

Decorating your table-definitions to add functionality to an App

Elevate DB, the Orix database, includes the capability to store meta-data within the structure of the database. All database objects (data-tables, columns, functions, stored-procedures, views etc.) can have a DESCRIPTION.

When your Orix App starts this DESCRIPTION is analysed and if key-words are present these are used to extend the operation of your App. These pieces of meta-data, added to the Descriptions of the database are called "decorations".

The decorations are used to control behaviour of the data-records in data-tables, how data displays, how the user can interact with it, and how the Business Object itself behaves.

Sometimes this behaviour is further supplemented by data stored in the system-tables. A decoration may require some record to exist in the "BusinessObjects" or "Resources" data-table in order to work properly. For example a decoration that adds a Drop Down List to the Edit Window may require a SQL Script to be added in the Resources data-table to control exactly what appears in the list.

Decoration is also used in Orix on Stored-Procedures to "surface" these into your App. How to do this is covered in a separate Help document.

Note: These decorations exist in addition to many default features that Orix uses to define your App. Just creating data-tables in the correct way, with relational links as per the Orix standard will add many aspects of behaviour that are useful. This topic covers additional extensions to the default behaviour which may be required in a limited set of circumstances.

What are the typical uses for data-table decoration, and why would I use it?

Decoration can be used to control who can edit, delete or update records. It can be used to control the type of data-entry control the user sees when a field is displayed in your App. It can be used to automate the data that is added to a table or column. It can be used to extend and define the relationships between different tables in your App.

There are many common tasks a programmer wants to achieve in a database (for example adding details of the Current User to a column in a database so you know who made a change to a record). The Orix Framework does what it can to simplify these tasks. The programmer just has to add decorations to the database, and the framework will automate the process.

Note: Many decorations require the existence of certain columns in a data-table, for example the "OnlyAuthorEdits" decoration, requires that a data-table includes a column with the name "AuthorID". Please ensure when you use a decoration that it is used in conjunction with the correct columns, if they are needed.

How do I add decorations to a data-table or column?

This can be done programmatically using SQL at any time. Simply write SQL which includes a DESCRIPTION for the database element formed with the correct syntax, execute the SQL and when your App next starts the Framework will add the specified functionality. All changes to database structure must be made with an "exclusive lock" on the database. This means when the change SQL is executed no other users can be connected to the database. Note that if your database is distributed with multiple instances / nodes, the change SQL must be executed on all instances / nodes.

Many decorations are added when a new Business Object is added to your App. If the the "Create BusinessObject" Form in the System DB Modeller is used to create a new BusinessObject, it will automatically add common decorations. When you are creating a new BusinessObject it is probably easiest to use the [System DB Modeller](#) and [Create BusinessObject](#)

Remember that the change SQL must be run with no connections to your Orix App, using the [Database Management Utility \(OrxDBUtil.exe\)](#)

Examples

```
ALTER TABLE <MyTable>
```

```
ALTER COLUMN <MyColumn> AS INTEGER DESCRIPTION '<Add Description here>'
```

```
ALTER TABLE Contracts
```

```
ALTER COLUMN ProductsID AS INTEGER DESCRIPTION 'ReuseLast'!
```

```
ALTER TABLE Contracts
```

```
ALTER COLUMN ProductsID AS INTEGER DESCRIPTION
```

```
'[Properties]
```

```
ReuseLast=1'
```

Notes on the above

1. When ALTERing a COLUMN, the SQL starts by calling for the whole the table to be altered.
2. Table alterations require an "exclusive lock" on the database. This means these changes can **only** be made when no other users are accessing the database.
3. Decorations are added within single quotation-marks after the key-word DESCRIPTION.
4. If just **one** property is being added, the key-word for that property can be added by itself. If multiple decorations / properties are being added they are written **Windows Ini-file format**, with a heading "Properties" in square brackets, and then other properties of each decoration added in the form <property-name>=<value>.
5. If the decoration adds a boolean / tick-box feature then "<decoration>=1" should be used when the "[Properties]" key-word has been added, as shown with the last example listed above.

Data-table Decoration

```
1 CREATE TABLE "ContractItems"
2 (
3 "ID" INTEGER DEFAULT UID() NOT NULL,
4 "ContractsID" INTEGER,
5 "DateStart" DATE DESCRIPTION '[Properties]
6 ReuseLast=1',
7 "DateEnd" DATE DESCRIPTION '[Properties]
8 ReuseLast=1',
9 "ProductsID" INTEGER DESCRIPTION '[Properties]
10 ReuseLast=1',
11 "ContractItemsTypeID" INTEGER,
12 "PeopleInvolved" VARCHAR(500) COLLATE "ANSI" DESCRIPTION '[Properties]
13 IDList=1
14 IDListTableName=People
15 IDListWHERESQL=Current=true',
16 "BudgetQuantity" FLOAT DEFAULT 0 NOT NULL,
17 "BudgetValue" DECIMAL(19,4) COMPUTED ALWAYS AS BudgetQuantity * UnitValue,
18 "VATValue" DECIMAL(19,4) COMPUTED ALWAYS AS IF(CarriesVAT THEN 0.2 * BilledValue ELSE 0),
19 "UnitValue" DECIMAL(19,4) GENERATED ALWAYS AS IF(Complete = true then UnitValue ELSE
20 IF(Chargeable = false THEN 0 ELSE ProductPrice(ProductsID))),
21 "CarriesVAT" BOOLEAN GENERATED ALWAYS AS (ProductCarriesVAT(ProductsID) AND NOT ContractVATExempt(ContractsID)),
22 "Chargeable" BOOLEAN DEFAULT true NOT NULL,
23 "BilledValue" DECIMAL(19,4) COMPUTED ALWAYS AS BilledQuantity * UnitValue,
24 "BilledQuantity" FLOAT DEFAULT 0 NOT NULL,
25 "Description" CLOB COLLATE "ANSI",
26 "StaffResponsibleID" INTEGER DESCRIPTION '[Properties]
27 ReuseLast=1',
28 "DateCreated" TIMESTAMP DEFAULT Current_Timestamp,
29 "StatusID" INTEGER DEFAULT StatusID('Planned'),
30 "PaidValue" DECIMAL(19,4) DEFAULT 0 NOT NULL,
31 "Complete" BOOLEAN DEFAULT false NOT NULL,
32 "ContractPaymentsID" INTEGER DESCRIPTION 'Dependent, CICPLIST, ContractsID',
33 "PercentWorked" FLOAT GENERATED ALWAYS AS ROUND((SumHours(ID) / BudgetQuantity * 100 TO 2) DESCRIPTION 'PercentBar',
34 "Name" VARCHAR(400) COLLATE "ANSI",
35 CONSTRAINT "PK_ContractItems" PRIMARY KEY ("ID"),
36 CONSTRAINT "ContractsID" FOREIGN KEY ("ContractsID") REFERENCES "Contracts" ("ID")
37 ON UPDATE NO ACTION ON DELETE NO ACTION
38 DESCRIPTION 'Parent',
39 CONSTRAINT "ContractItemsTypeID" FOREIGN KEY ("ContractItemsTypeID") REFERENCES "Types" ("ID")
40 ON UPDATE NO ACTION ON DELETE NO ACTION,
41 CONSTRAINT "ProductsID" FOREIGN KEY ("ProductsID") REFERENCES "Products" ("ID")
42 ON UPDATE NO ACTION ON DELETE NO ACTION,
43 CONSTRAINT "StatusID" FOREIGN KEY ("StatusID") REFERENCES "Status" ("ID")
44 ON UPDATE NO ACTION ON DELETE NO ACTION,
45 CONSTRAINT "ContractPaymentsID" FOREIGN KEY ("ContractPaymentsID") REFERENCES "ContractPayments" ("ID")
46 ON UPDATE NO ACTION ON DELETE NO ACTION
47 DESCRIPTION 'Default',
48 CONSTRAINT "StaffResponsibleID" FOREIGN KEY ("StaffResponsibleID") REFERENCES "Staff" ("ID")
49 ON UPDATE NO ACTION ON DELETE NO ACTION
50 )
51 DESCRIPTION '[Properties]
52 AddDuplicateRecord=1'
53 VERSION 1.00
54 READWRITE
55 IMMEDIATE
```

Data-table decoration

1. "ReuseLast" **Column** level decoration added. Repeated records inserted into this table will reuse the value entered by the user in the previous data-entry.
2. "Dependent, CICPLIST, ContractsID" **Column** level decoration. In the App, the "ContractPaymentsID" field will show a "dependent list", linking to records in ContractPayments.
3. "Parent" and "Default" **Constraint** level decoration. Constraints control data-linkages in Orixia. By describing a constraint as "Parent" the created table (ContractItems) becomes a "child" of the parent table joined by in the constraint (Contracts). This means ContractItems will be displayed in the App as a child-table, and users will be able to see lists of ContractItems linked to Contracts, also if a record is inserted into the ContractItems data-table while a Contracts record is open it will automatically be

3. populated with a ContractsID from the active Contracts record.

The "Default" **Constraint** level decoration. If a ContractPayment record is open in the App, and a new ContractItem is created, the ContractItem's ContractPaymentsID will be set to the value of the currently open ContractPayment record. The "Default" constraint is a scaled down version of the "Parent" constraint, without the built in links in the UI.

4. "AddDuplicateRecord" **Table** level decoration. This results in the addition of an **action** on the Edit Form of this Business Object, creating a menu-item allowing the user to click to create a duplicate record. This is shown in more detail before.

NOTE: Orixa uses decoration of many different parts of the database schema. Decorations can be added to the **TABLE**, **TABLE-COLUMN** and **TABLE-CONSTRAINT**. Orixa also uses decoration of other database schema elements such as **STORED PROCEDURES** and **VIEWS**. These different decorations are detailed in other help-articles.

Decorations which control data

The following decorations alter the behaviour of the data in your App, controlling who can update or change it, or how it is linked together.

OnlyAuthorEdits and OnlyAuthorDeletes (Table-level Decorations)

EXAMPLE

```
ALTER TABLE [Your-Table-Name]
DESCRIPTION '[Properties]
OnlyAuthorEdits=1'
```

Add just the simple text "OnlyAuthorEdits" or "[Properties] OnlyAuthorEdits=1" if other properties are also being added to the data-table DESCRIPTION. "OnlyAuthorDeletes" is added in the same way. This must only be done in tables which contain an "AuthorID" column. The AuthorID holds the ID linked to a specific user who originally created the record.

Once this is done for a data-table, the user who created a record is assigned as its exclusive author and no other users can make changes or delete. Note that these two key-words can be used individually or together for different effects.

It is important to emphasize that this control is implemented at the level of the App, not at the level of the database. If a user opens an "OnlyAuthorEdits" record in the App they will not be able to edit it. However, if an Administrator runs a SQL Script they can still change the data in the record. This set up is used to avoid a situation where records could be "stuck" in the database if a user leaves your organisation.

LockIfCompleteSecurityLevel (Table-level Decoration)

Note that Orixa includes default behaviour for all data-tables that include a boolean / tick-box column with the name "Complete". If such a column exists, once it is ticked all users will be cautioned with a message "Record is marked complete, are you sure you want to start editing" before the Complete tick-box is unticked and they are allowed to start editing. Also built into the Orixa framework is an audit log process. When a record is ticked-complete Orixa records the user who did this. If it is un-ticked (after the warning) Orixa also records this.

These default behaviours provide functionality to "soft lock" records without needing really complex security systems. Having good visibility to show / inform users as to who has changed records when, without requiring "hard" security checks.

The "LockIfCompleteSecurityLevel" **extends** Orixa's default behaviour. Add "LockIfCompleteSecurityLevel=<value>" to the **table** DESCRIPTION. This must only be done in tables which contain a "Complete" tick-box column. Once the new "LockIfCompleteSecurityLevel" is set, if a record is ticked "Complete" and the record is posted to the database it cannot be edited unless a user's SecurityLevel exceeds <value>.

CompleteCondition (Table-level Decoration)

In some situations it can be useful to **automatically** mark a record as complete once some other conditions are met. For example if a user has completed adding data to a number of columns and set a "StatusID" to a particular value.

Add "CompleteCondition=<value>" to the **table** DESCRIPTION to achieve this. The <value> must be a valid SQL WHERE clause.

Example

```
CompleteCondition=WHERE StatusID = 123 AND Memo is NOT Null
```

AddForecasts (Table-level Decoration)

Add "AddForecasts=1" to the **table** DESCRIPTION, and the Forecasting system will be enabled for this data-table.

Full description of how the Forecasting system works can be found here: [Using Forecasts in Orix](#)

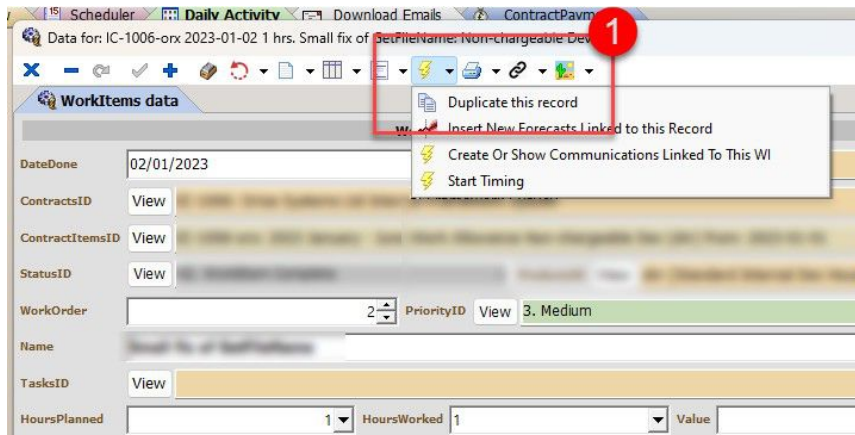
AddDataExport (Table-level Decoration)

Add "AddDataExport=1" to the **table** DESCRIPTION, and the Data-Export system will be enabled for this data-table.

Full description of the Data-Export system can be found here: [The Export Data Window](#)

AddDuplicateRecord (Table-level Decoration)

Add "AddDuplicateRecord=1" to the **table** DESCRIPTION. This will add an "Action" to the Edit-Form of the business-object with the text "Duplicate this record", as shown at 1., in the image below:



Add Duplicate Record Action

Example

```
ALTER TABLE SalesOrders
DESCRIPTION '[Properties]
AddDuplicateRecord=1'
```

NOTE: If a record has child records the user will be asked whether they wish to duplicate these as well as the main record. For example, if a SalesOrder record includes SalesOrderItems the SalesOrder and all SalesOrderItems will be duplicated (if the user says "yes")

SyncCompleteTables, SyncCompleteFields or SyncCurrentTables, SyncCurrentFields (Table-level Decorations)

Add "SyncCompleteTables=<tablename>, <tablename> ..." to a **table** DESCRIPTION. This must only be done if data-tables exist with the names listed, and all the data-tables must contain a "Complete" tick-box column. Then add "SyncCompleteFields=<fieldname>, fieldname ..." Once this is done, when a record in the data-table is ticked complete, the "complete" column in all listed data-tables will also be ticked, based on linking the ID of this table to the IDs listed in <fieldname>.

Exactly the same behaviour is possible for tables that contain the tick-box column "Current" by using the "SyncCurrentTables" and "SyncCurrentFields" key-words.

Example

```
ALTER TABLE SalesOrders
DESCRIPTION '[Properties]
SyncCompleteTables='SalesOrderItems'
SyncCompleteFields='SalesOrdersID'
```

In the above example, ticking the "Complete" column of a SalesOrder will immediately tick the "Complete" column of **all** SalesOrderItems linked to that SalesOrder by their SalesOrderID.

Note: There may be other database actions triggered by these changes. For example ticking complete for an OrderItem might finalize some computations for the record. These consequences must be analysed when designing your App.

SecurityLevelInsert, SecurityLevelEdit, SecurityLevelDelete, SecurityLevelVisible (Table and Column-level Decorations)

Add "SecurityLevelInsert=<value>" to a **table** DESCRIPTION, <value> should be a simple whole number such as 10, 90, etc. Once this is done only users with a SecurityLevel greater than <value> will be able to Insert, or in the case of SecurityLevelDelete Delete, the record.

SecurityLevelEdit can be applied in the same way as SecurityLevelInsert but is different because it can be applied to **tables or columns**. Once this is done the data in the individual column can only be edited by users with a SecurityLevel above <value>.

SecurityLevelVisible can be applied in the same way as SecurityLevelEdit, but is different because it can only be applied to **columns**. Once done users with security level less than <value> will not see these fields in edit-windows.

Note that with the **SecurityLevelDelete** constraint a **newly inserted** record **can** be deleted. Once data has been added to the record it becomes fixed, so users cannot delete it. This is done to allow "undo" processes following accidental addition of data.

Parent and Default (Constraint-level Decorations)

Add "Parent" or "Default" as a DESCRIPTION to a Constraint in a data-table, as shown at 3. in the image above. Constraints are used to link data-tables, allowing relationships between data. Constraints control the behaviour of your App, as the App uses them to define linkages and relationships. Details of how this works can be found in other parts of the help.

If a Constraint has an additional "Parent" or "Default" decoration added this extends the normal behaviour.

For Constraints marked as "Parent" it is assumed that these point to a controlling / parent data-table. For example SalesOrderItems could have a Constraint on SalesOrdersID linked to the SalesOrders data-table. If this was decorated as a "Parent" then SalesOrderItems would display alongside SalesOrders in the System Entities screen of your App, and SalesOrders would automatically have a "child-data-grid" view added to allow viewing SalesOrderItem records. Also, adding a New SalesOrderItem record from SalesOrders would automatically populate the SalesOrdersID in SalesOrderItems, defining a clear link between the parent and child data-table.

Use "Parent" as a decoration on a Constraint where there is a really tight binding between the parent and child data-tables. Where you want the two data-tables to be thought of and dealt with together by users.

The "Default" decoration key-word is similar to "Parent" except that the two tables are not grouped together on the System Entities screen. Default decoration allows a parent child relationship to be created, without the child being viewed together with the parent.

For example you might use the "Default" decoration when linking the Products data-table to the SalesOrderItems data-table. This allows users to view grids of SalesOrderItem records from Products, and means that if a Product record is open, when a new SalesOrderItem record is created the two will be linked, but it does **not** mean that you can create a new SalesOrderItem from the Products entity.

Using Parent and Default decorations has some complications and needs to be thought through carefully when architecting an App. For example Orixia can support infinite layers of parent - child - grandchild -great-grand-child, with interlinkage. However in the System Entities screen only Parents and Children are physically grouped together on the screen, as more complex grouping of entities can just be too confusing. Also, Orixia only supports child data-tables being parented to a single parent, for simple reasons of practicality. A data-table can have as many "Default" linkage relationships as necessary.

ReuseLast (Column-level Decoration)

Add the key-word "ReuseLast" or add "ReuseLast=1" after the "[Properties]" key-word, as shown at 1., in the image above.

Once this is done, the framework will store a record of the last value used in this column for each user. When a new record is added this prior value will be used to set the value of this column.

Many data-entry tasks are highly repetitive. Data for the same Customer or Person needs to be added multiple times. In such cases this key-word speeds up data-entry.

SoftUniqueCheck (Column-level Decoration)

Add:

```
"[Properties]  
SoftUniqueCheck=1"
```

To a Column definition to add a check whether a field in a table is unique **without blocking the action completely**. A CONSTRAINT can be added to the database to force a hard block on identical data in one or more fields. To create a "soft check" add the above decoration, then if a column has a value that is shared by other records in the table the user will be warned, but allowed to continue if they want to.

CurrentUser (Column-level Decoration)

Add the key-word "CurrentUser" or add "CurrentUser=1" after the "[Properties]" key-word, as shown for ReuseLast at 1., in the image above. This should only be done for ID / integer fields which can be used to store the ID of the currently connected user. Typical Columns on which this decoration would be added might be "InspectorID" or "StaffID" where the record is very likely to be created by the person recorded as "Inspector" or "Staff" for this record.

Once this is done, whenever a record is edited or inserted the ID of the current user will be used to set the value of this column.

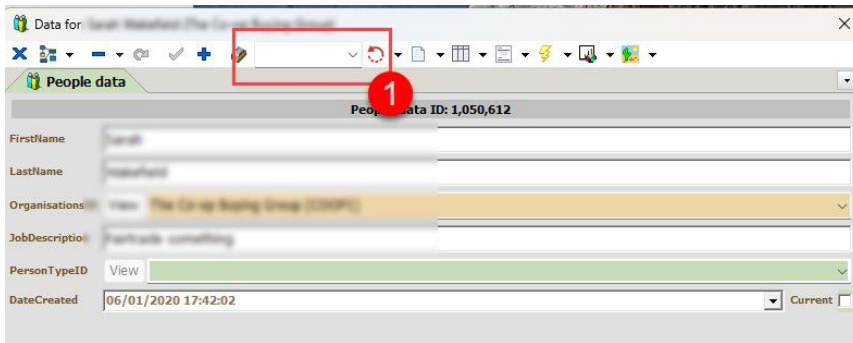
ReadOnly (Column-level Decoration)

Add the key-word "ReadOnly" to fix a column so that it's value can only be set programmatically. Note that if a column is already set using the COMPUTED or GENERATED key-words, it is not necessary to add the "ReadOnly" description, as the column will already be read-only by design.

Decorations which control data-display and selection

The following decorations control the user-interface of the Edit-Windows of your App, changing how users can select data.

HideEditFormSearchPanel (Table-level Decoration)



Edit Form Search Panel

By default the "Edit Form Search Panel" (shown in the image at 1.) is added to all Edit Forms. If the data-table is very large, or not usually searched in this way, the developer can **remove** or **hide** this feature using the following table decoration:

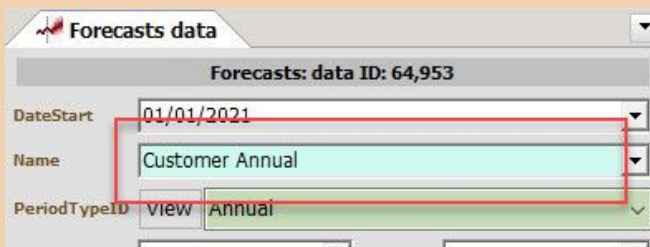
```
ALTER TABLE EditHistory
DESCRIPTION
'[Properties]
HideEditFormSearchPanel=1'
```

Note that the EditHistory data-table is automatically set to this feature as it is a really large table. Also, the EditFormSearchPanel will **not** appear unless the BusinessObject has a "ListScript" used to generate the list of records to choose from.

Distinct (Column-level Decoration)

Add the "Distinct" key-word to any VARCHAR column in the data-tables of your App. This will change the way a field is displayed in the edit window. Instead of a simple text-edit being shown, a light-blue edit will be shown, with a drop-down list that contains all previous entries in the table.

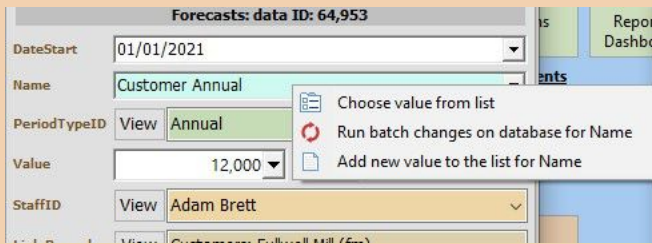
Note that drop-down lists come in several flavours in the Orixia Framework. The "Distinct list" is the simplest, as it simply works on the basis of retrieving all previously entered values in this column. If you want to create a short list of choices it is probably better to create a "Types" lookup list and store the data in an "ID" column.



Distinct Combo

- It is shown in a distinct light blue colour
- If the user clicks on the arrow on the right a list of previously entered values is shown.
- If the user right-clicks on the combo-box, a menu with 3 choices is shown. These give the user options to choose a value from a list, run batch changes to the data in this column, or add a new Distinct value.

If the "Distinct" decoration is added to a field, the framework automatically adds a specialized editOr to the edit-window.



Distinct Combo Menu

URL (Column-level Decoration)



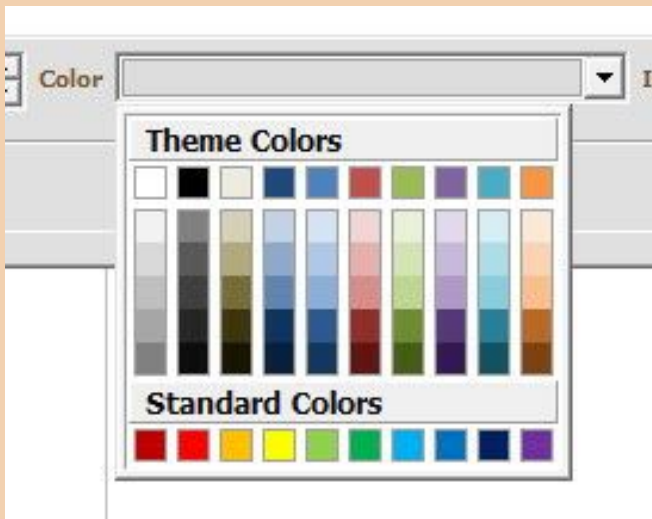
Website / URL Edit

Add the "URL" key-word to the Description of any VARCHAR column in the data-tables of your App. This will change the way a field is displayed in the edit window.

The decoration can also be added in the form:

```
ALTER COLUMN "[ColumnName]" AS VARCHAR(200)
DESCRIPTION 'Properties]
DisplayFormat=URL' !
```

ColorCombo (Column-level Decoration)



ColorCombo Colour Picker

ColorCombo

Add the "ColorCombo" key-word to the Description of any INTEGER column in the data-tables of your App. This will trigger the App to show a control like the one on the left. The App will store color-data picked by users, and allow users to select colors and link them to data-records.

Clicking on the arrow on the right-hand side of the Color-Combo shows a "Theme" colours and "Standard" Colours.



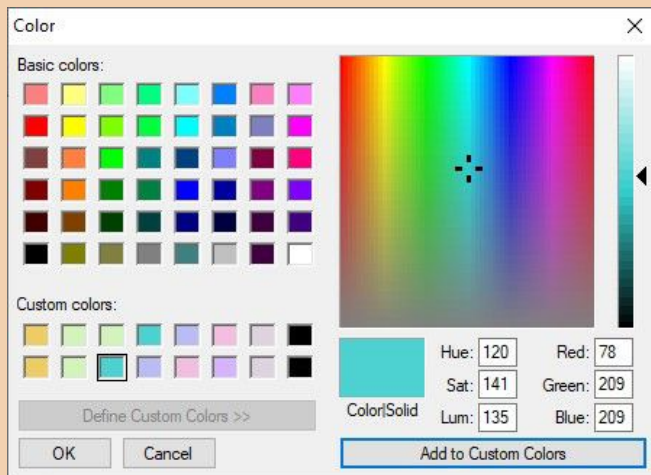
Non-Standard Colours from ColorCombo

Right-clicking on the ColorCombo shows a "Non-standard colours" menu. If the user clicks on this a "Color Dialog" opens with system color-picking tools, allowing the selection of any color.

The decoration can be added two ways:

```
ADD COLUMN "[ColumnName]" INTEGER DESCRIPTION
'ColorCombo' !
```

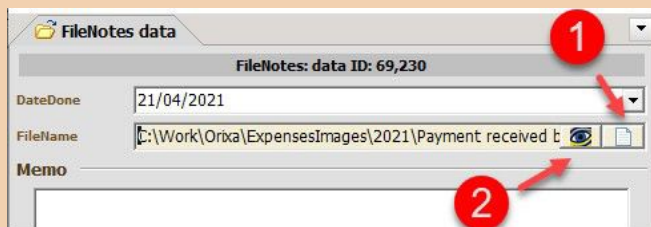
```
ADD COLUMN "Color" INTEGER
DESCRIPTION '[Properties]
```



Non-Standard Color Dialog

DisplayFormat=ColorCombo ' !

The first "shorthand" can be used if this is the only decoration being added to the field. The second longer form can be used if multiple decorations are being added.



FileName Combo

1. Click the "New" button to add a new File-name. A dialog-window will open allowing you to navigate to any accessible file.
2. Click the "View" button to open the file.

Filename Combo (column level decoration)

Adding the decoration:

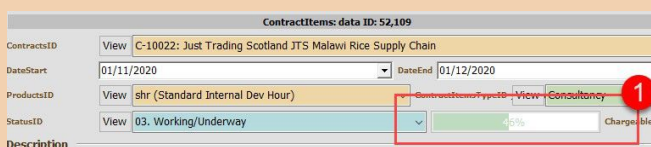
```
ALTER COLUMN "[ColumnName]" AS VARCHAR(200)
DESCRIPTION 'FileNameCombo'
```

or:

```
ALTER COLUMN "[ColumnName]" AS VARCHAR(200)
DESCRIPTION '[Properties]
FileNameCombo=1'
```

Automatically adds an edit-field which allows the user to select files from disk and include links to these files in their data-records.

Note: Orixa has the built in "FileNotes" mechanism, which allows data-records to be linked to any number of files and for these records to be viewed in different contexts in your App. Only use the "FileNameCombo" decoration where you need greater individual control than this.



Percentage Used Combo

Percentage Used Combo (column level decoration)

Adding the decoration "PercentUsed" or

```
"[Properties]
PercentUsed=1"
```

Creates a display on-screen of a "percentage bar."

This **must** be linked to a number / float field in the database which contains percentage-based numbers between 0 and 100. Numbers outside this range will display as 0 or 100. With suitable data, this combo shows a nice display of items such as the level of "completion" of a task, if it can be detailed in this way.

DayOfWeek Edit (Column level decoration)

An edit which shows a list of days in order, and maps these onto integers 1 to 7 stored in a database.

DayOfWeek Edit

1. A "ReadOnly" DayOfWeek edit, which automatically shows the day for the selected "DateStart"
2. A DayOfWeek edit in an Orixa Tasks-Management-System. In this case the user picks a "Frequency" (Daily, Weekly, Monthly etc.) and a DayOfWeek. This image shows the edit in its "dropped-down" position.

Orixa has the built in assumption that if a table contains a column **named** "DayOfWeek", in the Edit Window a DayOfWeek edit will be added. This will show a list of days of the week the user can pick from. For columns with other names, the same behaviour can be achieved by adding a "DayOfWeek" decoration to the column's description.

To define a DayOfWeek column in your table use the following SQL:

```
ADD COLUMN DayOfWeek INTEGER
```

You can also set any column to display a DayOfWeek Edit by adding the decoration "DayOfWeek" in the Description:

```
ADD COLUMN SelectedDay INTEGER DESCRIPTION '[Properties]
ReadOnly=1
DayOfWeek=1'
```

As with other Orixa systems, you can add "just" the text DayOfWeek, or add DayOfWeek=1 in a list under the heading [Properties].

How the DayOfWeek Edit Works

It is important to understand that if a DayOfWeek Edit is used, Orixa does not store the whole text of the days in the database. Instead it stores the days as integer (whole number values) between 1 and 7, where 1 = Monday, 2 = Tuesday etc.

This means when a Developer wants to Query the DayOfWeek to return data in Orixa they must use these numbers, not the text version of the day-name for the query. Note that part of the reason for this is that the SQL standard for days of the week also uses numbers from 1 to 7. Therefore using these same numbers in Orixa makes other SQL operations much easier.

Also, If they want to use the DayOfWeek in a Grid or other Resource they must use a database function in the SQL to show the text version in the Query Result.

Returning the day of the week in a Query

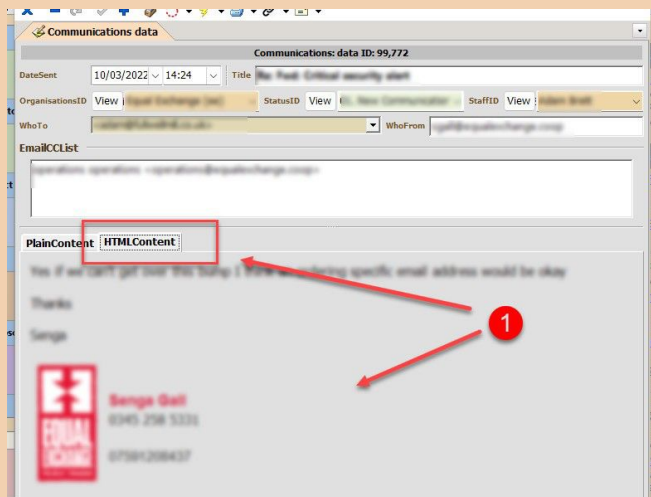
```
SELECT
    DayOfWeekName(DayOfWeek) as DayOfWeek,
    Name
FROM Tasks
WHERE DayOfWeek IN (4, 5)
```

The above SQL will return a readable text as the "DayOfWeek", and returns only those tasks that are scheduled for Thursday or Friday.

HTML (column-level Decoration)

Adding the decoration "HTML" or "DisplayFormat=HTML" forces the system to display the data stored in a CLOB field as if it was a web-page containing HTML.

Any data stored in this field will be displayed exactly like the data you are reading right now. Users can type and enter data into HTML columns, and it is displayed in a format similar to "rich-text", or standard HTML web-pages.



HTML Content in an Edit Form

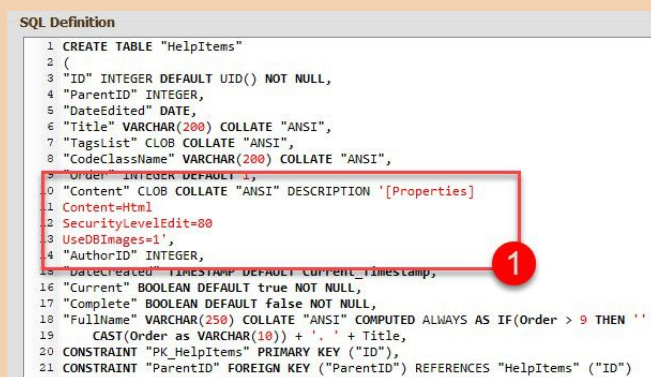
HTML Content

Basic HTML Formatting (as shown at 1., in the image) can be useful to allow users to input data with more complex content than simple text.

If users want to add simple font-styling, such as bold, headings etc., or wish to add images and links an "HTML Content" format can be very useful.

Once a database table is suitably decorated, Orixa will automatically display the content of data in this format.

Orixa applies a standardized **style sheet** to the HTML Column. This style sheet is stored in the ConfigurationSettings system-table. Developer users are free to re-write and edit this style sheet to extend its capability.



HTML Content SQL Decoration

SQL Decoration needed for HTML Content

Adding a DESCRIPTION with "Control=Html" to the SQL definition of your table in Orixa switches on the HTML Content control.

[Properties]

Content=Html

SecurityLevelEdit=80

UseDBImages=1

1. "Content=Html" this is all that is needed to switch on HTML Content for a field.
2. "UseDBImages=1" if this setting is added the HTML content will accept "HREF LINKS" in the form "img src="ID=1234" and display images stored in the "Images system-table. Just add the ID for the chosen image and it will display.

The HTML Editor will also accept "raw" binary data as images, and allows users to copy and paste images into HTML Content. Note that this does result in large amounts of data being linked directly into the database. The Orixa Images database is optimized to minimize data waste, so users are encouraged to try to use the images database to store images.

Tags-Search (Column-level decoration)

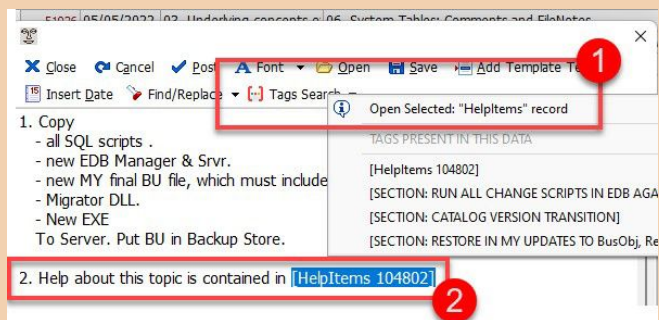
For more "techie" Apps, the ability to track tags in longer sections of text can be useful. The User types text inside square-brackets. The App keeps track of these tags, and allows the User to jump to them quickly from any location in the text.

Tags can also be formatted to allow users to jump to completely different records in their App.

This feature can be useful when an App includes something like a series of "how-to" procedures. The user can read an entry in the App and jump from it to many other locations using the tag.

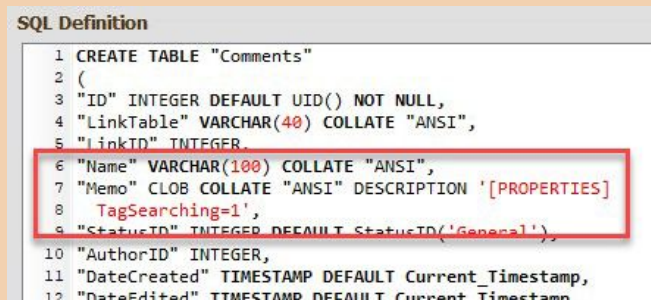
Tags Search in an App

1. In the Memo editor a "Tags-search" button will appear. Clicking on it will open a menu listing "Tags present in this data". Click on any of these items to jump to that point in the text.



Tags Search link to Specific BusinessObject Record

2. Once you click, the section of text you jumped to will be highlighted. If this is a tag containing the name of a BusinessObject and it's ID, then the user can "jump" direct to this record.



Tags Search SQL Definition

SQL Decoration needed for Tag-search

```
"Memo" CLOB COLLATE "ANSI" DESCRIPTION
'[ PROPERTIES]
TagSearching=1',
```

Simply adding "TagSearching=1" to the DESCRIPTION of the table-column will add this functionality to a CLOB (long-text) column

IDList (column-level decoration)

An "IDList" is a data-column in a table which stores a list of integer values. These integer values are IDs that link to records in the App Database. An IDList is a useful way to store data such as "PeopleInvolved" or "CountriesOfOrigin" where several items may be present, each of which are held in another data-table.

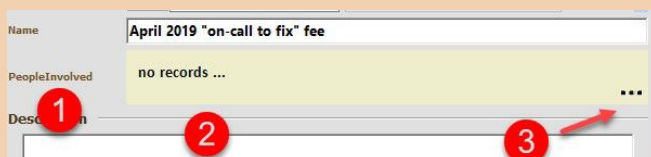
The decorations that can be used to create an IDList are flexible as it can be added in a number of ways.

IDLists allow the addition of a single CLOB field to a data-table, into which multiple IDs can be inserted and stored from any other BusinessObject in the App.

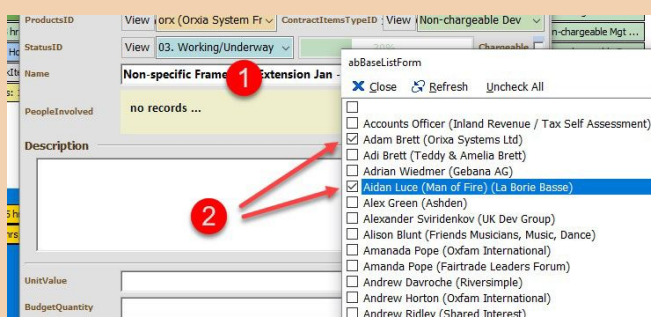
Note that with IDLists, data is stored in the database as a list of IDs. Data will display in Edit Windows using the **DisplayScript** for the BusinessObject.

If you wish to display this data outside of an edit window in other situations you must use the system Function `IDListToText(<fieldName>, <tablename>)`. This converts the IDs stored in the database into a comma-delimited list of display-names.

Care should be taken when adding IDList fields, as it can be hard to show lists of records for users to add / remove if the data-tables are very large. The "IDListWHERESQL" property can be used to limit the size of the list of records the user sees.



IDList Combo



Picking Records for an IDList

The combo will use the BusinessObject's List Script to create the list of possible records.

The decoration's optional Where script clause can be used to add a WHERE statement to reduce the number of records shown in the pick list.

1. "PeopleInvolved" field has IDList Decoration added.
2. Combo displays with light yellow colour.
3. click on ellipsis to open a list of People to pick from.

Once the pick-list is open, selection is done in the same way as in other parts of an Orix App

1. Double-click on the list or click on the ellipsis to open the list.

2. Once the list is open, hold down the Shift key and click as many records as you want to select.

```
CREATE TABLE "NutritionalDetails"
(
  "ID" INTEGER NOT NULL,
  "ManufacturedOnSite" BOOLEAN DEFAULT true NOT NULL,
  "Memo" CLOB COLLATE "ANSI",
  "DateDone" DATE DEFAULT Current Date,
  "Unit" VARCHAR(50) COLLATE "ANSI",
  "Source" VARCHAR(50) COLLATE "ANSI",
  "GeneticModificationID" INTEGER DEFAULT 146/246,
  "AdditivesLUList" VARCHAR(500) COLLATE "ANSI" DESCRIPTION 'IDList',
  "PreservativesLUList" VARCHAR(500) COLLATE "ANSI" DESCRIPTION 'IDList',
  "MicroTVGID" INTEGER,
  "MicroEntColiformsID" INTEGER,
  "MicroFColID" INTEGER
)
```

Example of simple Description to add "IDList"

Because many IDLists draw their data from the Types system-table which is used to hold lists, a short-hand syntax for adding a Types-linked DList is possible.

Add the simple decoration "IDList" to the column Description and the system will assume that this field will contain a list of names from the Types system table, constrained by the LinkField and LinkTable of the current Column and Table.

Records for the users to pick can then be added to the Types system-table, and users will be able to rapidly add a selection of choices from this list.

```
CREATE TABLE "Products"
(
  "ID" INTEGER DEFAULT UID() NOT NULL,
  "Image" BLOB,
  "ProdCode" VARCHAR(20) COLLATE "ANSI" NOT NULL,
  "Name" VARCHAR(100) COLLATE "ANSI" NOT NULL,
  "PublicDescription" CLOB COLLATE "ANSI",
  "PrivateDescription" CLOB COLLATE "ANSI",
  "MinOrderSize" FLOAT DEFAULT 0 NOT NULL,
  "WarehouseConcerns" CLOB COLLATE "ANSI",
  "CountriesOfOriginIDList" VARCHAR(500) COLLATE "ANSI" DESCRIPTION 'IDList, Countries',
  "HeightMetres" FLOAT DEFAULT 0 NOT NULL,
  "WidthMetres" FLOAT DEFAULT 0 NOT NULL
)
```

Example for "CountriesOfOrigin"

You can also create an IDList by adding the decoration: "IDList, <LinkingTableName>" to any column.

IDs for Records in <LinkingTableName> will then be stored in the field, and will be made visible in the App.

Users will be able to select Records to show in the field from a pop-up list, and once selected will display in the field of the edit-window.

The list will automatically contain **all** the records in the list, meaning this form of decoration is only suitable for data-tables with few records.

The LinkingTable must have a BusinessObject record which includes a DisplayScript. This will be used to convert the IDs stored in the database into human-readable data.

```
SQL Definition
1 CREATE TABLE "ContractItems"
2 (
3   "ID" INTEGER DEFAULT UID() NOT NULL,
4   "ContractsID" INTEGER,
5   "DateStart" DATE DESCRIPTION '[Properties]'
6   Reuselast=1,
7   "DateEnd" DATE DESCRIPTION '[Properties]'
8   Reuselast=1,
9   "ProductsID" INTEGER DESCRIPTION '[Properties]'
10  Reuselast=1,
11  "ContractItemsTypeID" INTEGER,
12  "PeopleInvolved" VARCHAR(500) COLLATE "ANSI" DESCRIPTION '[Properties]'
13  IDList=1
14  IDListTableName=People
15  IDListWHERESQL=Current=true',
16  "BudgetQuantity" FLOAT DEFAULT 0 NOT NULL,
17  "BudgetValue" DECIMAL(19,4) COMPUTED ALWAYS AS BudgetQuantity * UnitValue,
18  "VATValue" DECIMAL(19,4) COMPUTED ALWAYS AS IF(CarriesVAT THEN 0.2 * BilledValue ELSE 0),
19  "UnitValue" DECIMAL(19,4) GENERATED ALWAYS AS IF(CarriesVAT THEN 0.2 * BilledValue ELSE 0)
```

Fuller "Properties" based IDList Definition

If you need to add multiple decorations to one field, or you need to add a "WHERE" clause to limit the size of the list of records, add a "Properties" section to the data-field's DESCRIPTION

```
"PeopleInvolved" VARCHAR(500)
DESCRIPTION '[Properties]'
IDList=1
IDListTableName=People
IDListWHERESQL=WHERE Current=true'
```

The above SQL:

1. Creates and IDList
2. Links it to the "People" table, so it will display the data selected by the DisplayScript of the People BusinessObject record.
3. The IDListWHERESQL property has been set so that when the user opens the list to show "People" only those marked "Current" will be shown in the list.

Dependent lookup (Column-level decoration)

This is a more complex look-up list, based on not just a link to an external table, but also to a column in the current table.

Adding the Decoration "Dependent, <ListSQLScriptResourceName>, <LinkedIDFieldName>" to the Description of any column adds this specialized

look-up list, where the list generated is dependent on both the existing record and the value of a **second** column, the <LinkedIDFieldName>. The <LinkedIDFieldName> must be a column in the data-table. The <ListSQLScriptResourceName> must be the name of a Resource record in the Resources system-table. Orixa will use this Resource record's SQLStr to generate the list, and pass in the ID of the second <LinkedIDFieldName> column. The SQLStr must include the %d wild-card as part of a WHERE clause into which the ID integer of <LinkedIDFieldName> value will be passed, and must return a dataset with 2 columns, a "Name" or "Fullname" and an ID.

To understand the Dependent Lookup, a practical example is easiest: think of a data-table "StockCounts" which records a company's stock. Staff want to be able to record stock-items (Products) but link them to specific Purchase-records for that Product for traceability. Therefore the StockCounts table includes a ProductsID (linking to the Products data-table) and a PurchaseItemsID linking to PurchaseItems. If normal Orixa behaviour occurred, these constraints would create look-up lists that could contain **every** record in the Products and PurchaseItems data-tables. However this is not the desired behaviour. Once a Product has been selected, staff only want to see PurchaseItems **for the chosen product** so the content of the PurchaseItems look-up list is Dependent on the choice of Product.

This form of look-up list is extremely useful, so Orixa has automated its creation. Decorate a field correctly and the "Dependent" look-up list will populate based on the contents of a table limited by some ID in one of its columns.

Dependent look-up in an App

An Edit Window with a Dependent Look-up.

Note the distinctive cream/brown colour used by a Dependent Lookup.

In the example shown, the selection of the **Contract** from the ContractsID look-up list populates the "ContractItemsID" look-up list with only those **ContractItems** that are present in the chosen Contract.

This is an extremely useful programatic pattern, allowing "sub-lists" to be generated, such as "Staff working on this project" or "Products Available to this customer"

Dependent look-up Worked Example (using the above "ContractItemsID" case)

```
1 CREATE TABLE "WorkItems"
2 (
3   "ID" INTEGER DEFAULT UID() NOT NULL,
4   "StaffID" INTEGER DESCRIPTION '[Properties]'
5   CurrentUser=1,
6   "DateDone" DATE DEFAULT Current_Date,
7   "ContractsID" INTEGER DESCRIPTION '[Properties]'
8   ReuseLast=1,
9   "ContractItemsID" INTEGER DESCRIPTION 'Dependent, CIDependentLUList, ContractsID',
10  "StatusID" INTEGER COMPUTED ALWAYS AS IF((DateDone < Current_Date AND HoursPlanned = 0),
11    IF(HoursWorked = 0 THEN 62917 ELSE 62918)),
12  "Name" VARCHAR(250) COLLATE "ANSI",
13  "WorkOrder" INTEGER DEFAULT 1 NOT NULL,
14  "HoursWorked" FLOAT DEFAULT 0 NOT NULL.
```

Simplified "Dependent" definition

Simplified SQL definition for the "WorkItems" data-table

Note the section beside "ContractItemsID" which reads:

"ContractItemsID" INTEGER DESCRIPTION
'Dependent, CIDependentLUList, ContractsID'

The reference "CIDependentLUList" is used by the framework to locate the resource used to populate the list.

```
1 CREATE TABLE "Attendance"
2 (
3   "ID" INTEGER DEFAULT UID() NOT NULL,
4   "DateDone" DATE,
5   "StaffID" INTEGER,
6   "WagesID" INTEGER DESCRIPTION '[Properties]'
7   Dependent=1
8   ResourceName=AttendanceDependentLUList
9   TargetField=StaffID
10  TargetTable=Staff
11  SecurityLevelVisible=80',
12  "DayTypeID" INTEGER,
13  "Hours" FLOAT DEFAULT 0 NOT NULL,
14  "OvertimeHours" FLOAT DEFAULT 0 NOT NULL,
```

Longer Form Dependent Lookup definition

Fuller SQL Definition, for "Attendance" data-table.

Note that in this case the "Dependent" lookup is also mixed with a "SecurityLevel" setting, to limit visibility.

Field Definition

"WagesID" INTEGER DESCRIPTION '[Properties]
Dependent=1
ResourceName=AttendanceDependentLUList
TargetField=StaffID TargetTable=Staff
SecurityLevelVisible=80'

In **both** of the above versions resource SQL must be added using the "ResourceName" to give your App SQL to create the list.

Name	AttendanceDependentLULList
SQLStr	Description
1 SELECT	
2 CAST(W.DateDone as VARCHAR(10)) + ' ' +	
3 P.FullName,	
4 ID	
5 FROM Wages W	
6 LEFT JOIN People P ON (W.StaffID=P.ID)	
7 WHERE P.ID = %d	
8 ORDER BY FullName	

Resources Record used by Dependent lookup

Note how the database definition and decoration interacts with the framework system tables.

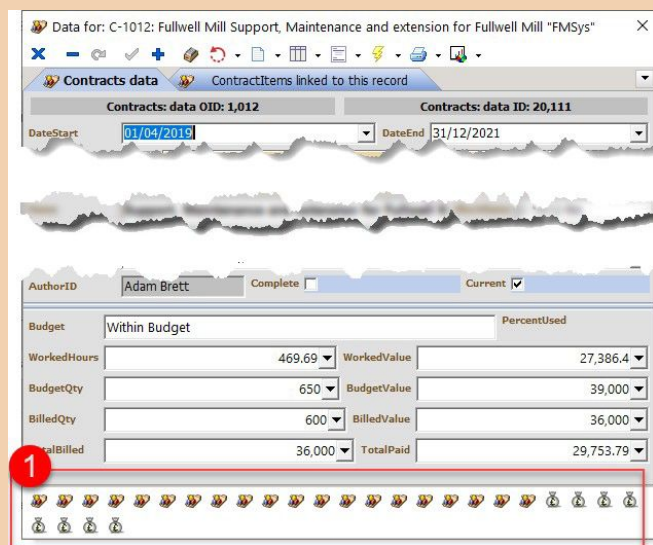
If this Dependent Lookup-List decoration is defined, but no Resources data-record is added the system will show an error "Resource not Found" and ask you to add it.

```
SELECT
    CAST(W.DateDone as VARCHAR(10)) + ' ' +
    P.FullName,
    ID
FROM Wages W
    LEFT JOIN People P ON (W.StaffID=P.ID)
WHERE P.ID = %d
ORDER BY FullName
```

Note the %d wild-card character is present, the framework uses when it creates the list, substituting in the StaffID for the selected record.

Controlling appearance of the "Shortcut Links" (Table level decoration)

Orixa works to make the links between data in your App very transparent, by adding a bar with "Shortcut Links" at the bottom of each Edit-Form. This immediately shows users icons representing items connected to the data they are viewing, with one icon for each connected record. The Icon set in the BusinessObjects data-table are used to represent each linked record, to make Shortcut Links a bit more user-friendly.



LinksListBar Visible in an App

The default behaviour of an Orixa App is to show all links in the Shortcut Links, as shown at 1., in the image on the left. Generally this is the best choice.

However some highly connected BusinessObjects can end up with very large numbers of linked records. In this case you may want to switch off / hide Shortcut Links. You can do this by adding a simple decoration to your data-table's description.

It can be useful to switch off Shortcut Links for very highly inter-connected BusinessObjects, where the resulting LinksListBar is very highly populated, making it hard to read, and possibly slow to fetch from the database.

Remember: If a BusinessObject is a Master with Extension BusinessObjects linked to it, you can switch off Shortcut Links for each data-table separately by adding to each tables DESCRIPTION.

1 CREATE TABLE "ContractItems"	
2 (
3 "ID" INTEGER DEFAULT UID() NOT NULL,	
4 "ContractID" INTEGER,	
5 "DateStart" DATE DESCRIPTION '[Properties]	
6)	
47 DESCRIPTION 'u fault',	
48 CONSTRAINT "StaffResponsibleID" FOREIGN KEY ("StaffResponsibleID") REFERENCES "Staff" ("	
49 ON UPDATE NO ACTION ON DELETE NO ACTION	
50)	
51 DESCRIPTION '[Properties]	
52 AddDuplicateRecord=1	
53 ShowLinksBar=0	
54 VERSION 1.00	
55 READWRITE	
56 UNENCRYPTED	

"ShowLinksBar" data-table Description SQL

In the Image the "ShowLinksBar=0" statement switches off the LinksListBar so it will be hidden.

Example SQL

```
ALTER TABLE ContractItems
DESCRIPTION '[Properties]
AddDuplicateRecord=1
ShowLinksBar=0 '
```

Add "Export Data from ..." option to the BusinessObject

To add a new Action allowing users to open a window to generate SQL statements that in turn create CSV Files which can be exported from an Orixia App, add the following SQL decoration to the data-table:

```
DESCRIPTION '[Properties]
AddDuplicateRecord=1
AddDataExport=1'
```

Custom Lookup List (Column-level decoration)

Add a decoration to a column in the form: 'Custom, <Linkingtablename>, <ListSQLScriptResourceName>' to a column's Description to create a customized look-up list on any ID column in a data-table which you wish to link to another data-table. The <ListSQLScriptResourceName> Resource record must exist with a valid SQLStr (as explained for the "Dependent" lookup list above). The data-table can also contain a Constraint linking the column to the linked table, as explained in the data-design sections of the help, but this is not required.

The purpose of this decoration is to allow customization of the look-up lists that are created by default by Orixia. Orixia's default look-up list uses the "ListScript" of the look-up table's BusinessObject record. This is usually adequate. However there may be a need to further restrict the list (if a table contains very large numbers of records) or to personalize a list (for example limiting it to show only records edited by the CurrentUser). In such cases add the "Custom" column-decoration, and a suitable Resources system-table data-record, and the look-up list will be populated as your system requires.

Types lookup List Customization (Column-level decoration)

Orixia includes the built-in idea of a "Types" lookup list, which allows records to be set into categories, such as "ProductType" or "CustomerType". These are easy to create and do not require any special coding, simply add an Integer Column and a Constraint to the Types table in the following form:

```
ALTER COLUMN "ContractsTypeID" AS INTEGER,
CONSTRAINT "ContractsTypeID" FOREIGN KEY ("ContractsTypeID") REFERENCES "Types" ("ID")!
```

However it may be that additional control is needed. For example, 2 different data-fields may want to share a list of "Types". To achieve this, add a decoration to the Description of a column in the form: 'Types, <LinkTableName>, <LinkFieldName>' to any ID-field to override the standard operation of a "Types" look-up list. The linking Constraint to the Types data-table is still required as before.

Using the Types lookup list customization decoration allows multiple table-columns to share a single list in the Types system-table.

The Types system-table is used to hold lists to categorize data. For example PersonTypeID, or ProductsTypeID. Orixia uses a link on the ID of the Types table to link to the Name field, which is displayed with the record. Orixia uses the name of the table and the name of the field (LinkTable and LinkField) to limit the list, so users only see correct items to choose from.

Sometimes tables share items they need to list. As a slightly simplified example a "StockCounts" table might include a "WarehouseLocationID", where names of WarehouseLocations are held in the Types system-table. If another table, say StockMovementItems is given "WarehouseLocationInID" and "WarehouseLocationOutID" columns, these would not normally see the warehouse location records for "StockCounts", it would see its own lists. This would require duplication of the records in the Types table, and would make it hard to run queries linking the two data-tables via their WarehouseLocationID. Duplication of data is a potential source of errors and should be avoided.

To solve this issue Orixia allows TypeID fields to link to customized lists, which are based on a single list of values, but potentially linked to many columns in many data-tables.

In the example above the two tables would have the following definitions:

```
CREATE TABLE "StockCounts"
(
  "ID" INTEGER DEFAULT UID() NOT NULL,
  ...
  "WarehouseID" INTEGER,
  ...
  CONSTRAINT "WarehouseID" FOREIGN KEY ("WarehouseID") REFERENCES "Types" ("ID")
)

CREATE TABLE "StockMovementItems"
(
  "ID" INTEGER DEFAULT UID() NOT NULL,
  ...
  "WarehouseInID" INTEGER DESCRIPTION 'Types, StockCounts, WarehouseID',
  "WarehouseOutID" INTEGER DESCRIPTION 'Types, StockCounts, WarehouseID',
  ...
  CONSTRAINT "WarehouseInID" FOREIGN KEY ("WarehouseInID") REFERENCES "Types" ("ID")
)
```

```
CONSTRAINT "WarehouseInID" FOREIGN KEY ("WarehouseInID") REFERENCES "Types" ("ID")
)
```