on stop undo, leave:  if phBuffer:before-buffer:row-state = row-created  then.  end. /\* do transaction \*/  return super:BeforeRowSave(phBuffer, hBuffBi, hDbBuff).  catch appError as Progress.Lang.Error:  ThrowError(input appError).  delete object appError.  return false.  end catch.  end method. |

### Invoking a DataModel or DataTransaction from a procedure

|  |
| --- |
| define variable dtuser as utils.context.dtuser no-undo.  define variable mainController as com.quarix.system.Controller no-undo.  define variable cGuid as character no-undo.  define variable cAppName as character no-undo.  {tric/server/dsusr.i}  dtuser = new utils.context.dtuser().  dtuser:ClearContext().  dtuser:ErrorManager:Purge().  run com/quarix/bin/getcontroller.p(output mainController).  assign  cGuid = guid  cAppName = 'ecommerce':u.  mainController:Application:Name = cAppName.  mainController:Application:StartSession(cGuid, 1800).  dtuser:SetFilter('ttusr':U, 'UsrID':U, '=':U, '1111111').  if not dtuser:DataFetch(output dataset dsusr by-reference)  then do:  dataset dsusr:empty-dataset ().  run GetQuarixErrors.  return.  end.  PROCEDURE GetQuarixErrors:    define variable hErrBuf as handle no-undo.  define variable hQuery as handle no-undo.  define variable cErrorCode as character no-undo.  define variable cErrorMessage as character no-undo.  define variable hTmpError as handle no-undo.  define variable iErrorLevel as integer no-undo.  hTmpError = dtuser:ErrorManager:GetTmpErrorHandle().  if not valid-handle(hTmpError)  then return.  hErrBuf = hTmpError:default-buffer-handle.  create query hQuery.  hQuery:set-buffers (hErrBuf).  hQuery:query-prepare (substitute('for each &1 no-lock where &1.errorLevel = 0 or &1.errorLevel = 1':U, hErrBuf:name)).  hQuery:query-open ().  hQuery:get-first ().  do while not hQuery:query-off-end:  assign  cErrorCode = string(hErrBuf:buffer-field ('errorCode':U):buffer-value ())  cErrorMessage = hErrBuf:buffer-field ('errorMessage':U):buffer-value ()  iErrorLevel = hErrBuf:buffer-field ('errorLevel':U):buffer-value ().  hQuery:get-next ().  end. /\* do while not hQuery:query-off-end \*/  hQuery:query-close ().  finally:  delete object hQuery no-error.  end finally.  end procedure. |

The example abowe contains a sample code about how we can invoke code built with the frameowork from a stabdard procedure. All we need to do is to set the session by giving the application name and a session id and run the desired method from the data model or data transaction. The sample code contains also an internal procedure that will return the possible error messages.

### DataRequest object

Each time a request is made, the request is sent by an xml format, this is than stored into a DataRequest object to be able to get the properties of the request is an easier way. The content of this object can be visualised anytime by using the DumpToXml method, this will create an xml file located in the work directory of the session or in the folder where the application server stores his log files.

Request in form of xml for the first set of data

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <datasetMsg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <action command="sendRows" responseFormat="json"/>  <daoMsg id="ttcustomer" command="sendRows" responseFormat="json" startRowId="first" startRow="0" batchSize="10" skipRow="true" prefetch="false">  </daoMsg>  </datasetMsg> |

Request in form of xml for the next set of records

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <datasetMsg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <action command="sendRows" responseFormat="json"/>  <daoMsg id="ttcustomer" command="sendRows" responseFormat="json" startRowId="0x000000000000006a" startRow="0" batchSize="10" skipRow="true" prefetch="false">  </daoMsg>  </datasetMsg> |

Filters in the DataRequest

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <datasetMsg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <action command="sendRows" responseFormat="json"/>  <daoMsg id="ttcustomer" command="sendRows" responseFormat="json" startRowId="first" startRow="0" batchSize="10" skipRow="true" prefetch="false">  <filter fld="name" op="matches" val="\*ju\*"/>  </daoMsg>  </datasetMsg> |

### DataContext object

This object was developed to replace and extend the the DataRequest object. When a request is made all data passed to the dispatcher.p in form of an xml is stored in the DataRequest object than the filters, batching information, query names (properties) and sorting options are stored in the DataContext object. This object is available throught the application in all layers, data transaction, data model, data access and can be manipulated during the runtime.

Behind the DataContext object is a dataset called dscontext, in the following section this dataset will be detailed.

* ttFilter – used to store filters applied to the generated query in the DataAccess layer
  + tableName – the name of the temp table
  + fieldName – the name of the field from the temp table for witch the filter is defined
  + dbFieldName – the name of the corresponding field from the database buffer
  + operName – operator (=, <>, >, <, <=, >=, begins, matches)
  + fieldValue – filter value
* ttSort – table used to store sorting options on records
  + tableName – the name of the temp table
  + fieldName – the name of the field from the temporary table on witch the sorting will be applied when the query is generated
  + descOrder – logical field for ascending or descending sorting
  + sortOrder – the order of the sorting options how they will be inserted in the generated query
* ttInfo – information regarding the records and batching
  + ttName - to which temp table from the query the information reffers
  + lastBatch – if batching is used this indicates if there are more records available
  + firstBatch – if batching is used this indicates if we are in the beginning of the set
  + useIndex – force the using of some index when the query is generated
  + batchSize – number of records to be fetched from the database
  + startRowid – starting rowid of the record from witch new set of records must be fetched
  + startRow – starting row from witch the new set of records must be fetched
  + skipRow – when the new set of records is fetched skip the row with the starting rowid
* ttRowId – information about each record, this serialized to json and sent back to the frontend
  + ttName – the name of the temporary table
  + ttRowId - the rowid of the record from the temporary table
  + dbRowId – the rowid of the record in the database
  + NumRec – the order of the record
* ttProperty – properties/query names used through the application
  + propertyName – the name of the property
  + propertyValue – the value

Request for the first set of data

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <dsContext xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <ttInfo>  <ttName>ttCustomer</ttName>  <ttRowPos/>  <lastBatch>true</lastBatch>  <firstBatch>true</firstBatch>  <useIndex/>  <batchSize>10</batchSize>  <startRowid>first</startRowid>  <startRow>0</startRow>  <skipRow>true</skipRow>  <isSearchReq>false</isSearchReq>  </ttInfo>  </dsContext> |

Request for the next set of records

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <dsContext xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <ttInfo>  <ttName>ttCustomer</ttName>  <ttRowPos/>  <lastBatch>true</lastBatch>  <firstBatch>true</firstBatch>  <useIndex/>  <batchSize>10</batchSize>  <startRowid>0x000000000000006a</startRowid>  <startRow>0</startRow>  <skipRow>true</skipRow>  <isSearchReq>false</isSearchReq>  </ttInfo>  </dsContext> |

DataContext containing the response

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <dsContext xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <ttInfo>  <ttName>ttCustomer</ttName>  <ttRowPos/>  <lastBatch>false</lastBatch>  <firstBatch>true</firstBatch>  <useIndex/>  <batchSize>10</batchSize>  <startRowid>first</startRowid>  <startRow>0</startRow>  <skipRow>true</skipRow>  <isSearchReq>false</isSearchReq>  </ttInfo>  <ttRowId>  <ttName>ttCustomer</ttName>  <ttRowId>0x000000000000f900</ttRowId>  <dbRowId>0x0000000000000061</dbRowId>  <NumRec>1</NumRec>  </ttRowId>  ...  <ttProperty>  <propertyName>QueryRepositionFailed</propertyName>  <propertyValue>no</propertyValue>  </ttProperty>  <ttProperty>  <propertyName>RepositionToRowFailed</propertyName>  <propertyValue>no</propertyValue>  </ttProperty>  </dsContext> |

Presence of filters in DataContext

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <dsContext xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <ttFilter>  <tableName>ttCustomer</tableName>  <fieldName>name</fieldName>  <dbFieldName>Customer.Name</dbFieldName>  <operName>matches</operName>  <fieldValue>\*ju\*</fieldValue>  </ttFilter>  <ttInfo>  <ttName>ttCustomer</ttName>  <ttRowPos/>  <lastBatch>true</lastBatch>  <firstBatch>true</firstBatch>  <useIndex/>  <batchSize>10</batchSize>  <startRowid>first</startRowid>  <startRow>0</startRow>  <skipRow>true</skipRow>  <isSearchReq>false</isSearchReq>  </ttInfo>  </dsContext> |

Fetching data using list of id’s

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <dsContext xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <ttProperty>  <propertyName>IdListTmpHandle</propertyName>  <propertyValue>1463</propertyValue>  </ttProperty>  <ttProperty>  <propertyName>QueryName</propertyName>  <propertyValue>FetchDataByIdList</propertyValue>  </ttProperty>  </dsContext> |

A property is set to indicate in the data access layer that the query must be built using a join between the temp table holding the id’s and the buffer on tha database table.

##### Common methods of the DataContext

* AddWhereClause – Insert complex where clause in the query that cannot be generated through filters
* ClearContext – clear the context dataset
* ClearFilters – clear only the filters
* ClearProperties – clear only the properties
* ClearProperty – remove the specified property in the parameter
* ClearQueryData – clear filters and properties
* ClearSort – clear the data regarding the sorting
* CopyFilters – copy the filters from one temp table into another one
* DumpToXml – used for debugging, the dataset is dumped in xml format in the working directory or in the folder holding the application server log files
* getAllRecords – get the value of the property that specifies if all records must be fetched if no filters are defined at all
* getBatchSize – get the batch size of the buffer
* GetIdListTmpHandle – get the tmporary table handle used to fetch the data by a list of id’s
* GetQueryName – get the name of the query from the property table
* GetQueryRepositionFailed – if the reposition in the query fails in case of batching a property is available for this
* getStartRowid – the starting rowid of the record in case of fetching the records in batches
* RemoveFilter – removes a filter
* RemoveSort – removes sort information
* setAllRecords – in case no filters are defined we must use this method to set a property that allows to fetch all the records, if this is not set an error is generated and no records are fetched
* setBatchInfo – set batching information
* SetFilter – add filters
* SetProperty – add properties
* SetQueryName – set a query name
* UpdateTtRowId – used to update the temp-table rowid information when the data is copied from one temporary table to another, in this case we need the rowid in the target tem table

### Logging and debugging!

Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a program, thus making it behave as expected. Debugging tends to be harder when various subsystems are tightly coupled, as changes in one may cause bugs to emerge in another.

In Quarix the try – catch mechanism is used at all levels, this means that if an error occurs the error is catched and sent back to the frontend as an error message, this is the reason why no error messages are present in the application server log files.

The framework supports log levels this means that depending on this setting messages are ignored or logged. The possible log levels are 1, 2, 3 and 4. When developing/debbuging an application it is recomended to set the log level as high as possible to be able to notice all the errors that could occur during the runtime. Also the logging service must be configured in order to inspect the generated query and the performance measurements related.

To set the log level please use the core\_app table from the quarix database, search for your application and set the log\_level field to the desired value.

The following methods are available to send messages to the frontend:

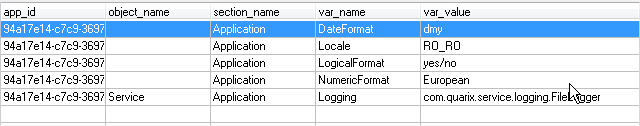
* ThrowError – If messages are logged with this method the response sent back to the frontend will contain the http error code 500 and the execution stops.
* ThrowClientError – Messages logged with this method will not generate the http status code 500, however these messages must be handled by the frontend. This works only when the log level is set at least at 1.
* ThrowWarning – Works only if the log level is set at least at 2.
* ThrowInfo - Works only if the log level is set at least at 3.
* ThrowDebug - Works only if the log level is set at least at 4, these messages are not serialized, they are only debug messages that are logged by the log service into a file. Be carefull with the usage of the ThrowDebug method when called in loops, the performance of the application is affected by excessive method calls. It is highly recomended to remove the calls of the method when they are no longer needed, it’s not enough to set the log level to a lower level.

In the example presented below you can notice a major difference in the execution time when the method is called.

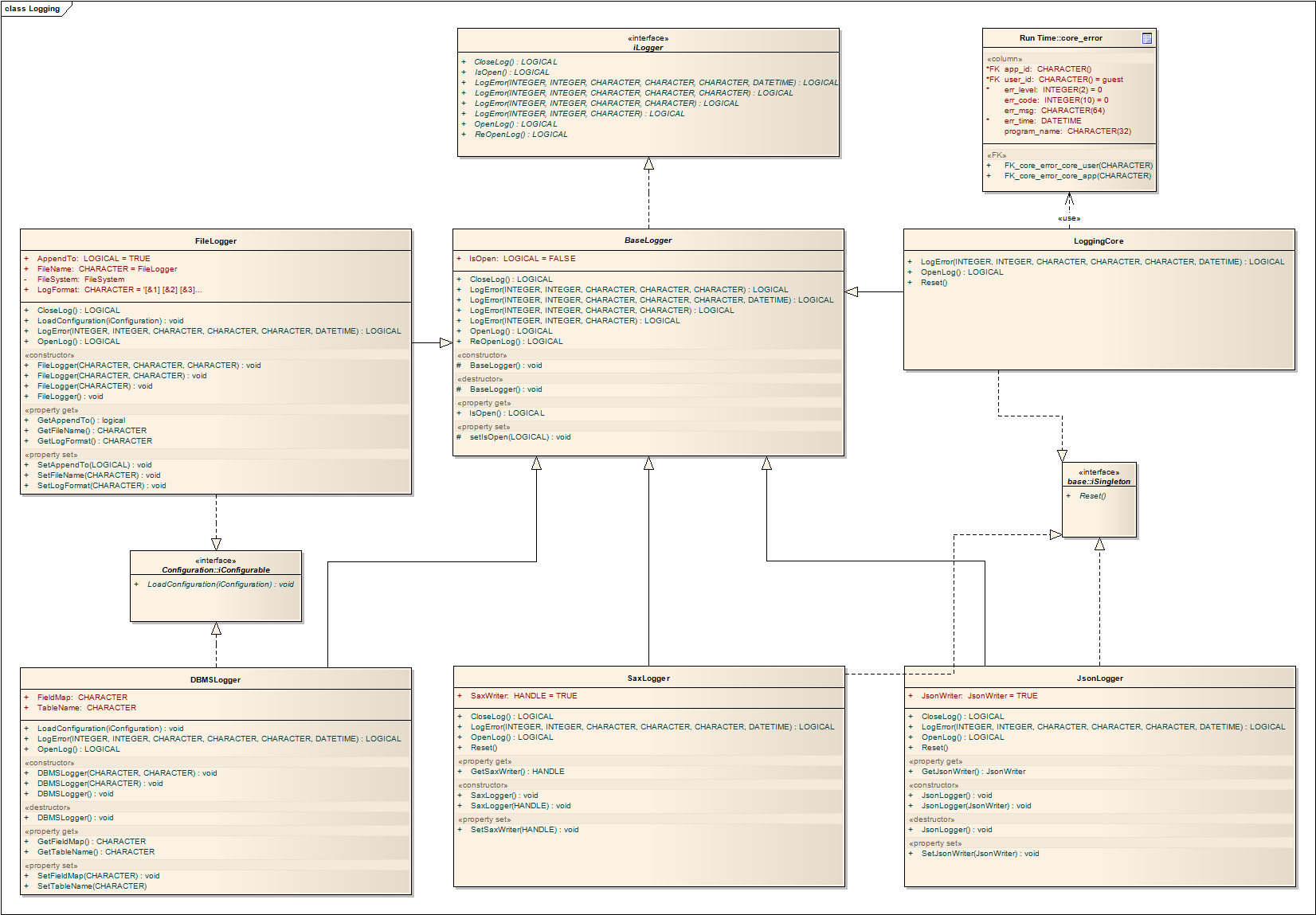
|  |
| --- |
| class myclass:  method public void doit():  end method.  end class. |

|  |
| --- |
| define variable i as integer no-undo.  define variable omycl as myclass no-undo.  define variable lDebugLevel4 as logical no-undo init false.  if LogLevel = 4 /\* Checking the log level from the framework \*/  then lDebugLevel4 = true.  assign omycl = new myclass().  etime(true).  do i = 1 to 1000000:  if lDebugLevel4 then omycl:doit().  end.  message etime(false) view-as alert-box.  delete object omycl. |

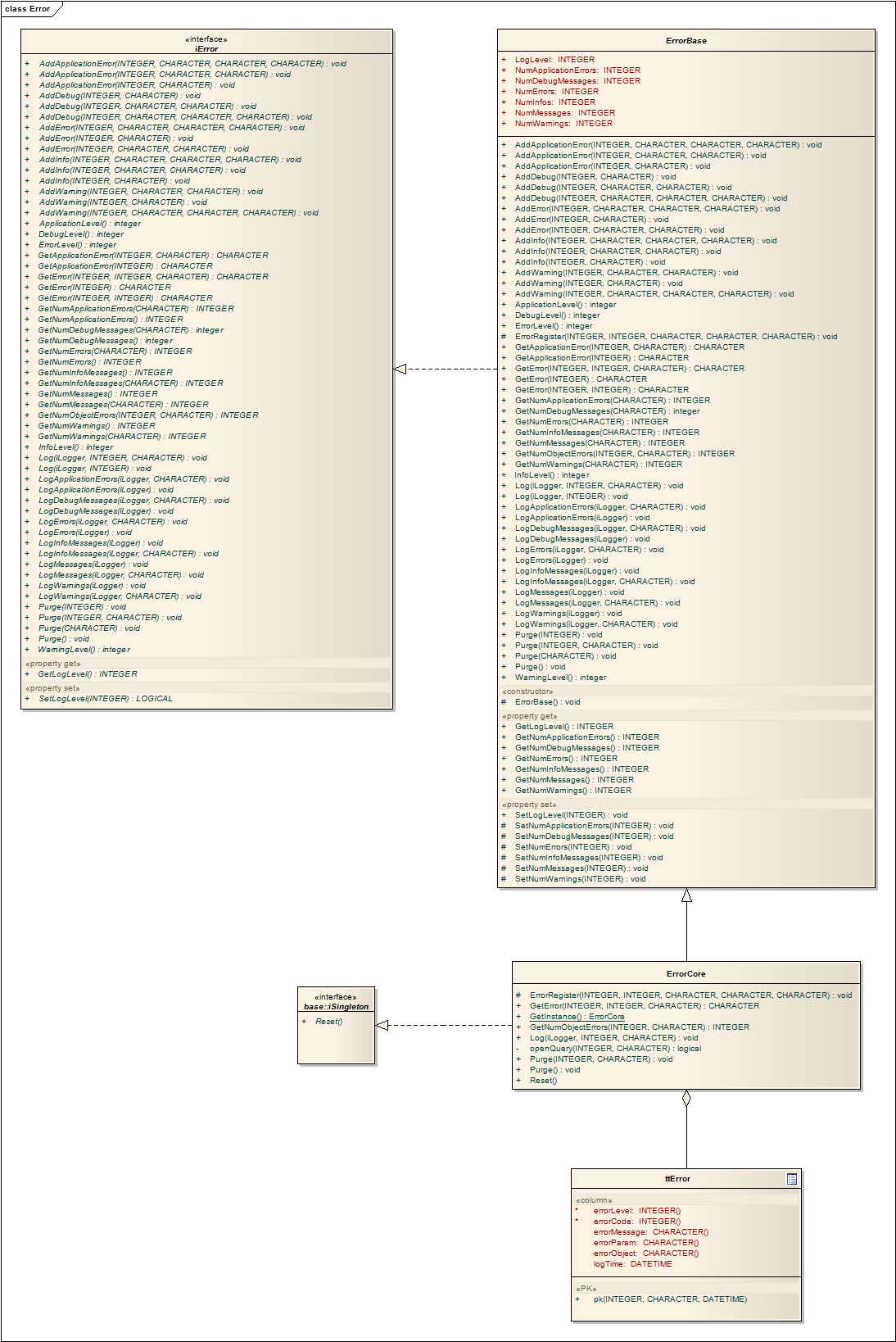
Please check the screenshot below how the logging service is configured, in the example we are using the Filelogger which creates a quarix.log file in the session working directory or in the folder containing the application server log files.



#### Architecture of the logging mechanism



#### Error manager archirecture



### Saving the data

The main entry point in the framework for saving the data in the data models is the method named dataUpdate.

The following methods are called during the save operation: BeforeSaveData, SaveData and AfterSaveData inside a transaction block, this means if one of the methods fails the whole transaction is undone and the changes are not saved.

None of the methods dataUpdate or SaveData should be overriden during the application development because they contain the correct logic for saving a data model.

The actual saving process of the records is implemented on the data access layer in the method named SaveData. The save mechanism makes use of the before image and after image records available in the temp table.

The changes sent by the frontend always contains also the before image records.

In case you would like to alter some of the records as part of the business logic you always have in mind to keep tgrack of the changes by setting the attribute TRACKING-CHANGES of the temp table object.

Before actually saving a record the framework validates them following the next rules, if one of them fails the save process is aborted and the transaction is undone:

* A primary key should exist on the temp table
* Read-only fields are not allowed to be changed
* Check if the database record is locked
* Call the BeforeRowSave, AfterRowSave, BeforeRowDelete and AfterRowDelete methods for programmer validations or custom logic
* Checking after duplicate records
* Checking after modified records, if the before image record is different than the databse record an error is thrown
* Checking for missing records, this means that the before image record is no longer available in the database

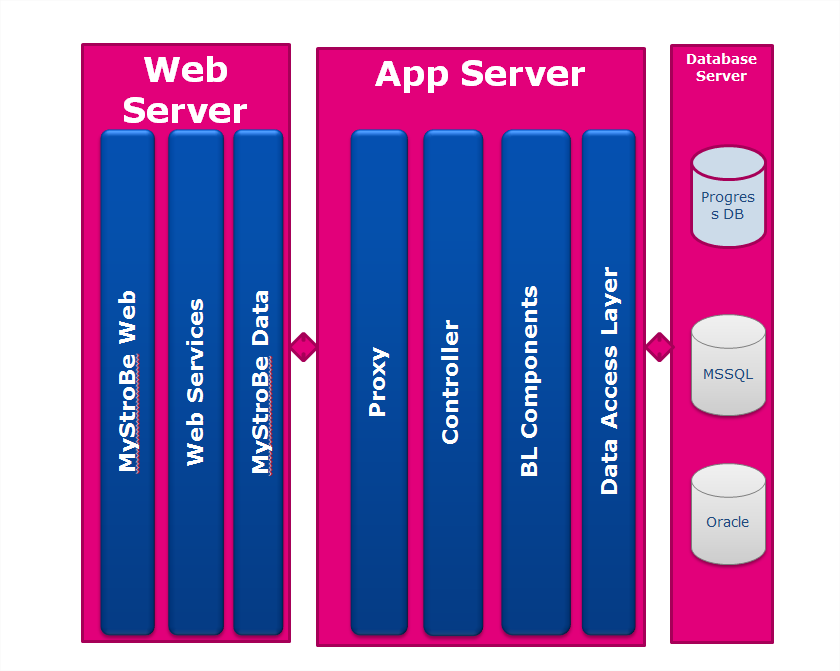
For data transactions the save mechanism must be implemented manually, this approach was chosen because the purpose of a data transaction to be able to create complex intercations between other data models and data transactions, following this idea an automated save is not possible to be implemented in the framework.

The main entry point of a save operation in a data transaction is the method named DataUpdate, in this case we have also available the method calls BeforeDataUpdate, SaveData, and AfterDataUpdate but they are not part of a transaction block.

To implement the save logic the method SaveData must be overriden and filled with custom logic where the dataUpdate method of other data models or data transactions is called.

|  |
| --- |
| method private logical SaveData(input-output dataset dsordordline):  define variable orderidlist as wicketds.server.orderidlist no-undo.  define variable hdsOrderLine as handle no-undo.  define variable hdsOrderTmp as handle no-undo.  define variable hdsOrderLineTmp as handle no-undo.  define variable httOrderLine as handle no-undo.  define variable hQuery as handle no-undo.  define variable iSortOrder as integer no-undo.  define variable iLinenum as integer no-undo.  Orderidlist = cast(GetInstance('wicketds.server.orderidlist':u), 'wicketds.server.orderidlist':u).  hdsOrderLine = getdsOrderLineTmpHandle().  if not valid-handle(hdsOrderLine)  then return false.  httOrderLine = hdsOrderLine:get-buffer-handle (1).  if not valid-handle(httOrderLine)  then return false.  for each btorder  where row-state(btorder) = row-deleted  no-lock:  orderidlist:addOrderNum(btorder.Ordernum).  end.  if orderidlist:IdListHasData()  then do:  dmorderline:ClearContext().  if not dmorderline:getbyorderidlist(orderidlist, output dataset-handle hdsOrderLine by-reference)  then return false.    create query hQuery.  hQuery:set-buffers (httOrderLine).  hQuery:query-prepare (substitute('for each &1 no-lock', httOrderLine:name)).  hQuery:query-open ().  hQuery:get-first ().  for each ttorderline  no-lock  by ttorderline.SortOrder descending:  iSortOrder = ttorderline.SortOrder.  leave.  end.  do while not hQuery:query-off-end:  find first btorderline  where btorderline.Ordernum = httOrderLine:buffer-field ('Ordernum'):buffer-value ()  and btorderline.Linenum = httOrderLine:buffer-field ('Linenum'):buffer-value ()  no-lock no-error.    if not available(btorderline)  then do:  temp-table ttorderline:tracking-changes = no.  find first ttorderline  where ttorderline.Ordernum = httOrderLine:buffer-field ('Ordernum'):buffer-value ()  and ttorderline.Linenum = httOrderLine:buffer-field ('Linenum'):buffer-value ()  no-lock no-error.  if available(ttorderline)  then delete ttorderline.  iSortOrder = iSortOrder + 1.  create ttorderline.  buffer ttorderline:buffer-copy (httOrderLine, 'SortOrder').  ttorderline.SortOrder = iSortOrder.  temp-table ttorderline:tracking-changes = yes.  delete ttorderline.  temp-table ttorderline:tracking-changes = no.  end. /\* if not available(btorderline) \*/  hQuery:get-next ().  end. /\* do while not hQuery:query-off-end: \*/  hQuery:query-close ().  delete object hQuery no-error.  hdsOrderLine:empty-dataset ().  end. /\* if orderidlist:IdListHasData() \*/  find first ttorderline  where row-state(ttorderline) = row-created  no-lock no-error.  if available(ttorderline)  then do:  dmorderline:ClearContext().  dmorderline:SetSort('ttOrderline', 'Linenum', yes).  dmorderline:DataContext:setBatchSize('ttOrderline', 1).  if not dmorderline:dataFetch(output dataset-handle hdsOrderLine by-reference)  then return false.  httOrderLine:find-first ('', no-lock) no-error.  if httOrderLine:available  then iLinenum = httOrderLine:buffer-field ('Linenum'):buffer-value ().  hdsOrderLine:empty-dataset ().  temp-table ttorderline:tracking-changes = no.  for each ttorderline  where row-state(ttorderline) = row-created  no-lock:  iLinenum = iLinenum + 1.  ttorderline.Linenum = iLinenum.  end.  end. /\* if available(ttorderline) \*/  wicketds.server.dsorderwrapper:createOrder(  input dataset dsordordline by-reference,  this-object:ToString(),  output dataset-handle hdsOrderTmp by-reference  ).  wicketds.server.dsorderwrapper:createOrderLine(  input dataset dsordordline by-reference,  this-object:ToString(),  output dataset-handle hdsOrderLineTmp by-reference  ).  if not valid-handle(hdsOrderLineTmp) or not valid-handle(hdsOrderTmp)  then return false.  do transaction  on error undo, throw:  dmorder:ClearContext().  dmorderline:ClearContext().  if not dmorder:dataUpdate(input-output dataset-handle hdsOrderTmp by-reference)  then undo, return false.  if not dmorderline:dataUpdate(input-output dataset-handle hdsOrderLineTmp by-reference)  then undo, return false.  end.  finally:  UnloadInstance(orderidlist).  Util:DisposeDataset(hdsOrderTmp).  Util:DisposeDataset(hdsOrderLineTmp).  Util:DisposeDataset(hdsOrderLine).  delete object hQuery no-error.  end finally.  end method.  method public override logical SaveData():  return SaveData(input-output dataset-handle datasetHandle by-reference).  end method. |

### Deployment Model



### Fill the dataset using an external procedure