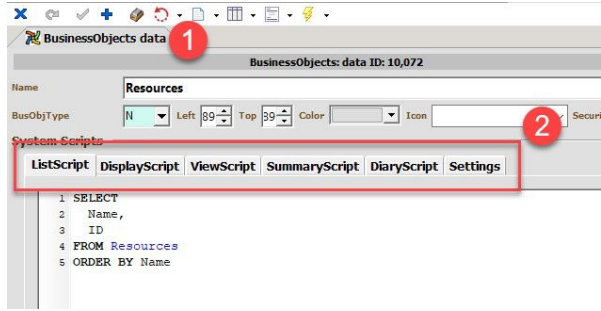


# The Purpose of the BusinessObjects Scripts and how to use them

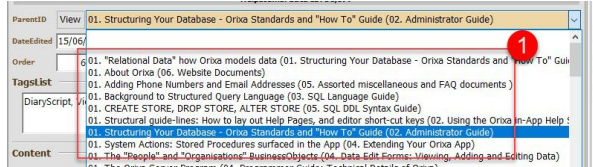
The Orixa framework uses the BusinessObjects framework-table to create your App. The "Script Columns": ViewScript, ListScript, DisplayScript, SummaryScript and DiaryScript contain SQL which is used by your App to display data in different situations.



BusinessObjects framework-table Edit Form

1. Developers can access the "BusinessObjects" framework table like any other data-table in an App.
2. The data-columns which contain the key Script columns are then laid out as tabs, each with its own SQL-editor. The scripts can be edited directly in this window, with the help of SQL Editing tools built into the SQL editor.

## ListScript



Data from a ListScript displayed in an App

### Purpose

Wherever a List of the records in a BusinessObject is displayed (as in the image above, at 1.), this script will be used to create the list. The many "lookups" that exist in Orixa use this script, and it is used when Search-boxes and other windows open for a user to select one or more records from the data.

A few BusinessObjects which are never viewed in Lists in your App may have a blank ListScript, but this is rare.

### Structure

Orixa requires that all ListScripts contain 2 fields, a "Name" or "FullName" field and an "ID" field. The ID field is always the ID of the chosen BusinessObject, the "Name" field can be far more complicated. For example a SalesOrders BusinessObject might include the name of the Customer, a date, a value

or other data in a List Script.

It is often useful to add a "Name" or (computed) "FullName" field to an Orixa data-table, as this makes the creation of these resources easier.

A ListScript must be **ordered**. It is usual to add "ORDER BY Name" or "ORDER BY FullName" to the end of the list script.

## DisplayScript (required)



Data from a ListScript displayed in an App

### Purpose

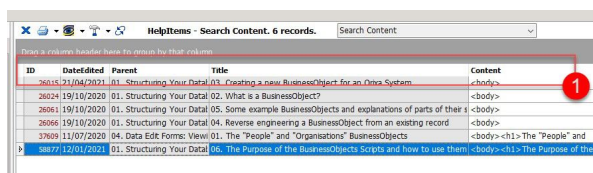
Wherever a single record in a BusinessObject is displayed, this script will be used. At the top of an Edit Window for one record, in the many "lookups"

### Structure

Orixa requires that all ListScripts contain 2 fields, a "Name" or "FullName" field and an "ID" field. The ID field is always the ID of the chosen BusinessObject, exactly like the ListScript. However the exact contents of the displayed data can be different. It can be useful to have more or less information in a DisplayScript, so Orixa allows Developers to vary the structure of these scripts according to their needs.

A DisplayScript must return a single record. This is ensured by always adding a "WHERE ID = %d" statement at the end of the script. Orixa substitutes in the ID of a chosen record where "%d" appears in the script, so that only one record is returned.

## ViewScript (required)



ID	DateEdited	Parent	Title	Content
26024	19/10/2020	01	01. Structuring Your Data 01. Creating a new BusinessObject for an Orixa System.	<body>
26024	19/10/2020	01	01. Structuring Your Data 02. What is a BusinessObject?	<body>
26061	19/10/2020	01	01. Structuring Your Data 05. Some example BusinessObjects and explanations of parts of their	<body>
26066	19/10/2020	01	01. Structuring Your Data 04. Reverse engineering a BusinessObject from an existing record	<body>
37609	11/07/2020	04	04. Data Edit Forms: View 01. The "People" and "Organisations" BusinessObjects	<body><h3>The "People" and
38877	12/01/2020	01	01. Structuring Your Data 06. The Purpose of the BusinessObject Scripts and How to use them	<body><h3>The Purpose of the

ViewScript generating a Grid in an App

### Purpose

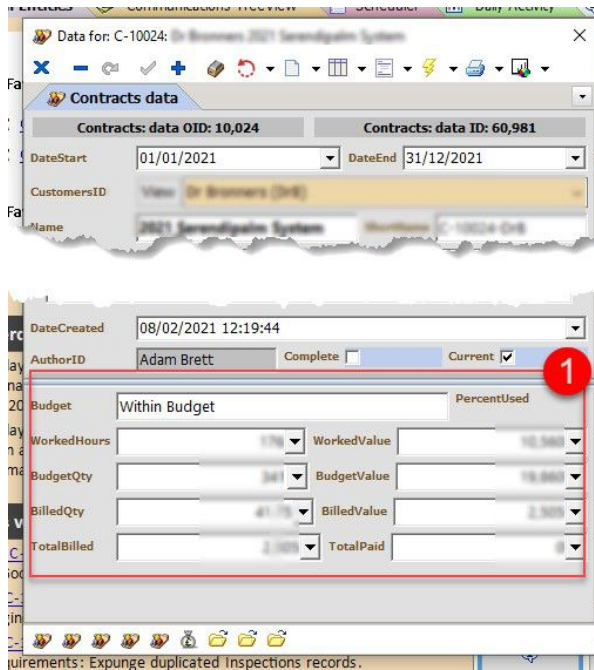
The ViewScript is used by the ViewGrid, to display data from a BusinessObject for the user in a flexible Grid structure. A ViewScript should return all the data that users regularly want to view en-mass from a BusinessObject. This usually includes data from other data-tables.

### Structure

A ViewScript always contains an "ID" field, and adds "%s" or "%0:s" wild-cards in the locations where a SQL WHERE statement would be included.

ViewScripts can usefully include data pulled from multiple data-tables and from VIEWS. The easiest way to review the potential of Orixa ViewScripts is to look at some of the scripts that are built in to an existing App.

## SummaryScript (optional)



Summary Script (at 1., above)

### Purpose

When a user opens an Edit Window for one record in a BusinessObject, a "Summary Panel" can be made to appear at the bottom of the window, if that record has a SummaryScript. The Summary Panel will show data from the Summary Script, the Edit Window will automatically pass the ID of the current record as a parameter to the Summary Panel to allow it to retrieve data.

The SummaryScript should useful data related to one record. For example the Sales for a customer, or the Costs of the Ingredients in a Product.

Care must be taken when writing SummaryScripts to ensure that they return data reasonably quickly. If the SummaryScript has to sum large numbers of records it may make your App very slow.

### Structure

A should never itself contain an ID field, as this will confuse the App. It should add selection of additional "computed" or "sum" fields from other tables. Note that due to complexities in the auto-creation of objects on an Edit Window the field-names of the Summary Script must be different from the field-names of the Edit Window. If there are identical names an error will be raised.

Generally the Summary Script will end with a "WHERE <tablename>ID = %d" statement. It is common for SummaryScripts to include SUM, AVERAGE, COUNT and other SQL Summary commands, and to include a GROUP BY statement.

When a user views an Edit Window, Orixia substitutes the "%d" for the ID of the selected record, and returns the data to be displayed in the Edit Window. Developers can then layout the data as they wish.

If a single row of data is returned, the App will display a set of fields, like the main Edit Window. If more than one row of data is returned, the App will display a Grid.

## Example Summary Script

The following script returns a single row of summary data for one "Contracts" record, note the way in which some fairly complicated maths is used to create useful values such as whether work on the chosen Contract is over budget or under budget etc.

```
SELECT
ContractsID as ID,
SUM(WorkedHours) as WorkedHours,
SUM(WorkedValue) as WorkedValue,
SUM(BudgetQuantity) as BudgetQty,
SUM(BudgetValue) as BudgetValue,
SUM(BilledQuantity) as BilledQty,
SUM(BilledValue) as BilledValue,
IF(SUM(BudgetValue) = 0 THEN IF((SUM(WorkedHours) /
SUM(BudgetQuantity) > 1) THEN 'Hours Exceeded' ELSE 'Within Hours')
ELSE IF(SUM(WorkedValue) / SUM(BudgetValue) > 1 THEN 'Over Budget'
ELSE 'Within Budget')) as Budget,
IF(SUM(BudgetValue) = 0 THEN ROUND(SUM(COALESCE(WorkedHours, 0)) /
SUM(COALESCE(BudgetQuantity, 0)) * 100 TO 2)
ELSE ROUND(SUM(COALESCE(WorkedValue, 0)) / SUM(COALESCE(BudgetValue,
0)) * 100 TO 2)) as PercentUsed,
    COALESCE(MIN(TotalBilled), 0) as TotalBilled,
    COALESCE(MIN(TotalPaid), 0) as TotalPaid
FROM
(SELECT
CI.ContractsID,
CI.ID as ContractItemsID,
CI.BudgetQuantity,
CI.BudgetValue,
CI.BilledQuantity,
CI.BilledValue,
CI.PaidValue,
SUM(COALESCE(WI.HoursWorked, 0)) as WorkedHours,
SUM(COALESCE(WI."Value", 0)) as WorkedValue
FROM ContractItems CI
LEFT JOIN WorkItems WI ON CI.ID = WI.ContractItemsID
WHERE CI.ContractsID = %d
GROUP BY ContractItemsID) as CI1
LEFT JOIN
( SELECT
ContractsID,
ROUND(SUM(COALESCE(ValueBilled, 0)) TO 2) as TotalBilled,
ROUND(SUM(COALESCE(ValuePaid, 0)) TO 2) as TotalPaid
FROM ContractPayments
GROUP BY ContractsID ) as CP ON CP.ContractsID = CI1.ContractsID
GROUP BY ID
```

Seond exmple, in this case note how in data for "Inspections" of one farmer data is returned related to Non-Conformiities of that farmer. This allows an Inspector to see any past non-conformities of the farmer whenever they are being inspected.

```
SELECT
    CAST(Q.Question as VARCHAR(100)) AS "NCQuestions",
```

```

I.DateDone as FormerDateDone,
T.Name as "Type",
CAST(NC.CorrectiveAction AS Varchar(100)) AS "CorrectiveAction",
NC.DateDeadline as "Deadline"
FROM Inspections I
LEFT JOIN Answers A ON (I.ID = A.InspectionsID)
LEFT JOIN NonConformities NC ON (NC.ID = A.ID)
LEFT JOIN Questions Q ON (A.QuestionsID = Q.ID)
LEFT JOIN Types T ON (NC.NonConformitiesTypeID = T.ID)
WHERE (I.PersonID = (SELECT
                    PersonID
                    FROM Inspections
                    WHERE ID = %d))
AND NC.ID IS NOT NULL
AND NC.Complete = false

```

## DiaryScript

### Purpose

If a BusinessObject has a DiaryScript its name will be listed in the "What To Show" box in the Diary /Scheduler of your App. Ticking an item in this list will add Events to the Diary. Orixia uses the content of the DiaryScript to determine what is shown.

An Event can display a **single record** or it can show data, and link to a **resource**. Orixia requires slightly differently structured DiaryScripts to produce these different outputs.

### Structure

Displaying data in the Diary/Scheduler is complex, so Orixia has clear rules on how to create a DiaryScript which allows for variation in how data is displayed.

1. A single record displayed by date, without a particular time.
2. A single record displayed at a precise time and date.
3. A single record displayed with a precise start and end time and date.
4. A "holder" display event, which can be clicked to open a list of records, grid, chart, report or cube. Note that in this case, the DiaryScript includes a reference to the **name** of a record in the **Resources** framework table, which is used to display the data.

### Required Columns

- **ID**, (integer) used to select the record.
- **TableName**, (varchar) used to select the BusinessObject to Open.
- **Title**, (varchar) used to generate text displayed in the Diary.
- **DateStart**, (date or timestamp) used to position an event. If this field is a date Orixia will add the event to the selected date. If a timestamp is passed Orixia will also set the time of the event.

- **ColorID**, an integer used to set the colour of an event. Note that SQL can be used to change the Color depending on the "state" of a record. Orixia has a simple pallet of colours that can be allocated to events using integers from 0 to 30. Above this range a LongInt value can be used to set any color required.
- **ReadOnly**, a boolean true/false field, used to determine whether the user can open the event. If ReadOnly = true the event will not respond when the user clicks.

## Optional Columns

- **Memo**, (varchar) used to generate text displayed in the Diary.
- **DateEnd**, (date or timestamp) this optional field is used to set the end-date and end-time for events with duration. Events can span multiple days.
- **Hint**, (varchar) this optional field is used to set the details visible in the "flyover hint" a user sees when their mouse hovers over an event.
- **ResourceName**, (varchar) this optional field is used to name a record in the "Resources" table which will be activated when the user clicks on the event. More details of how to use the "ResourceName" field are laid out below.

## Structure

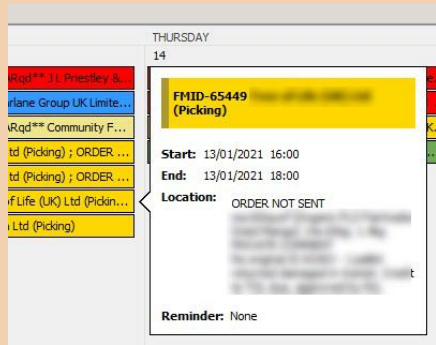
Every DiaryScript will include the above required columns and a WHERE clause which allows the App to insert dates from the date-range selected by the user. The WHERE Clause is always in the following form:

```
WHERE C.DateCreated BETWEEN CAST('%0:s' AS TIMESTAMP)
AND CAST('%1:s' AS TIMESTAMP)
```

With the "%0:s" and "1:s" wild-card characters used to set the Start and End Date for the data being selected.

The WHERE clause can contain other conditions, via additional AND statements, to further limit displayed data if desired.

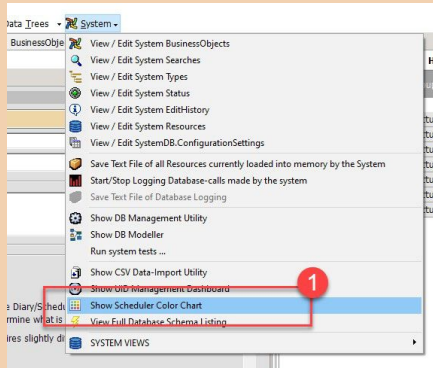
## Sample Images: Diary Script



A "Hint" displaying beside an event in the Diary/Scheduler.

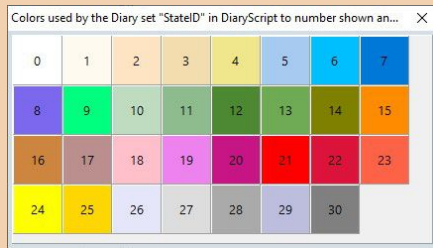
The Scheduler picks up the Title, start and end of the Event data, and also shows the Hint field under a "location" heading.

## Scheduler Hint



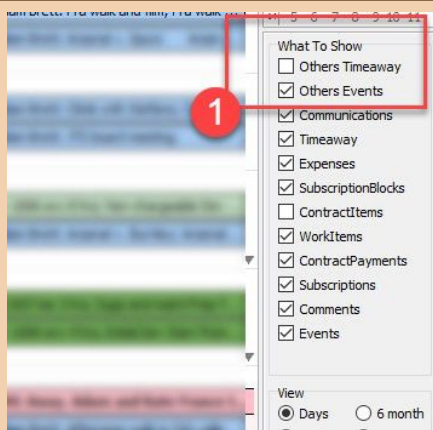
View the potential colours available to use in different parts of Orixia via the "Color Chart" from the System menu

System Menu Scheduler Color Chart



The small range of colours which can be accessed using integer values from 0 to 30. This list of possible colours can be extended. You will see the colours as they have been set for your system when you open this window.

Diary Scheduler Color Chart



The "What to Show" list in the Diary /Scheduler.

Note that the list is auto-generated from any BusinessObject record which includes a Diary Script.

Scheduler What To Show list

## Example DiaryScripts

```
SELECT
  C.ID,
  'Comments' as TableName,
  'Comment: ' + LinkTable + ': ' +
  CAST(COALESCE(C.Name + ' ', '') + COALESCE(C.Memo, '')) AS
VARCHAR(25)) as Title,
  COALESCE(C.Name, '') + #13 + COALESCE(Memo, '') as Memo,
  C.DateCreated as DateStart,
  27 as ColorID,
  false as ReadOnly
FROM Comments C
WHERE C.DateCreated BETWEEN CAST('%0:s' AS TIMESTAMP)
      AND CAST('%1:s' AS TIMESTAMP)
AND AuthorID = [CurrentUser]
```

```
SELECT
  ID,
  CAST('ContractItems' AS VARCHAR(40)) as Tablename,
  CAST(DateStart as VARCHAR(10)) as ControlDate,
  'CI-START ' + C.ShortName as Title,
  CAST(C.OID as VARCHAR) + ': '
  + O.Name + ' '
  + P.Name + ' '
  + CAST(T.Name AS VARCHAR(60)) as Memo,
  P.Name + ' '
  + CI.Name as Hint,
  CAST(DateStart as TIMESTAMP) as DateStart,
  true as ExecData,
  2 as ColorID,
  false as ReadOnly
FROM "ContractItems" CI
LEFT JOIN Contracts C ON C.ID = CI.ContractsID
LEFT JOIN Organisations O ON O.ID = C.CustomersID
LEFT JOIN Products P ON P.ID = CI.ProductsID
LEFT JOIN Types T ON T.ID = C.ContractsTypeID
WHERE CI.DateStart BETWEEN CAST('%0:s' AS TIMESTAMP) AND CAST('%1:s'
AS TIMESTAMP)
```

UNION ALL

```
SELECT
  ID,
  CAST('ContractItems' AS VARCHAR(40)) as Tablename,
  CAST(DateStart as VARCHAR(10)) as ControlDate,
  'CI-END ' + C.ShortName as Title,
  CAST(C.OID as VARCHAR) + ': '
  + O.Name + ' '
  + P.Name + ' '
  + CAST(T.Name AS VARCHAR(60)) as Memo,
  P.Name + ' '
```



```

+ CI.Name as Hint,
CAST(DateEnd as TIMESTAMP) as DateStart,
true as ExecData,
5 as ColorID,
false as ReadOnly
FROM "ContractItems" CI
LEFT JOIN Contracts C ON C.ID = CI.ContractsID
LEFT JOIN Organisations O ON O.ID = C.CustomersID
LEFT JOIN Products P ON P.ID = CI.ProductsID
LEFT JOIN Types T ON T.ID = C.ContractsTypeID
WHERE CI.DateEnd BETWEEN CAST('%0:s' AS TIMESTAMP) AND CAST('%1:s'
AS TIMESTAMP)

SELECT
  MIN(ID) as ID,
  'Communications' as TableName,
  'Communications: ' + CAST(COUNT(*) as VARCHAR(10)) + ' received'
as "Title",
  CAST(DateCreated AS DATE) + INTERVAL '6' HOUR as DateStart,
  2 as ColorID,
  true as ReadOnly,
  'CommunicationsSchedulerGridReceived' as ResourceName
FROM Communications C
WHERE C.DateCreated BETWEEN CAST(CAST('%0:s' AS TIMESTAMP) AS DATE)

AND CAST(CAST('%1:s' AS TIMESTAMP) AS DATE) AND Outgoing = false
GROUP BY CAST(DateCreated AS DATE)

```

### Notes on the DiaryScripts

1. In the 3 examples above use different sets of fields, from the minimum to maximum number.
2. The final example adds a "ResourceName" CommunicationsSchedulerGridReceived, which is explained below.
3. All include a WHERE statement which sets the **date-range**. This is done using the BETWEEN keyword.
4. Note how the third example adds 6 hours to the "DateCreated" so that all its Events will display at 6am.

## Summarizing data in the Diary/Scheduler using "ResourceName" Column of the DiaryScript

Say your App wants to display data in the DiaryScheduler, but for a particular BusinessObject there are too many records to display easily. For example a Company might make hundreds of sales per day, and want to see daily sales data in the Diary, but not want to see each individual record.

In this case, the DiaryScript should create a holder-record for each date where there are Sales, and add a record to the Resources framework-table to access the details, and display them. The record in the Resources table must be completed with the correct SQL structure to allow Orixia to use it.

In the third example DiaryScript above, a "Communications" Business-Object is queried, and a COUNT

of all Communications for each day is generated. When the App fills the Scheduler one single Event will be added to each day. When the user clicks on any of the "Communications" Events, Orixia will pull out the ResourceName (in this case "CommunicationsSchedulerGridReceived") and find a record in the Resources table with that name.

### Example Resource SQL for Resource called by "ResourceName"

```
SELECT
    C.ID,
    C.DateSent,
    C.Title,
    O.FullName as Organisation,
    CAST(WhoToList AS VARCHAR(80)) as WhoToList,
    C.WhoFrom
FROM Communications C
LEFT JOIN Organisations O ON (C.OrganisationsID = O.ID)
WHERE CAST(DateSent AS Date) = CAST('%s' AS Date)
AND Outgoing = false
```

The above SQL is stored in the SQLStr column of the Resources framework table record with the name "CommunicationsSchedulerGridReceived."

This Resources record has a "ComponentID" of "Grid" so the returned data will display as a Grid.

The App will substitute the **date** the user has clicked in place of the '%s' wild-card in the SQL.

### How wild-cards work in Resources called from the Diary-Scheduler

- If a "%s" wild-card is present, the App will substitute in the date the user has clicked.
- If a "%0:s" wild-card is present, the App will substitute in the minimum-visible-date in the scheduler.
- If a "%1:s" wild-card is present, the App will substitute in the maximum-visible-date in the scheduler.
- If the "%d" or "%0:d" wild-cards are present, the App will use the ID held by the Event.

Resource SQL can contain multiple SELECT statements brought together by UNION clauses. The above options allow Developers to return a wide range of useful data to the user. Review examples in your BusinessObjects table if you want to understand this in more detail.