Exility Core Reference Manual

EXILANT Technologies Pvt. Ltd.		
#27, P Kalinga Rao Road Bangalore – 560 027 INDIA	20195, Stevens Creek, Blvd #220 Cupertino, CA 95014 USA	
		Version. Revision 5.0.0
		Dated: 15-Jun -2015
		Reference Exility Version : 5.0.0
Web site: www.exilant.com		

Exility Core Reference Manual

EXILANT is the company behind Exility, a rapid enterprise web application development framework.
EXILANT Technologies Private Limited provides Information Technology (IT) solutions with marked focus on measurable value. It offers services and products in enterprise mobility, business intelligence and analytics, user experience/interface (UI/UX) design and development, enterprise application integration to BFSI, Hi-Tech, Retail, Telco, Transportation/Logistics, Textiles, Apparel, Dairy and Construction industry segments.
EXILANT was founded in 2004 and is headquartered in Bangalore, India. It is ISO 9001:2008 & ISO 27001:2005 certified and employs over 1400 people with operations in India, USA, UK, and Singapore. For more information, visit www.exilant.com.

DISCLAIMER:
This document is for informational purposes only and is provided "AS IS." The information set forth in this document is intended as a guide and not as a step-by-step process, and does not represent an assessment of any specific compliance with laws or regulations or constitute advice. We strongly recommend that you engage additional expertise in order to further evaluate applicable requirements for your specific environment. EXILANT MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND, EXPRESS OR IMPLIED, AS TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS DOCUMENT AND RESERVES THE RIGHT TO MAKE CHANGES TO SPECIFICATIONS AND PRODUCT/SERVICES DESCRIPTION AT ANY TIME WITHOUT NOTICE.
EXILANT RESERVES THE RIGHT TO DISCONTINUE OR MAKE CHANGES TO ITS SERVICES OFFERINGS AT ANY TIME WITHOUT NOTICE. USERS MUST TAKE FULL RESPONSIBILITY FOR APPLICATION OF ANY SERVICES AND/OR PROCESSES MENTIONED HEREIN. EXCEPT AS SET FORTH IN TERMS OF AGREEMENT YOU SIGN WITH EXILANT, EXILANT ASSUMES NO LIABILITY WHATSOEVER, AND DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO ITS SERVICES INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT. Except as expressly provided in any written license agreement from EXILANT, the furnishing of this document does not give you any license to patents, trademarks, copyrights, or other intellectual property.
All other product names and trademarks used in this document are for identification purposes only to refer to either the entities claiming the marks and names or their products, and are property of their respective owners. We do not intend our use or display of other companies' trade names, trademarks, or service marks to imply a relationship with, or endorsement or sponsorship of us by, these other companies.

HOW TO SUBMIT ERRORS, OMISSIONS AND SUGGESTIONS

If you find errors, omissions or have improvement suggestions, you can send an e-mail to zaglabs@exilant.com . Please clearly outline the following in your e-mail:

1. A detailed description of the error, omission or your suggestion
2. Your contact e-mail
3. Explicitly mention if you want your name included in credits list

We firmly believe in giving credit where it's due. We would like to publish the names, of individuals who help us improve this document. If you want your name to appear in credits list, please do mention explicitly in your email. If your suggestion is accepted and published, and if you have explicitly given us permission to publish your name, we would add your name to the credits list in appropriate document.

Contents

Overview

Exility is a set of tools, techniques, utilities, design aids and architectural philosophy meant for building well-engineered application software. This description is more appropriate rather than Exility being called as an ERP framework.

Here is why Exility is clearly set apart from the large number of ERP frameworks.

1. Exility views your application software as two *independent* applications
    a. A web application that interacts with users,
    b. A server application that delivers a set of services simultaneously.

    A typical framework, on the other hand, implements these as loosely coupled layers.
2. A framework provides you stubs or interfaces to be implemented in each of the layers of its multi-tier architecture. That is, each of your functionality (use-case) is implemented by creating components in each of the layers. In the case of Exility, you are unaware of the layers. You just use declaratives to implement your user interactions (UX) and service implementations. Exility provides all the necessary infrastructural components to deploy your application in a state-of-the-practice deployment architecture. As a result, your application is 'future proof'. As the state-of-the-practice for deployment changes, Exility will revise its infrastructural components to make use of these features. And your application requires no or very little change to migrate to the new technology.

Rather than resorting to comparison, let us first understand Exility on its own. We will use comparison and contrast at times as a learning technique in order to understand how Exility works in itself.

Exility design and development process prescribes a 'Client First' approach for your application. We recommend that you start with a client application. It has been our experience that when we ask users to review a *'prototype'*, we generally get only cursory feedback. Small changes here and there, about fonts and background colours. We get serious and useful feedback about the actual usability aspects only when they actually *use* the system. And, typically that happens during user acceptance phase, or much worse – after going live. Software Industry has tried all kinds of techniques to change this, and has tried to get useful feedback earlier in the game. Several new development processes have replaced the legacy 'water-fall' model. However, it is very clear that the real feedback comes only after users use the system. With Exility, it is easier to go to customers with a fully functional client application earlier in the development cycle, thereby reducing the pains of acceptance testing.

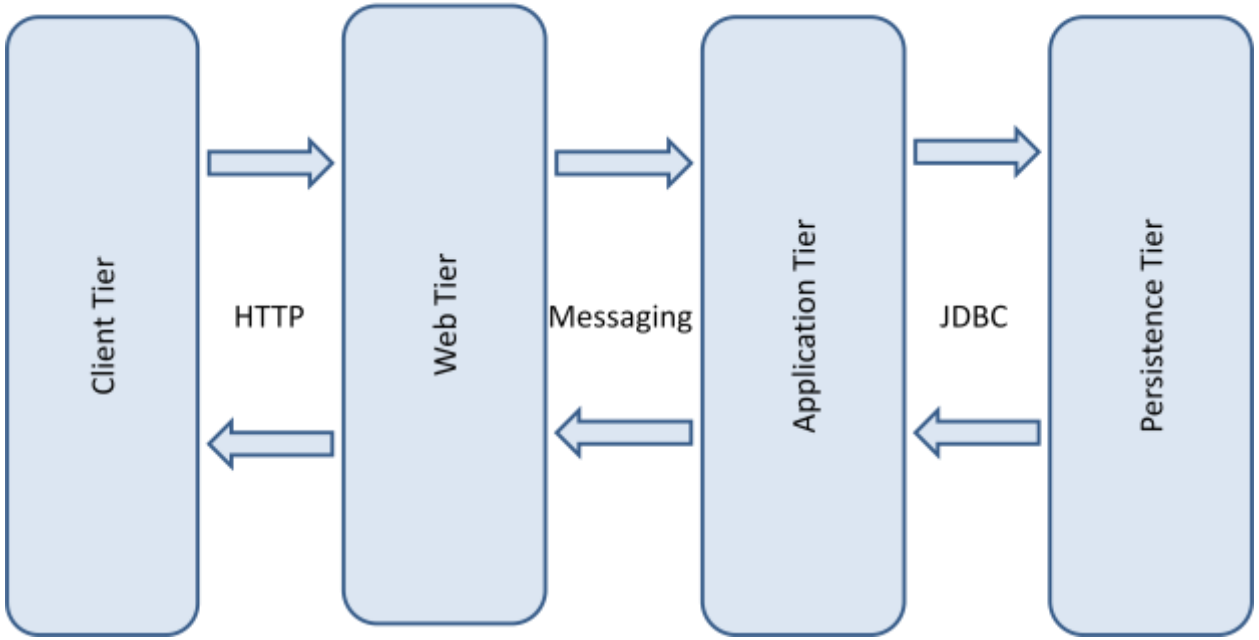
If you have used any other framework, one thing you will immediately observe is that, unlike those models, we do not talk about architectural concepts such as 'Aspect Oriented Programming', 'Object Oriented principles', 'Entity Beans', 'Dependency Injection', 'O/R Mapping', 'DAO', 'DTO', 'Action Mapping', etc. This does *not* mean that we are not for any of them. On the contrary, we firmly believe that all these concepts must be integral part of any well-engineered application. The difference here is that Exility internalizes all these principles, and releases the application developers from the burden of implementing them. As an application developer, you think of your design components, and Exility takes care of plumbing them together the right way to deliver the desired functionality in a flexible deployment scenario.

No Entity Beans? You heard it right. No Entity Beans. It is our view that business logic can be implemented in an elegant way using generic data structure, rather than encapsulating them into objects. For example, take a fairly complex pricing logic. If we take a pure object-oriented principle, we think of several objects like customer, product, order, order-history so on and so forth. And where do we attach the pricing method? Would it be Cusomer.getPriceForProduct(Product product,,,,,) or Product.getPriceForOrder(Order order,,,,) It would be fairly complex to implement a theoretically right encapsulation principle. But then, why do we encapsulate? There is a purpose. We want to minimize the code that needs to be looked-at when we have to change the pricing algorithm. If you accept the objectives as more important than the chosen method, we offer an alternate principle and evaluate it for the stated purpose. How about traditional functional programming with suitable data-structure? It has been our experience that such an approach is simpler than a strictly encapsulated object.

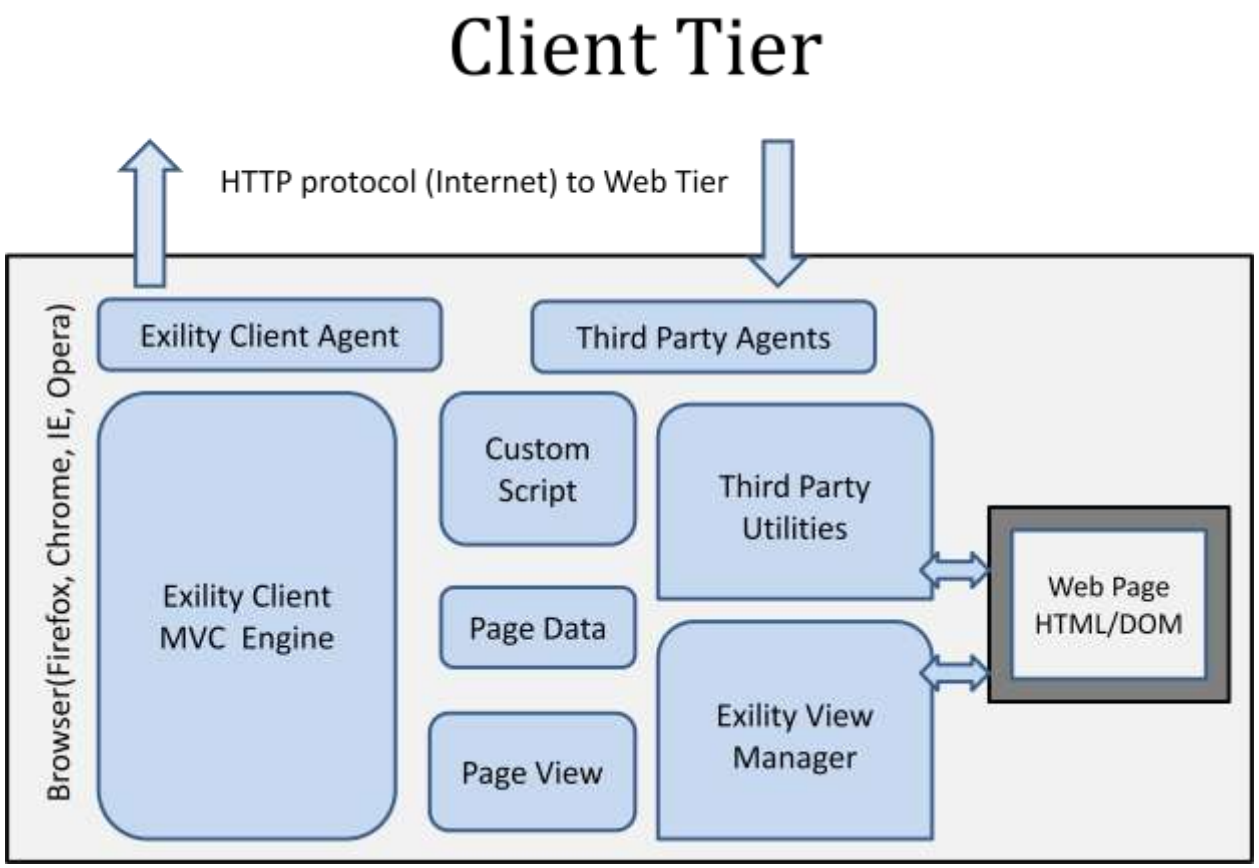
In fact, we believe that the very presence of public getter/setter methods on private attribute itself means that the encapsulation is not suitable for the problem on hand. Object.attribute = "value" or Object.setAttribute("value") is just a matter of syntax. The real issue is whether the fact that an object has a specific attribute is to be known to others or not? In case of ERP applications, Data Model is typically the foundation of the whole system, and its scope goes beyond the software system: it is a business requirement that customer should have name. It is not appropriate to encapsulate name inside customer class. Once again, we would like to reiterate that we are not against Object Oriented Design. Exility itself is built on very strict Object Oriented principles. You will see very good use of most of the OO principles. Our argument is about how to implement a service on the server side of an ERP application. And that is where we are recommending use of data-structures rather than encapsulated attributes of classes.

Exility uses declarative based XML files as design-time work products. However, the run-time environment conforms to commonly used industry standards on all the '*tiers'* as well as the protocols used to communicate between these '*tiers'*. For example, application tier consists of simple java objects (POJOs). XML files that users develop or maintain are loaded as object instances. This approach provides maximum flexibility for the application to be deployed at run time to suit available infrastructure as well as demands on performance. The following diagram is a simple depiction of the tiers and how they interact at run time.

Deployment Architecture



We briefly describe these tiers/layers.



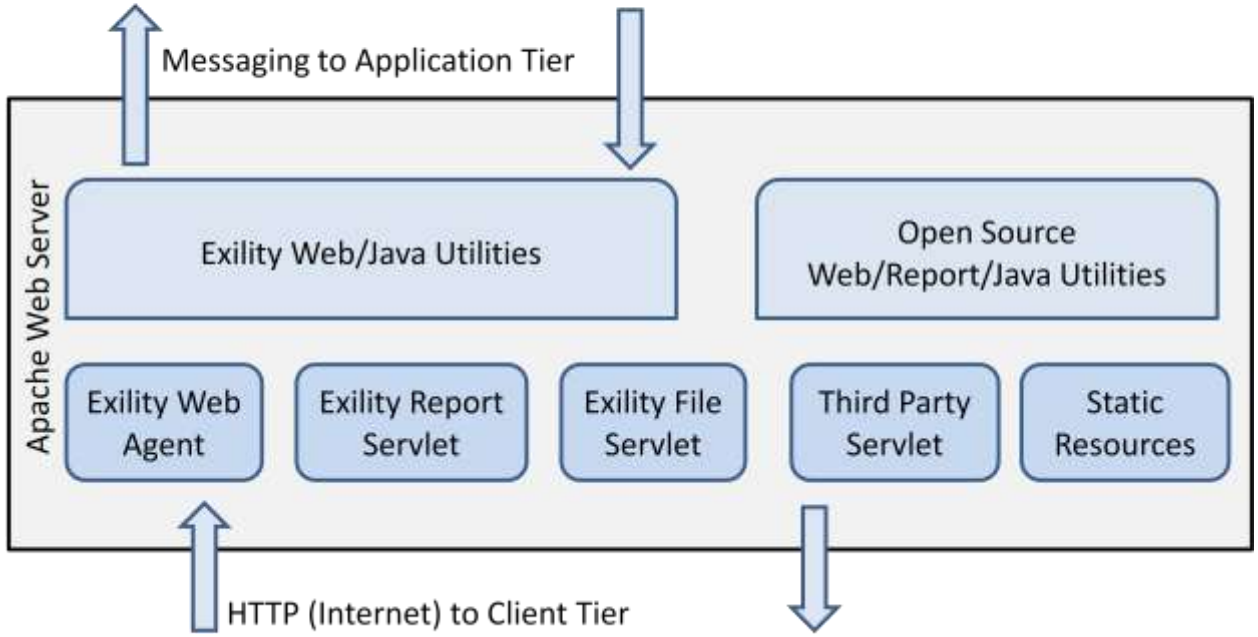
Client application is completely managed inside the browser using Javascript, html and css. Exility uses a Model-View-Controller based engine at client side. Data is organized on the client as Javascript data structure (Model). HTML is used to render the

page(View). Controller tracks user interaction with the view, and keeps the data-model synchronize with it. Controller also ensure that any data received from server into the Model is also pushed to the View as part of this synchronization process.

While Exility's primary focus is data, it also provides a set of widgets, like list and grid, for a user-friendly UI. However, users are encouraged to use the state-of-the-art utilities for UI. For example, pages can exploit the large number of widgets that are available in the open source world like jQuery. In case your application does not need a strong client-side MVC, and a fancy UI is more important, you may do away completely with the Exility engine, and use a small API to communicate with the server.

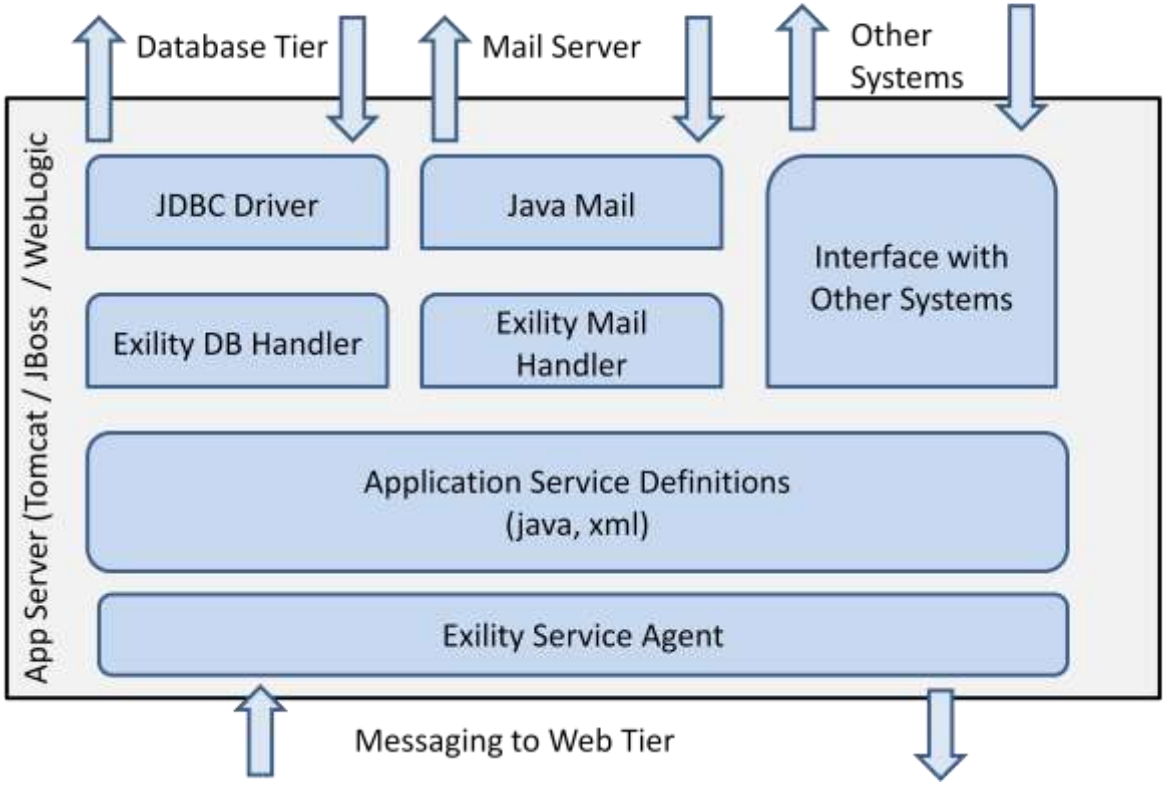
We emphasize the need to look at the client and the server tiers more like independent applications. But this "independence" is in their "architecture", that is how they do the role that is allocated to the. When it comes to what they do, they are very tightly coupled to deliver the delivered functionality. We use standard HTTP protocol to implement a simple communication protocol between the client and the server. Exility Agent uses asynchronous http calls to connect to server (This technique is popularly known as AJAX). Agent always picks up data from the client-side data-model, but not directly from html/dom. This provides maximum flexibility in managing the view independent of the data.

Web Tier


Messaging to Application Tier
Apache Web Server
Exility Web/Java Utilities
Open Source
Web/Report/Java Utilities
Exility Web
Agent
Exility Report
Servlet
Exility File
Servlet
Third Party
Servlet
Static
Resources
HTTP (Internet) to Client Tier

Exility's unique feature is that there is only one entry point at web-tier, and there are no service specific components in the web-tier. Web tier is kept very thin. It manages sessions and basic data-type based validation of requests. It also houses specific components for reports and file upload/download utilities.

Application Tier



In almost all applications that are built using Exility so far, application and web tiers are merged. Either Tomcat or JBoss are used as servlet containers, and the application layer is also managed in the same instance. However, it is possible to have a server farm as application layer with a load-balanced set of web servers as fronts to receive http requests behind a router.

Large part of application functionality is implemented using declarative syntax of Exility Service. However, users do write java code to implement business logic that is not readily amenable to the utilities provided by Exility.

Exility uses a thin wrapper around JDBC to restrict its use to a smaller standardized subset. This is to provide maximum portability across RDBMS servers without compromising on the features that a specific RDBMS may provide. However, user written java code, on a need basis, can utilize all features of JDBC.

Projects have written java code using which an Exility service can communicate with other applications.

It is important to keep in mind that the Exility environment is not exclusive, and it can co-exist with other components.

A simplistic view of the sever is that it

- Is a repository of all business data
- Enforces all business logic
- Responds to request for any of the services published through ‘service list’.

Basic Server Concepts

Exility provides utilities on the server side for you to implement services using declaratives. That is, you specify what you want to do in terms of data base interaction and data processing.

Following are the work products that are stored in .xml files

1. **DataTypes** : This xml specifies the data-types that are used in the application
2. **DataDictionary**: This xml specifies all the fields that are used in the application
3. **Messages :** This xml has the list of messages and their classification
4. **Record :** A logical data model for all the data that the application deals with. This is an alternative to DataDictionary, and is designated to replace it completely in the next version.
5. **ServiceList** : this is the initial entry point to the server. It has the list of services that can be called from any client
6. **Service Spec :** Defines input and output parameters for each service that is available to clients. It also defines basic validations for these parameters.
7. **Service**: Defines how a service is delivered / executed.
8. **SQL :** a template definition of a sql which is converted into a valid sql statement at run time for extraction or any other database operation. This is also used for triggering stored procedures.
9. **Table Definition:** table structures based on which table-based *read/insert/update/delete* operations can be carried out.

Data Types

Value Type

Exility based applications handle "data" – pieces of information. Data is typically organized into a "data base". First name of customer is a text-data while date of birth is a date-data. If you are used to any programming language, you represent such data in a "value type". For example, Java programming language offers boolean, char, byte, short, int, long, float, double, date and string as types to hold values. (For the purpose of this discussion, we do not differentiate between an Object and a Value). We notice that the values are primarily Numeric, Textual, Date (possibly with time) or Boolean. Further, treating a number as either decimal or integral has business implications. Hence, Exility supports the following value types.

1. Integral
2. Decimal
3. Text
4. Date (possibly with time)
5. Boolean

But then, why do we not provide finer definitions like byte, short, int and long for integral values? For the type of applications that Exility is designed to deal with, such finer optimization of memory is not important. However, keeping it simple with one integral data type avoids several value-size related issues. Also, the memory used by these data is "transient", in the sense they are used only during execution of a transaction. Their permanent storage is inside databases, and Exility does not put any constraints on the type of data elements that the database uses. For example, the underlying data element in a database may use byte as its storage. This takes-up one byte on the disk, while Exility may take up 8 bytes in memory during the execution of service that uses this data element.

Exility uses long data type provided by java that can hold 18 digits. This is good enough for all known business applications. Similarly, the double data type provided by java is good enough for any decimal data type.

Need for Data Type

In an application, there may be few hundred or a couple of thousand different pieces of data that you may be tracking. Several of these may be bound by same restrictions on their values. For example, you may have "description" as a field for fifty six different entities. For sake of standardization and simplicity, you may want a description field to be restricted to a max of 250, 500 or 1000 characters. It makes sense for you to define three "data-types": short-description, medium-description and long-description. While defining a description field, you simply choose one of these three.

Similarly, you may standardize on three sizes for amounts. Note that these restrictions are driven by business requirements, and not meant to optimize computing resources.

Then there are business constraints on some values. Date of birth has to be in the past, and expected delivery date of an order has to be in the future. You may insist that a user-id be formed with only alpha-numeric characters starting with an alpha character. As you can see, these business rules put additional restrictions on the range of possible values within that type of value.

shortDescrition, mediumDescription, smallAmount, pastDate, userId are all examples of data-types in Exility. We will give precise definition later in this module.

How do you know what data types you need?

Though we explained data type as the first basic concept, its usage comes once you start creating your data elements in your data dictionary. With a view to build concepts bottoms up, we explained data type before data elements. Obviously, you do not need any of them till you do your data base design. Exility gives you a small set of predefined data types that are used internally by Exility. You will start with this list, and add others on a need basis as you design your data base.

Naming Data Types

A good naming convention goes a long way in making an application maintainable. Data types should be named based on their intended use rather than based on their attributes. For example "shortDescription" is a better name than "text250". Suppose you choose text250. After a year, your business decides that the description fields be restricted to 200 characters instead of 250. See what happens to the name if you change its length to 200 now. PartNumber, name, age, fileName, yesNo are all examples of good data type names. date1, char12, customerName, employeeDateOfBirth are examples of not-so-good data type names.

How to create data types

Exility provides some basic data types already defined. (These are the ones that are used internally by Exility) You may find them inside WEB-INF/exilityResource/dataType folder. This folder may contain one or more .xml files. All data types defined in these files are accessible to your project. You should not touch these files. Also, you should not re-define them in your files to use them differently. If you do (by mistake ☺) Exility will refuse to over-ride the built-in meaning, after giving a warning at the time of starting the server.

Attributes that define a data type are described below.

Common attributes for all data types

Name	Type	Reqd?	Default	Explanation
name	string	yes	-	Unique name across the project. This has to be unique across modules. Name of the qualified data-type to be used in data dictionary.
description	string	yes	-	Describe why you created this data type, its intended uses and misuses.
messageName	string	yes	-	Message to be displayed if the data does not match the data-type definition. This name must be defined in the messages.xml file. If messageName is not specified, description is used. Note that specifying messageName enables you to deploy the application for different languages.
formatter	string	no	-	Name of a formatting function to be used by the client application to format this data for rendering. Exility client application provides lcase, amt, inr, inr2, eur, eur2, usd and usd2 as built-in formatting functions. You can add your

Name	Type	Reqd?	Default	Explanation
				own formatters in your myProject.js file. (refer to documentation about myProject.js) Note that this attribute is not used by server.
sqlType	string	no	-	Used to link this data type to a sqlType in RDBMS. Used by utility functions that would provide some mapping of RDBMS to data dictionary. This is also used in automating RDBMS table creations from data dictionary based tables.

Integral Data Type

Name	Type	Reqd?	Default	Explanation
allowNegativeValue	boolean	no	false	Specifies if the value can contain negative numbers or not
minValue	number	no	-	Minimum value that can be entered in this field. It is an error if you provide a negative value here and set allowNegativeValue=true;
maxValue	number	no	-	Maximum value of the integer that can be entered in this field. Default max has 18 digits that is good enough for any business application.

Decimal Data Type

Name	Type	reqd	Default	Explanation
allowNegativeValue	boolean	no	false	Specifies if the value can contain negative numbers or not
minValue	number	no	-	Minimum value that can be entered in this field.
maxValue	number	no	-	Maximum value of the integer that can be entered in this field
numberOfDecimals	number	no	2	Specifies the number of digits that are to be present after the decimal. The number is rounded off to the nearest decimal value if this is result of a calculation.

Text Data Type

Name	Type	Reqd?	Default	Explanation
regex	string	no	-	To specify a regular expression to validate the value in the field .Please refer to http://www.regular-expressions.info/quickstart.html for a introduction on how to use and form regular expressions
minLength	number	no	0	Minimum number of characters that need to be entered. Note that this restriction is applied only if the value is specified. Hence, you should not use 0 as min length.
maxLength	number	yes	-	Maximum number of characters that can be entered. It is a theoretical argument that one does not want to restrict the length. In business applications, you MUST restrict the length to a practical value.

Date Data Type

Name	Type	Reqd?	Default	Explanation
maxDaysBeforeToday	number	no	-	Is there a minimum? If so, how far in the past can it be from today? Note that this field can be negative to indicate that the min date is in the future.
maxDaysAfterToday	number	no	-	Is there a max? If so, how far in the future could it be from today? If this is negative, max date is in the past.
includesTime	boolean	no	False	Does this contain time of the day also?

Boolean Data Type

Name	Type	reqd?	Default	Explanation
trueValue	string	No	1	A Boolean data type is always internally stored as bool in java and bit/Boolean in rdbms. Server will use 0 for false and 1 for true while communicating with client application.
falseValue	string	No	0	specifies the valid value to mean “false” e.g. “0” ,”No”

Data Dictionary

Data dictionary is a repository of all the data elements used in the application. Exility enforces that every data element used in the application MUST be defined in the data dictionary.

Why MUST ? Well, that is what a dictionary means. I am sure you would be disappointed if you look for a Kannada word that you encountered that is not found in Kannada-English dictionary. Data dictionary serves this purpose. It is a repository where you find the meaning, and possibly more details like some documentation about the business logic around that data element.

In an application, you may use the same name to refer to different piece of information. For example, "code" can be used in several different entities. Customer code could be a text field, while employee code could be a number. Exility requires that you create different data elements if the business meaning is same, but require different data type. We recommend employeeCode and customerCode rather than just "code" so that the data element name is meaningful even if the context around it is not visible.

Data elements used to be defined inside a dictionary.xml file. This method is discouraged after we introduced the concept of records and fields. Skip to records if you are going to create a new project. If you have to understand and maintain an Exility based project that uses data dictionary.xml, continue with this section.

Data Element Attributes

Name	Type	reqd?	Default	Explanation
Name	String	yes	-	Name is unique within an application. It is possible that you may refer to two different data elements (like columns in two different tables) by the same name. This is not a problem, so long as these two have the same data type. In general you should not define such an element twice. But, in case you follow some standard convention that makes more sence for you to mention these two columns, then Exility does not throw an error. Historically, we used to recommend a naming convention with groupName_name as unique. Based on experience followed by the projects, we now recommend that the group concept be done away with.
Label	string	yes	-	Label to be displayed if this field is used on the screen. This is the default English label.

Name	Type	reqd?	Default	Explanation
				With translation facilities, it is possible to extend this to multi-lingual UX. If your design requires you to use different label for different iinstallation, like an ERP product uses an org structure, you can use custom labels. Refer to section on custom label.
Description	string	yes		Description as seen by end-users of the application. This is not used by Exility so far.
businessDescription	string	no	-	Document the business meaning of the element here
technicalDescription	string	no	-	Document the technical notes for this element if any
isInternalElement	boolean	no	No	If true, this element is for internal use, and is not something a business user can understand. For example, designers create parameters to increase the flexibility of the system. These are internal fields, and not business data. Utility functions are yet to be developed to make use of this attribute.
dataType	string	yes		One of the declared data types
listServiceId	string	no	-	Good example is countryCode. If the values that this data element can take are limited to a small list of values that are stored on the server, then you can specify the list service name here. (refer to list service concept on the page on the client side) Any for field that uses this data element will automatically use this list service.
valueList	string	no	-	If the values that this data element can take are very small, and are not kept in a data base, they can be specified here as a comma separated list. This will be used as valueList for form fields (refer to valueList of selectionField on teh client side)
messageName	string	no		Used by the client to show an error message if this data element fails validation.
Formatter	string	no		Refer to formatter of data type. This entry over-rides data type attribute.
customLabels	xmls	no		This feature is meant for products that are configured for each installation. Some data elements may be internally the same across all installations, but are known by different name. For example organization units may be called as “Zone”, “Department”, “Region” etc.. Custom labels are a set of key-value pairs. Key represents a specific installation, and the value represents the label to be used in that installation. These are entered as xml child nodes. This feature is deprecated and should not be used.

How to create data elements?

Data elements are organized into files under dataDictiory folder. Exility uses set of data elements for its internal use. These are found under WEB-INF/exilityResource/dataDictionary folder. You may use any of these file as a reference and create your dictionary files.

Note that you should not re-define data elements that are already defined by Exility.

You will notice that the dictionary file has a concept of group. Use of group is deprecated, and you should use a standard where one file has exactly one group.

You should use a suitable folder structure to organize your dictionary files. Exility will parse through the folder structure and load all files, and the actual folder structure is of no consequence. It is for your convenience.

Record

Term record comes from the days when computers were not used for 'record keeping'. A record consists of fields. We have three types of records:

1. Storage: represents a record that is stored as part of your data base. A storage record maps directly to a table in your RDBMS. Note that only storage records can be used for writing to the data base. (new/modify/delete)
2. View: represents a record as seen by end-users. Typically a view has either a subset of fields from a storage record, or it is formed by picking fields from a set of related storage records. For example when you show details of a customer on a page, you may want to show total order value of this customer, number of pending orders etc.
3. Structure: this is a set of related fields that are utilized by some part of your application, but they are not directly mapped to any field in a storage record. These fields are temporary data elements that are calculated at run time implement some business rules.

Note that if you define records, you do not have to create dictionary.xml fields for data elements. Records simplify definition of other components like page, service, table and sqls.

Record Attributes

Name	Type	reqd?	Default	Explanation
name	String	yes	-	Name is unique within an application. We recommend camelCase notation. In case you cannot use such a notation in your RDBMS, we still recommend that you use tableName attribute to keep teh rdbms name different from record name.
aliasName	string	no	-	If you have already defined a record, and you want the name of this record to be an

Name	Type	reqd?	Default	Explanation
				alias to that record, just specify the name in this attribute, and do not bother about the rest of the attributes in this record
recordType	string	yes	Storage	storage – this record corresponds to a data base table. view – this record represents arrow of data to be returned to the client. It typically has fields from more than one table. structure – this is a collection of fields that are used during a service execution, and not part of data base.
description	string	yes		Description
tableName	string	no	name	In case of a storage or view, this is the name as defined in the database
parentRecord	string	no	-	For a storage record, use this to link it to the parent table, as used in classical sense. Like orderHeader is parent for orderliness. It is for documentation only. For view, this is the base record from which the view is derived.
relatedRecords	string	no	-	Relevant only for a view. Comma separated list of records from which the view is created
isAudited	boolean	no	false	Should Exility automatically write audit logs for this record? Refer to relevant secion for other details.
keyToBeGenerated	boolean	no	false	Should Exility generate next internal primary key while inserting new row into this table?
Fields				List of fields in this record (refer to next section)

Field

We chose this term because we use it to define all aspect of the data element: storage, processing, and UI. Once a field is fully defined, Exility picks up appropriate attributes from it depending on the context where it is used. For example if the field is used in a page, then the UI attributes are used for rendering, and data type attributes are used for validating the value entered by the user.

Field names are unique within a record. Same field name may be used in another record. However, the two fields MUST be if the same value type. This restriction is put by Exility because the name of the field is used in certain contexts where the record that it belongs to is not known, and the field is to be parsed into its value. We encourage qualified field names. For example it is better to call name of customer as customerName rather than just name.

Field Attributes

Field has attributes for storage as well as for UI. Idea is to have all the relevant information in one place. Note that the UI attributes will be used across pages. In case, for whatever reason, you have some UI attributes different in different pages, you the option of over-riding them in that page.
For fields in a view, you should specify just the referredRecord and referredField. Other attributes are always copied from this field. Typically view is used as output in a page, and hence the UI attributes are not relevant anyways.

Name	Type	reqd?	Default	Explanation
name	String	yes	-	Name is unique within a record.
label	string	yes	-	Label to be displayed if this field is used on the screen. This is the default English label. With translation facilities, it is possible to extend this to multi-lingual UX.
description	string	yes		Description
columnName	string	no	name	As in the data base, if it is different from name
columnType	string	no	-	Refer to the next table that describes the possible values
referredRecord	string	no	-	For fields in a view this is mandatory. Record to which this field is referring to
referredField	string	no	-	For fields in view, this is mandatory
dataType	string	no	-	One of the declared data type
valueList	string	no	-	If the values that this data element can take are very small, and are not kept in a data base, they can be specified here as a comma separated list. Use comma separated values. In case the internal value is different from the value being displayed to users, use the format internalValue1:displayValue1,internaValue2,displayValue2.....
isOptional	bool	no	false	Is this field optional in this record. Is the record meaningful without a value in this field?
defaultValue	string	no	-	In case the field mandatory, would you provide a default value.
basedOnField	string	no		Sometimes, the field is optional. But if a related field is specified, then this field becomes mandatory. In such a case, set isOptional=”true” and set this attribute to name of that field..
basedOnValue	string			A variation of the above. This field is optional, except when a related field is set to a specific value. In such a case, use this attribute to specify the value associated with the basedOnField.
mutuallyExclusiveField	string			Reverse of basedOnField. If you have a pair of fields, and if one of them has value, then the other one should not have value. Specify this attribute for one of the field. Note that the two fields are together optional.
fromField	string			If you have a pair of fields that together specify a range of values, like dates and

Name	Type	reqd?	Default	Explanation
				amounts, use this attribute on the field that has the upperLimit. Note that, for a pair, you specify this attribute for the toField, and do not set anything on the other field.
toField	string			This is to enable a field to participate in more than one range. Once again, note that for a given range, you need to specify either fromField="" or toField="" and never both.
defaultInputType	text	yes		If this field is used in a page for user to input the value, what field type should we use? Note that you can over-ride this in the page, if you need different field types in different page, for some specific reason.
messageName	string	no	-	Error message to be flashed to end-user in case this field fails validation as per details given for this field. If this is not specified, message associated with the dataType is used.
listServiceName	string	no		Name of the service to be used to contact server for list of values for this field at run time. This service is used as a listService if nbrCharsToShowSuggestions is set to 0. Otherwise it is used as suggestionService. Note that listService is valid for assistedInputfield and selectionField only. the page field is selectionField. If nbrCharsToShowSuggestions is set to non-zero, then this is us used as suggestionService.
listServiceFields	string	no		Comma separated list of fields either from this record, or from others that are to be sent as data for the list service.
descriptionServiceName	string	no		Name of the description service to be used to either validate this field or get some data based on this value at run time.
descriptionServiceFields	string	no		Comma separated list of fields either from this record, or from others that are to be sent as data for description service. Note this field is sent by default.
formatter	string	no	-	When this field is used as output, you can use a formatter on the client side. Exility provides inr, usd, inr2 and usd2 as built-in formatters. You may add your formatters on the client side. Refer to myProject.js documentation for the same.
isNullable	bool	no	false	Can this field be null in the database.

Column types

Column Type	Explanation
data	This is a normal field in a storage record, and is not any of the special fields listed below
primaryKey	This is the actual primary key. This is used in both storage and views
logicalPrimaryKey	This column is to be unique, but it is not defined as primary key. costomerCode could be logical primary key, but in the data base we could have used customerId, in internally generated number, as a primary key.

Column Type	Explanation
parentKey	Column that links this record to its parent. In this case, name of the parent record has to be specified in referredRecord.
foreignKey	Key into a related table. Like financialYearId. Once again, the record to which this column links is to be specified in referredRecord
createdTimeStamp	Special field in the table
createdByUser	Special field in the table
modifiedTimeStamp	Special field in the table
modifiedByUser	Special field in the table
view	All fields in view MUST have this type
temp	All fields in structure should have this type

Following input fields are valid for defaultInputType

- textInputField
- assistedInputField
- selectionField
- textAreaField
- radioButtonField
- checkBoxField
- hiddenField
- outputField means that this field will remain to be output though the panel is meant for input
- checkBoxGroupField
- imageField
- StyleField

Messages

Messages provide a means of externalizing descriptive text associated with an error, warning or just about any text. It is more like a code and description. Internally we always use code, and whenever this has to be displayed, we use the corresponding text in the desired language from messages file.

Messages are organized in .xml files under message folder. Exility internally uses a set of messages. You can find them under WEB-INF/exilityResource/message folder. You can choose to put all messages in one file, or organize them by module into different files. Note that the message name has to be unique across files.

A message can be designed to have run time parameters. Place holders for such parameters are indicated with @1 etc.. For example Customer @1 does not have the desired minimum balance of @2. Exility provides feature for the designer to supply customer name and min balance amount for this message at run time.

Name	Type	reqd?	Default	Explanation
name	string	yes	-	Name to refer the message in other files . Exility comes with certain pre-defined messages for common errors. All these messages have names starting with "`exil`"
severity	string	yes	-	The severity parameter can have the following values • `error` : This message will stop the execution of the branch of code and return with this severity to the caller • `warning`: This message will be added on to the set of message collection and the execution proceeds. • `info`: This message will be added on to the set of message collection and the execution proceeds • ignore: for a given installation, this messageId is not relevant, and hence it is to be ignored. That is, logic should behave as if this messageId is never added.
text	string	yes	-	Text of the message ; this text can have space holders for parameters to be substituted at runtime using the syntax "@1" , @2 ..
forClient	boolean	no	false	Whether this message should be passed on to the list of messages used by the client. All messages used by the client, including the messages associated with data-types, are converted into a javascript file and made available to the client application.

Service List

Service list concept is being phased out. If you are going to work on a new project, you do not need this anymore.

This is a list of all services that a client can ask for. This list is meant for the "middle tier" that typically is implemented on the web server. Clients are connected to application server through this web server. Web server looks-up this list to validate a service request coming from a client. We call an entry in this list "serviceEntry". This is to differentiate such an entry from the actual service that is implemented on the application server.

A service entry specifies the data that need to be accompanied with the request, as well as the data that is delivered as a response. Exility component sitting on the web-server uses this specification to do a preliminary data validation for a service request. In case the client has not sent the right input data, a suitable response is sent from here, without contacting the application server.
<we need to put a diagram to illustrate this>

Service entries are organized into files within a folder named serviceEntry. We use qualified names for service that mimics the folder structure. For example a service called "order.createOrder" will be matched with a service entry named "createOrder" inside a file called "order.xml" inside serviceEntry folder. In case you have small set of services, or in case you want to have some unqualified service names, you can put them in a file called "serviceList.xml" under your resource root folder.

You may choose to have more than one level of folder structures, in which case the qualified name will have more than one "dotted prefixes".

ServiceList was not mandatory in our first version. We used to create a service entry on-the-fly for any service that the application has created. We have now made this mandatory to ensure that the application retains deployment options in the future. With this restriction in place, we can state that the purpose of the server application is to implement all the service entries defined in the service list. Conversely, if a service defined in the application is not linked to a service entry, directly or indirectly, then it is not used on the server application.

Service Entry Attributes

Name	Type	reqd?	Default	Explanation
name	string	yes	-	Name of the service, as referred by the client would be moduleName.name where modukeName is the name of the file in which this entry is made. By default, this is also the name of the service that this entry refers to.
serviceName	string	no	name	In case the service name is different from entry name. Note that it is possible to a mix-and-match between the service names used by the client and the ones used by the server, but this will be quite confusing. We recommend that you resort to this only if there is some other issue in using the same name.
description	string	yes		Short description of the service
allowAllInput	boolean	no	depends	If the I/O spec is present for the service , then the default is “false”. Setting this field to “true” would override the spec to allow the service to refer to all the fields in the input. However if the spec is not present , then this field defaults to “true”
allowAllOutput	boolean	no	depends	If the I/O spec is present for the service, then the default is “false”. Setting this field to “true” would override the spec to allow the service to output all the fields. However if the spec is not present , then this field defaults to “true”
hasUserBasedSecurity	boolean	No	false	Is the access to this service restricted based on user type or is it allowed for all logged-in users?
dataAccessType	string	No	none	Does this service require data base access? : none : no database access required. `readOnly` : access is required, but no updates. `readWrite:` data base is going to be updated. A commitment control is implemented for this service. That is either all the updates for this service go thru, or none. In case of any error in the middle of the service, updates are rolled back to the state before the start of this service.
fullyQualifiedClassName	string	no	-	By default, a service is implemented using a service.xml file. However, if this service is implemented using java/c# class, specify the class name here. Note that this class has to implement ServiceInterface.
submitAsBackgroundProcess	boolean	no	false	Should this service be submitted in the background, and return to the caller immediately, without waiting for the completion of the service? If this is set to true, a job id is generated and returned to the client in a field

Name	Type	reqd?	Default	Explanation
				named "_backgroundJobId". Result of this service is cached on the server, and can be fetched by the client with the help of a special service named "getBackgroundJobResult". Client can call this service for the job id, and the resultant dc is returned to the client, if the job is done, or the current status of the job if it is not done. Refer to FAQ on "how to run a service in the background".
toBeCached	Boolean	no	false	Should the result of this service be cached and reused for subsequent calls? Such an arrangement be very useful to manage relatively static data like org structures. Data is likely to be changed only few times a year.
specName	String	no	name	In case you a common spec can be re-used across services, we can use his attribute to over-ride the name of service spec to be used for this service.

Advanced types of service entries have been designed to take care of some special needs. You may want to skip these on your first pass, and revisit these after understanding services. We are documenting them here for logical grouping purpose.

Task Entry

Suppose you have a service that has exactly one step in that. And that step is a task step. You can use taskEntry as a short-cut creating such a service.xml file. Task entry allows you to specify the task right inside the service entry file.

This concept is obsolete, and we discourage use of this short-cut, but provide this feature only for backward compatibility.

Batch Entry

Transaction processing systems have a notion of a "batch job". A batch job typically executes a predefined transaction for a set of entities. Payroll processing is a good example. Payroll is processed for a list of employees. In this case, payroll processing is the process that is executed.

Traditionally, list of entities (like employee list) is supplied as a "batch-file". In our approach, we name grid as "inputGridName". This grid has one row for each entity to be run in batch. inputServiceName is the service you designate that, when executed, creates this

grid. If generation of this grid is a simple sql task, then you can use inputSqlName instead of inputServiceName.

You name your main service that is to be executed for each row as “batchServiceName”. This service is called after copying a row from the grid into values collection of the dc.

You may have to do some winding-up activity after the batch service has executed. This is indicated as “logServiceName”.

Here are the additional attributes of a batchEntry.

Name	Type	reqd?	Default	Explanation
inputGridName	String	yes	-	Name of the service, as referred by the client would be moduleName.name where modukeName is the name of the file in which this entry is made. By default, this is also the name of the service that this entry refers to.
inputServiceName	String	no	-	In case the service name is different from entry name. Note that it is possible to a mix-and-match between the service names used by the client and the ones used by the server, but this will be quite confusing. We recommend that you resort to this only if there is some other issue in using the same name.
inputSqlName	String	no	-	
batchServiceName	string	yes	-	
logServiceName	string	no	-	

Group Entry

Group entry allows you to expose a service name to client, but actually execute a series of service entries on the server. It is to be noted that each entry is executed in its own transaction. One of the common uses of this approach is to separate a long read-only service from a small update service. Let us say we have to generate a fairly complex report that requires reads from several tables and several thousand rows. After generating the report, we have to update one table to log the fact that the report was generated. By default, you would write a service with several steps, with the last step that updates the table. Since the transaction control is at the service level, the whole service is executed under transaction control, and hence the RDBMS will end up locking thousands of rows. (We will explain later as to why we have this rule of one-service-one-transaction). To avoid this, we split the service into two services.

First one is a read-only service that does not require commitment control. Hence the report will be much faster.

Auto Entry

This is not a type of entry, but a feature for the project to use qualified service names as service-entry without explicitly making an entry in the list. Whenever a requested entry is not found in the service list, Exility looks for a service that matches this name. If such a service is found, it creates a temporary entry for this entry, simulating a default entry made for this service. This feature is deprecated, and projects MUST add service entries instead of using this feature. This is to ensure that the layering architecture is not violated and we retain flexibility for deploying the application in a variety of ways to suite performance needs. (like a load-balanced farm of servers behind a single web server)

Service Spec

Service Spec concept is being phased out. If you are going to work on a new project, you need not go through this.

Service spec specifies the parameters (data elements) that are required as input to the service, as well as the data elements that the service outputs. While this is optional, it is highly recommended that you specify these, so that the service is well documented. This also helps in generating stubs and default panels for test automation.

Service specs are stored under spec folder in your resource folder. Name of the file is same as the service name. Note that you may have a module/folder structure, in which case, the file stored under its sub-folder. Also this service is referred with its fully qualified name as module.serviceName in serviceList.

We first describe some basic concepts are terms required to define service spec.

Field Spec

A filed is an instance of a data element being used for some purpose. When you expect the client to send data, we call them inout fields. .Purpose of a field spec is to validate the value that the client is sending. Field attributes are very similar to validation attributes of a field in a page. This is by design, as the purpose is similar.

Name	Type	reqd?	Default	Explanation
name	string	yes	-	Name to the data element
description	string	no	-	Always helps to document your design!!
dataElementName	string	no	name	entry in the data dictionary to which this data element refers to. Note that this has to be groupName_elementName if your project follows a standard where element names are not unique across groups. If this element is not found in data dictionary, a warning is generated, but Exility assumes this to be a text element. It is highly recommended that you avoid this warning. Exility has deprecated this feature that would require you to add than referring to another element.
label	string	no	label from dictionary	label is used in test automation and panel generation. It defaults to the label field in data dictionary.
isOptional	bool	no	false	is a non-blank value expected in this field? Attribute name is coined in

Name	Type	reqd?	Default	Explanation
				such a way that the default (false) means that the value is required.
isFilterField	boolean	no	false	filter field is a special field. Refer to client side manual for its use.
basedOnField	string	no	-	Is this field relevant only if another field is given? For example, spouseName field may be mandatory, but only if isMarried is specified. Specify the field name based on which this field may become mandatory.
basedOnFieldValue	string	No	-	If this field is mandatory based on some specific value of another field, but not just based on whether the field is specified or not. In the above example, you specify “1” in this field, so that the spouseName is mandatory ONLY if isMarried has a value of 1.
fromField	string	no	-	Is this field part of a pair of from-to field? If so, specify the name of the from-field for this field. This field is assumed to be the to-field for that field.
descServiceId	string	no	-	Refer to page (client side) for more explanation on the concept of description service. Idea of specifying here is to repeat the validation on the service, to make application hack-proof, and more secure, without having to write custom code for the same.
descFields	string	no	-	Commas separated list of fields that are returned by the above service that should be pushed into input data. Once again, this facility is to avoid accepting fields that are sent from client, that could be subjected to manipulation. Instead of accepting whatever comes, we get them again from server.
isUniqueField	boolean	no	-	This attribute is relevant only if descServiceId is specified. If the value in this field is expected to be a new key, for example a new userId, then set this attribute true. The way this works is that Exility first executes the descServiceId. If a row is fetched, and this attribute is set to true, then it generates an error.
validValues	string	no	-	If this field has to have a set of design time-known values, this is the best way to carry out that validation. Specify a comma separated list of values.

List Spec

Exility allows list (array) as a data structure. But, this feature is found to be not so useful. Since no project used this feature, we have made this obsolete. A grid/table with one row is always a suitable substitute to list..

Grid Spec

Grid spec specifies validation needs of a grid/table of data coming from client, as well as going back to client. A grid/table is like a sheet in your spread sheet : first row is a header and subsequent rows have data. Alternately, a grid can be viewed as an ordered collection of columns. Each column in the grid has exactly same number of elements. Each element in the column is of same data type.

Grid attributes

Name	Type	reqd?	Default	Explanation
name	string	yes		Name of the grid
isOptional	isOptional	no	false	Is is OK if client does not send this grid at all?
minRows	int	no	0	Minimum number of rows expected in this grid.
maxRows	int	no	0	Maximum number of rows expected in this grid

Grid spec will have a list of field spec, each field spec representing a column in the grid.

Data Collection

Exility uses a generic data structure that carries data across layers, as well as between different components within a layer. This data structure is called 'data collection', or dc for short. Exility uses this as a 'carrier' and not input and output. By that what we mean is that the components in different layers receive dc, they pick their inputs from dc, and put their output onto it. A data collection has the following:

- Fields : collection of name-value pairs. If dc is carrying details of a customer, you will probably find customerName="exilant" etc.. in this collection.
- Grids: dc has name-grid pairs that carry rows of data. For example, getOrderHistory service may put a grid named orderHistory in dc that has one row for each order. Internally, Exility models a grid as a collection of columns, and does not attach any specific meaning to order of columns. With the result, whenever a table on the client is bound to a grid in dc, order of the column need not match. Exility goes by name of the column (header) rather than the order.
- Messages: messages are accumulated into dc during service execution.
- Lists : list is an array of data. We noticed that no one used this for several years, and hence we have deprecated this feature. A list can always be simulated as a grid with a single column.

If you have to write custom java code, understanding dc, and its parts, is critical. Refer to technical details in java-docs before you write your first line of java code.

Approach to Implementing Service

You need to retrieve data from data base, and save new or modified values into database. (manage persistence).
Three broad categories of work you do on server are:

1. Data persistence: retrieve required data from data base, and save new and modified data.
2. Implement business logic : Detect conditions and compute new values based on known values. (This is quite simplistic view of business logic, but is sufficient most of our time)

Infrastructural requirements (non-functional requirements like security, performance, logging, notifications etc..)Task is a term used by Exility to denote a basic unit of work on the server. A service is typically implemented with the help of one or more tasks.

Task is a term used by Exility to denote a basic unit of work on the server. A service is typically implemented with the help of one or more tasks.

Data update, data insert, data retrieval and data manipulation are the most common tasks carried out in application software. Exility caters to these common needs with some ready-to-use components. Tasks that cannot be done using these components, or are too complex to implement using these components require custom java code to be written.

Tasks are designed with some possible run-time parameters to improve their reusability across services. While gridName is a common parameter that is made available to all tasks, an array of text(string) values is made available as a means of passing any other set of values that are relevant for the task.

Exility deals with RDBMS as of now. Utilities are provided to interface with an RDBMS for manipulation and retrieval of data.

Expression

We have borrowed this word from programming paradigm. While implementing any algorithm or logic, you need to check for conditions and compute new values from known values. Expression is the basic building block of any calculation. For example, amount is calculated as quantity times price. In this case,
orderQuantity * orderPrice
is an expression. Following are some more examples
a + b
principal + (interestRate / 100 * year)
discountAmount <= authorizationLimit

An expression consists of variables and constants connected with operators, and possibly grouped with parathesis.

Points to be noted about an Exility expression:

1. A variable is any parameter that is available in the dc at the time of execution. Note that Exility is a type-safe environment. That is, every parameter has design-time determined basic data type (one of integral, decimal, text, Boolean or date)
2. Valid number is a numeric constant. Number with a decimal point is considered a decimal constant, otherwise an integral constant. Text (string) constants are to be enclosed in double quotes. Since the entire expression is generally expressed as an attribute of an xml tag, you will have to use the escape sequence " to represent a quote. For example <dummyStep executeOnCondition="userOption = "a"" …. >. A date constant is to be enclosed in single quotes (like '2008-02-24 23:11:49'). Note that date format on server is fixed as yyyy-mm-dd hh:mm:ss.fff, width time and fraction being optional. true and false, with no quotes around them, represent Boolean constants.
3. Arithmetic operands supported are +, -, *, /, and ^. Note that these are appropriately carried out for integral and decimal expressions.
4. A text field can have + as an operand for appending. Note that either the left or the right operand can be text field.
5. A date + integer is valid, and the resultant is a date. Similarly, date – integer is a date, and date – date is an integer.
6. – as a unary operator has the normal arithmetic meaning.
7. ! is negation for a Boolean operand.
8. ? is a special unary operator to check existence. Note that a variable is considered to exist if it is found in the collection being considered or if it is NULL. That is, for a variable to be considered as 'exists' it should be a non-null value.
9. | and & are logical operators. Note that these are not bit operators, (you may be used to treating them that way in your

java/c/c++/c# languages) Since we do not have bit operations, we have simplified the notation to | rather than ||. Note that the & characters has to be escaped in xml as &

10. Exility generates an error if a variable in the expression does not exist in the collection (normally dc) when the expression is executed, except if that variable has a unary "? " operator. Most programming paradigms optimize "and" and "or" operators. In an expression like "a and b" if a is false, b is not even evaluated, because the expression is false irrespective of what b would be. Exility DOES NOT do this optimization at this time. And this is important, because, in the above case, if b is not a valid variable, Exility generates error.

You can create new variables, or change value of a variable using expressions.

Data Collection – DC

We have written this for people with programming background. We will be rewriting this later, but meanwhile, if you do not understand this, call us and we will be glad to explain this in your language!.

All modern programming environments allow you to create arbitrary data structures. Object oriented languages allow you to pack the data structure and its associated functionalities nicely into object. In Exility, we have created a simple data structure called "Data Collection", "DC" in short.

- DC has a collection of name-value pairs. This collection is called "values". That is, "values" is a collection of variables or fields. cusotmerName, orderAmount are possible members in this collection.
- Any name used in your application, must be defined in the data dictionary. This is the best way to document all fields used in the application, and ensure that they are used consistently across pages and modules. Once you define a field in data dictionary, its value-type is known to Exility, and it carries out operations with this connotations.
- Allows tabular data to be stored. This is called "grid" or "table".
- A grid similar to a sheet in your spread-sheet. It has columns of data. Each row of the grid has exactly the same number of columns, in the same order. That is if you have 10 columns and 300 rows, there will be 10 * 300 cells of data.
- Name of the columns must be defined in data dictionary, from where we know the type of value this column has.
- DC collects all messages, errors, warnings and information, during the execution of a service.

DC is the only data structure we support in Exility. All the business logic and algorithm need to be implemented using DC as a data structure. This looks like a huge restriction in the beginning. When the world has moved well into the world of objected based algorithm, why is that we are still talking about this generic data structure, but not allow arbitrary data structure. Well, the answer is

simple and straight forward. Since Tabular data is extensively used in business applications. Most business functions are still managed with spread sheets. Users have no difficulty in modelling all their data needs into spread sheets.

Programming community who have benefitted with Object Oriented Design would certainly argue for such an approach in application software. We certainly agree that there will be scenarios where a good object oriented code will be required to implement complex business logic. Exility provides seamless integration of its built-in utilities with the custom code that may have to be written for implementing such business logic in an application. While DC is used as a common carried of data across different layers of the application, a custom Java code can easily have its own data structure to implement its logic.
We are using this DC also as a common data structure between client and server. DC is a data carrier as well. Client sends data to server in the form of a DC, and the server responds back with a DC.

Table

We keep repeating, like a good parrot, that Exility's target applications deal with data bases. Few years back, database was almost synonymous with RDBMS. Today, there is lot of talk (and a little bit of usage as well) about no-sqls. We use the term "table" to logically model data into rows and columns. Like a spreadsheet. This is the perspective of the application designer. It is entirely up to the data base designer, and the supporting tools, to map this logical view into whatever form the data is physically managed.

Exility very clearly separates application design from data base design. Design concepts are kept fairly flexible to handle different persistence strategies. However, every single use of Exility has been with RDBMS so far. Hence, you may expect that the underlying utilities are well tuned to handle RDBMS, while it remains at conceptual level for possible other models. If any project wants to use a non-RDBMS database, we will be eager to work with them and enhance Exility to take care of their needs.

Having put that on record, we continue with the rest of this document as if RDBMS is THE persistence strategy.
Exility "table" represents an RDBMS table or a view. It can also be used to represent any row of data that you want to model for your application. In its simplest form, we define all the columns in the table. Note that you should have added each of these column names into data dictionary.

There are some common design practices while using RDBMS Exility supports the following design practices.

1. One-up number as an internal primary key. For example, one may have customerCode, a text field as a primary key as seen by users for customer table. However, it will be fairly expensive in terms of indexing if customer code is used as foreign key in many tables. In such a case, a one-up number is used as an internal key. This value is never shown to the user, but used internally for efficiency sake. Different RDBM systems provide tools/verbs to take care of this. Exility uses a common approach. It stores the last key used in a common table. (exilKeys)

   Whenever you manipulate such a table outside of the application (say data-upload, restoring data from a back-up, or manual entry of data directly into database), the last key maintained by Exility will be out-of-synch. You should manually go and reset this column in exilKeys table (Utility will be provided in the future to manage this through a page)

   When a new transaction is created (say order entry), the new key generated for the header is used as foreign key for the child

rows. Exility supplies the generated key automatically to the next step for this purpose.

2. Running sequence numbers with prefix based on other fields. For example an invoice may have 'invoiceNumber" as a running sequence number by financial year, say 2008001 etc.. Or it could be by department. Exility allows you to specify such fields, and the total width of the key, so that the generated key can be 0 filled on the left.

3. timestamping. It is a common practice to track created/modified time as well as created/modified users. Exility will maintain these fields automatically for you, once you mark such columns.

4. Concurrency: In a multi-user system, system has to be designed to avoid concurrent update by more than one user. For example, user 1 asks a row to be modified. She changes 'authorizedAmount' from 100 to 20. Meanwhile, before this user has pressed submit button, another user asks for the same row for update. He would see 'authorizedAmount' as 100. Based on this he takes some decision, and changes another field. First user presses submit now. Second user submits after words. The database is in an invalid state now, because second use has updated it based on 'stale' data. Time-stamping is a common technique used here. In such a technique, a time-stamp (usually system-time) is set whenever the row is updated, and it is guaranteed that the time-stamp is always different. Before updating a row, value of time-stamp in the data base is compared with the one that has come back for update. If they differ, then it means that someone else has updated the row, and hence the update is rejected and sent back to the user.

   Exility implements such a technique automatically if you specify modified-time field.

5. audit log: Audit trails may be required for some tables, to trace changes if required. Exility can do this job for you automatically. An insert/delete log is created at the row level. On update, logs are generated only for the columns that were updated with different value.

With the introduction of Record design, you can use a recordName attribute instead of tableName in tasks. You need not create table.xml. Exility uses record.xml instead of table.xml, but the concept explained above is in tact.

Table Attributes

Name	Type	reqd?	Default	Explanation

Name	Type	reqd?	Default	Explanation
name	string	yes	-	Name of this definition. Generally same as table name, except when you need to write more than one definitions for a given table. Also, your naming convention in application and for data base may not be compatible. In that case, use name as the one to be used in the application, and tableName is the one used in database.
tableName	string	no	name	Name of the table in the database.
module	string	no	-	Module name. For documentation only.
Description	string	no	-	description
okToDelete	Bool	no	false	Can rows be deleted from this table? Most RDBMS designers do not allow deletion of rows of most of the table in a transaction, They are typically deactivated, or set to some status. However, some transaction tables may allow deletion of rows. If this value is not set to true, Exility will generate an error if any serviceStep uses a deleteTask based on this table
keyToBeGenerated	Bool	no	false	is the primary key an internal one-up-number, to be generated by Exility.
keyPefixFieldNames	string	no	-	If sequence number is to be based on other fields. (Refer discussion above)
keyColumnWidth	number	no	-	One of our customer was using a one-up number, but the field was used by end-users also, and they wanted it to zero-fill on the left t make it a fixed width text field. We do not approve of this design, but we had to deliver such functionality. This can be used in a meaningful way when you use key prefix. Field is of text type, and hence it makes sense to make it fixed-width. Exility will zero-fill the keys before appending it to the prefix.
createdTimeStampName	Text	no	-	Column name that tracks the created time for the row. Once you specify the column name here, you do not include it in the list of columns.

Name	Type	reqd?	Default	Explanation
modifiedTimeStampName	text	no	-	Column name that tracks the time at which the row is last modified. Also note that this field automatically triggers a time-stamp check for concurrency of updates.
createdUserName	Text	no	-	Exility tracks the logged-in user id, and updates this field automatically
modifedUserName	Text	no	-	changed whenever the row is updated
createdTimeStampColumnName	Text	no	-	In case the db column name is different from name.
modifiedTimeStampColumnName	Text	no	-	In case the db column name is different from name.
createdUserColumnName	Text	no	-	In case the db column name is different from name.
modifedUserColumnName	Text	no	-	In case the db column name is different from name.
parentTableName	Text	no	-	for documentation only. Not used as of now, but we have plans to use this in the future.
writeAuditLog	Bool	no	false	Automatically write insert/update/delete audit log for this table.
sqlName	Text	no	-	This is the name of a sql template. This is an alternate to using sql template directly. You should ensure that the columns you specify are in-line with the selected fields in this sql. In other words, you use this feature instead of specifying output parameters. We plan to use table definitions to validate page designs and service list validations. That is the advantage of using this approach over using sql as it is.

Column Attributes

Name	Type	reqd?	Default	Explanation
Name	string	yes	-	Name of this column as used in your application. Generally same as name of column in your RDBMS table. You may have issues with

Name	Type	reqd?	Default	Explanation
				the naming convention used in the RDBMS, in which case you may choose the column name to be different. See next attribute.
columnName	string	no	name	name of the column in the RDBMS table. Use this ONLY IF it is different from name. You should follow a good naming convention in your project to ensure that application developers do not get confused. For example, SQL standard for RDBMS has an issue with case sensitivity. It requires the RDBMS to keep all names in upper case and be case insensitive when it comes to entity names. It also requires users to use double quote mark (“) in case users want to use case sensitive naming. We certainly do not approve of all caps names. In this case, you may use camelCase as naming standard in your application, and use corresponding all_lower_case standard for your RDBMS entities. That is, you set name=”customerCode” and columnName=”customer_code”. In your application, you use customerCode while your stored procedures and sqls use customer_code.
Description	string	no	-	Description
isKeyColumn	bool	no	false	Though we recommend that a table should have internal primary key, it is possible that you may have a combination of columns as primary key. To facilitate such a design, we allow you to mark columns as keys, rather than asking for keyColumnName as a table level attribute.
isOptional	bool	no	false	Is it okay if the client does not provide value for this column while inserting a new row? If this is true, then you should either provide a default value, or the column is nullable. We strongly recommend that you live with defaults rather than make the field nullable. Most programmers make mistakes while handling nulls.
basedOnColumnName	string	no	-	Sometimes, one column depends on the other. Let us say emailed is

Name	Type	reqd?	Default	Explanation
				required but only if user has opted for mail as communication channel. In this case you would have three columns in the table, communicaitonChannel which can take value “mail” and “phone”. You will have mailed and phoneNumber as other two fields. For mailed, you set basedOnColumnName=”communicationChannel” and basedOnColumnValue=”mail”. Similarly, for phoneNumber you set basedOnColumnName=”communicationChannel” and basedOnColumnValue=”phone”. If this column is required so long the other column has some value, then you skip setting basedOnColumnValue.
basedOnColumnValue	string	no	-	Refer basedOnColumnName.
defaultValue	string	no	-	Value to be used while inserting a new row if client does not supply value for this column. You should have set isOptional=”true” for this default value to be used.
otherColumnName	string	no	-	If you have two columns in your table, but you expect value in only one of them, then use this feature. This implies that your design Take the example we used for basedOnColumnName. Suppose the business requirement is same, but you decided to design it differently. You skip communcationChannel column. You design mailed and phoneNumber as two columns, but you assume that the user will specify one of them (client page design ensures that user either enters phone number, or mail id, but not both). Your communication channel is decided accordingly. Use otherColumnName=”phoneNumber” for mailId and otherColumnName=”mailId” for phoneNumber.
label	string	no	-	Defaults to label Specified in data dictionary.
dataSource	text	no	-	This is one of the most confusing and misunderstood (and hence misused) attributes. So please read carefully. Suppose you are doing a bulk-insert using this table. By default,

Name	Type	reqd?	Default	Explanation
				Exility picks-up values for each column from each row of the underlying grid. However, it is possible that you have a foreign-key in this table, and the value for that column is not repeated in each of the rows. Its value is available as a field inside dc.values (collection of name-value pairs) In such a case, use the name of that field as dataSource for this column. DO NOT use this attribute if you are not using a grid as a source of data. DO NOT use this to use get value for this column from another field. You should copy the value to the desired column using appropriate service step before calling this table-task, rather than using dataSource for this column. When Exility looks for value for a column from a DataCollection (that is the common data structure that carries all data within the application across different execution units) it d name that tracks the created time for the row. Once you specify the column name here, you do not include it in the list of columns.
listServiceName	text	no	-	Used for generating test data for this table. List service is executed to get possible set of values for this column.
enumeration	text	no	-	Comma separated list of values to be used as possible set of values, in the absence of listService for test data generation.
enumerationType	text	no	-	Fully qualified class name of the java enum class that defines possible values for this columns. Relevant only if listService and enumeration are not specified.

SQL Template

Sql Template, sql in short, is a utility to create a SQL statement at run time to be used to extract or update data. Approach is very similar to prepared statement, but we have extended the concept a bit to provide more flexibility.

To understand this utility, it is better to start with an example. Suppose you have to extract all details of a customer from customer for the specific customer say 'exilant'. You would have written a sql

```
select a,b,c... from customer_master where customer_name = 'exilant'
```

sql template for the same is

```
select a,b,c... from customer_master where customer_name = @1
```

As you can see, this is fairly straight forward. You specify one input parameter for this template. Exility takes the value of this parameter at run time, creates the sql at run time, fires it, and extracts data into dc.

In its simplest form, you can have any number of input parameters defined for a sql, and you can use them in the text of the sql with this notation of "@n". In this form, it is a simple string-substitution utility.

One common use of dynamic sql is to filter rows as per user request. User can give one or more filtering criteria, each criterion being a field as compared to a value. That comparison could be equal, not equal, greater than etc...

Sql template has specific features to handle such scenarios.
You can declare a parameter as "optional". Take the following sql

```
select a,b,..... from order where financial_year = @1 and customer_name like '@2%' and
orderValue < @3
```

And you allow user option of not specifying customer_name or OrderValue. At run time, the actial sql could be

```
select a,b, ... from order where financial_year = 2012 and orderValue < 10000
```

or it could be

```
select a,b, ... from order where financial_year = 2012 and customer_name like 'ex%1'
```

or

```
select a,b, ... from order where financial_year = 2012
```

given, we need to chop-off some part of the sql. You specify this chunk using a pair of flower brackets. Like

```
select a,b,..... from order where financial_year = @1 {and customer_name like '@2%' } {
and orderValue < @3}
```

You make a parameter optional, and provide a pair of braces around that parameter in the sql text. Note that it is a syntax error if you do not provide the braces.

What if, in the above example, even the financial_year is optional? We may have to produce a sql that may be as simple as
`select a,b,..... from order`

In this case, in case none of the parameters is specified, we may have to knock-out the verb "where" as well. For this, we allow you to put an extra pair of braces around the possible remainder to be knocked. The sql template would look like

```
select a,b,..... from order {where 1 = 1 } {and financial_year = @1} {and customer_name
like '@2%' } { and orderValue < @3}
```

We threw a surprise at you with that "1 = 1". That is a simple trick to ensure that each condition has and before that. (All rdbms brands are smart enough to ignore a clause like 1 == 1)

We still need to figure out what to do if the comparator also can change at run time. Actual sql could be

```
select a,b,..... from order where and orderValue between (10000, 20000)
```

or

```
select a,b,..... from order where and orderValue > 50000
```

That is the “condition” itself is decided at run time. We provide a feature called “searchField” as type of parameter. The sql template would look like

```
select a,b,..... from order {where 1 = 1 } {and @1} {and  @2 } { and @3}
```

Parameters 1,2,3 are all marked as “searchField”. Exility has a complimentary feature on the client side that allows user to select the comparator as well as value. Exility client and Exility server work in tandem to send appropriate set of information beneath the carpet and make this work for you.

With all this, you still remember that sql template is still a string-substitution utility, and has no intelligence about sql syntax. For example you can use a sql template to decide the table name at run time. Like

```
select a,b,c... from @1 where code = @2.
```

Here is the summary of features and working of sql template

1. You mark a place holder inside your sql with @1 etc. @1 is the place holder for first parameter, @2 is for the second and so on.
2. You may use a given parameter more than once in a sql. That is it is OK to have @1 occur more than once in the sql.
3. If you miss out a place holder, it is not a syntactic error. You will not get any warning or error. The corresponding parameter is just ignored.
4. A parameter is marked as either optional or mandatory.
5. When a parameter is optional, every occurrence of that parameter must be preceded with a ‘{‘ and followed with a ‘}’. If the parameter is supplied at run time, it @n is replaced with that value, and the braces are just removed. If the parameter is not supplied at run time, the entire text between the braces, (including the braces) is removed.
6. An error is generated, and the sql is not used, if a mandatory parameter is not supplied at run time.
7. At the time of substituting the value for a parameter, the value is ‘suitably’ formatted for the desired RDBMS. For example if

first parameter is date/time field, then @1 is replaced with something like '2008-02-21 22:01:43.123' for sql server, but it is formatted as TODATE("2008-02-21 22:01:43.123", "yyyy-mm-dd hh:mm:ss.fff") for oracle. (This setting is taken care of by your project leader in applicationParameters.xml file)

8. If you want to use the parameter for a purpose other than for a where-clause value, you may not want the value to be formatted. Use doNotFormat attribute for the parameter.
9. Three special purpose parameters are supplied. A filter-field takes care of formatting the entire comparator clause. For example if customerType is a filter field, then in the sql you specify ….. Where…. customerType @1. Exility takes care of putting the write comparator and the value for it based on what the user has chosen in the client.
10. A list parameter is used to cater to the "in" clause of sql. If you want to use something like … where customerType in ('val1', 'val2',…) use list as the type of parameter. Value for such a parameter could be either a regular text field in which you can have comma separated values, or a list, or a grid with this field as a column.
11. A combinedField takes care of a possible flexible UI, in which users can use the normal input field itself to specify the comparator. (Apple for one uses such a standard). For example, if there is a field in the page called employeeName, user can type "%swamy" to search for names starting with swamy, "%swamy%" for names that contain swamy. For a numeric field, one can type "<2" or ">23" etc..
12. If your naming convention is suitable, or if you use the right 'AS' clause for the fields you select, then you do not need to specify output parameters. Exility uses the run-time environment to find out the names of the output fields, as well as their data type. It also optimizes by taking this only the first time, and uses it for subsequent executions.

Sqls are stored under sql folder. If your project uses modules, then you may have sub-folders under which individual sqls are stored. In such a case a sql has to be prefixed with its module when it is referred in tasks.

SQL attributes

Name	Type	reqd?	Default	Explanation
Name	string	yes	-	must match the name of the file in which the sql is stored. For example if you have sql1.xml is the file name under a sub-folder modul1, then this name attribute should be set to "sql1" and it should be referred elsewhere as module1.sql1.
Module	text	no	-	Module name if you use modules in your project. You should ensure

Name	Type	reqd?	Default	Explanation
				that the module name matches the sub-folder under which you save this.
description	text	no	-	For documentation
toBeExecuted	bool	no	false	Is this a sql that extracts data or one that manipulates data. Default is that it is an extractor. If this is set to false, then it is assumed that it is a "SELECT" type of sql. Else it is of other types like "UPDATE", "INSERT" , "DELETE" etc.. If you set this to true by mistake for a select query, you will not get any error at run time, but you will not see any data being extracted.
minParameters	number	no	-	Do you need a min number of input parameters to be supplied at run time for the call to be valid? For example you would have designed your sql with the assumption that at least one of the where-clause parameter is supplied at run time. Note that if a specific parameter is required, you set that as part of that parameter. This attribute is for the total number of parameters being supplied.
sql	string	yes	-	The text of sql. Alternately, you can use a CDATA section with sql as its tag. This option allows you to put a sql as it is, without having to escape special characters. This is ignored if sqlType is set to "storedProcedure". In case you are using sqlType="preparedStatement", you have to ensure that the number of "?" match with the number as well as sequence of input parameters.
sqlType	text	no	sql	We extended the original design to allow prepared statement and stored procedure.
storedProcedureName	text	no	-	If this is a stored procedure, provide the name here. This field is mandatory when you set sqlType to "storedProcedure", and is ignored if it is set to other values. It is possible to write a sql to execute a stored procedure. However,

Name	Type	reqd?	Default	Explanation
				we strongly recommend specifying stored procedure as it is simple and straight forward.
inputRecordName	string	yes	-	We have made this mandatory for new projects. Existing projects depend on data dictionary to get the parameter types. Note that this record must have all the input parameters listed for this sql. Of course it can contain other fields as well, but they have no relevance for this sql.
outputRecordName	string	yes	-	Mandatory for new projects. Output parameters can be omitted if it has all the fields in the same order as in this record. You have to ensure that the output fields are aligned with the fields in this table

inputParameters is a collection of sqlParameter.

SQL Parameter Attributes

Name	Type	reqd?	Default	Explanation
Field Attributes	set	-	-	Field attributes defined earlier.
parameterType	string	-	normal	normal, list, filter and combined, as explained above.
doNotFormat	bool	-	false	By default, values are formatted for the sql based on the data type. For text, value is enclosed in single quote, and required function name is inserted for date for oracle etc.. If you use the parameter in a context other than value, then you do not want quotes. For example if table name itself is determined at run time. This attribute is valid only for text parameter type. Note that, we still escape single quotes with two single quotes, as per sql syntax.
justLeaveMeAlone	bool	no	false	Value of the parameter is inserted as it is. This is a very special case. We DO NOT RECOMMEND this at all, because it will expose the sql to sql-injection security threat from client. Some projects had

Name	Type	reqd?	Default	Explanation
				specific requirements that we catered to using this feature, but it should be generally avoided.
toUpper	bool	-	false	This feature is useful if you want to make an case-insensitive comparison. E.g. toUpper(field1) = @1. Exility converts the text to upper case before inserting it into the sql.
gridName	text	-	-	If the field value is to be taken from a column of a grid, you can specify the name of the grid, In such a case, fieldname is assumed to be the name of the column.
Index	number	-	-	relevant if gridName is specified. If you do not want the value from the first row, you can specify the row number.
basedOnColumnName, basedOnColumnValue	text	-	-	This is an advanced feature. This is applicable if you specify as grid name. If you want to select the row based on matching a column value, then use this feature. For example gridName=grid, basedOnColumnName=col1 basedOnColumnValue=N, fieldname=col2. In this case, col1 in the grid is searched for a value of N. First row that has this value is selected as the row. Value of col2 in this row is used as the field value.

OutputParameters is a collection of parameter(field) attributes. Refer to attributes of "field".

NOTE:
We have observed that programmers write quite complex sqls. In fact, one of the disturbing trends we have seen is that designers and programmers are moving away from writing good stored procedures and java code for processing. Instead they try to push even the logic into complex SQLs.

Here are our guidelines:

1. Use table based tasks as much as possible. Use sql ONLY if table task is not decent.
2. You should use only joins, group-by and where-clause in sql. Injecting pl-sql like code (like using if-then-else or switch etc..) is clear NO-GO.
3. You should try to define a view using the tools that the RDBMS provides, and use this view as a table in your task, rather than

making complex joins in sql tasks.

4. Many a times, we think about one sql that will provide exactly what we want. Consider breaking this into multi-step approach. You may write a simpler sql that selects reasonable number of rows (say few hundreds to few thousands). Use this sql to get data into a grid. Then you can use elegement Java code to process this further, or use grid processor utilities to filter it further.

Grid Processors

Grid or table is a common data structure that is used to carry data back and forth between the rdbms and UI. Exility provides some useful utilities to manipulate a grid.

Grid, on the server side, is a collection of "columns". A column has a name and carries a type-specific list of values. (Remember that grid has a more simplistic view on the client. It is viewed as rows and columns, first row being the label and the rest of the rows carry data)

For most part, order of columns is not important. All columns are of same length.

Grid processors are quite handy when you have to manipulate table. Many a times we have seen that a complex sql that a programmer has written can be re-factored into simple sql and a few grid processors.

Most programmers try to get the final result in one sql. We encourage you to think in terms of splitting this into steps, each of which is either interacting with db (like fetching rows) or modifying rows.

For example, suppose you have to insert rows into a table based on existing rows in another table, and some business-logic is to be applied to filter-out some of the rows. A typical programmer writes an insert-sql with an inner query that selects rows. Instead, we recommend that you first write a simple sql to extract rows into grid, use grid processors to manipulate data, possibly your own java code to manipulate, and prepare the final set of rows that can be inserted using a simple table-insert task.

We used to have a concept called "local services". These were services that ran completely on the client. And then we tried to give full compatibility between client and server services. Some of the utilities were similar, but not same. We ended up with a list of grid processors that are confusing. We have now cleaned-up this mess and made some utilities and attributes "obsolete". Please ensure that you do not use the obsolete ones in your project, but you can retain the one that are already working fine.

Copy Grid

Make a copy of a grid. This is a clone, and not calling the same table by two different names.. After copying grid1 to grid2, if you change grid1, it is not reflected in grid2. Attributes are fromGridName and toGridName.

Optionally, you can specify copying only a subset of columns. Specify list of columns to be copied as a comma separated value into the attribute “fromColumns”. The names of these columns in the new grid can also be specified optionally, using toColumnNames.

You can also copy subset of rows using actionFieldName attribute. However, we recommend using filter-utility rather than this attribute.

Name	Type	reqd?	Default	Explanation
fromGridName	string	yes	-	Name of the grid to copy from.
toGridName	string	yes	-	Name of the grid to copy to. Could be same as fromGridName.
fromColumns	string	no	-	List of columns to be copied using from-grid as a comma separated value.
toColumns	string	no	-	Specify new or same Column names for to-grid as a comma separated value. Defaults to fromColumns
actionFieldName	string	no	-	Do not use this feature.

Example :

```
<gridProcessorStep>
   <processor>
      <copyGrid fromGridName="indianFlowers"
          toGridName="flowers"
          fromColumns="flowerId,flower"
          toColumns="id,flower"
      />
   </processor>
</gridProcessorStep>
```

Before copy grid operation.

indianFlowers

flowered	flower	state
1	Water lily	Andhra Pradesh
2	Lady's Slipper	Arunachal Pradesh
3	Foxtail Orchids	Assam
4	lotus	Haryana

After copy grid operation.

indianFlowers

flowerId	flower	state
1	Water lily	Andhra Pradesh
2	Lady's Slipper	Arunachal Pradesh
3	Foxtail Orchids	Assam
4	lotus	Haryana

flowers

id	flower
1	Water lily
2	Lady's Slipper
3	Foxtail Orchids
4	lotus

Rename Grid

Just changes the name of the grid. Attributes are fromGridName and toGridName.

```
<gridProcessorStep>
   <processor>
      <renameGrid fromGridName="indianFlowers" toGridName="flowers"/>
   </processor>
</gridProcessorStep>
```

Remove Grid

Removes a grid. Attribute is gridName.

```
<gridProcessorStep>
   <processor>
      <removeGrid gridName="indianFlowers"/>
   </processor>
</gridProcessorStep>
```

Filter Grid

Applies a filter, and possibly remove some rows from the grid. This is obsolete. Use filter rows utility instead.

Purge Grid

Deletes data, but retains the structure. After this operation, grid exists, but with no data in it.

```
<gridProcessorStep>
   <processor>
      <purgeGrid gridName="indianFlowers"/>
   </processor>
</gridProcessorStep>
```

Before purge operation	After purge operation
indianFlowers	indianFlowers

flowerId	flower	state
1	Water lily	Andhra Pradesh
2	Lady's Slipper	Arunachal Pradesh
3	Foxtail Orchids	Assam
4	lotus	Haryana

flowerId	flower	state

Grid To Values

A specific row of a grid is copied to dc with the column name as their name. Common use is when you have only one row that is to be used as values instead of a row of a grid.

Name	Type	reqd?	Default	Explanation
gridName	string	yes	-	Name of the grid
removeGrid	bool	no	false	should the grid be removed/deleted after copying the column
index	number	no	0	zero based row number to be copied.
prefix	string	no	-	prefix to be attached to each column name while copying them to dc. For example if id is the column name, and customer_ is the prefix, id will be copied to dc with the name customer_id.

```
<gridProcessorStep>
<processor>
<gridToValues gridName="indianFlowers" index="1" prefix="india_" removeGrid="true"/>
</processor>
</gridProcessorStep>
```

indianFlowers	Values:
indianFlowers **flowerId** / **flower** / **state** 1 / Water lily / Andhra Pradesh 2 / Lady's Slipper / Arunachal Pradesh 3 / Foxtail Orchids / Assam 4 / lotus / Haryana	india_state: Arunachal Pradesh india_flower : Lady's Slipper india_flowerId: 2

Add Column

A column of specified data type is added to the grid. Value of this column is an expression that is evaluated for each row of data. That is, the expression may contain columns from this grid, as well as fields from dc. Expression is evaluated for each row, and the value is saved into the new column in that row.

Name	Type	reqd?	Default	Explanation
gridName	String	yes	-	Name of the grid
columnName	String	yes	-	Name of the column to be added
columnType	Type	yes	text	one of text, integral, decimal, date or Boolean.
expression	String	no	-	should evaluate to the right type. Same value is used to initialize the entire column.

```
<gridProcessorStep>
   <processor>
      <addColumn gridName="indianFlowers" columnName="rating" columnType="integral" expression="5"/>
   </processor>
</gridProcessorStep>
```

Before add column operation.

indianFlowers

flowerId	flower	state
1	Water lily	Andhra Pradesh
2	Lady's Slipper	Arunachal Pradesh
3	Foxtail Orchids	Assam
4	lotus	Haryana

After add column operation.

indianFlowers

flowerId	flower	state	rating
1	Water lily	Andhra Pradesh	5
2	Lady's Slipper	Arunachal Pradesh	5
3	Foxtail Orchids	Assam	5
4	lotus	Haryana	5

Clone Column

One column of a grid is cloned and added with a new name.

Name	Type	reqd?	Default	Explanation
gridName	string	yes	-	Name of the grid
columnName	string	yes	-	Name of the column to be cloned
newColumnName	string	yes	-	Name of the column to be added

```
<gridProcessorStep>
    <processor>
      <cloneColumn gridName="indianFlowers" columnName="state" newColumnName="cloned_state"/>
    </processor>
</gridProcessorStep>
```

Before clone column operation.

indianFlowers has 5 rows and 3 columns

flowerId	flower	state
1	Water lily	Andhra Pradesh
2	Lady's Slipper	Arunachal Pradesh
3	Foxtail Orchids	Assam
4	lotus	Haryana

After clone column operation.

indianFlowers has 5 rows and 4 columns

flowerId	flower	state	cloned_state
1	Water lily	Andhra Pradesh	Andhra Pradesh
2	Lady's Slipper	Arunachal Pradesh	Arunachal Pradesh
3	Foxtail Orchids	Assam	Assam
4	lotus	Haryana	Haryana

Copy Column

Same as clone column. Obsolete, and should not be used.

Rename Column

Name of a column is changed

Name	Type	reqd?	Default	Explanation
gridName	string	yes	-	Name of the grid
columnName	string	yes	-	Name of the column to be cloned
newColumnName	string	yes	-	Name of the column to be added

Remove Column

Removes a column from the grid

Name	Type	reqd?	Default	Explanation
gridName	string	yes	-	Name of the grid
columnName	string	yes	-	Name of the column

```
<gridProcessorStep>
   <processor>
      <removeColumn gridName="indianFlowers" columnName="state"/>
   </processor>
</gridProcessorStep>
```

Before remove column operation

indianFlowers

flowerId	flower	state
1	Water lily	Andhra Pradesh
2	Lady's Slipper	Arunachal Pradesh
3	Foxtail Orchids	Assam
4	lotus	Haryana

After remove column operation

indianFlowers

flowerId	flower
1	Water lily
2	Lady's Slipper
3	Foxtail Orchids
4	lotus

Merge Columns

This utility helps you in creating one column by merging two or more columns. This is definitely not advisable because it violates basic data handling principles. We generally insist on "atomicity" of a field : one field should carry one piece of information. However, you may have to deal with client applications who may insist on such "combined" data elements. And why would they put such "unreasonable demand" on you? Because a client application's focus is usability, and user convenience, and not data modelling principles. Once you establish that there is indeed a need to have a column that merges two columns, only question you ask is who should merge the columns? Client Application or server application? We will leave that question to be asked and answered appropriately by the design team of the application. We will provide you the flexibility to do it either on the server or on the client.

A product may be sold in different sizes. Size is one field and quantity is another field. Say 20 of size "small". Client may want this information in one field like "s-20". After extracting data into a grid, you merge the two columns into one before sending the grid to client.

Name	Type	reqd?	Default	Explanation
gridName	string	yes	-	Name of the grid
mergedColumnName	string	yes	-	Name of the resultant (new) column. Its data type is text.
columnNamesToMerge	string	yes	-	comma separated names of columns to be merged
fieldSeparator	char	no	'-'	Specify the character used to separate the values if it is other than '-' (hyphen)

```
<gridProcessorStep>
   <processor>
      <mergeColumns gridName="ordrLines" mergedColumnName="quantityBySize"
      columnNamesToMerge="productSize,orderQuantity" fieldSeparator="-"/>
   </processor>
</gridProcessorStep>
```

Before Merging

orderLines

product	productSize	orderQuantity
Shirt	s	20
Shirt	m	30
Kurta	xl	56

After Merging column.

orderLines

product	productSize	orderQuantity	quantityBySize
Shirt	s	20	s-20
Shirt	m	30	m-30
Kurta	xl	56	xl-56

Split Columns

This is the reverse of merge columns. Taking the same example, this utility allows you to create two columns, "size" and "quantity" from a single column that the client may send.

When we create new columns, we refer to data dictionary to decide the data type of the new columns. So, you should ensure that column names are defined in data dictionary.

Name	Type	reqd?	Default	Explanation
gridName	string	yes	-	Name of the grid
columnNameToSplit	string	yes	-	Name of the column to be split
columnNamesToSplitInto	parameters	yes	-	list of column names to split into. These column names must be found in data dictionary, We pick-up their data types from data dictionary.
fieldSeparator	char	No	'-'	Specify the character used to separate the values if it is other than '-' (hyphen)

Before split Columns operation.

ordeLine

product	quantityBySize
Shirt	s-20
Shirt	m-30
Kurta	xl-56

After split Columns operation.

orderLine

product	quantityBySize	productSize	orderQuantity
Shirt	s-20	s	20
Shirt	m-30	m	30
Kurta	xl-56	xl	56

```
<gridProcessorStep>
   <processor>
      <splitColumns gridName="orderLine"
            columnNameToSplit="quantityBySize"
            columnNamesToSplitInto="productSize,orderQuantity"
            fieldSeparator=","/>
   </processor>
</gridProcessorStep>
```

Copy Columns

This is obsolete. Use copy columns across grid instead.

Copy Columns across Grids

Allows columns from one grid to be copied to another based on key value matches. This is typically useful in copying data across parent-child grids.

We start with the child grid. For each row in the child grid, we first identify the corresponding row from parent grid by matching key-value. Then we copy columns values from parent row to the child row. We expect that there is one and only one row in the parent grid

for a row in the child grid. If such a row is not found, the columns remain empty. If more than one row is found in the table, we do not guarantee which row will be taken. We may take any row. Hence, you should ensure that that the key value is unique in the parent grid.

.

Name	Type	reqd?	Default	Explanation
fromGridName	string	yes	-	Name of the grid to copy columns from
toGridName	string	no	-	Name of the grid to copy the columns to. Defaults to fromGridName.
keyColumnNameToMatchFrom	string	yes	-	comma separated list of column names in the from-grid to be used for matching rows
keyColumnNameToMatchTo	string	no	-	defaults to from names. Number of names must match in the two attributes of both are specified.
columnNamesToCopyFrom	string	yes	-	comma separated list of columns to be copied from
columnNamesToCopyTo	string	no	-	defaults to from columns.

```
<gridProcessorStep>
   <processor>
      <copyColumnsAcrossGrids
            fromGridName="indianFlowers"
            toGridName="scientificName"
            keyColumnNameToMatchFrom="flowerId"
            keyColumnNameToMatchTo="flowerId"
            columnNamesToCopyFrom="state,flower"
            columnNamesToCopyTo="copied_state,copied_flower"
      />
</processor>
</gridProcessorStep>
```

Before copyColumnsAcrossGrids operation.

indianFlowers

Flowered	flower	state
1	Water lily	Andhra Pradesh
2	Lady's Slipper	Arunachal Pradesh
3	Foxtail Orchids	Assam
4	lotus	Haryana

scientificName

flowered	name
1	Nelumbo
2	Cypripedioideae
3	Rhynchostylis gigantea
4	Nelumbo nucifera

After copyColumnsAcrossGrids operation.

scientificName

flowerId	name	copied_state	copied_flower
1	Nelumbo	Andhra Pradesh	Water lily
2	Cypripedioideae	Arunachal Pradesh	Lady's Slipper
3	Rhynchostylis gigantea	Assam	Foxtail Orchids
4	Nelumbo nucifera	Haryana	lotus

Copy Rows

"Append Rows" would have been a better name for this utility. Rows from one grid are appended to another. Ideally, the two grids should have the same columns in the same order. In case the columns differ, we copy only the common columns, and leave any other column in the to-grid empty.

It is possible to omit certain rows while copying. Use actionFieldName attribute. Rows with a value of "delete" in this column will not be copied.

Name	Type	reqd?	Default	Explanation
fromGridName	string	yes	-	Name of the grid to copy from
toGridName	string	yes	-	Name of the grid to copy to

```
<gridProcessorStep>
   <processor>
      <copyRows fromGridName="new_city" toGridName="city"/>
   </processor>
</gridProcessorStep>
```

Before copyRows operation.

new_city has 4 rows and 2 columns

new_id	new_cities
4	hubli
5	dharwad
6	bidar

city has 4 rows and 2 columns

id	cities
1	bangalore
2	mysore
3	tumkur

After copyRows operation.

city has 7 rows and 2 columns

id	cities
1	bangalore
2	mysore
3	tumkur
4	hubli
5	dharwad
6	bidar

Filter Rows

Filter rows from a grid based on a value from a column. We go thru each row and compare the value of the specified column with the supplied value. If comparison is true, we retain the rows, else delete the row.

Name	Type	reqd?	Default	Explanation
fromGridName	string	yes	-	Name of input grid
toGridName	string	yes	-	Name of output grid
columnName	string	yes	-	Column name from input grid
comparator	type	yes	exists	Choose comparators among (exists, doesNotExist, equalTo, notEqualTo, lessThan, lessThanOrEqualTo, greaterThan, greaterThanOrEqualTo)
value	string	yes	-	Value to be used for comparison

```
<gridProcessorStep>
   <processor>
      <filterRows fromGridName="indianFlowers"
            toGridName="sorted_indianFlowers"
            comparator="equalTo"
            columnName="state"
            value="Haryana"/>
   </processor>
</gridProcessorStep>
```

Before filter operation

indianFlowers

flowerId	flower	state
1	Water lily	Andhra Pradesh
2	Lady's Slipper	Arunachal Pradesh
3	Foxtail Orchids	Assam
4	lotus	Haryana

After filter operation

sorted_indianFlowers

flowerId	state	flower
4	Haryana	lotus

Split Rows

Not used any more. Use de-aggregater instead.

Merge Rows

Obsolete. Use Aggregator instead.

Aggregater

Many a times, you are interested in some kind of a summary data, and not the actual detailed data. For example, you want total order by region. In this case, you are looking for "sum" as an aggregation on column orderQuantity with region as grouping columns. An aggregator creates a new grid that has the same data as the first grid, but at summary level for each distinct value in the group-by column. You can have more than one aggregate column. Like total order, averageOrderQuantity, maxOrderValue etc.

Every column that is required in the output grid must be included in the list of columns. Columns that form the key, must be marked as key, but any other column that

Name	Type	reqd?	Default	Explanation
inputGridName	string	yes	-	Aggregate operations will be performed by using input grid.
outputGridName	string	yes	-	A new grid will be created with outputGridName containing the aggregated columns.

There are 10 kinds of aggregators. They are:
0. key
1. sum
2. average
3. count
4. first
5. last

6. Min
7. max
8. list
9. append

Name	Type	reqd?	Default	Explanation
inputColumnName	string	yes	-	Specify the column name in the input grid to perform aggregation.
outputColumnName	string	yes	-	Specify alias or same name as input column, which appears on output grid after performing aggregation.
aggregationFunction	string	Yes	-	Type of aggregate function. Types of aggregators are mentioned above.

```
<?xml version="1.0" encoding="utf-8" ?>
<service name="aggregatorExample"
                    xmlns="http://com.exilant.exility/schema"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://com.exilant.exility/schema ../../schema.xsd"
                    dataAccessType="none"
                    >
<steps>
  <!-- columns -->
 <list name="state" values="Karnataka,Karnataka,Andra pradesh,Andra pradesh,Kerala,Kerala"/>
 <list name="districts" values="Bangalore, Mysore, chitoor, anantapur, kochi, thiruvananthpuram"/>
 <list name="population" values="100, 80, 120, 90, 75, 85"/>
  <list name="famousFor" values="garden, palace, tirupati, anantapur, lakes, Ayurveda"/>
 <list name="languages" values="kannada, kannada, telugu, telugu, malayalam, malayalam"/>
 <!-- table -->
 <grid name="IndianStates" columns="state, districts, population, famousFor, languages"/>
 <!-- Aggregate list of districts by statewise-->
<gridProcessorStep>
        <processor>
                <aggregater inputGridName="IndianStates" outputGridName="IndianStatesGroup" >
                <columns>
                        <aggregaterColumn inputColumnName="state" outputColumnName="state" aggregationFunction="key"/>
                        <aggregaterColumn inputColumnName="districts" outputColumnName="districts"
                                        aggregationFunction="list"/>\
                        </columns>
                </aggregater>
        </processor>
</gridProcessorStep>
</steps>
</service>
```

IndianStates

IndianStates

state	districts	population	famousFor	languages
Karnataka	Bangalore	100	garden	kannada
Karnataka	Mysore	80	palace	kannada
Andra pradesh	chitoor	120	tirupati	telugu
Andra pradesh	anantapur	90	anantapur	telugu
Kerala	Kochi	75	lakes	malayalam
Kerala	Thiruvananthpuram	85	Ayurveda	malayalam

IndianStatesGroup

IndianStatesGroup

state	districts
Karnataka	Bangalore,Mysore
Andra pradesh	chitoor,anantapur
Kerala	kochi,thiruvananthpuram

```
<gridProcessorStep>
<processor>
        <aggregater inputGridName="IndianStates" outputGridName="IndianStatesGroup" >
        <columns>
                <aggregaterColumn inputColumnName="state" outputColumnName="state" aggregationFunction="key"/>
                <aggregaterColumn inputColumnName="districts" outputColumnName="firstDistrict" aggregationFunction="first"/>
                <aggregaterColumn inputColumnName="districts" outputColumnName="lastDistrict" aggregationFunction="last"/>
                <aggregaterColumn inputColumnName="districts" outputColumnName="districtCount"       aggregationFunction="count"/>
                <aggregaterColumn inputColumnName="population" outputColumnName="populationSum" aggregationFunction="sum"/>
                <aggregaterColumn inputColumnName="population" outputColumnName="populationMin" aggregationFunction="min"/>
                <aggregaterColumn inputColumnName="population" outputColumnName="populationMax" aggregationFunction="max"/>
                <aggregaterColumn inputColumnName="population" outputColumnName="populationAvg" aggregationFunction="average"/>
                <aggregaterColumn inputColumnName="famousFor" outputColumnName="famousList" aggregationFunction="list"/>
                <aggregaterColumn inputColumnName="districts" outputColumnName="district_language" aggregationFunction="list"/>
                <aggregaterColumn inputColumnName="languages" outputColumnName="languagesAppend"
aggregationFunction="append"/>
        </columns>
        </aggregater>
        </processor>
</gridProcessorStep>
```

Example 3 Aggregate column

IndianStatesGroup has 4 rows and 10 columns									
state	**First District**	**Last District**	**districtCount**	**Population Sum**	**Population Min**	**Population Max**	**Population Avg**	**Famous List**	**district_language**
Karnataka	Bangalore	Mysore	2	180.0	80.0	100.0	90.0	garden,palace	Bangalore-kannada,Mysore-kannada
Andra pradesh	Chitoor	anantapur	2	210.0	90.0	120.0	105.0	tirupati,anantapur	chitoor-telugu,anantapur-telugu
Kerala	Kochi	thiruvananthpuram	2	160.0	75.0	85.0	80.0	lakes,Ayurveda	kochi-malayalam,thiruvananthpuram-malayalam

```
<gridProcessorStep>
        <processor>
                <aggregater inputGridName="IndianStates" outputGridName="IndianStatesGroup" >
                <columns>
                        <aggregaterColumn inputColumnName="state" outputColumnName="state" aggregationFunction="key"/>
                        <aggregaterColumn inputColumnName="districts" outputColumnName="districts" aggregationFunction="list"/>
                        <aggregaterColumn inputColumnName="population" outputColumnName="populationAvg"
aggregationFunction="append"/>
                        </columns>
                </aggregater>
        </processor>
</gridProcessorStep>
```

Example 4 Aggregate List

IndianStatesGroup	
state	**districts**
Karnataka	Bangalore-100,Mysore-80
Andra pradesh	chitoor-120,anantapur-90
Kerala	kochi-75,thiruvananthpuram-85

DeAggregater

Obviously, the opposite of aggregater. Process to do the reverse of aggregation. However, we cannot de-aggregate columns that were aggregated using function other than concatenate/list. For example, if you used average, we cannot get back the list of numbers that produced this average. But a list can be split back into the individual value that was created with.

It is possible to de-aggregate appended list also. But, as of now, we do not handle that directly. You first de-aggregate the list into an output grid. In this grid, the column would have an appended value (like "s-20") which you can split into two columns using SplitColumn utility.

Each row in the input grid will repeat as many times as many values are found in the column after splitting it. Each of the output rows has the same column values except the column being split.

In case the column is empty in a row, that row is not copied to the output grid.

Name	Type	reqd?	Default	Explanation
inputGridName	string	yes	-	DeAggregate operations will be performed by using input grid which has a column with list of values aggregated.
outputGridName	string	yes	-	A new grid will be created with outputGridName containing the deAggregated columns.
deaggrgatedInputColumnName	String	yes	-	Name of the column to deAggregate.
deaggrgatedOutputColumnName	String	yes	-	alias or same name as deaggrgatedInputColumnName, **which** appears on output grid after performing deaggregation.
additionalOutputColumnName	String	no	null	Specify name for the column, if deaggrgation list has appended

Name	Type	reqd?	Default	Explanation
				values in it.

```
<gridProcessorStep>
        <processor>
                <deaggregater  inputGridName="IndianStatesGroup"
                               outputGridName="IndianStates"
                               outputColumnNames="state,districts"
                               deaggrgatedInputColumnName="districts"
                               deaggrgatedOutputColumnName="districts"
                               additionalOutputColumnName="population"
                                >
                </deaggregater>
        </processor>
</gridProcessorStep>
```

IndianStatesGroup has 4 rows and 2 columns

IndianStatesGroup

state	districts
Karnataka	Bangalore-100,Mysore-80
Andra pradesh	chitoor-120,anantapur-90
Kerala	kochi-75,thiruvananthpuram-85

IndianStates has 7 rows and 3 columns

IndianStates

state	districts	population
Karnataka	Bangalore	100
Karnataka	Mysore	80
Andra pradesh	chitoor	120
Andra pradesh	anantapur	90
Kerala	kochi	75
Kerala	thiruvananthpuram	85

Service

Service defines how a published service should be carried out on the server. You have two ways of implementing a service.

a. Use service.xml format to define steps that this service involves. Here you make use of all the utilities we have described so far. You can also interlace these with java code that you may write.
b. Write your own java class that implements a prescribed interface. Refer to Exility Programmer's manual for details.

Our objective is to make service definition as readable and maintainable as possible. Hence we have defined a few "verbs" that describe primarily "what" is to be done and not "how" it should be done.

A service expects certain input data. This is described as a set of input records. Exility validates input as per this record and the fields in them. In case of any error, request is returned to client without invoking the service.

Before introducing the concept of records, this expectation of data used to be described as part of service entry.

Input Record Attributes

Name	Type	reqd?	Default	Explanation
recordName	text	yes		Fully qualified name of the record
fieldNames	text	no		If you are expecting a subset of fields, give us a comma separated list of fields. You have to ensure that the subset of fields contains required from-to pairs etc..
isGrid	text	no		Are we expecting data in a grid? (otherwise we expect name-value pairs in dc)
minRows	number	no	0	Required if isGrid is true. Note that an empty grid, with only header and no data, is put into dc even if no rows are sent from client.(If required, we may add an attribute to influence this feature)
maxRows	number	no		This is required if isGrid is true. A project-wise max is set in applicationParameters. This is a safety measure against a client sending large number of rows that may choke the system. You should set reasonable limit if the client is interactive. If you are designing this service for batch-update that may bring in millions of rows, we certainly recommend a custom solution rather than this service based approach. You may have to have some sequential processing : that is read-process-

Name	Type	reqd?	Default	Explanation
				discard algorithm rather than reading all into memory and then processing.
forFiltering	boolean	no	false	Fields in this record are used for defining the filtering criterion for a search/filter operation on the client. Standard Exility client feature sends an operator and possibly a to-field for every field sent for filtering. This convention is utilized to extract the required data from input.
allFieldsAreOptional	boolean	no	false	Disregard the isOptional attribute of the field in record, and assume every field is optional. This option is useful when you are filtering, or when you allow update operation on subset of fields that re sent from client.

A service consumer expects certain data as output from the service. This expectation is described as output records. recordName, fieldNames and isGrid are the three attributes of output record, with same meaning as in input reocord.

You describe what the service has to do using service steps.. A step is similar to a statement of a programming language. Exility starts with the first step, and goes sequentially to other steps. However, it is possible to change this default sequence with jump and loop structures. Also, the execution can be stopped with an error, or an exception can be raised at any stage.

A step is where you make use the "basic work units" of Exility (i/o tasks and logic tasks are considered to be the basic work units).

A step can be "conditional". That is, it is executed only if certain conditions are met. Three possible conditions are explained in the common attributes below. Note that these attributes are applicable to any step.

Step Attributes

Name	Type	reqd?	Default	Explanation
label	text	no		This is the name of this step. It is used only if this step is a target of a go to from other step.
description	text	no		For documentation purpose
techNotes	text	no		For documentation purpose
skipOnMessageIds	text	no		Comma separated list of messaged Ids. This step is skipped(not executed) if any of these messages are found in dc. This feature is useful if you need conditional execution based on some validation results in the

Name	Type	reqd?	Default	Explanation
				previous steps. That is to say, if any of the previous steps had added any of the message ids to the dc, then skip this step.
executeOnMessageIds	text	no		Converse of the above attribute i.e. step is executed if any of the messagesIds are present in DC.
executeOnCondition	text	no		a valid "expression" that should evaluate to a Boolean. If specified, this expression is evaluated at run time by taking values for any variable in this expression from DC. If the expression evaluates to false, then the step is skipped. Refer to section on Expression for details.

We now describe all possible steps of a service

Dummy Step

Common use of a dummy step is as a destination of a goTo of another step. Note that it does not make sense to have any attribute other than label for a dummy step.

Expression Assignment Step

This is like "`a = b * c - d`" in a programming language. Set Value is an alternative to this. Existing value is replaced with this new value.

Note that Exility uses this field name as "variant". That is, the value type of the field is determined based on the result of the expression evaluation. However, we strongly recommend that you use data elements entered into data dictionary, and never change the data type of a field in a given application.

Name	Type	reqd	Default	Explanation
stepAttributes	Set	-	-	stepAttributes as specified above
fieldname	Text	yes	-	To which value is assigned.
expression	text	no	-	Valid expression that evaluates at run time with no errors. In case this expression itself is known at run time, then leave this attribute unspecified, and use the next attribute.
expressionFieldName	text	no	-	In case the expression to be evaluated is to be determined at run time,

Name	Type	reqd	Default	Explanation
				this field should have a value that is a valid expression.

Set Value Step

User can set some value to a variable using this option.

Name	Type	reqd?	Default	Explanation
`condition`	string	yes	-	Obsolete. Should not be used
`name`	string	yes	-	Name of the field to which the user is assigning the value
`expression`	string	yes	-	should evaluate to the right type. Same value is used to initialize the entire column.

Stop Step

Execution stops, of course if the condition associated with this step is evaluated to true.

Function Assignment Step

Exility does not allow functions within an expression. This requires a multiple-step approach in case you have a complex expression that involves function evaluation. This step allows you to execute a function with parameters and assign the result to a field. This is like your x = foo(p1, p2, ..).

Name	Type	reqd	Default	Explanation
stepAttributes	Set	-	-	stepAttributes as specified above
fieldname	Text	yes	-	To which value is assigned.
functionName	text	yes	-	Name of the function to be executed. This function should be registered with an entry in functions.xml.
inputParameters	text	no	-	Comma separated field names whose values are passed to the function at run time.

Validation Step

Objective of this step is to add a message to dc with required parameters. Condition attached to step acts as validation test.

Name	Type	reqd	Default	Explanation
stepAttributes	set	-	-	stepAttributes as specified above
messageName	text	yes	-	Message to be added to dc.
messageParameters	text	no	-	Comma separated field names that are supplied to the message.
continueOnError	bool	no	false	Should teh execution continue even if this added message is an error

Error Check Step

An error check step allows you to branch out based on any error encountered so far during the execution of steps so far. You can check the over-all error level of the service (as reflected in dc) and decide how you want to deal with it.

Name	Type	reqd	Default	Explanation
stepAttributes	set	-	-	stepAttributes as specified above
severity	enum	-	error	**error, warning, info and ignore** are the possible values. Control shifts to the label specified in the next attribute if the current level of dc >= this level. Note that the **default is error**. This condition is in addition to any of the general step conditions you may have specified.
goToStep	text	yes	nextStep	label of the step to which the control should jump to

Switch Step

A switch step is similar to a switch-case in higher level programming languages. It has field name, and a collection of switch actions. Each switch action has a value and the label of the corresponding step to jump to if the value in the field matches this value. Control flow after the initial jump is also similar to your higher language. It keeps going unless you use a jump again. You may use jump facility of any of the task steps, or may use a dummy step with an unconditional jump. It is good practice to have a dummy step as the common point after the switch block to which the dummy steps at the end of each case block may jump to.

Name	Type	reqd	Default	Explanation
stepAttributes	set	-	-	stepAttributes as specified above
fieldname	text	yes	-	Name of the field whose value is to be used for switching
switchActions	List of actions	-	-	This is a tag (object) and not an attribute. It has two attributes, value and labelToGoTo. Note that the value is text, but you must have a constant of the right data type in that. Common mistake is to put a text constant just like that. You must enclose a text constant with a pair of &quo;.

Loop Step

A loop step allows you to carry out a set of tasks for each row of a grid. The block of steps are executed as many times as many rows are there in the grid. Also, columns from the corresponding row are copied to dc with name = prefix + columnName. At the end of the loop, optionally, a set of column values can be set back from dc. It is possible for you to get out of the loop in the middle using any of the goTo facilities by jumping beyond the first or last step of the loop.

Name	Type	reqd?	Default	Explanation
stepAttributes	set	-	-	stepAttributes as specified above
tableToLoopOn	text	yes	-	Name of the grid based on which to loop. If the grid has no rows, then the loop block is just skipped. It is an error if the grid does not exist in dc.
lastStepOfTheBlock	text	yes	-	label of the step that marks the end of the loop block
inputFields	text	no	all	By default, all columns of the grid are copied to dc. If you use a small subset of the columns, you may provide the list of columns as a comma separated list.
outputFields	text	no	none	You have the option to calculate values for one or more columns in the loop block. Such column names can be specified as a comma separated list. If the grid already has these columns, they are replaced, else new columns are added. Note that each of the data type of the column (as indicated by the name) must match the data type calculated by the step.
noInputFieldsPlease	bool	no	false	inputFields defaults to all. What if you do not want any input fields? This attribute over-rides the default value for inputFields attribute

Task Step

A task is a basic work horse in Exility. Task is where either a data base I/O or calculation happens. taskEntry in serviceList has the same structure as task step here.

Tasks have a notion of work done. For example, a sql extraction task returns number of rows extracted as work done. An update sql returns number rows affected as work done. A task will not have a predefined meaning for failure to extract. It just returns this condition, and it is up to the designer of the step to decide what to do. A task can be used in more than one service. It could be an error if no rows are extracted in one service, while another it would be the other way round.

Task step has attributes to respond to the work done returned by the task step.

Name	Type	reqd?	Default	Explanation
taskname	string	yes	-	Name to the task. Depending on the task type, this name will refer to either an SQL template, a table, or logic name etc..
taskParameters	string	no		Parameters for the task. Can be coma separated list. Meaning of the parameters is specific to the task on hand.
raiseExceptionIfNoWorkDone	Boolean	no	false	If the task returns 0 as work done, then an exception is to be raised, and the execution of service stops. The transaction is rolled back.
ifNoWorkDoneMessageName	string	no		Message Id to be pushed to dc if no work is done. If you have set above attribute to true, then you must specify a message name, so that an appropriate message is sent to the caller.
ifNoWorkDonemessageParameters	string	no		Comma separated list of fields, whose values are to be supplied to the message id as parameters.
ifWorkDoneMessageName	string	no		Some time, your design is such that the task is expected to return empty handed. That is it is not supposed to do any work, And it is a condition you want to handle if work is done. Use this and the next attribute to handle such situations.
ifWorkDonemessageParameters	string	no		Comma separated list of fields, whose values are to be supplied to the message id as parameters.
ifWorkDoneGoToStep	string	no		Facility to jump to a step if the task returns a non zero work done value
ifNoWorkDoneGoToStep	string	no		Facility to jump to a step if the task returns zero work done value
repeatForRowsInGrid	bool	no	false	Many tasks are designed in such a way that can be repeated for each row of a grid. For example, a sql task can be repeated for each row of a grid.

Name	Type	reqd?	Default	Explanation
				In this case, the sql is formed for each row, and executed. This is similar to loop-step for a single task.
gridName	string	no	-	Meaning depends on the task. For example, for a sql task with above parameter set to false, data is extracted into this grid. But if above attribute is set to true, this is the grid for which the task is repeated.

Grid Processor Step

A table (or grid) is the only data structure that Exility provides. Simple view is to treat this as rows and columns, as in a spreadsheet. First row is the heading, providing names for the columns with which they can be referred to. Subsequent rows contain data. One restriction is that each column has to stick to its data type. Each column name must refer to a data element in the data dictionary. Note that any value could be NULL.

You may also treat a grid as an ordered collection of "columns". A column is a data structure with its name and data type, and a list of data (values). All columns in a grid have to have exactly the same number of values.

You typically end up having a grid as an input from a page, or when you extract data using a sql. You may need to manipulate the data in such a grid. Exility provides a set of grid processors. This step is the wrapper using which you can invoke a grid processor. (Grid processors ares explained later)

bulkType

Name	Type	reqd?	Default	Explanation
taskAttributes	set	-	-	taskAttributes as specified above
ifWorkDoneGoTo	string	no		Label of the step to be jumped to if this processor returns a non zero value as work done. Most grid processors return the number of rows as work done. Please refer to the individual processor's specification before using this feature.
ifNoWorkDoneGoTo	string	no		refer above

Service Step

A service can be reused in another service as a step. This flexibility encourages designers to reuse group of tasks across services. Suppose you have a set of tasks which together implement a meaningful business transaction. This is required to deliver more than one service exposed to the clients. In such a case, you create a sub-service (syntax is same as creating a service, but you do not put that in service list) and use this sub-service as a step in the services that you have exposed to clients. Some important points to keep in mind

1. A new dc is created for the sub-service based on the spec if provided. Only fields in this dc are available to tasks of this service
2. Updates are part of the same transaction as the parent service. No new connection is used, or no new transaction is started.
3. outspec of the sub-service is used to copy fields from sub-service dc to the main service dc when the sub-service completes.
4. Any number of levels of call is allowed. That is a sub-service may use another service step, including itself recursively.

Name	Type	reqd?	Default	Explanation
taskAttributes	set	-	-	taskAttributes as specified above
serviceName	string	yes		Name of the service to be executed. This service need not be in the service list.

Task

Task is one chunk of work that can be used as a step inside a service. Task can also be used as a simple service by entering the task directly in serviceList. Task returns work done. If returned value is zero, it means that the task was not successful, (but no errors though). For example, for a sql task, this is the number of rows retrieved. Task attributes allow you to carry out some validations based on how the task went.

```
<taskStep>
  <task>
  <customCodeTask taskName="getFiles" fullyQualifiedClassName="com.exilant.exility.ProjectName.GetFiles"
                  taskParameters=”folderName,fileName” gridName=”filesGrid”/>
  </task>
</taskStep>
```

Example 6 Custom Task

Task Attributes.

Name	Type	reqd?	Default	Explanation
taskName	string	yes	-	name of the underlying task (like sql, or table definition) that you use to carry out this task.
taskParameters	string	no	-	comma separated parameter names if you use them. Use of these parameters is left to the task, and the service step where this is called.
raiseExceptionIfNoWorkDone	bool	no	no	Should an exception be raised, if the task returns work done as 0
raiseExceptionIfWorkDone	bool	no	no	Should an exception be raised, if the task returns work done as non zero
ifNoWorkDoneMessageName	string	no	-	if no work done, add this message to dc
ifNoWorkDoneMessageParameters	string	no	-	parameters that go along with the above message
ifWorkDoneMessageName	string	no	-	if work done, add this message to dc
ifWorkDoneMessageParameters	string	no	-	parameters that go along with the above message
gridName	string	no	-	if the task requires a grid, this the name of the grid. Otherwise, this is the name of the grid for which the task will be repeated
repeatForRowsInGrid	bool	no	-	the underlying task is to be repeated for each row the grid.

sql Task

sql task is created using an underlying sql template. taskName refers to the sql name. Note that, if you use module, taskName should specify fully qualified sql name (module.name). Exility generates the sql based on data at run time, and executes or extracts the sql.

If toBeExecuted is true, number of affected rows is returned as work done. Grid name is ignored unless repeatForRowsInGrid is specified. In that case, the sql is generated for each row of the grid, and total affected rows are returned as work done.

If it is an extraction sql, number a record set is created with the generated sql. If grid name is specified, all rows are extracted into this grid, else only the first row is extracted into dc. If repeatForRowsInGrid is specified, then the first row data is extracted into the corresponding row of the grid.

Table Insert Task

Use this to insert a row into a table. If you have not included any of the columns in the database table, then it must be either nullable, or should have a default value set in the table design. Also, any column you have defined as optional must be either nullable or have a default value specified in data base design. If the key is to be generated, then the generated key is put back into dc, so that you can use it for any foreign key in its child tables.

Table Update Task

Use this task to update a row based on values sent by the client along with the primary key and time stamp if relevant. A sql is generated to update only the fields that are sent. This sql has a where clause for the primary key, and if required, for matching the time stamp. If the row was updated in-between, there will be no error, but it returns work done as zero. You can use this in your service to return an appropriate message to the client.

Table Save Task

It is a common practice to have a common page on the client for both creation and editing of a document. In such a case, the sent row has to be either inserted or updated. This task is designed for such a condition. It invokes tableUpdateTask if the primary key is available. Else, a tableInsertTask is executed.

Table Delete Task

Use this delete a row of a table. Primary key is to be supplied by the client, and the table specification should have okToDelete set to true.

Bulk Task

This is meant for grid editing. When a grid is sent to the client for editing, it comes back with rows with a column that indicates what is to be done with each row. This column, called actionColumn, is set by Exility client side, to one of 'add', 'modify', 'delete'. This task makes use of such a column, and initiates an appropriate task.

All the table based tasks can be carried out in 'bulk' mode. That is, each task can be executed for each row of a grid. In such a case, work done is set to the sum of affected rows. Note this is similar to bulk task above, except that the operation on each row is predetermined, and not based on a column.

Stored Procedure Task

Stored procedures can be executed using this task. When defining parameters for the procedure, define in the order in, inout, out. The same order (in, inout, out and the params within) should be used in the task as well. No return values will be handled (as return value may be table itself)

Business Logic Task

This is obsolete.

Crafted Logic Task

Obsolete.

User Task

Obsolete.

Custom Code Task

This task is used to execute a custom task you have written in Java. Your java code should implement CustomCodeInterface, and you specify the fully qualified name of the class as attribute of this task.

Audit Log

Exility provides feature to automate audit logs.

First of all, you have to decide on a naming convention for name of audit table. We insist that the name of the audit table starts with the name of the table followed by a suffix that you define for your entire project. For table customer_details you may have audit table as customer_details_audit. In this case, you have shoes "_audit" as suffix to be used by Exility while writing audits.

Following parameters are to be set properly in applicationParameters:

- audittableSuffix: tableName+suffix is the name of the audit table. Like '_adit' we mentioned above.
- enableAuditForAll: Do you want every table to be auditable. If you set this to true changes to every table is logged. Normally, that is not what you want. You may choose to set this as false, and use "isAudited" option at the table level to do selective logging.
- auditConnectionString : The connection string to the audit schema. Not recommended Use the next parameter instead.
- auditDataSource : secure way of specifying how to connect to the database using jndi.

Note that Exility does not create the tables. Your database administrator has to create these tables as clones of the tables that they log, with one additional text column names "action". Whenever a row is added, modified or deleted, Exility pushes a new row into the audit table with action column representing these actions, while other columns are populated with the same values as in the table.

If you modify the table with any of your sqls or stored procedure, Exility cannot detect these operations automatically, and hence you should take care of audit logging as well.

Background Jobs

Any enterprise class web based application does have a few operations that are time consuming. A good design practice mandates, that instead of showing the user a “rotating circle” for a very long time, we submit the job for processing, and when the work is complete, we have a way of informing the user. While the job is processing, the user is free to perform other actions.
With this intent, the concept of background jobs was introduced.

How It Works

The client makes a service call. If the service call is marked in the service entry as "submitAsBackgroundProcess=true" then the Exility server opens a new thread and executes the service in the new thread. The original service call immediately returns with the value of a job id in the DC. The job id in the returned DC is stored in the values collection with the special name "_backgroundJobId". Additionally, each and every call to the server contains the status of all submitted jobs in the returned DC in the values collection under the special name "_backgroundJobs". The format is [jobid]=[status];[jobId]=[status]. The status can have only two values - "running" and "done". Once a thread completes the execution, it creates a file in the "temp" folder of the web deployment. The file name is same as the job id. The job id is a UUID. The file contains a serialized DC that was the result of the job. To know the status of a specific job, you can call a server action with the special name "getBackgroundJobResult". If the job is complete, then this method will read the DC stored in the file, delete the file and return the job status in the values collection field as "done". Please note, the field name "_backgroundJobStatus" is used to return the status. This time, this only contains the status as "done" or "running". The result of the background job is contained in the returned DC.

What is Done on Exility Server and Client Side

A new object called "backgroundJobs" will always be available to the client as an attribute of the "PM" (PageManager) object. Every call to the server, as stated before, returns with the status of the background jobs. Exility client side automatically updates the status of each job. The developer does not have to do anything. The "PM.backgroundJobs" is a collection of all background jobs that got submitted during the session. If you decide to know the status of a specific job, by the job id, you can call a server action. The service id must be the special service name "getBackgroundJobResult" and the input job id should be in values collection with the special name "_backgroundJobId". Once the server action returns, once again, Exility client side updates the status of the job automatically and result is stored in the "PM.backgroundJobs" object.

Sequence of JS File Inclusion

The clientConstants.js file must be the first file to be included in the index.htm file. Also, backgroundJobs.js file must be included in index.htm before the pageManager.js and serverAgent.js file

Limitations

There is no specific limitation. We only need to remember, that if the user refreshes the screen then status of all background jobs will be lost. This is consistent with current Exility behaviour.

Workflow

Workflow has different connotations at different contexts. We use it as a modelling technique. That is, you model your server side functionality as a workflow, so that its implementation can be simplified.

Exility provides only design time utilities for application designer to implement a workflow. There is no support for run-time creations of workflow by end users.

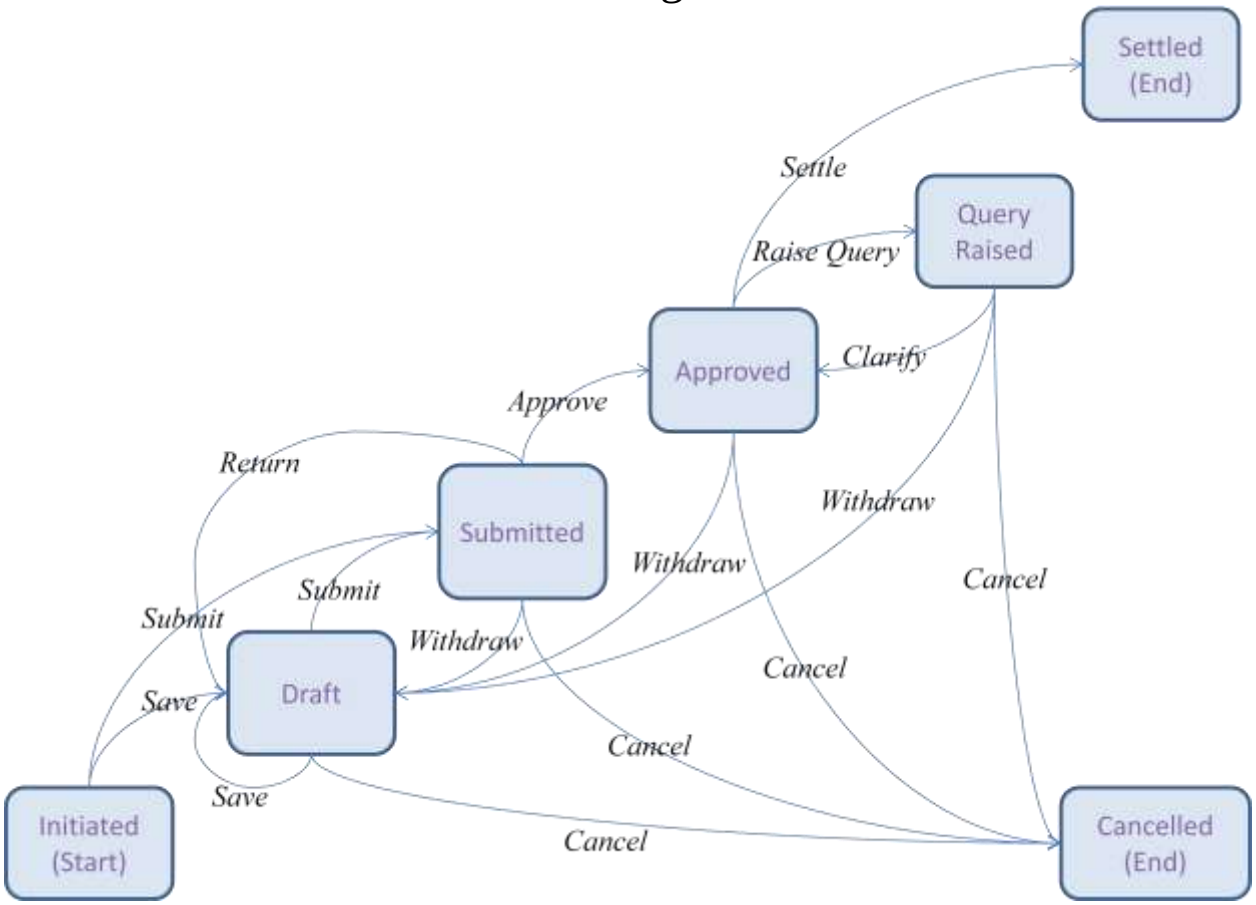
Let us take "expense reimbursement" as an example.

1. User fills up an expense sheet. She can just save it be reviewed by her later, or submit it.
2. A saved document can be saved again and again, cancelled, or submitted.
3. An expense claim can be cancelled at any time before closed. Once cancelled, claim is done with, and cannot be touched.
4. An expense claim can be withdrawn any time after it is submitted, but before it is done-with. Withdrawn claim is exactly like a saved claim.
5. Her direct reporting manager is the only one who can approve it. And, for simplicity, let us say one level of approval, irrespective of the claimed value is enough for accounts to pay this amount. If the manager has some issue with this, and does not want to approve, she can return it to the claimant. (Of course necessarily with some comments). A returned cliam is treated exactly like a saved claim.
6. Once approved, a designated accountant posts this for payment as part of payroll. Once posted, this workflow is closed. That is, actual payment outside of this workflow. It happens as part of payroll process. Also, once posted, the claim cannot be modified or cancelled. No changes to this will happen thru another correction process that is outside the scope of this workflow.
7. Accountant may have some query or clarifications that he may ask. Claim is not posted unless the claimant provides clarification. However, no further approval is required by the manager. The cycle of seeking and getting clarification can go on.

Purpose of a workflow is to take an underlying document from starting state one of the end states through a series of steps, each step being an action taken by a role because of which the document goes from one state to another, or retain it in the same state.

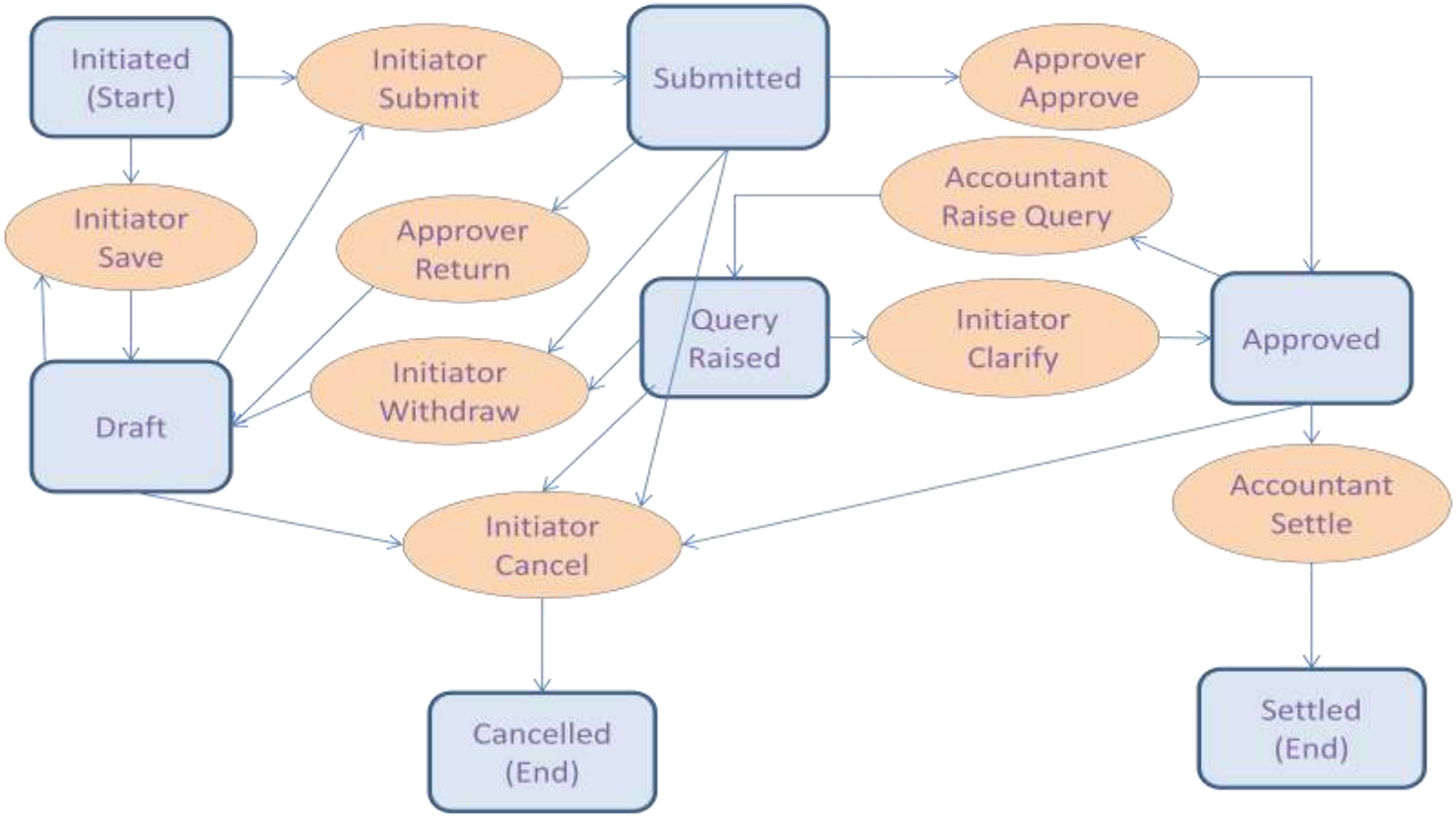
With that rather long sentence as our definition, let us delve into its parts.

Leave Approval Process – State Transition Diagram



This is a state-transition diagram for the workflow. It is state centric view. It is easy to understand, but this diagram does not show who the actors are. Next diagram is another way of presenting the same workflow. It looks far more complex, but the only difference is that we have added the actors to the action. Both represent the same workflow, and you can choose the one that is intuitive to you while reading the detailed text.

Leave Approval Process – Actor and Action Diagram



Document

Document is quite straight forward in most case. It is the entity that we are working on. In some cases, we could be working with two or more documents. Like order modification after it is invoiced. You may treat any one as the underlying documents. Or you may create a mythical "change order request" as the document that flows through this workflow. It is up to you to see which way of modelling makes it easier to understand and represent other aspects well.

In our current example, expense claim is the document.

A document may go through several workflow processes in its life cycle. Also, it is possible that the same document could be simultaneously flowing through different workflows. Remember that a document flowing through a work flow, going from one in-box to another, is only for our visualization. Document in reality is always safely sitting in our database. ☺

Actors

Any user of the application working on a specific role who can take action in a workflow is called an actor. Actor is different from user as well as from role.

In our example here are our actors.

Actor	Description
initiator	This is a mandatory value for every workflow. Every workflow has to have an initiator, the one who starts the workflow. Business rules define who can be an initiator for this workflow. Your standard operating procedure (SOP) may say that only permanent employees can claim expenses through this procedure. Assuming.
approver	Who can approve claim from the initiator. Business rule could be that the reporting manager of the initiator is the approver.
accountant	One who verifies that the claim is complete in all aspect and has the knowledge to post it to the right account heads in the books. Business rule may say any employee playing the role of 'account assistant" can do this. Or, there may be one account assistant for every department. So, the account assistant in charge of the department to which the manger belongs, could be the actor for this workflow.

State

Document transitions through different states during a workflow. It is quite intuitive. However, we require a more rigorous definition for state for us to apply some standard logic to the work flow. A state is a state if and only if the actions that can be taken are determined just by knowing the state, and nothing else. That is state + action must take it to a predetermined state.

Following are the states in our example.

Actor	Description
initiated	This is a mandatory state. This is defined as the starting state for every workflow. Note that this is a theoretical, transient state that allows us t define initial set of actions and actors.
draft	This is the state of the claim when the initiator decides not to submit right away, but saves it after entering all details. This is the state if initiator edits it and saves again. This is also the state when initiator withdraws the claim before or after approval.
submitted	Once the initiator submits the claim it gets into this state.
approved	This state is attained when the claim gets approved. This is also the state when initiator responds to any query raised by accountant, or the accountant cancels the query he raised.
query-raised	This is an optional state in the workflow. A claim reaches this state if the accountant raises a query.
settled	Once the accountant posts this as a financial transaction into the financial system, he marks claim as settled, possibly by referring to posting reference number. This is an end state. That means, once the document reaches this state, the workflow ends, and no more actions are allowed on this claim.
cancelled	Claim reaches this state once initiator cancels it. This is an end state.

Step

When the document is in a state, an actor takes a valid action and the document moves to its next state. This is called a step. Work flow has a number of steps through which it moves around till it reaches one of the end states.

Current state	Actor	Action	New state	Description
initiated	initiator	save	saved	User saves the claim document after entering details. She becomes

				initiator.
initiated	initiator	submit	submitted	User submits the claim document after entering details. She becomes initiator.
saved	initiator	save	saved	Initiator retrieves a saved claim, looks at it for a minute, gets cold feet and decides not to submit ☺. Saves it again with some changes
saved	initiator	submit	submitted	Initiator retrieves a saved claim, and gets the courage to submit it!!
saved	initiator	cancel	cancelled	Initiator retrieves the claim and decides not to go ahead with it.
submitted	initiator	withdraw	saved	Initiator retrieves a claim that was already submitted but finds something is not right. So, withdraws it with an intention to correct it.
submitted	initiator	cancel	cancelled	Initiator decides not to claim this. Cancels it.
submitted	approver	approve	approved	Approver finds everything in order and approves the claim.
submitted	approver	return	saved	Approver is not comfortable with the claim. Returns with some comments of course.
approved	initiator	withdraw	withdraw	Initiator wants to make some change. Withdraws it.
approved	initiator	cancel	cancelled	Initiator decides to drop the claim.
approved	accountant	post	settled	Accountant is fine with the claim, and creates necessary payment instructions in another system.
approved	accountant	raise-query	query-raised	Accountant requires some additional information. Probably a copy of the bill..
query-raised	initiator	withdraw	saved	Initiator wants to make some change. Withdraws it.
query-raised	initiator	cancel	cancelled	Initiator decides to drop the claim.
query-raised	initiator	clarify	approved	Initiator provides necessary clarification

We see that this simple work flow has large number of steps. That is primarily due to repetition of cancel and withdraw actions at every state. We can have a notation, say * for current state to imply any state, of course other than start and end states.

When you refer to the work flow diagram, a step is a connector.

Attribute	Description
name	This is the name of the workflow that has to be unique in a project. You may use a dot

	notation to use module.workflow-name. A workflow is also a valid service. You have to use workflow.workflow-name to refer to a workflow. For example, on the client side, service name would be workflow.expenseCliam.
description	For documentation purpose
tableName	This is the fully qualified name of table (as in table.xml) that represents the primary document that flows through this workflow. Exility takes care of reading and saving this document.
otherTablesToBeRead	If this workflow requires data from other tables, mention them in a comma separated list.
docIdName	Key field name of the primary table.
customWorkflowClassName	This is the java class you have written to implement business logic behind this workflow. You will write it only if such business logic cannot be implemented using the options given by Exility.
letCustomClassHandleEveryThing	Default is false. Set this to true, only if your workflow is quite different from the model that Exility implements. When you do this, the custom java class you have written will handle all aspects of the work flow.
dataAccessType	This is an attribute of a service. Normally you need to set this to readWrite. Other possible values are readOnly, autoCommit and none.

So, how does it work? To answer that question, let us go back to our design paradigm. We said that there are two different applications, client and server. That client is designed to interact with the user, and the server is built to deliver whatever service client wants. We also suggested that we start from the client.

So, let us start with the client of a workflow. Client application typically has to first show the details of the document, and allow the user to take appropriate action. In this scenario, client would ask for just two services:

- getPossibleActions – client wants to get all the action that the current user can take on the document that is being viewed. Note that the client would have used a different service to get all the details to be shown along with this document. There is no "workflow" aspect. It is like any other view-modify page. Once we respond back to client with possible actions the user can take, it (client application) may either show these in a drop-down, or show them as buttons to be clicked.

Client needs to just send the id of the document. Logged-in user is always known to server.

- act: Yes. Just act. That is, user decides to take one of the possible actions on the document, and pushes appropriate button. Client requests sever to take that action. It will also send all relevant data along with that request of course. Server has to validate everything again, including a check to see if the user is authorized to take that action.

We expect that the workflow design will evolve with usage. It is used by just one project, that too to a very limited extend. We will work with you to design feature to suit your requirements.

Test Automation

Applications built using Exility can use any of the standard testing tools and utilities.

We have designed a spread-sheet based simple utility for server side test automation. You specify the input data and expected output data in a spread sheet, and Exility carries out the test, and reports any deviation from expected result. You may use test-harnessing. That is, you can use manual testing using regular client, and Exility captures input and output data into a test-case spread sheet. You may use this spread-sheet as template to create several other test scenarios.

This utility has not seen any serious usage so far. We will be glad to work with you and help you in setting up, and make any necessary enhancements.