OrderFlow Reporting Guide

OrderFlow Ltd.

Document Version: 4.2.4

Document Built: 2024-02-16

Overview

An important distinguishing characteristic of any warehouse management system (WMS) is the degree to which it provides easy access to information.

OrderFlow's reporting capability is critical in enabling it to achieve this objective. In addition to a wide range of *built-in* reports, OrderFlow provides a powerful custom reporting capability. In fact, custom reports are used in OrderFlow in a wide variety of situations:

- management and information reports
- paperwork generation reports, for example for despatch notes and courier labels
- printable picking reports, for shipment batches and for stock move tasks
- strategy reports used to determine the source or target locations and quantities for stock move tasks.

In addition, OrderFlow's reporting framework is used throughout the system to generate views and fragments for both the desktop and mobile versions of the application, providing easy visual access to key performance indicators. These collections views can be easily enhanced, modified or extended by suitably trained technical users.

In all cases, reports can be added to the system 'on the fly'. There is no need for a new software build or system restart when adding reports.

Custom Reports

OrderFlow includes a set of built-in reports which cover typical management information requirements. However, because the system can be used in so many different ways, needing to cover such a disparate range of requirements, an extended custom reporting facility is essential.

Custom reports can only be created by users with the appropriate administrator access to the system.

These reports can be created in a variety of formats. The most common formats are CSV (Comma-Separated Values) and XLS (Microsoft Excel spreadsheets). These serve the needs for users who are looking to extract unformatted or raw data from the system.

For printable custom reports that need to be generated with pixel-perfect accuracy, OrderFlow uses Jasper Reports, the leading open-source reporting technology. Typical examples of these kinds of reports are barcode label generation reports, despatch notes and picking reports.

Custom vs Built-in Reports

It is helpful to explain exactly what is meant by a custom report, and its cousin, the built-in report:

Built-in Reports

Built-in reports are reports which are provided with the software. They are actually delivered in files included as part of the OrderFlow build.

Built-in reports are *supported* in the sense that they will be kept up to date with changes to OrderFlow report. For example, if an OrderFlow database table's column is renamed, then built-in reports which use this column will be changed accordingly.

Built-in reports are developed within the OrderFlow codebase and regression tested with every new OrderFlow release. If a built-in report does not work as it should, the problem will be treated as a bug and the work required to fix it will not be chargeable.

Built-in reports cannot be modified directly. However, these reports can be *overridden*. By overriding a built-in report, the report administrator effectively creates a custom report with the intention of reusing some elements of the built-in report, but modifying other elements. For example, if the administrator wants to slightly change the report display, but retain the report data source, this can be achieved by overriding a built-in report.

Note that once a built-in report has been overridden, it effectively becomes a custom report.

Custom Reports

Custom reports are reports which are added to the operating environment. They are in fact installed in the OrderFlow database. Note that custom reports are not supported in the same way as built-in reports.

In general, OrderFlow developers will try to allow for any negative impact that system upgrades may have on custom reports, and will, where possible, apply changes automatically to custom reports to accommodate these changes.

However, no guarantee can be provided that custom reports will never be broken by a system change. In our example of a column change above, custom reports, which rely on the old column name, will no longer work after the column change unless they are manually modified.

Custom reports can be developed by the OrderFlow support team or by any OrderFlow user with the necessary privileges. They are not regression tested against new OrderFlow releases - bug fixes and change requests will be charged on a time and materials basis. Custom reports should always be tested in the OrderFlow staging environment before new OrderFlow releases are used in a live environment.

Custom Data Reports

The most common custom reports are data reports which are aimed simply at extracting data from the system for subsequent analysis and manipulation. These reports are also the simplest reports in that they do not involve any complex presentation; the output tends to be simple tabular formats such as CSV or XLS.

Conceptually, any report involves two elements: a *data source* or query, which is used to extract the relevant data, and a *design*, which is used to render the report in the relevant format.

Report authors need to have the following knowledge:

- familiarity using SQL
- an understanding of the OrderFlow application, and knowledge of the relevant tables in the OrderFlow database schema
- for CSV-based reports, knowledge of how to use the [FreeMarker(http://freemarker.org/) templating engine, which can be easily acquired from the FreeMarker web site.

Data Sources

OrderFlow supports the use of a variety of data sources. The most commonly used are based on Structured Query Language (SQL), allowing report administrators to design reports which *safely* query data directly from the OrderFlow database.

An example of an SQL statement is shown below:

```
SELECT
    p.externalReference AS 'product',
    p.description AS 'description',
    s.externalReference AS 'site',
    o.externalReference AS 'organisation',
    l.externalReference AS 'location',
    lt.externalReference AS 'locationtype',
    lt.damaged AS 'damagedLocation',
    sc.id AS 'id',
    sc.type AS 'adjustment_type',
    sc.newQuantity AS 'new_quantity',
    sc.previousQuantity AS 'previous_quantity',
    sc.changeQuantity AS 'change_quantity',
    sc.user AS 'user',
    sc.created AS 'timestamp',
    sc.note AS 'comment'
FROM stock_change sc
JOIN product p ON p.id = sc.productId
JOIN organisation o ON o.id = p.organisationId
JOIN location l ON l.id = sc.locationId
JOIN location_type lt ON lt.id = l.logicalTypeId
JOIN site s ON s.id = l.siteId
WHERE
    (1=1)
AND
    (sc.type in ('positive_adjustment','negative_adjustment')
    OR
    (sc.type in ('move_in','move_out') and lt.damaged = true))
AND p.deleted = FALSE
#[organisations:p]
ORDER BY sc.id
```

The example above is used to query changes. Of course, familiarity with the OrderFlow database schema is also necessary for writing non-trivial reports.

Parameters

Reports can have parameters to restrict the data to be returned. Typical parameters include those for the site, channel or organisation(s) for which the data should apply. It is also very common to restrict the report parameters by date range.

In the example below, the parameters `from_date` and `to_date` are used to restrict a data by date range. The parameters used in this way need to be declared in the report's *Parameter Source* field. For more details on how this is done, see the Report Configuration GUI section.

An example parameter declaration is shown below:

```
   organisation_id:Organisation,from_date:From date:required:datetime, to_date:To
date:required:datetime,productReference:Product Reference:optional:string
```

The example above supports a required organisation, from date and to date parameters, and an optional product reference parameter.

More detail on the use of date parameters are contained in the Date Parameters section.

The parameters are used directly in the SQL of custom reports, as is shown in the example below.

```
SELECT externalReference from order_item o
WHERE
date(o.completed) >= ${from_date}
AND
date(o.completed) <= ${to_date}
```

Note that three parameter values are very commonly used in multi-channel and multi-site environments:

- `channel_id` : used to filter report data by channel
- `organisation_id` : used to filter report data by organisation
- `site_id` : used to filter report data by site

Note that parameters can be *required* or *optional*. When using optional parameters, it is often useful to inclusion of the parts of the SQL conditional on whether a value has been supplied for the parameter.

In the example below, the clause `p.externalReference = ?` will only be added to the SQL if a value is supplied for the `productReference` parameter.

```
#[include:productReference]p.externalReference = ${productReference}#[include]
```

Note that in the above example, the `${productReference}` segment is translated into SQL that uses bound parameters, which can be beneficial for query performance. In the example above, the SQL used after preprocessing would be `p.externalReference = ?` . At times, it is more convenient just to insert the parameter literal. This can be done using the `literal:` prefix.

In the example below, if the value `SKU123` is supplied for the parameter `productReference`, then the SQL segment

```
#[include:productReference]p.externalReference like '%${literal:productReference}%'#[include]
```

would be translated into `p.externalReference like '%SKU123%'` before being executed.

PARAMETER FORMAT AND TYPES

Parameters are defined in reports using the following format:

`Identifier:Name:required or optional:type:Default value`

Name	Example	Description
Identifier	`productReference`	The identifier which is used in the report, typically as a parameter in the SQL itself.
Name	`Product Reference`	The human readable name for the parameter, which appears as the name of the parameter when parameter values need to be captured on the report launching screen.
Required or option	`required`	If present, must be either `required` or `optional`.
Type	`string`	The type of the parameter.
Default Value	`123`	The default value for the parameter is none is supplied by the user.

Note that of the above values, the *required or optional* and the type values can be determined implicitly:

- if no `required` or `optional` value is set, then the parameter is assumed to be optional.
- if no type is set, the type is assumed to be `string`.

Type	Usage
string	Free text entry values
password	Password entry values
integer	Standard integer values
date	Dates without time component
datetime	Dates with time component
boolean	Simple true or false
double	Double-precision floating point (decimal) values
float	Single-precision floating point (decimal) values
long	Long integer values

RESERVED PARAMETER CONVENTIONS

The following are reserved parameters that define a convention that should be followed.

Note that these parameters conventions should be closely observed for reports that are to be used as a basis for periodic reports.

Name	Title	Usage
`channel_id`	Channel, Used to filter data by channel. Used in channel scoped periodic reports.	
`organisation_id`	Organisation, Used to filter data by organisation. Used in organisation scoped periodic reports.	
`site_id`	Site, Used to filter data by site. Used in site scoped periodic reports.	
`from_date`	From Date, "Used to define the starting point for date-based filters. Suitable for daily, weekly and monthly periodic reports.	
`to_date`	From Date, "Used to define the end point for date-based filters. Suitable for daily, weekly and monthly periodic reports.	
`from_datetime`	From Date, "Used to define the starting point for date-based filters. Suitable for hourly, daily, weekly and monthly periodic reports.	
`to_datetime`	From Date, "Used to define the end point for date-based filters. Suitable for hourly, daily, weekly and monthly periodic reports.	

DATE PARAMETERS

The main thing to understand about the semantics of date parameters is the use of the *to* date or timestamp.

- for *date* parameters, the from and to are both *inclusive*. So if the to date is 31/01/2016, then the report data should include all data that is on that date, regardless of what time of day this occurred.
- for *date time* parameters, the from date is inclusive but the to date is exclusive. So if you want to include all data on 31/01/2016, the to date will need to be defined up to midnight on 01/02/2016.

Date Parameters

SQL

The inclusive to date is captured in the SQL by a combination of the use of the `date()` function and the <= sign in before the `${to_date}`.

```
date(datefield) >= ${from_date} and
date(datefield) <= ${to_date}
```

Parameters

Note that the date parameter is of type `date`.

```
from_date:From date:required:date, to_date:To date:required:date
```

Date Time Parameters

SQL

The exclusive to datetime is captured using a < sign. There is no need for any function to be applied to the compared field value.

Note that datetime parameters are only suitable for date fields in which the values are captured as date times (with the date and time element in the stored data).

```
datefield >= ${from_datetime} and
datefield < ${to_datetime}
```

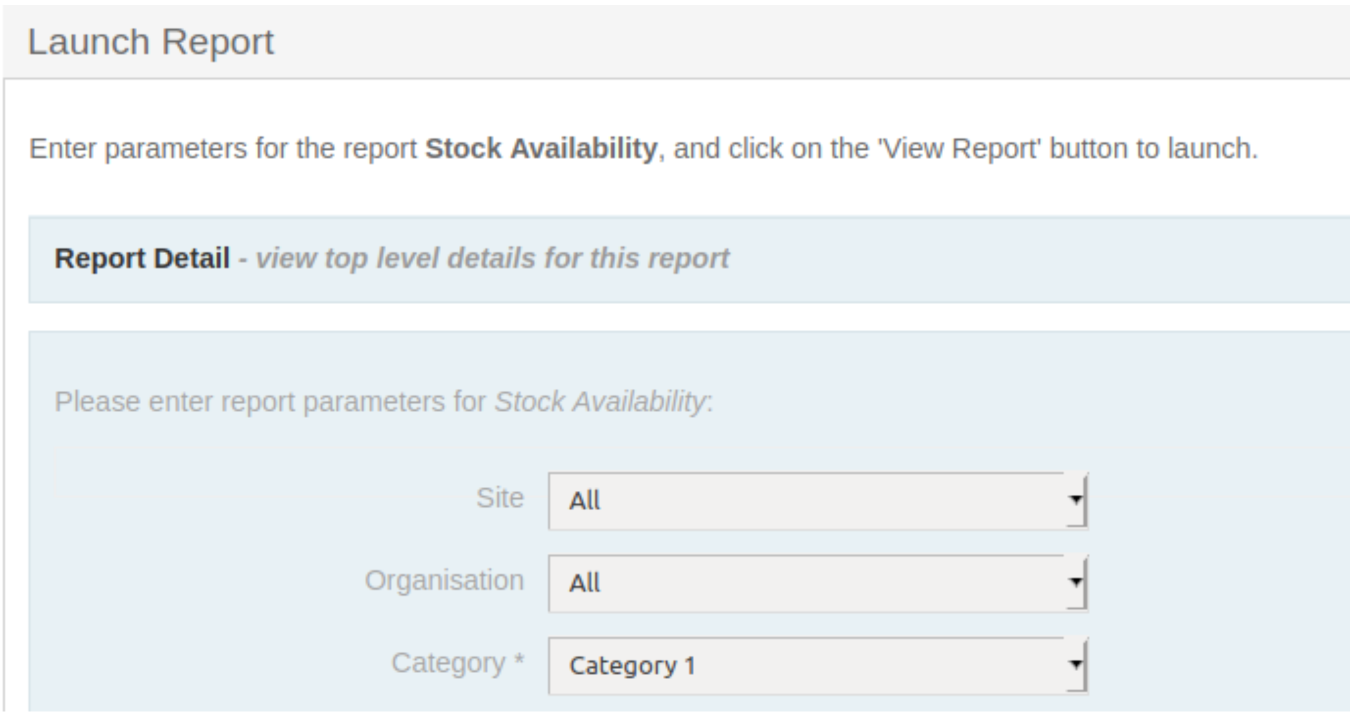
Parameters

Note the use of the `datetime` type, which prompts for time element be captured as input when running a report with this parameter.

```
from_datetime:From date:required:datetime, to_datetime:To date:required:datetime
```

DROP DOWN PARAMETERS

On occasions it is necessary for the selectable parameter values to be from a pre-configured list. This is possible from OrderFlow versions 4.0.8.1 and later.



Launch Report

Enter parameters for the report **Stock Availability**, and click on the 'View Report' button to launch.

Report Detail - *view top level details for this report*

Please enter report parameters for *Stock Availability*:

Site	All
Organisation	All
Category *	Category 1

The setup of these categories is done as below:

```
site_id:Site, organisation_id:Organisation, include_unavailable_products:Include unavailable
products:optional:boolean, category:Category:required:string:(cat1=Category 1|cat2=Category 2)
```

Note that in the example above, the operator will need to set a category parameter, and either the value 'cat1' or 'cat2' will be passed to the report query.

This mechanism is a simple way to restrict the parameter choices available to a user. However, it does require each parameter choice to be preset in the report configuration.

In the next section, we show how to support pre-selectable options that are themselves determined from a report, which will be useful in more dynamic environment.

REPORT-BACKED PARAMETERS

It is possible to have parameters set up on OrderFlow that themselves use reports to specify default values.

For example, for areas, a report named 'area_options' may be used. In this case, an example report parameter will take the form:

```
organisation_id:Organisation:required,site_id:Site:required, area_id:Area:required:integer:null:
'area_options'
```

Note that the `null` value in this case indicates no default value.

It is worth bearing in mind that if the report-back parameter is *required*, then any other parameter that specifies the scope of the report (specifically `site_id`, `organisation_id` or `channel_id`) will also need to be set as mandatory. In this case, the scope-defining parameter values will need to be selected before the report-backed options are presented to the user.

All of this is necessary because the report-backed options typically depend on the current report scope. For example, the selected areas will need to depend on the the selected site for the report.

Report 'Hardening'

It is sometimes required to 'harden' a report so that it does not expose data relating to organisations, channels or sites to which the user does not have access, for obvious reasons.

This is achieved by adding particular directives to a report's SQL data source. The directives each take the following form:

```
#[<*entity to restrict*>s:<*alias of restricted-entity-aware table in 'from' clause*>]
```

For example, the following directive is present in the example SQL statement above:

```
#[organisations:p]
```

where `p` is the alias for the organisation-aware `product` table.

As well as for organisations, directives should be declared where required for *channels* and *sites*. The syntax for these expressions follows that used for organisations:

```
#[channels:channelAwareEntityAlias]
#[sites:siteAwareEntityAlias]
```

These directives should be placed in the `WHERE` clause of a query.

Note that if a `channel_id`, `organisation_id` or `site_id` parameter is present in the report configuration, the relevant value supplied by the user will automatically be applied when the hardening expression is expanded during report execution.

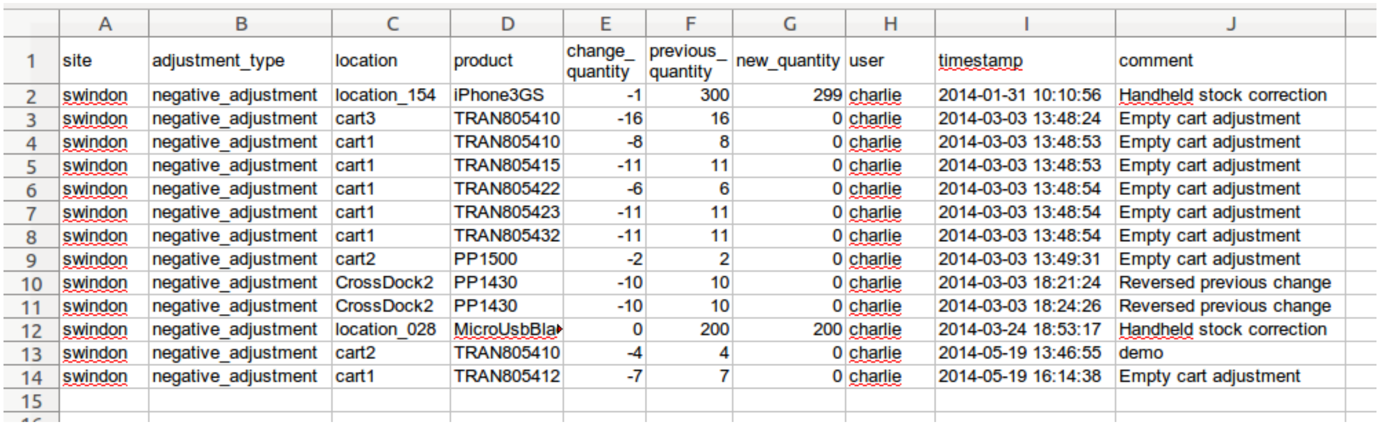
On creation and update, if a report is active then its SQL data source is checked against its 'Hardening Requirement' value, to ensure that it contains the necessary directives.

Design

Regarding the design element of report generation, the simplest format is XLS (the Microsoft Excel spreadsheet format). OrderFlow supports the creation of Excel reports simply by listing the column headers. For the SQL above, the corresponding design for an Excel format report would be:

```
site,adjustment_type,location,product,change_quantity,previous_quantity,new_quantity,user,timestamp,comment
```

Example output:



	A	B	C	D	E	F	G	H	I	J
1	site	adjustment_type	location	product	change_ quantity	previous_ quantity	new_quantity	user	timestamp	comment
2	swindon	negative_adjustment	location_154	iPhone3GS	-1	300	299	charlie	2014-01-31 10:10:56	Handheld stock correction
3	swindon	negative_adjustment	cart3	TRAN805410	-16	16	0	charlie	2014-03-03 13:48:24	Empty cart adjustment
4	swindon	negative_adjustment	cart1	TRAN805410	-8	8	0	charlie	2014-03-03 13:48:53	Empty cart adjustment
5	swindon	negative_adjustment	cart1	TRAN805415	-11	11	0	charlie	2014-03-03 13:48:53	Empty cart adjustment
6	swindon	negative_adjustment	cart1	TRAN805422	-6	6	0	charlie	2014-03-03 13:48:54	Empty cart adjustment
7	swindon	negative_adjustment	cart1	TRAN805423	-11	11	0	charlie	2014-03-03 13:48:54	Empty cart adjustment
8	swindon	negative_adjustment	cart1	TRAN805432	-11	11	0	charlie	2014-03-03 13:48:54	Empty cart adjustment
9	swindon	negative_adjustment	cart2	PP1500	-2	2	0	charlie	2014-03-03 13:49:31	Empty cart adjustment
10	swindon	negative_adjustment	CrossDock2	PP1430	-10	10	0	charlie	2014-03-03 18:21:24	Reversed previous change
11	swindon	negative_adjustment	CrossDock2	PP1430	-10	10	0	charlie	2014-03-03 18:24:26	Reversed previous change
12	swindon	negative_adjustment	location_028	MicroUsbBla	0	200	200	charlie	2014-03-24 18:53:17	Handheld stock correction
13	swindon	negative_adjustment	cart2	TRAN805410	-4	4	0	charlie	2014-05-19 13:46:55	demo
14	swindon	negative_adjustment	cart1	TRAN805412	-7	7	0	charlie	2014-05-19 16:14:38	Empty cart adjustment
15										

The other principal format for data-centric reports is CSV. For these, the tool of choice is the open source FreeMarker templating engine. The equivalent CSV-based design for the Excel formatted report above is shown below:

```
site,adjustment_type,location,product,change_quantity,previous_quantity,new_quantity,user,timestamp,comment
<#list results as result>
${result.site},${result.adjustment_type},${result.location},${result.product},${result.change_quantity},$
{result.previous_quantity},
${result.new_quantity},${result.user},${result.timestamp?string('yyyy-MM-dd HH:mm:ss')},${result.comment}
</#list>
```

Example output in CSV:

```
site,adjustment_type,location,product,change_quantity,previous_quantity,new_quantity,user,timestamp,comment
swindon,negative_adjustment,location_154,iPhone3GS,-1,300,299,charlie,2014-01-31 10:10:56,Handheld stock
correction
swindon,negative_adjustment,cart3,TRAN805410,-16,16,0,charlie,2014-03-03 13:48:24,Empty cart adjustment
swindon,negative_adjustment,cart1,TRAN805410,-8,8,0,charlie,2014-03-03 13:48:53,Empty cart adjustment
swindon,negative_adjustment,cart1,TRAN805415,-11,11,0,charlie,2014-03-03 13:48:53,Empty cart adjustment
swindon,negative_adjustment,cart1,TRAN805422,-6,6,0,charlie,2014-03-03 13:48:54,Empty cart adjustment
swindon,negative_adjustment,cart1,TRAN805423,-11,11,0,charlie,2014-03-03 13:48:54,Empty cart adjustment
swindon,negative_adjustment,cart1,TRAN805432,-11,11,0,charlie,2014-03-03 13:48:54,Empty cart adjustment
swindon,negative_adjustment,cart2,PP1500,-2,2,0,charlie,2014-03-03 13:49:31,Empty cart adjustment
swindon,negative_adjustment,CrossDock2,PP1430,-10,10,0,charlie,2014-03-03 18:21:24,Reversed previous change
swindon,negative_adjustment,CrossDock2,PP1430,-10,10,0,charlie,2014-03-03 18:24:26,Reversed previous change
swindon,negative_adjustment,location_028,MicroUsbBlackberry,0,200,200,charlie,2014-03-24 18:53:17,Handheld
stock correction
swindon,negative_adjustment,cart2,TRAN805410,-4,4,0,charlie,2014-05-19 13:46:55,demo
swindon,negative_adjustment,cart1,TRAN805412,-7,7,0,charlie,2014-05-19 16:14:38,Empty cart adjustment
```

As the examples above show, CSV reports require slightly more effort to set up than spreadsheet-based reports. However, they do allow for fine-grained control of the formatting of certain data (such as dates) in a way that is not supported with Excel-based reports. The output data can also be previewed on screen prior to download, a capability that is not supported with Excel-based reports.

OrderFlow Report Administration

In this section we describe how the report administration functionality can be used in OrderFlow.

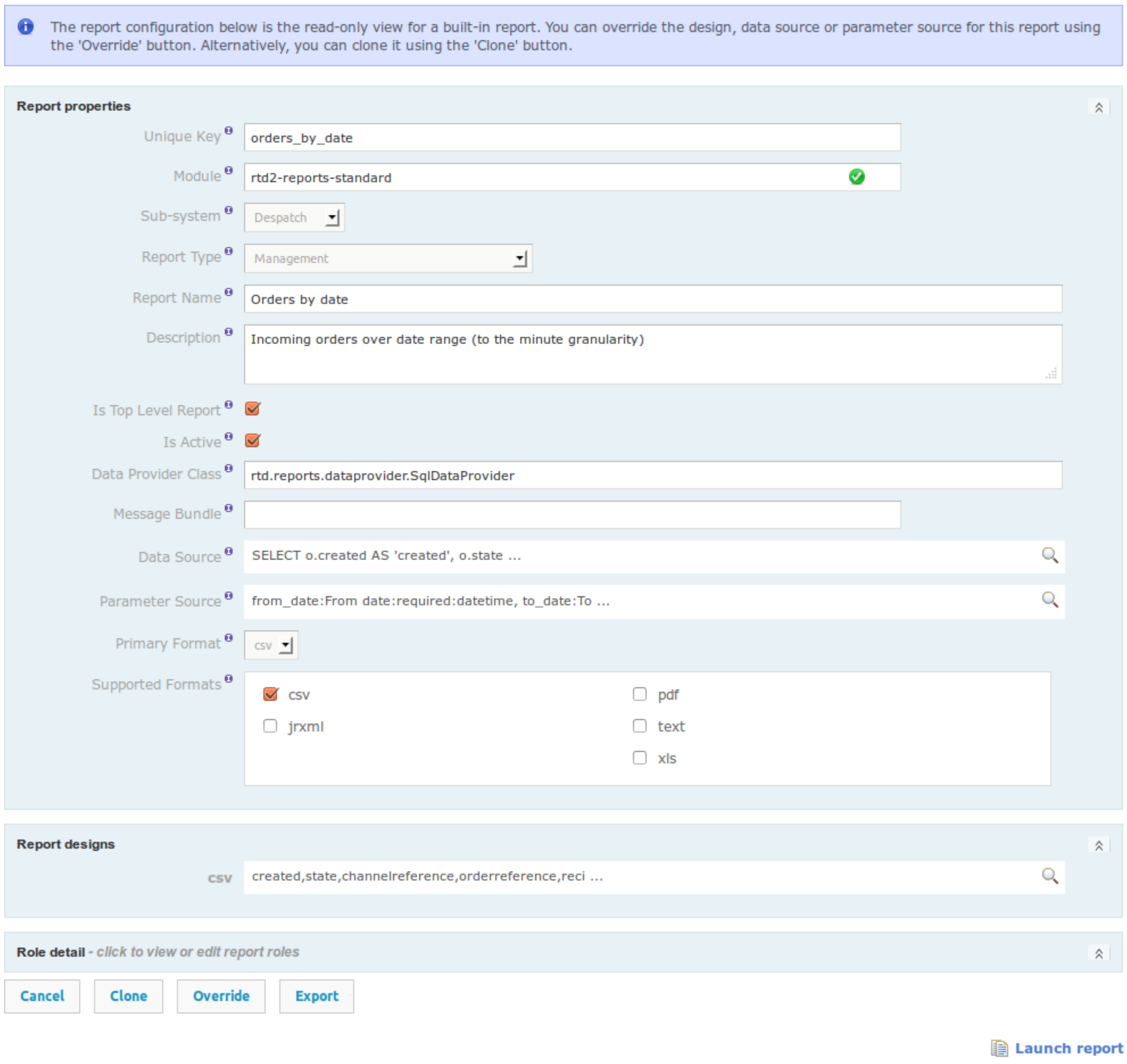
When creating a new custom report, four options are available:

- **create** a new custom report from scratch. This involves the most effort.
- **clone** an existing report. The report that you clone can be either a built-in report or a custom report. When cloning an existing report, a new report is created with a name and identifier that is different from the old one, but with other properties (e.g. the data source) that are the same as the old one. Cloning a report provides a convenient starting point for creating the new custom report, as it is usually possible to find an existing report on the system which has similar characteristics to the one you are attempting to create.
- **override** an existing built-in report. Note that when overriding a built-in report, you are not actually creating a new one. Instead, you are making a change to the way the existing report appears or behaves. Essentially, you are creating a custom report from an existing built-in report, making it not automatically supported by the system.
- **import** an existing report. When copying a report from another environment (e.g. from test to live), the most convenient way to do so is to import the report. This process allows you to import all of the configuration associated with a report, in a single operation, rather than having to copy individual fields manually. This makes the process of migrating a report much quicker and less error-prone.

Report Configuration GUI

The report configuration GUI supports the creation and modification of custom and overridden reports on OrderFlow. This section describes the purpose and usage of the main editable fields.

An example of the report configuration screen for an existing report is shown below.



The report configuration below is the read-only view for a built-in report. You can override the design, data source or parameter source for this report using the 'Override' button. Alternatively, you can clone it using the 'Clone' button.

Report properties

Unique Key	orders_by_date
Module	rtd2-reports-standard
Sub-system	Despatch
Report Type	Management
Report Name	Orders by date
Description	Incoming orders over date range (to the minute granularity)
Is Top Level Report	✓
Is Active	✓
Data Provider Class	rtd.reports.dataprovider.SqlDataProvider
Message Bundle	
Data Source	SELECT o.created AS 'created', o.state ...
Parameter Source	from_date:From date:required:datetime, to_date:To ...
Primary Format	csv
Supported Formats	[✓ csv] pdf jrxml text xls

Report designs

csv	created,state,channelreference,orderreference,reci ...

Role detail - *click to view or edit report roles*

Unique Key

The reference that is used in other parts of the application to identify the report. As the name suggests, this field must contain a unique value.

Module

Defines a Java module to look up for a report. This module must have visibility to the *Data Provider Class*. By default the *rtd2-reports-standard* module is used.

Sub-system

A drop-down menu that, along with individual user configuration, defines the visibility of the report, i.e. which sub-system (Despatch, Warehouse, Import, etc.) has access to it.

Report Type

As the name suggests, defines the type of the report - Despatch note, Admin, Barcode, Periodic, etc.

Report Name

The name the user sets to easily identify the report. This differs from *Unique Key* such that the unique key is used by the application, whereas the report name helps the user to distinguish between different reports.

Description

A description of what the report does and/or how it is used or any notes regarding it.

Is Top Level Report

A check box, which determines whether the current report is a main report or a sub-report. If it is a sub-report (i.e. the check box is not ticked), then it will not appear in the main list of reports, but it can be used by top level reports.

Is Active

Self-explanatory: determines whether the report is active, i.e. if it can be used or not.

Data Provider Class

The Java class, used to provide data for the report. The most commonly used data provider is *rtd.reports.dataprovider.SqlDataProvider*, which is used to return a list of rows using a single SQL query.

Message Bundle

A more advanced field - it is useful when producing multi-language reports (e.g. despatch notes). After uploading a message file resource, it can be used as a base key for looking up message resources.

Data Source

Used to query data from the OrderFlow database by using SQL statements. See Data Sources above for more detail on SQL queries in reports. Parameters can also be defined in this field, which are basically user inputs that help when querying the database.

Parameter Source

Used to define the source of any parameters that the query uses. These parameters are entered once a report is launched. The format, used to determine parameters is as follows:

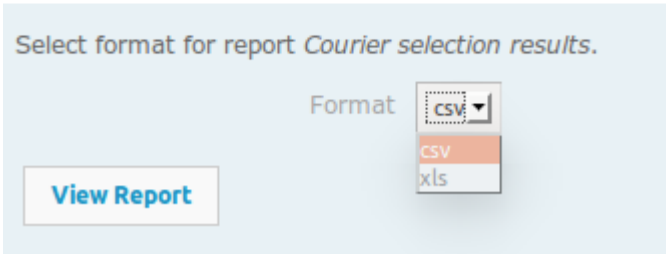
```
name:title:optional/required:type:default-value
```

Each element is divided by a colon and the *type* is a string by default. The *default-value* can be omitted. The *name* element is a reference to the name, used in the SQL query in the *Data Source* field. An example parameter source value is:

```
from_date:From date:required:date, to_date:To date:required:date, channel_id:Channel, site_id:Site
```

Primary Format

The default format, in which the report is executed. If multiple formats are supported by the same report, you can define which is the default format by using this field. Once the report is launched, this can be changed, so a different format is used. In the example below *csv* is the default format, but just before executing the report, the format can be changed to *xls* if desired.


Select format for report *Courier selection results*.

Format: csv

Supported Formats

Determines the formats in which a report can be produced.

Report Designs

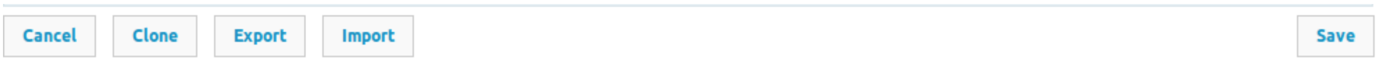
These specify the design(s) of the report, depending on the format that is used. See design above for more detail on formatting XLS and CVS reports and report design below for JRXML.

Role Detail

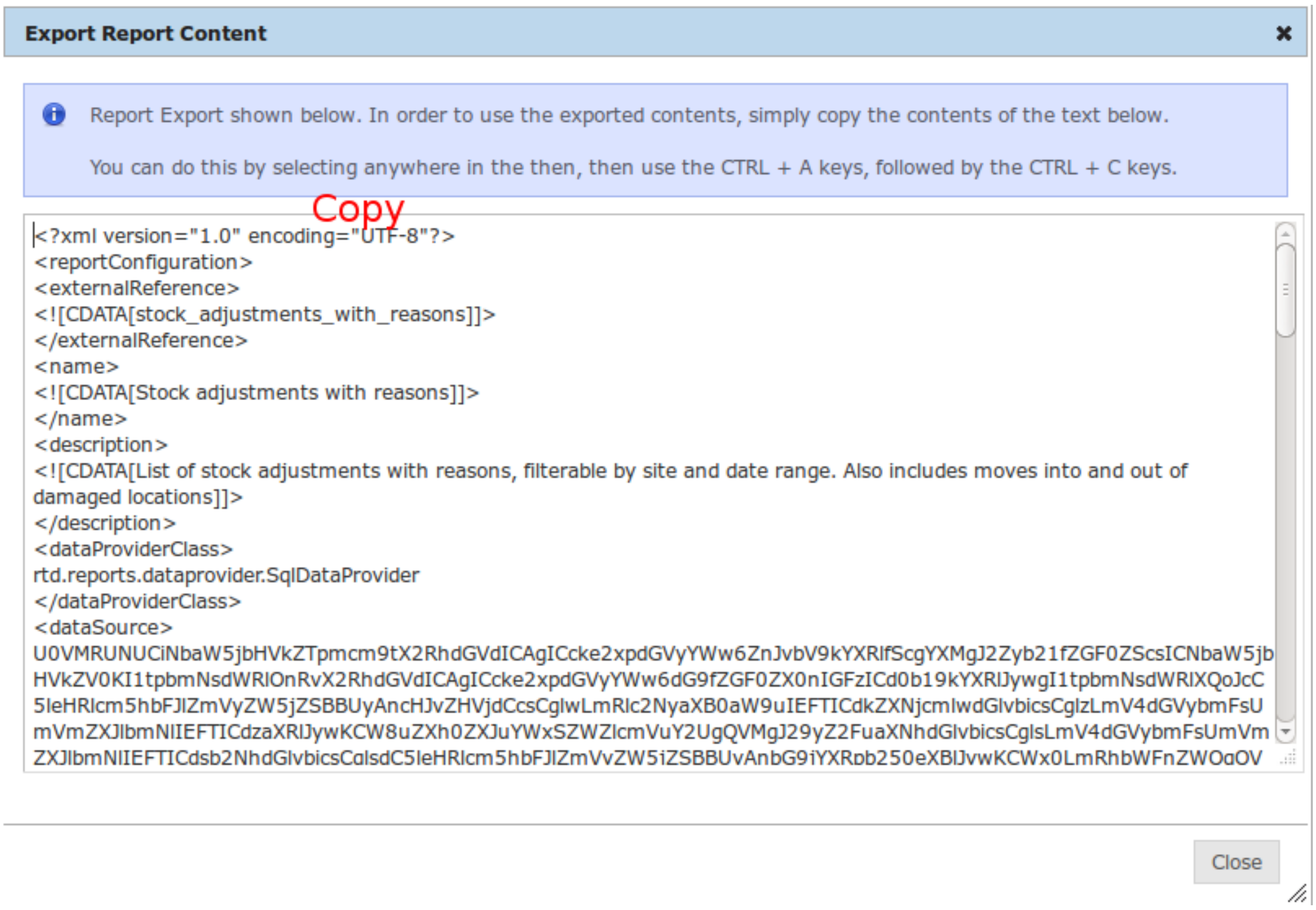
These properties are used to restrict access to the report to different user groups - e.g. Packer, Stock handler, Import operator.

Importing and Exporting Reports

In order to export a report, simply click on the *Export* button at the bottom of the report configuration screen of the report to be exported:



A pop-up dialog then opens with all the configurations of the report, encoded in an xml format. Select and copy all the contents of the text in the text box.



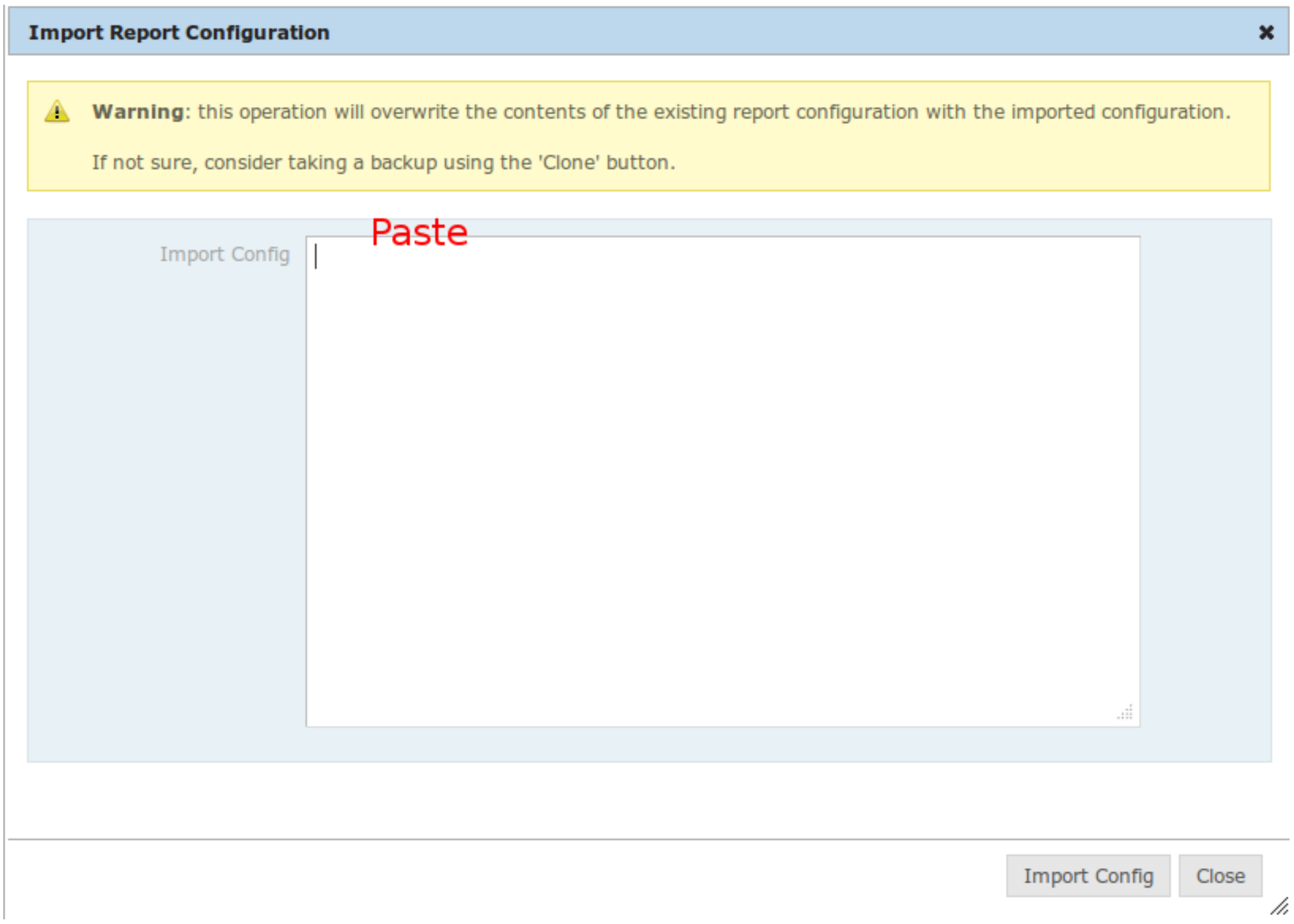
Report Export shown below. In order to use the exported contents, simply copy the contents of the text below.

You can do this by selecting anywhere in the then, then use the CTRL + A keys, followed by the CTRL + C keys.

```
<?xml version="1.0" encoding="UTF-8"?>
<reportConfiguration>
<externalReference>
<![CDATA[stock_adjustments_with_reasons]]>
</externalReference>
<name>
<![CDATA[Stock adjustments with reasons]]>
</name>
<description>
<![CDATA[List of stock adjustments with reasons, filterable by site and date range. Also includes moves into and out of
damaged locations]]>
</description>
<dataProviderClass>
rtd.reports.dataprovider.SqlDataProvider
</dataProviderClass>
<dataSource>
U0VMRUNUCiNbaW5jbHVkZTpmcm9tX2RhdGVdICAgICcke2xpdGVyYWw6ZnJvbV9kYXRlfScgYXMgJ2Zyb21fZGF0ZScsICNbaW5jb
HVkZV0KI1tpbmNsdWRlOnRvX2RhdGVdICAgICcke2xpdGVyYWw6dG9fZGF0ZX0nIGFzICd0b19kYXRlJywgI1tpbmNsdWRlXQoJcC
5leHRlcm5hbFJlZmVyZW5jZSBBUyAncHJvZHVjdCcsCglwLmRlc2NyaXB0aW9uIEFTICdkZXNjcmlwdGlvbicsCglzLmV4dGVybmFsU
mVmZXJlbmNlIEFTICdzaXRlJywKCW8uZXh0ZXJuYWxSZWZlcmVuY2UgQVMgJ29yZ2FuaXNhdGlvbicsCglsLmV4dGVybmFsUmVm
ZXJlbmNlIEFTICdsb2NhdGlvbicsCglsdC5leHRlcm5hbFJlZmVyZW5jZSBBUyAnbG9jYXRpb250eXBlJywKCWx0LmRhbWFnZWQgOV
```

Once this is done you need to make sure that the target report is in place. If necessary, you may need to create a new report, either by creating a new report from scratch or by the easier method of cloning an existing report. Make sure that the *Unique Key* of the two reports match, or the attempt to import will be rejected.

Once the target report is ready, you can import the contents of the source report from the browser's clipboard in the following way. First, click on the *Import* button at the bottom of the screen. This opens a dialog, similar to the export one, but with a warning message and an empty text box. To import the report, simply paste the contents from the export pop-up into the text box and click *Import Config*, as shown below.


Import Report Configuration

Warning: this operation will overwrite the contents of the existing report configuration with the imported configuration.

If not sure, consider taking a backup using the 'Clone' button.

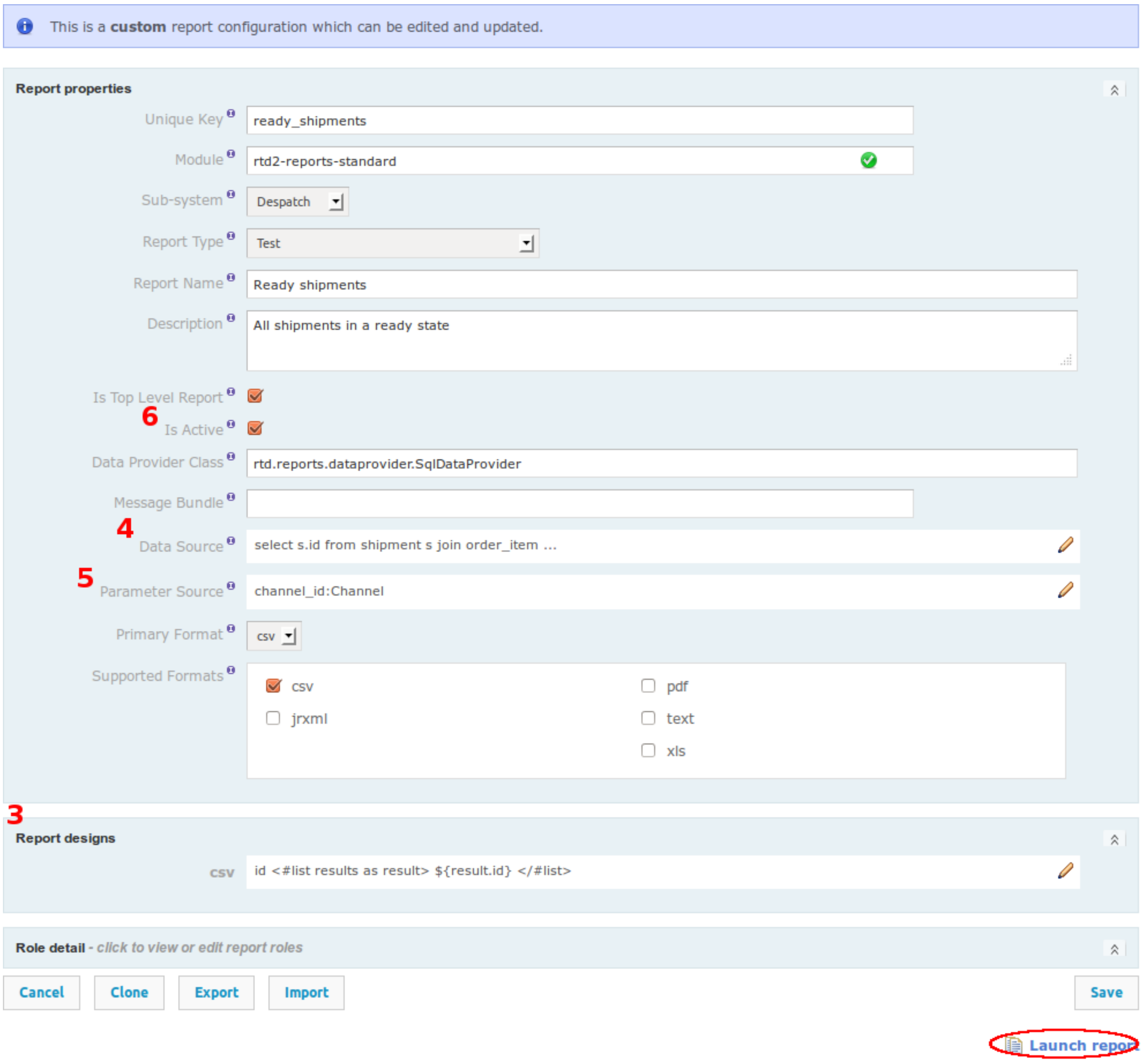
If successful, a small information window will pop at the top right corner of the screen, indicating that the import has been successful.

Custom Report Development Phases

A typical custom report will be developed in the phases described below.

1. Use a database query editor to experiment with and subsequently refine the query being used to generate the report data.
2. Clone or override an existing report to create the configuration entry which the report is to use. (Save the cloned report before proceeding.)
3. Create the report design.
4. Insert the report data source (query), adding any additional parameters necessary.
5. Create the parameter source to capture input parameters for the report.
6. Activate the report. Once activated, the report can be run using the *Launch report* link on the report configuration screen.

The screenshot below illustrates where steps 3-6 can be configured from.



This is a **custom** report configuration which can be edited and updated.

Report properties

Unique Key	ready_shipments
Module	rtd2-reports-standard
Sub-system	Despatch
Report Type	Test
Report Name	Ready shipments
Description	All shipments in a ready state
Is Top Level Report	✓
Is Active	✓
Data Provider Class	rtd.reports.dataprovider.SqlDataProvider
Message Bundle	
Data Source	select s.id from shipment s join order_item ...
Parameter Source	channel_id:Channel
Primary Format	csv
Supported Formats	✓ csv, ☐ pdf, ☐ jrxml, ☐ text, ☐ xls

Report designs

csv	id <#list results as result> ${result.id} </#list>

Role detail - *click to view or edit report roles*

Once the report has been created, it can be iteratively refined by editing, then saving the report configuration, then rerunning the report. It is quite convenient to have two browser tabs open simultaneously to serve this purpose.

Formatted Printable Reports

The custom reports we have described in the previous section are data reports, typically in an CSV or XLS format.

OrderFlow also supports a mechanism to publish and run printable reports, such as barcode labels, picking reports and despatch notes, These reports can be produced to pixel-perfect accuracy for Jasper Reports, the world's most popular open source reporting framework.

The design for the report can be produced using the iReport tool, provided for free by JasperSoft Corporation, the company behind Jasper Reports. In addition, JasperSoft now provide an alternative design tool, called Jasper Studio. Both tools provide a 'what you see is what you get' (WYSIWYG) editor which allows you to position report elements exactly as they need to appear on the output document.

This document does not cover in any detail the mechanics of JasperReports, a powerful, feature-rich piece of software but also one with quite a steep learning curve for the uninitiated. However, we will make comments on particular aspects of JasperReports, relevant to the way it has been used in OrderFlow, for example, in areas such as subreports, handling of fonts and handling of images.

For serious usage, we recommend purchasing a book on JasperReports. We also recommend that before attempting to produce printable paperwork using JasperReport, you familiarise yourself with the custom reports framework, ideally through creation of a few custom reports.

Despatch notes and other printable reports use the same reporting framework as the OrderFlow custom reports. However, they differ in two key respects:

- custom reports typically use some form of the SQL data provider, which supports the extraction of data for the report using SQL. For despatch note reports, the data is provided as part of a more wide-ranging business process that involves other changes to the shipment (such as the creation of print item and shipment document entries).
- the majority of the custom reports output data rather than a printable document. For JasperReports-based paperwork, the *design* of the report is the JRXML document, which is an XML file that can be most easily edited using a design tool such as *iReport*.

Despatch Notes

As part of the pack and despatch process, customer paperwork invariably needs to be generated to be included with the outgoing shipment. This paperwork is typically referred to as the 'despatch note', and sometimes as the 'customer invoice'. The despatch note will typically include:

- the name and address of the recipient.
- details on the merchant, including site and return addresses, telephone numbers, etc.
- details on the lines despatched for the order, including product name, quantity, and often pricing information.
- details on the total price of the order, including shipping costs.
- instructions, plus terms and conditions, on returns.

Despatch notes and other paperwork can be produced to pixel-perfect accuracy in OrderFlow. OrderFlow achieves this by building on the capabilities of Jasper Reports, the world's most popular open source reporting framework.

Despatch notes and other printable reports use the same reporting framework as the OrderFlow custom reports. However, they differ in two key respects:

- custom reports typically use some form of the SQL data provider, which supports the extraction of data for the report using SQL. For despatch note reports, the data is provided as part of a more wide-ranging business process that involves other changes to the shipment (such as the creation of print item and shipment document entries).
- the majority of the custom reports output data rather than printable documents. For JasperReports-based paperwork, the *design* of the report is the a JRXML document, which is an XML file that can be most easily edited using a design tool such as *iReport*.

Report Design

The report design document for an OrderFlow despatch note is a JRXML file, the standard design file format for JasperReports. An example is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports
      http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
name="Sample Despatch Note" language="groovy"
pageWidth="595" pageHeight="842" columnWidth="567"
leftMargin="14" rightMargin="14" topMargin="14" bottomMargin="14"
isFloatColumnFooter="true" uuid="cccea0e2-75fa-4d3e-a0f0-91c946de3713">

 <style name="default" isDefault="true" fill="Solid"
        hAlign="Left" vAlign="Top" fontName="Verdana" fontSize="9"
      isBold="false" isItalic="false" isUnderline="false"
      isStrikeThrough="false"/>

<parameter name="logo" class="java.lang.String"/>
<parameter name="shipmentReference" class="java.lang.String"/>
<parameter name="totalNetPrice" class="java.lang.String"/>
...
<field name="sku" class="java.lang.String"/>
<field name="description" class="java.lang.String"/>
    ...
<pageHeader>
```

```
    <band height="265">
      <image isLazy="true" onErrorType="Blank">
        <reportElement uuid="ea357aa3-012a-494c-b13b-6ebca85b87ae"
            x="0" y="28" width="215" height="25"/>
        <imageExpression><![CDATA[$P{logo}]]></imageExpression>
      </image>
      <staticText>
        <reportElement uuid="a0472efb-bc4e-4b88-9ac8-2046407d9738"
            style="default" x="306" y="45" width="96" height="12"/>
        <textElement textAlignment="Right" markup="none">
          <font size="8"/>
        </textElement>
        <text><![CDATA[Shipment Reference:]]></text>
      </staticText>
      <textField isBlankWhenNull="true">
        <reportElement uuid="7d8c0c59-0c87-40a8-b854-193cb2927209"
            style="default" mode="Transparent" x="410" y="45"
            width="157" height="12" forecolor="#000000"/>
        <textElement verticalAlignment="Middle">
          <font size="8"/>
        </textElement>
        <textFieldExpression><![CDATA[$P{shipmentReference}]]></textFieldExpression>
      </textField>
      ...
    </band>
  </pageHeader>
  <columnHeader>
    <band height="25">
      <staticText>
        <reportElement uuid="146763e7-e3be-44b9-9db4-62795c6a4ab3"
            style="default" positionType="FixRelativeToBottom"
            x="2" y="5" width="63" height="12"/>
        <textElement>
          <font isBold="true"/>
        </textElement>
        <text><![CDATA[SKU]]></text>
      </staticText>
      ...
      <line>
        <reportElement uuid="00c99d61-c45d-44c9-9f6c-05dede5c0027"
            style="default" positionType="FixRelativeToBottom"
            x="0" y="20" width="565" height="1"/>
      </line>
    </band>
  </columnHeader>
  <detail>
    <band height="15">
      <textField isStretchWithOverflow="true">
        <reportElement uuid="a100709f-94d1-420a-b8fc-d53015ab675f"
            style="default" x="0" y="2" width="60" height="11"/>
        <textElement>
          <font size="8"/>
        </textElement>
        <textFieldExpression><![CDATA[$F{sku}]]></textFieldExpression>
      </textField>
      ...
    </band>
  </detail>
  <columnFooter>
    <band height="103">
      <staticText>
        <reportElement uuid="c53ae53b-0090-467a-b686-b1537bee6245"
            style="default" x="348" y="37" width="111" height="13"/>
        <textElement textAlignment="Right">
          <font isBold="true"/>
        </textElement>
        <text><![CDATA[Total Net Price:]]></text>
      </staticText>
      <textField isBlankWhenNull="true">
        <reportElement uuid="7d8c0c59-0c87-40a8-b854-193cb2927209"
            style="default" mode="Transparent"
          x="464" y="37" width="103" height="14" forecolor="#000000"/>
        <textElement verticalAlignment="Middle"/>
        <textFieldExpression><![CDATA[$P{totalNetPrice}]]></textFieldExpression>
      </textField>
      ...
    </band>
  </columnFooter>
</jasperReport>
```

The JRXML document contains both precise positioning information, and declarations of *parameters* and *fields*, for which information is extracted with the help of various expressions (explained in detail further in this chapter).

Editing this document directly by hand is generally not necessary. As with any formatted printable report on OrderFlow, the design be edited via a drag-and-drop interface using the freely available iReport or Jasper Studio tools.

Styles

Jasper Reports has the option to create *styles*, which help with the design of the report. Instead of entering the same property values for different elements throughout the report, you can simply create a single style and apply it to all required elements. In the example above, the style, called *default*, is defined at the top of the JRXML file and is used for all text fields and labels. Then only the properties, which differ from that style, are added in (e.g. the company name font size being bigger, some labels being right-aligned, elements being bold, etc.)

This is useful when making sure that the formatting throughout the report is identical, also saves time when adding in new fields or editing existing ones and automatically ensures that the code in the JRXML is as clean as possible.

Parameters and Fields

The despatch note generation environment will prepare the data, required to render the report. Typically, a despatch note is generated for each shipment.

Data is provided to the report in the form of **parameters** (for non-repeating data, such as the order header and shipment details) and **fields** (for repeating data, such as order lines).

Data can be extracted from any of these parameters or fields (a number of which are declared in the despatch note design example) using Java or Groovy expressions. While it is possible to access the OrderFlow data model directly from within the despatch note JRXML file, this is not desirable, as it means that the JRXML designer also needs to have a knowledge of scripting, a wider and technically more demanding skill set.

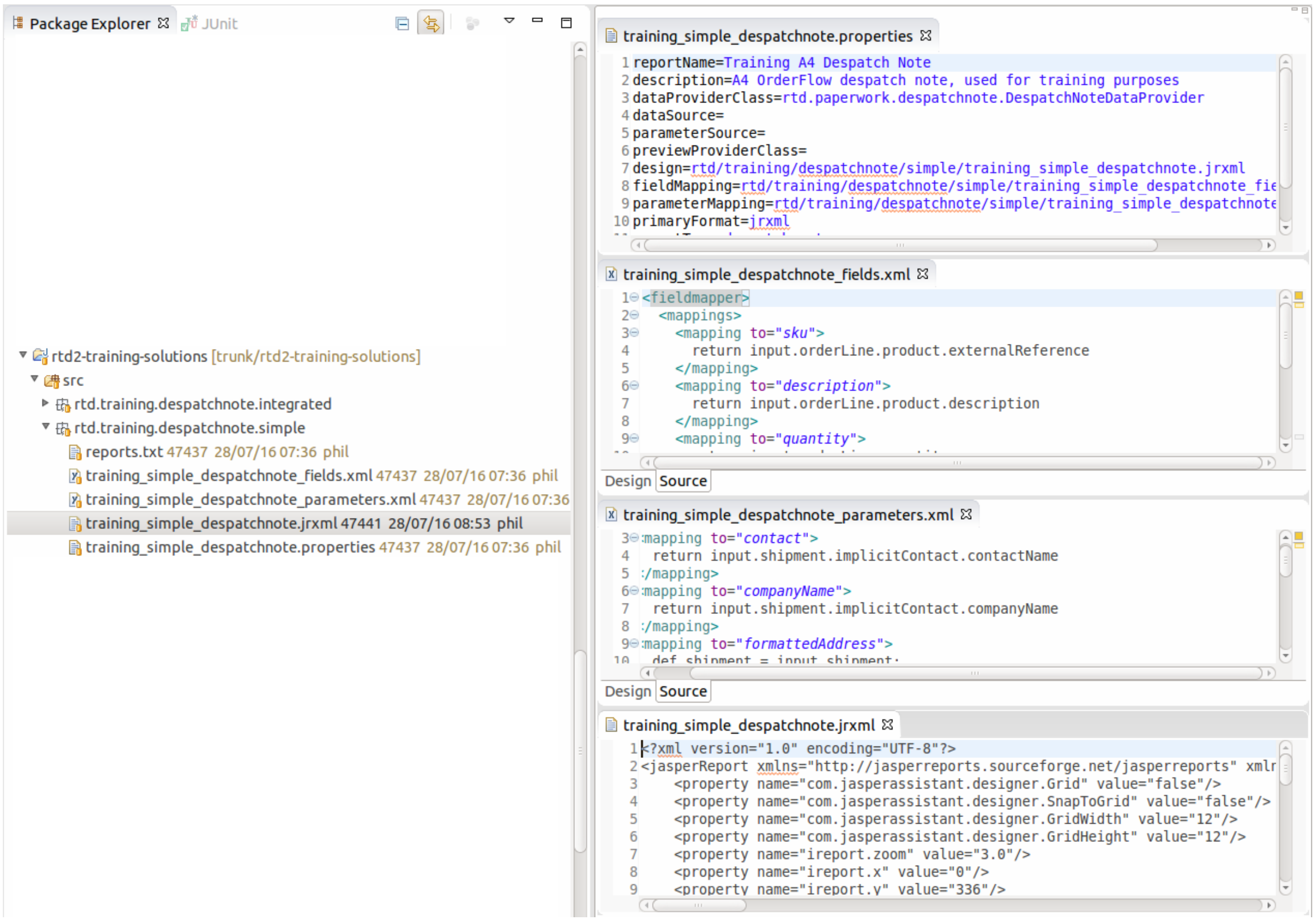
Instead, the recommended approach is to explicitly declare in the JRXML all of the *parameters* and *fields* individually as simple values, preferably just as strings. The JRXML file can in this way be limited purely to *design*.

This allows the report design to be easily edited by anyone with little or no programming knowledge, with data extraction left to programmers or the OrderFlow Technical Support team.

To facilitate this, OrderFlow provides a mechanism for populating the parameters and fields through a separate script outside of the JRXML.

By convention, this is done in two mapping files, for parameters and fields respectively, which by convention are named using based on name of the report (the JRXML file), followed respectively by *_parameters.xml* or *_fields.xml*.

The organisation of these files in a built-in despatch note report in the Eclipse IDE is shown below.



```
▼ rtd2-training-solutions [trunk/rtd2-training-solutions]
  ▼ src
    ▶ rtd.training.despatchnote.integrated
    ▼ rtd.training.despatchnote.simple
        reports.txt 47437  28/07/16 07:36  phil
        training_simple_despatchnote_fields.xml 47437  28/07/16 07:36  phil
        training_simple_despatchnote_parameters.xml 47437  28/07/16 07:36
        training_simple_despatchnote.jrxml 47441  28/07/16 08:53  phil
        training_simple_despatchnote.properties 47437  28/07/16 07:36  phil
```

training_simple_despatchnote.properties

```
reportName=Training A4 Despatch Note
description=A4 OrderFlow despatch note, used for training purposes
dataProviderClass=rtd.paperwork.despatchnote.DespatchNoteDataProvider
dataSource=
parameterSource=
previewProviderClass=
design=rtd/training/despatchnote/simple/training_simple_despatchnote.jrxml
fieldMapping=rtd/training/despatchnote/simple/training_simple_despatchnote_fie
parameterMapping=rtd/training/despatchnote/simple/training_simple_despatchnote
primaryFormat=jrxml
```

training_simple_despatchnote_fields.xml

```
<fieldmapper>
  <mappings>
    <mapping to="sku">
      return input.orderLine.product.externalReference
    </mapping>
    <mapping to="description">
      return input.orderLine.product.description
    </mapping>
    <mapping to="quantity">
```

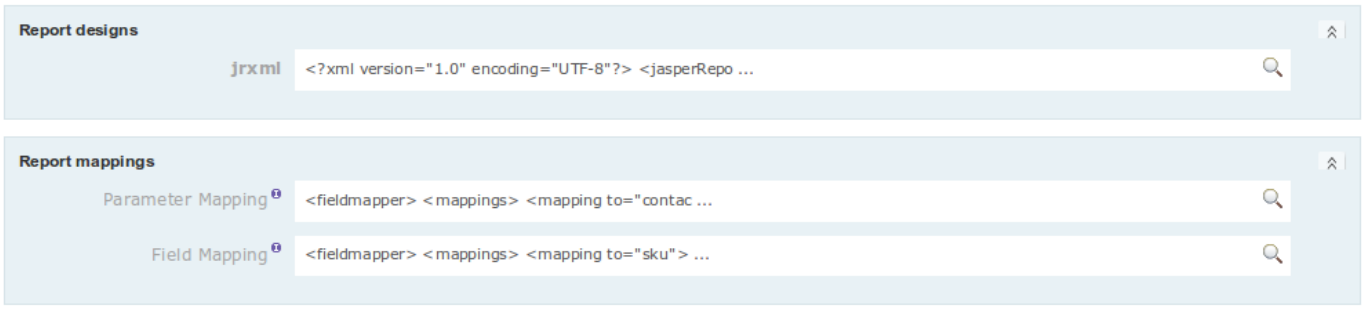
training_simple_despatchnote_parameters.xml

```
:mapping to="contact">
  return input.shipment.implicitContact.contactName
:/mapping>
:mapping to="companyName">
  return input.shipment.implicitContact.companyName
:/mapping>
:mapping to="formattedAddress">
  def shipment = input.shipment;
```

training_simple_despatchnote.jrxml

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlr
    <property name="com.jasperassistant.designer.Grid" value="false"/>
    <property name="com.jasperassistant.designer.SnapToGrid" value="false"/>
    <property name="com.jasperassistant.designer.GridWidth" value="12"/>
    <property name="com.jasperassistant.designer.GridHeight" value="12"/>
    <property name="ireport.zoom" value="3.0"/>
    <property name="ireport.x" value="0"/>
    <property name="ireport.y" value="336"/>
```

The same entries are visible on the OrderFlow GUI as shown below:



jrxml <?xml version="1.0" encoding="UTF-8"?> <jasperRepo ...

Parameter Mapping <fieldmapper> <mappings> <mapping to="contac ...

Field Mapping <fieldmapper> <mappings> <mapping to="sku"> ...

Note that the JRXML fields and parameters are declared individually in the JRXML, as in the example below:

```
<parameter name="formattedAddress" class="java.lang.String"/>
<parameter name="returnsAddress" class="java.lang.String"/>
<parameter name="returnsURL" class="java.lang.String"/>
<parameter name="returnsInformation" class="java.lang.String"/>
<parameter name="vatNumber" class="java.lang.String"/>
<parameter name="telephone" class="java.lang.String"/>
<parameter name="email" class="java.lang.String"/>
<parameter name="orderDate" class="java.util.Date"/>
<parameter name="shipmentReference" class="java.lang.String"/>
<parameter name="contact" class="java.lang.String"/>
<parameter name="logo" class="java.lang.String"/>
<parameter name="totalNetPrice" class="java.lang.String"/>
<parameter name="tax" class="java.lang.String"/>
<parameter name="totalShippingPrice" class="java.lang.String"/>
<parameter name="totalPrice" class="java.lang.String"/>
<parameter name="companyName" class="java.lang.String"/>
<parameter name="organisationName" class="java.lang.String"/>
<field name="quantity" class="java.lang.Integer"/>
<field name="sku" class="java.lang.String"/>
<field name="description" class="java.lang.String"/>
<field name="unitLinePrice" class="java.lang.String"/>
<field name="totalLinePrice" class="java.lang.String"/>
```

The parameters are declared in the JRXML file and used in the design document itself. For example, the `shipmentReference` parameter value will be accessible in the document using a declaration such as:

```
<textFieldExpression><![CDATA[$P{shipmentReference}]]></textFieldExpression>
```

In the next sections, we describe how these field and parameter values are populated through scripts.

Parameters

The parameter values for a despatch note report are extracted using scripts defined in the parameters mapping file, already introduced earlier. An example is shown below.

```
<fieldmapper>
 <mappings>
  <mapping to="formattedAddress">
    def shipment = input.shipment;
    return rtd.domain.Address.formattedAddress(shipment.implicitAddress, '\n');
  </mapping>
  <mapping to="shipmentReference">
    return input.shipment.externalReference
  </mapping>
  <mapping to="orderDate">
    def date = input.shipment.orderItem.placed;
    if (date != null) {
       return date;
    }
    return input.shipment.orderItem.created;
  </mapping>
  <mapping to="logo">
    def channel = input.shipment.orderItem.channel.externalReference;
    return 'logos/' + channel + '.png';
  </mapping>
  <mapping to="returnsAddress">
    def sharedProperties = input.sharedProperties;

    rtd.reports.util.ReportUtils.concatenate('\n', sharedProperties,
   'returns.business.name',
   'returns.site.address.line1',
   'returns.site.address.line2',
   'returns.site.address.line3',
   'returns.site.address.line4',
   'returns.site.address.posttown',
   'returns.site.address.postcode'
   ).toUpperCase();
  </mapping>
  <mapping to="returnsURL">
    return input.sharedProperties.get('returns.info.url')
  </mapping>
  ...
  <mapping to="totalNetPrice">
    return String.format('%.2f', input.shipment.orderItem.implicitGoodsPrice.net) + ' ' +
input.shipment.orderItem.currency;
  </mapping>
  ...
 </mappings>
</fieldmapper>
```

Scripting Context

Note that in each case, the `input` variable is used to reference the contextual data available to the script.

The following parameters are available for use in the parameters mapping script include the list below. Note that it is possible, but not best practice, to access these values directly in the JRXML template.

Expression	Type	Description
input.order	`rtd.domain.OrderItem`	Holds a reference to the order, containing the shipment, for which the despatch note is generated. Java object, so `OrderItem` domain model can be navigated to extract values.
input.shipment	`rtd.domain.Shipment`	Holds a reference to the shipment, for which the despatch note is generated. Java object, so `Shipment` domain model can be navigated to extract values.
input.sharedProperties	`java.util.Map`	Holds reference to property values for properties in the group 'courier_shared'.
input.addressProperties	`java.util.Map`	Holds reference to property values for properties in the group 'address'.
input.Properties	`java.util.Map`	Holds reference to property values for properties in the property group associated with the shipments currently selected courier. Use the **Setup -> Courier** to determine the courier's current property group.

For each of the property-related values, you can use the **Setup -> Properties -> Search** menu to determine the list of individual properties available and values set for your current channel, organisation and site context.

Examples

The code contains examples of different ways to extract data and format it.

The *formattedAddress*, as the name suggests, represents the shipping address, formatted correctly by using the method *formattedAddress* from the *Address* class and the data from *input.shipment.implicitAddress* from the RTD domain.

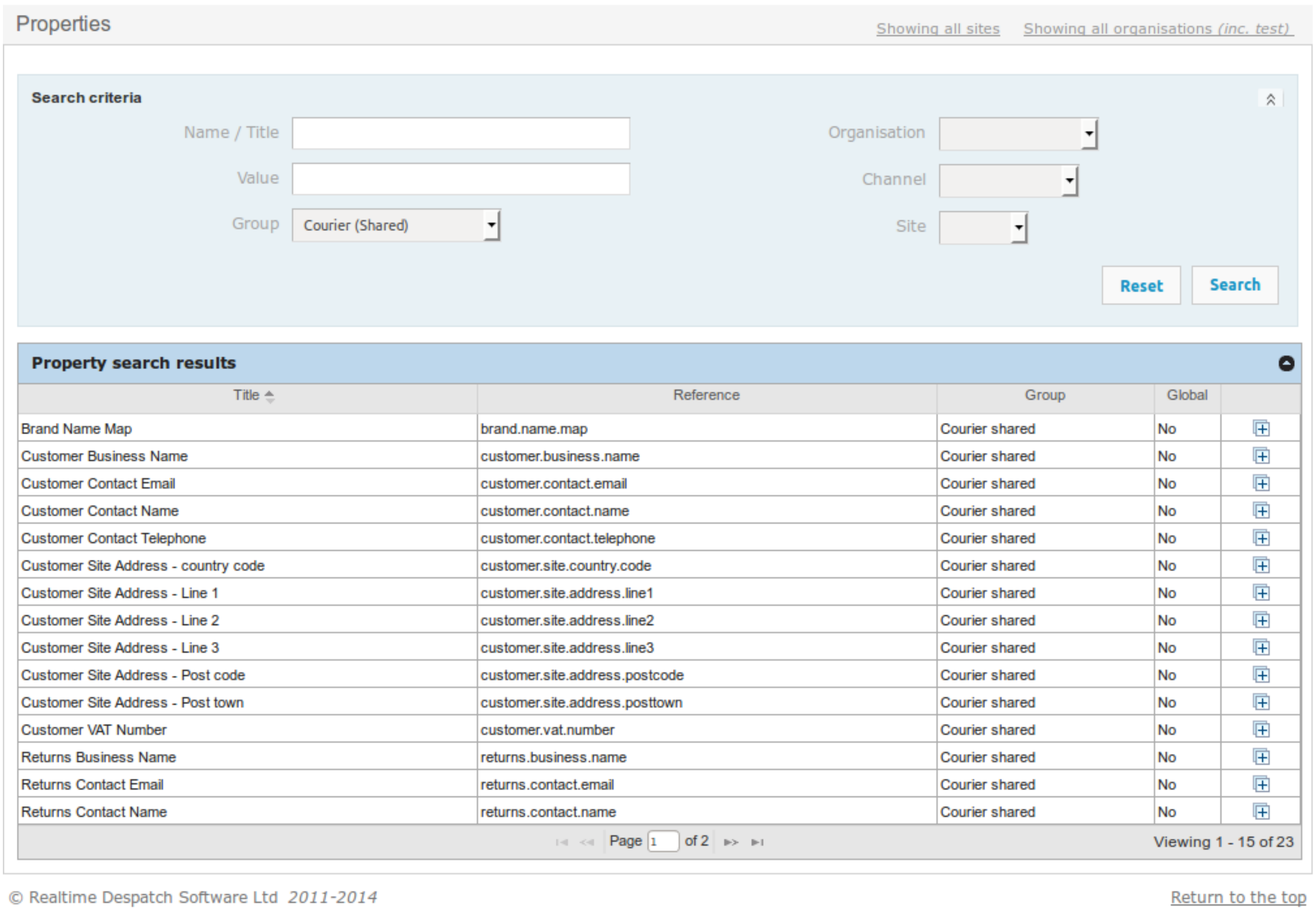
shipmentReference is the most straightforward example - it simply returns the output from *input.shipment.externalReference*.

orderDate contains a conditional statement - it checks whether the date of placing an order exists and if so, returns that data. Otherwise, it returns the date of creating the order.

logo actually contains a path to the image and the specifics around *logos and images* - themselves are described in more detail further in this document.

The *returnsAddress* uses the RTD *concatenate* method to create one text field that contains the returns business name and the whole address. If separate text fields were used instead, this would have created gaps (empty lines) where data was omitted, e.g. the address contains only 3 lines.

For the *returnsURL*, as well as all other information, related to returns, the *Couriers (Shared)* properties have to be accessed. In the example above, only the URL is depicted with the rest being omitted, but they can easily be accessed by using the *sharedPropertes*' *get* method. All fields can be viewed from the *Setup* tab - they are located at the bottom of that page. Below is a screenshot of some of the *Courier (Shared)* properties:



Property search results

Title	Reference	Group	Global	
Brand Name Map	brand.name.map	Courier shared	No	
Customer Business Name	customer.business.name	Courier shared	No	
Customer Contact Email	customer.contact.email	Courier shared	No	
Customer Contact Name	customer.contact.name	Courier shared	No	
Customer Contact Telephone	customer.contact.telephone	Courier shared	No	
Customer Site Address - country code	customer.site.country.code	Courier shared	No	
Customer Site Address - Line 1	customer.site.address.line1	Courier shared	No	
Customer Site Address - Line 2	customer.site.address.line2	Courier shared	No	
Customer Site Address - Line 3	customer.site.address.line3	Courier shared	No	
Customer Site Address - Post code	customer.site.address.postcode	Courier shared	No	
Customer Site Address - Post town	customer.site.address.posttown	Courier shared	No	
Customer VAT Number	customer.vat.number	Courier shared	No	
Returns Business Name	returns.business.name	Courier shared	No	
Returns Contact Email	returns.contact.email	Courier shared	No	
Returns Contact Name	returns.contact.name	Courier shared	No	

Page 1 of 2 Viewing 1 - 15 of 23

© Realtime Despatch Software Ltd 2011-2014 Return to the top

The last property in the example XML file is a total price, in particular - the *totalNetPrice*. This has been formatted by using the Java *String.format* method in order to display two digits after the decimal separator and also include the currency after the price.

Fields

The field mappings allow you to access values for repeated data within the report, which in the case of despatch notes will normally be in the form of order lines. One common exception is for despatch notes that are also used as picking notes, the field values access the order line location entries.

In the example field mapping file below, only five field values are populated by navigating data in the order line.

Populating these fields is relatively straightforward, and typically more easily done than for parameters. For the JRXML above, data for the fields is extracted as follows:

```
<fieldmapper>
 <mappings>
   <mapping to="sku">
     return input.orderLine.product.externalReference
   </mapping>
   <mapping to="description">
     return input.orderLine.product.description
   </mapping>
   <mapping to="quantity">
     return input.orderLine.quantity
   </mapping>
   <mapping to="unitLinePrice">
     return input.orderLine.implicitUnitPrice.gross
   </mapping>
   <mapping to="totalLinePrice">
     return input.orderLine.implicitTotalPrice.gross
   </mapping>
 </mappings>
</fieldmapper>
```

All data is extracted from the *orderLine* field, since items from the shipment are contained within it.

Scripting Context

The values available to the fields mapping script will include either of the following in the table below.

As with the parameters, the field mapping file is aimed at users familiar with the structure of the OrderFlow domain model.

Expression	Type	Description
input.orderLine	`rtd.domain.OrderLine`	Holds a reference to the current order line. Java object, so `OrderLine` domain model can be navigated to extract values.
input.orderLineLocation	`rtd.domain.OrderLineLocation`	Similar to the `orderLine` field. The main difference is that it holds a separate value for each order line and location combination. For order lines picked from multiple locations, multiple entries will appear. Suitable for cases where the despatch note is also used as a picking note.

The question on whether to access data via the `OrderLine` or the `OrderLineLocation` depends on which **data provider** is used.

Data Providers

Data providers were previously introduced in Custom Reports chapter. For despatch note generation, a number of specialist data providers have been created.

The most commonly used ones are listed with a description below:

Class	Field Type	Description
DespatchNoteDataProvider	`OrderLine`	Most commonly used data provider which simply provides a list of order lines in the current shipment.
LocationDespatchNoteDataProvider	`OrderLineLocation`	Data provider which provides a list of order line locations. Used for despatch notes which are also used as picking reports.
OrderItemDespatchNoteDataProvider	`OrderLine`	Data provider which for multi shipment reports, will make available to the despatch note all order lines, for the order, including ones in other shipments in the same order. Useful for despatch notes where lines in 'shipments despatched' separately need to be listed.
PartShippedDespatchNoteDataProvider	`OrderLine`	Variation on `DespatchNoteDataProvider`, with mechanism to set the `isLastShipment` parameter if there are no more shipments to be sent out for the curernt order.
PackageAwareDespatchNoteDataProvider	`OrderLine` with embellishments	Extension of `DespatchNoteDataProvider`, which allows for a separate despatch note document to be created for each package in a shipment, with package specific despatch notes detailing the contents of the specific package.

There are other built-in despatch note data providers available, but the use of these is more esoteric.

Logos and Images

OrderFlow despatch notes and other printed document can contain images, particularly useful for displaying company logos.

There are a number of elements to be understood in the way that images should be used on OrderFlow.

- the image element in the design (using an image element in the JRXML file).
- the mechanism for identifying the image (using an image expression within the image element)
- the mechanism for provisioning images (using a File Resource entry).

Declaring Images

Images are declared using an `imageExpression` element in the JRXML design document.

```
<image isLazy="true" onErrorType="Blank">
 <reportElement uuid="ea357aa3-012a-494c-b13b-6ebca85b87ae"
         x="0" y="28" width="215" height="25"/>
<imageExpression><![CDATA[$P{logo}]]></imageExpression>
</image>
```

Two of the element properties are worth considering in more detail: *isLazy* and *onErrorType*.

isLazy

A flag that specifies whether the image should be loaded and processed when filling the report or when exporting it - in case the image is not available at fill time. For logos and other images, the value of `true` should be used. This allows the image to be cached more effectively, removing the need for the same image data to be embedded each despatch note generated. This improves performance and reduces bandwidth, used in producing despatch notes.

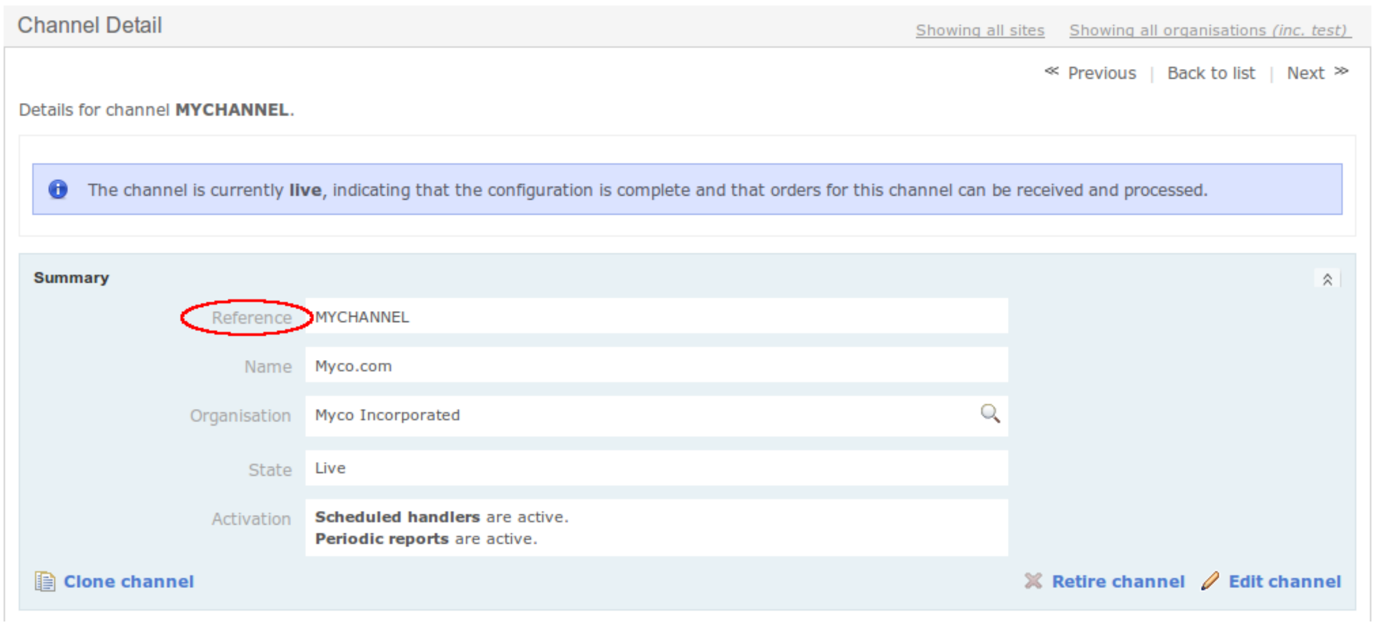
onErrorType

Specifies the output if the image is unavailable when the system tries to load it. There are three possible options for this property - *Error*, *Blank* and *Icon*. For despatch note generation, the second option, *Blank*, is best. If an image is not present at the point where it is viewed or printed, a blank space will be outputted, rather than having an exception thrown or an icon appear in the image's place.

Uploading Images

To add a new logo to the file resources from OrderFlow, click on the *Setup* tab and from the left-hand side menu select *File Resources*. Then either select the submenu *New*, or click *Add new file resource* from the page on the right. Both of these actions lead to the *New File Resource* page.

Any image detail can be set from this page. In this case, we want the *Reference* to be something like *logos/mychannel.png*. To find the exact reference of the channel, from OrderFlow, go to *Setup*, then *Channels* and find the channel you need. The name that is required is the channel *reference* - in the example below it is *MYCHANNEL*.


Channel Detail

Showing all sites
Showing all organisations (inc. test)
« Previous | Back to list | Next »

Details for channel **MYCHANNEL**.

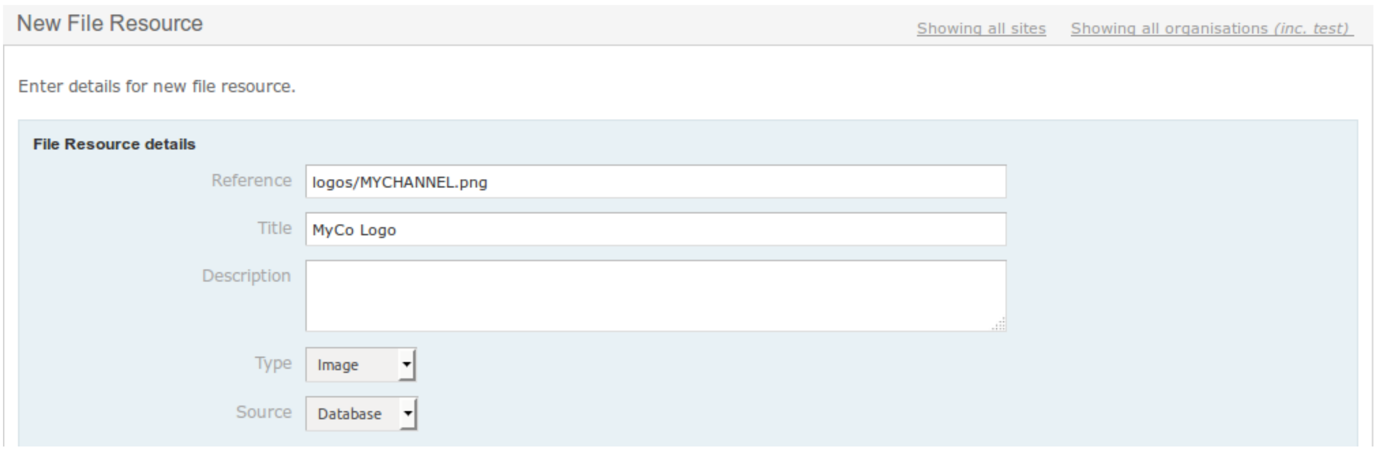
The channel is currently **live**, indicating that the configuration is complete and that orders for this channel can be received and processed.

Summary

Reference	MYCHANNEL
Name	Myco.com
Organisation	Myco Incorporated
State	Live
Activation	**Scheduled handlers** are active. **Periodic reports** are active.

Clone channel
Retire channel
Edit channel

The *Title* can be the name of the channel, followed by "Logo" (e.g. MyChannel Logo) to make it easier to distinguish amongst the file resources. Once the *Reference* and *Title* have been set, click *Create* at the bottom of the page. This will create a new file resource with the data that you have set up for it.


New File Resource

Enter details for new file resource.

File Resource details

Reference	logos/MYCHANNEL.png
Title	MyCo Logo
Description	
Type	Image
Source	Database

Before uploading the image, make sure that the *Type* of the file resource is set to *Image*. Then click on *Upload image* at the bottom of the screen, browse for and select the image you want to appear as your logo and click *Update*. Your logo should now be displayed at the bottom of the page.

Image Dimensions

An understanding of how image dimension are calculated is important in ensuring high quality image printouts.

JasperReports works on the basis of 72 DPI (Dots per Inch). However, images typically require a 300 DPI resolution for good quality printing.

The example despatch note contains a logo with a size of 215 by 25 pixels in *iReport* (with a 72 DPI resolution). This converts to approximately a 76x9mm image when printed. For 203 DPI (some Citizen and Zebra printers) a 606 by 70 pixels image is required. For printers operating at 300 DPI, best results are achived if the image is of size 896 by 104 pixels.

The conversions above are illustrated in the table below:

Millimetres (mm)	Pixels - 72 DPI (JRXML)	Pixels - 203 DPI	Pixels - 300 DPI
75.85	215.01	606.2	895.87
8.82	25	70.49	104.17

A note on some specific examples follows:

JasperReport Pixels to Millimetres

Divide the pixels by 72 (to get the number of inches), then multiply the value by 25.4, i.e. **(pixels/72)*25.4**.

Image in Millimetres to Pixels

You will need to do this conversion when working out the required dimensions of an image in pixels, for a given space in millimetres.

First convert the millimetres in inches and work from there, by dividing the size in millimetres by 25.4, i.e. **millimetres/25.4**. From there, converting to the respective size in pixels is straightforward - the image size in inches is simply multiplied by the DPI:

Millimetres(mm)	Inches	Pixels - 72 DPI (JRXML)	Pixels - 203 DPI	Pixels - 300 DPI
x	y = x/25.4	y*72	y*203	y*300

Using the above calculations will be helpful in ensuring that logos (and other image) in the report is printed out with the best possible quality. For labels with embedded barcodes, this is particularly important in ensuring that that the barcodes can be scanned off the printed label stock.

Subreports and Concatenated Reports

When preparing despatch notes, advanced techniques such as the use of **subreports** and **concatenated** reports are often required.

Subreports

Subreports allow for a primary or master report to *embed* a secondary or subreport. This feature is particularly useful for cases where a part of a report needs to be reused.

A good example is when the despatch note embeds a courier label, with the latter being detatchable from the main document via a peel-off section of integrated stationery. The courier label has a design which is specific to the courier being used, while the rest of the despatch note has a bespoke design for the customer, with a custom layout, use of customer logos, etc.

In order to make it easier for different sized courier labels to be embedded, the despatch note by itself is also embedded as a subreport, therefore the master report only contains the code for integrating those two.

Subreports, as the name suggests, are reports, which are embedded in other reports. In the aforementioned example, the main despatch note and the courier label are the embedded reports, whereas the master reports simply contains placeholders for these.

Below is an example (the JRXML file) of such a master report with an embedded 190x90 courier label at the bottom of the page and the embedded header of the despatch note at the top of the page:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports
    http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
    name="integrated_a4_190x90_bottom" language="groovy"
    pageWidth="595" pageHeight="842" columnWidth="595"
    leftMargin="0" rightMargin="0" topMargin="0"
    bottomMargin="0" uuid="1906a486-9497-44aa-a9f3-aa644caf2260">
<background>
<band splitType="Stretch"/>
</background>
<detail>
<band height="559" splitType="Stretch">
  <subreport isUsingCache="true" runToBottom="false">
    <reportElement uuid="c06fbe33-f558-4564-bca8-58dd91c5ecc1"
             x="0" y="0" width="595" height="537"/>
    <parametersMapExpression>
        <![CDATA[$P{REPORT_PARAMETERS_MAP}.courier_label_parameters]]>
    </parametersMapExpression>
    <dataSourceExpression>
        <![CDATA[$P{REPORT_PARAMETERS_MAP}.despatch_note_datasource]]>
    </dataSourceExpression>
    <subreportExpression>
        <![CDATA[$P{REPORT_PARAMETERS_MAP}.despatch_note]]>
    </subreportExpression>
  </subreport>
</band>
</detail>
<lastPageFooter>
<band height="283">
  <subreport runToBottom="false">
    <reportElement uuid="aedb7a2a-8fd2-4393-9be2-f1e04776a5d0"
            x="28" y="0" width="538" height="255"/>
    <parametersMapExpression>
        <![CDATA[$P{REPORT_PARAMETERS_MAP}.courier_label_parameters]]>
    </parametersMapExpression>
```

```
      <dataSourceExpression>
          <![CDATA[$P{REPORT_PARAMETERS_MAP}.courier_label_datasource]]>
      </dataSourceExpression>
      <subreportExpression>
          <![CDATA[$P{REPORT_PARAMETERS_MAP}.courier_label]]>
      </subreportExpression>
    </subreport>
  </band>
  </lastPageFooter>
  </jasperReport>
```

PARAMETER MAPPINGS IN SUBREPORTS

Parameter mappings can be used in subreports.

Suppose for example we want to use a parameter mapping to output the return address using scoped properties. This can be done using the following parameter mapping entry.

```
<fieldmapper>
  <mappings>
    <mapping to="returnsAddress">
      def sharedProperties = input.sharedProperties;
      return rtd.reports.util.ReportUtils.concatenate('\n', sharedProperties,
                'returns.business.name',
                'returns.site.address.line1',
                'returns.site.address.line2',
                'returns.site.address.line3',
                'returns.site.address.line4',
                'returns.site.address.posttown',
                'returns.site.address.postcode'
                ).toUpperCase();
    </mapping>
   </mappings>
</fieldmapper>
```

When the subreport parameter mapping code is evaluated, the resulting value is stored against a parameter map that is visible only to the subreport. This parameter map is stored using the the key `%subreport%_parameters`.

As a result, the `parametersMapExpression` in the previous section *in the master report* (which contains the reference to the subreport) is required as shown below to allow the JRXML template to access the `returnsAddress` parameter described above.

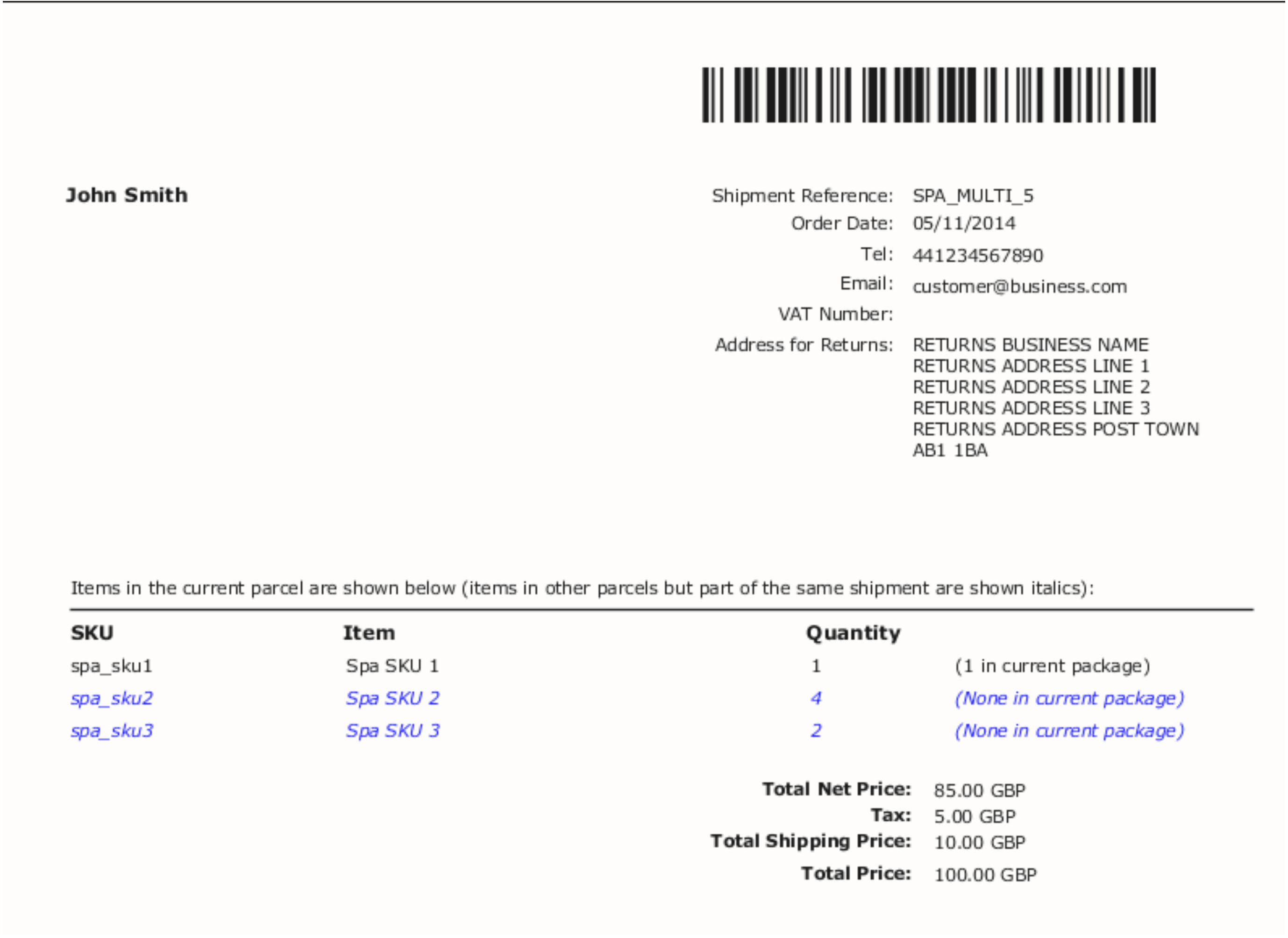
```
<![CDATA[$P{REPORT_PARAMETERS_MAP}.courier_label_parameter)]]>
```

Concatenated Reports

Concatenated reports are a more advanced feature of Jasper Reports, primarily used with subreports and report groups. Generating a concatenated report is not covered in this document, but if required, OrderFlow can provide information on how to create them or can help by generating a custom concatenated report to cover your needs.

Package Aware Despatch Notes

As of OrderFlow 3.7.7, for multi-package shipments it is possible to use a specific data provider that allows for the creation of a separate document per package. An example is shown below:



John Smith

Shipment Reference:	SPA_MULTI_5
Order Date:	05/11/2014
Tel:	441234567890
Email:	customer@business.com
VAT Number:	
Address for Returns:	RETURNS BUSINESS NAME RETURNS ADDRESS LINE 1 RETURNS ADDRESS LINE 2 RETURNS ADDRESS LINE 3 RETURNS ADDRESS POST TOWN AB1 1BA

Items in the current parcel are shown below (items in other parcels but part of the same shipment are shown italics):

SKU	**Item**	**Quantity**	
spa_sku1	Spa SKU 1	1	(1 in current package)
spa_sku2	*Spa SKU 2*	*4*	*(None in current package)*
spa_sku3	*Spa SKU 3*	*2*	*(None in current package)*

Total Net Price:	85.00 GBP
Tax:	5.00 GBP
Total Shipping Price:	10.00 GBP
Total Price:	100.00 GBP

This feature is enabled through the use of the despatch note data provider `rtd.paperwork.despatchnote.PackageAwareDespatchNoteDataProvider`, plus changes to the JRXML document.

The JRXML is modified for a group to be declared per package, which refers to the `groupKey` field. This results in a new page started with each package.

```
<field name="groupKey" class="java.lang.String"/>
<group name="Package" isStartNewPage="true">
   <groupExpression><![CDATA[$F{groupKey}]]></groupExpression>
   <groupHeader>
      <band height="270">
         ... group header content
      </band>
   </groupHeader>
   <groupFooter>
      <band height="71">
         ... group footer content
      </band>
   </groupFooter>
</group>
<detail>
  <band height="15" splitType="Stretch">
  ... band content
  </band>
</detail>
```

The `groupHeader` and `groupFooter` elements are then used instead of the `pageHeader` and `pageFooter` bands for the delivery address and return addresses and other top level information required in the despatch note.

Internationlised Despatch Notes

An internationalised despatch note is one that supports local language text through a single report. Without internationalisation, creating separate language version of despatch notes would require a separate despatch note report per language, resulting in a substantial amount of work in setting up and maintaining multi-language environments.

In the next section we describe how despatch notes are internationalised in OrderFlow, and follow this with a discussion on how different language versions are identified on the system.

Report Setup

The starting point for an internationalised despatch note is normally an existing working despatch note which contains text strings which are 'hard-coded' in the JRXML file in a single language. The idea is to replace these with resource references, where the actual text can be retrieved from a set of external files. This set of files is known as a *message bundle*.

If these message resources are not supplied as files built in to the application, they can be uploaded manually from the **Setup -> File Resources** menu. The example below shows a couple of built-in message resource files.

Message Resource

The message resources are properties files, with entries such as the following (for English).

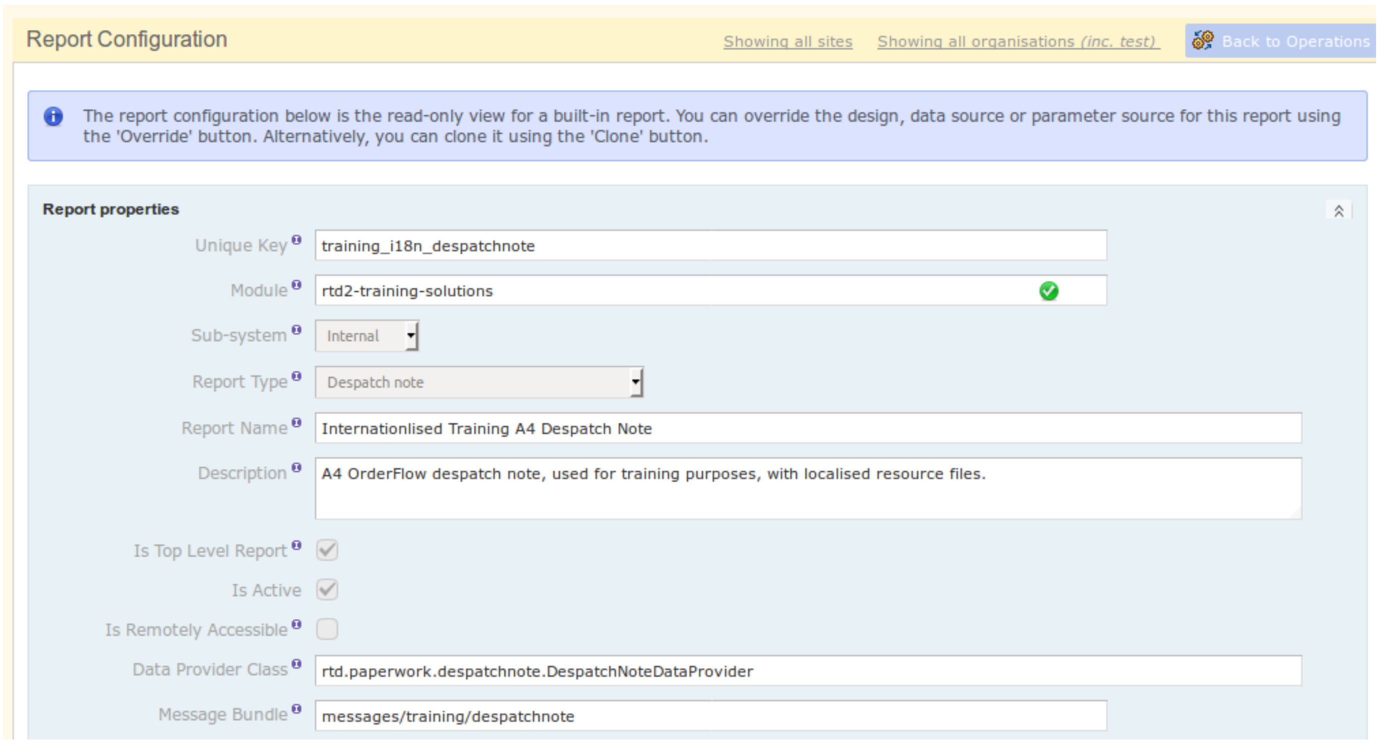
```
page.header.help=Can we help?
page.footer.footer.text=Thank you for shopping with us. We hope you are happy with your order.
```

The equivalent entries in the German language message resource are shown below:

```
page.header.help=Wie können wir Ihnen behilflich sein?
page.footer.footer.text=Vielen Dank für Ihre Bestellung. Wir hoffen Ihre Bestellung war zufriedenstellend.
```

With the resource bundle in place and ready to be used, we need to update the despatch note to use internationalised text.

The first step here is to identify the message bundle to be used. This can be done using the Message Bundle field on the Report Configuration screen for the despatch note, shown below.



The report configuration below is the read-only view for a built-in report. You can override the design, data source or parameter source for this report using the 'Override' button. Alternatively, you can clone it using the 'Clone' button.

Report properties

Unique Key	training_i18n_despatchnote
Module	rtd2-training-solutions ✓
Sub-system	Internal
Report Type	Despatch note
Report Name	Internationlised Training A4 Despatch Note
Description	A4 OrderFlow despatch note, used for training purposes, with localised resource files.
Is Top Level Report	✓
Is Active	✓
Is Remotely Accessible	
Data Provider Class	rtd.paperwork.despatchnote.DespatchNoteDataProvider
Message Bundle	messages/training/despatchnote

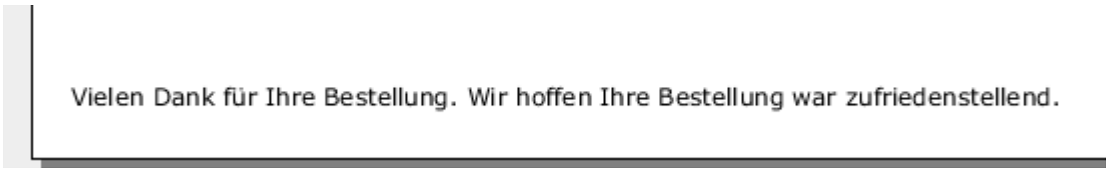
Note that the value here corresponds with the reference of the resource entry (excluding the '.properties' suffix). The exact mechanism for this is discussed in more detail in the Locale Identification section.

Next, the report design needs to be updated. The JRXML example shown below.

```
<textField>
   <reportElement uuid="85d01f90-e814-4aab-aa39-8e7590b4a84b" style="default" x="0" y="9" width="565"
height="14">
   </reportElement>
   <textElement markup="none"/>
   <textFieldExpression><![CDATA[$R{page.footer.footer.text}]]></textFieldExpression>
</textField>
```

Note that the message reference 'page.footer.footer.text' corresponds with the key for one of the entries in the message property file.

In the example above, the localised German text may appear as in the below.



Vielen Dank für Ihre Bestellung. Wir hoffen Ihre Bestellung war zufriedenstellend.

Locale Identification

The message files that are described in the previous section can vary according to language, country, and can also have a variant. This combination of information is known as a *locale*.

For example, it is possible to have separate resource files for Swiss and Austrian German, each with their own sets of text.

One question is how the *Locale* to be used can be identified in OrderFlow. The language is determined from the `languageCode` field of the order. The country code is mapped from the order or shipment's delivery country code field. It is also possible to have further variations in the resource file used based on the order's `source` field. This for example, would allow for different messages to be displayed for web site versus Amazon or eBay orders, even for the same country or language combination.

The message resource files are named used the following naming covention:

```
[message bundle reference]_[locale suffix].properties
```

The locale suffix has values such as those shown in the next table.

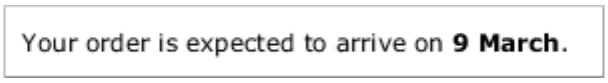
Value	Description
`despatchnote.properties`	The default resource file, used if no locale information is specified.
`despatchnote_de.properties`	A resource file used for German orders.
`despatchnote_de_CH.properties`	A resource file used for Swiss German orders. Note the use of the CH country code to include Switzerland in the locale.
`despatchnote_de_CH_amazon.properties`	A resource file used for Swiss German orders that have originated from the Amazon channel.

Parameter and Field Mappings with Localised Messages

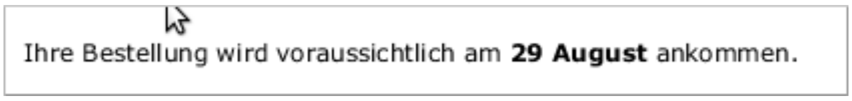
The use of resource entries such as `$R{page.footer.footer.text}` support simple cases where one simply needs to replace some static text with a single language-specific value.

More complex cases may require some scripting. Suppose you wish to support output of the following text:

In English:


Your order is expected to arrive on **9 March**.

In German:


Ihre Bestellung wird voraussichtlich am **29 August** ankommen.

In the above case, we cannot simply use resource references in the JRXML, as there is additional dynamic information to be supplied in populating the text string.

In order to achieve the above, from Orderflow 3.8.0 we can use a Parameter Mapping, from which we can populate the message string appropriately. The mechanism for doing this is described next.

Firstly, the JRXML will contain an additional field declaration and design element, as shown below.

```
<parameter name="expectedArrivalText" class="java.lang.String"/>
...
<textField isBlankWhenNull="true">
   <reportElement uuid="7d8c0c59-0c87-40a8-b854-193cb2927209" style="default"
      mode="Transparent" x="2" y="208" width="563" height="14" forecolor="#000000">
   </reportElement>
   <textElement verticalAlignment="Middle" markup="styled"/>
   <textFieldExpression><![CDATA[$P{expectedArrivalText}]]></textFieldExpression>
</textField>
```

We then populate the parameter value using an entry such as the following:

```
<mapping to="expectedArrivalText">
  def dateText = new java.text.SimpleDateFormat('d MMMM').format(input.shipment.requestedDeliveryDate)
  return messages.message('shipment.expected.arrive.text', dateText);
</mapping>
```

Note that the message resource entry can be obtained using the expression `messages.message('shipment.expected.arrive.text')`. However, we can additionally specify multiple arguments for the dynamic values we want to supply.

The bundle resources themselves need the additional entries, which can be added as follows for English.

```
shipment.expected.arrive.text=Your order is expected to arrive on <b>{0}</b>.
```

The German equivalent entry is shown below.

```
shipment.expected.arrive.text=Ihre Bestellung wird voraussichtlich am <b>{0}</b> ankommen.
```

Note that the placeholders are defined using the convention {0}, {1}, etc, with the number corresponding to the argument index in the `messages.message(...)` expression.

Labels

One of the primary uses of Jasper Reports in OrderFlow is in the creation of labels. Typically, labels will include barcodes to allow them to be scanned using a handheld terminal (HHT).

The most common types of labels produced in OrderFlow include the following:

- product labels with barcodes
- location labels
- shipment courier labels
- labels for licence plates
- custom labels to support particular operational processes

Label Reports

As with despatch notes, labels are created in OrderFlow using using Jasper Reports. Labels can either be created using *standalone* reports, or using reports that are *embedded* into other processes. For examples, location labels would typically be created using standalone reports, while shipment courier labels are created as part of a 'Print Label' operation, normally invoked by a packer on the packing screen.

Standalone Labels

Standalone Labels are created using a combination of the following:

- a data source, typically defined using SQL, to retrieve the data to be used to generate the label
- optional parameters to restrict the data set
- a Jasper Reports JRXML design

The JRXML design will typically include one or more barcodes.

An example of a standalone label is shown below:

Data Source

The data provider class is typically `rtd.reports.dataprovider.SqlDataProvider`, with a data source such as the following:

```
SELECT
   p.externalReference AS 'sku',
   p.barcode AS 'barcode',
   p.description AS 'description',
   c.name AS 'category'
FROM product p
LEFT JOIN product_category c ON p.categoryId = c.id
WHERE
p.deleted = 0
AND
p.activated is TRUE
#[include:text_string] AND
 (p.externalReference LIKE '%${literal:text_string}%'
OR
  p.barcode LIKE '%${literal:text_string}%'
OR
  p.description LIKE '%${literal:text_string}%')
#[include]
ORDER BY p.externalReference
```

The above example is for a product barcode generating report, which generates barcodes for labels matching the `text_string` parameter.

Design

The Jasper Report design for the label above includes the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport
      xmlns="http://jasperreports.sourceforge.net/jasperreports"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports
                          http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
      name="A5Label" pageWidth="290" pageHeight="290" orientation="Landscape"
      whenNoDataType="NoDataSection" columnWidth="270"
      leftMargin="10" rightMargin="10" topMargin="10" bottomMargin="10">
   <style name="default" isDefault="true" fontName="Verdana" fontSize="8">
      <box topPadding="0" leftPadding="0"/>
   </style>
   <style name="small" style="default" fontSize="7"/>
   <style name="header" vAlign="Middle" fontName="Verdana" fontSize="8"/>
   <field name="sku" class="java.lang.String">
      <fieldDescription><![CDATA[]]></fieldDescription>
   </field>
   <field name="barcode" class="java.lang.String">
      <fieldDescription><![CDATA[]]></fieldDescription>
   </field>
   <field name="description" class="java.lang.String">
      <fieldDescription><![CDATA[]]></fieldDescription>
   </field>
   <field name="category" class="java.lang.String">
      <fieldDescription><![CDATA[]]></fieldDescription>
   </field>
   <detail>
      <band height="270">
            ...
            <componentElement>
            <reportElement x="4" y="9" width="171" height="58">
               <printWhenExpression><![CDATA[!org.apache.commons.lang.StringUtils.isNumeric($F{barcode})]]></
printWhenExpression>
            </reportElement>
            <jr:Code128 xmlns:jr="http://jasperreports.sourceforge.net/jasperreports/components"
                        xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports/components
                                            http://jasperreports.sourceforge.net/xsd/components.xsd"
                        moduleWidth="2.0" textPosition="none">
               <jr:codeExpression><![CDATA[$F{barcode}]]></jr:codeExpression>
            </jr:Code128>
         </componentElement>
      </band>
   </detail>
</jasperReport>
```

Notice of the use of the `Code128` component; in this case as Code 128 barcode is being generated. More detail on the barcode is configuration is included in the 'Barcodes' section below.

Process-embedded Labels

Apart of the content, the main difference between standalone and embedded labels is in the data source. While standalone labels use a generic SQL-oriented data provider, embedded labels use data providers that only work within specific processes or interactions.

For shipment courier labels, some of the available data providers are shown below (**note** that the documentation for these applies as of OrderFlow 3.7.7):

Name	Usage
`rtd.courier.label.ParametersOnlyLabelDataProvider`	Used for labels for which uses only the supplied parameter values only. No Jasper Report Fields are used
`rtd.courier.label.EmbeddedImageLabelDataProvider`	Labels with a single embedded label image, accessible using either 'labelImageFile' or 'labelImageStream' Parameter ($P{}) values.
`rtd.courier.label.EmbeddedImagesLabelDataProvider`	Labels with one or more embedded images (usually, one per pacakge), accessible using either 'labelImageFile' or 'labelImageStream' Field ($F{}) values.
`rtd.courier.label.IndexedShipmentOrPackageLabelDataProvider`	Labels for which the elements are defined within the Jasper Report, allowing for one or more field values.

Note that for all shipment courier labels, many of the parameter values are set up using the `LabelDataPopulator` class.

Name	Usage
`shipment`	The current shipment for which the label is being produced.
`package`	If the label is being produced just for a single package, then identifies this package.
`shipmentOrPackage`	Allows data that is common to shipments and packages to be referenced in a uniform way.
`shipmentCourierOptions`	Represents courier options for the selected courier and service.
`despatchNoteProperties`	Scoped properties from the 'despatch_note' property group.
`sharedProperties`	Scoped properties from the 'courier_shared' property group.
`addressProperties`	Scoped properties from the 'address' property group.
`[courier]Properties`	Scoped properties from the courier-specific property group.
`shipmentCourierProperties`	Represents courier-specific data that has been populated against the shipment.
`despatchReference`	The shipment or package despatch reference.
`countryLookup`	Can be used to look up the country name from the country code stored against the shipment.

An example JRXML file which uses `EmbeddedImagesLabelDataProvider` is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports http://
jasperreports.sourceforge.net/xsd/jasperreport.xsd" name="4 by 6 label" pageWidth="288" pageHeight="432"
columnWidth="288" leftMargin="0" rightMargin="0" topMargin="0" bottomMargin="0">
   <field name="labelImageFile" class="rtd.domain.DataFile"/>
   <field name="labelImageStream" class="java.io.InputStream"/>
   <detail>
      <band height="432">
         <image scaleImage="RetainShape" hAlign="Center" vAlign="Middle">
            <reportElement mode="Opaque" x="0" y="0" width="288" height="432" forecolor="#000000"
backcolor="#FFFFFF"/>
            <imageExpression class="java.io.InputStream"><![CDATA[$F{labelImageStream}]]></imageExpression>
         </image>
      </band>
   </detail>
</jasperReport>
```

An example JRXML file which uses `IndexedShipmentOrPackageLabelDataProvider` is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports
   http://jasperreports.sourceforge.net/xsd/jasperreport.xsd" name="4 by 6 label"
   pageWidth="283" pageHeight="425" columnWidth="263" leftMargin="10" rightMargin="10" topMargin="10"
bottomMargin="10">
   <parameter name="formattedDeliveryAddress" class="java.lang.String"/>
   <parameter name="contactName" class="java.lang.String"/>
   <parameter name="shipmentReference" class="java.lang.String"/>
   <parameter name="now" class="java.lang.String"/>
   <field name="despatchReference" class="java.lang.String"/>
   <field name="packageInfo" class="java.lang.String"/>
   <detail>
      <band height="400">
         <textField isStretchWithOverflow="true">
            <reportElement uuid="10aaa896-d7c9-4728-b2b9-adfeab764fd3" mode="Transparent" x="18" y="32"
width="223" height="41" forecolor="#000000"/>
            <textElement textAlignment="Left" verticalAlignment="Bottom">
               <font size="15" isBold="false"/>
            </textElement>
            <textFieldExpression><![CDATA[$P{contactName}]]></textFieldExpression>
         </textField>
         <textField isStretchWithOverflow="true">
            <reportElement uuid="2b555a0d-f79e-4d3c-aa99-a04d0214de33" mode="Transparent" x="18" y="77"
width="223" height="198" forecolor="#000000"/>
            <textElement textAlignment="Left" verticalAlignment="Top">
               <font size="15" isBold="false"/>
            </textElement>
            <textFieldExpression><![CDATA[$P{formattedDeliveryAddress}]]></textFieldExpression>
         </textField>
         <textField>
            <reportElement uuid="5d3281c1-52c6-434f-83f3-5ccc4a37777f" mode="Transparent" x="23" y="378"
width="218" height="12"/>
            <textElement textAlignment="Right" verticalAlignment="Middle">
               <font size="9" isBold="false"/>
            </textElement>
            <textFieldExpression><![CDATA[$P{shipmentReference}]]></textFieldExpression>
         </textField>
         <componentElement>
            <reportElement uuid="57727cbf-0860-47de-bb41-3960b58a1eb4" x="18" y="285" width="223" height="42"/
>
            <jr:Code128 xmlns:jr="http://jasperreports.sourceforge.net/jasperreports/components"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports/components http://
jasperreports.sourceforge.net/xsd/components.xsd" moduleWidth="2.0" textPosition="none">
               <jr:codeExpression><![CDATA[$F{despatchReference}]]></jr:codeExpression>
            </jr:Code128>
         </componentElement>
         <textField pattern="">
            <reportElement uuid="475a6957-0faf-49f4-901d-d05ab295941c" x="23" y="347" width="218" height="12"/
>
            <textElement verticalAlignment="Bottom" markup="html">
               <font size="7"/>
            </textElement>
            <textFieldExpression><![CDATA[$F{packageInfo}]]></textFieldExpression>
         </textField>
      </band>
   </detail>
</jasperReport>
```

Note that the `packageInfo` and `despatchReference` fields are generated using the field mapping:

```
<fieldmapper>
  <mappings>
    <mapping to="packageInfo">
      return 'Package ' + input.packageIndex + ' of ' + input.packageCount;
    </mapping>
    <mapping to="despatchReference">
      return input.shipmentOrPackage.despatchReference;
    </mapping>
  </mappings>
</fieldmapper>
```

Label Sizing and Printing

One of the most critical things to get right with labels is size.

JRXML work on the basis of a 72 pixel per inch representation. This can be used to determine the number of pixels for each dimension in your report.

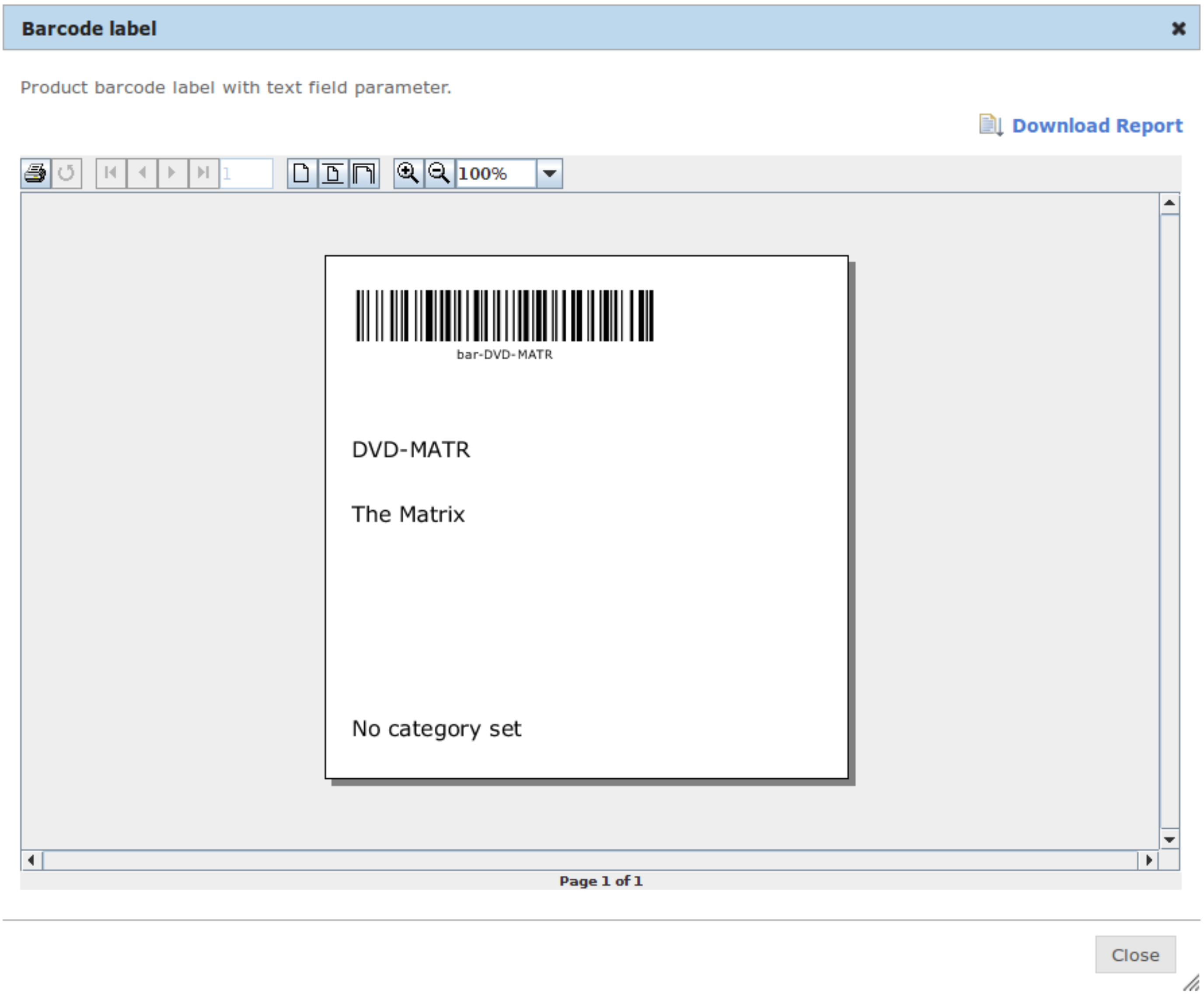
For example, a 4 by 6 label will require a report that is 72 x 4 = 288 pixels wide, and 72 x 6 = 432 pixels in height.

In order to get the label to print correctly, the printer label stock settings will need to be checked to ensure that these match the *actual* dimensions of the label stock.

A typical routine for verifying the use of a label printer for printing labels will be as follows:

- measure the label stock in each dimension, converting the measured values to inches.
- multiply the number of inches to determine the required dimension for the JRXML.
- create a simple *standalone* report for a label with these dimensions. Include in this report a barcode element, so that you can verify that the report and printer are set up correctly for barcode printing.
- verify the printing of this label by running the report, then feeding the report to the printer.
- ensure that the barcode appears to be appears to be crisp and free of jagged edges, and that it scans without any difficulty.

The printing step above can be done from the report output screen, by clicking on the 'Print' icon, as shown below.


Barcode label

Product barcode label with text field parameter.

bar-DVD-MATR

DVD-MATR

The Matrix

No category set

Note that your browser will need to have Java applets enabled for the screen shown above to work.

Alternatively, you can download the printed report as a PDF document. Note, however, that printing a PDF output uses different image rendering and printing software from that typically used in OrderFlow operations, so a test of a PDF version of the document can only be regarded as a partial test.

Barcodes

Jasper Report has built-in support for barcode generation; a design tool such as iReport will easily allow you to embed a barcode such as the following into your report:

```
<jr:Code128 xmlns:jr="http://jasperreports.sourceforge.net/jasperreports/components"
            xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports/components
                                http://jasperreports.sourceforge.net/xsd/components.xsd"
            moduleWidth="2.0"
            textPosition="none">
   <jr:codeExpression><![CDATA[$F{barcode}]]></jr:codeExpression>
</jr:Code128>
```

Depending on the length of the text, and the size of the barcode, the `moduleWidth` parameter may need to be adjusted to maike

Images

As with despatch notes, it is quite common to embed images onto labels. For example, the label itself may in its entirety contain a label image generated using a third party courier system. The Despatch Notes chapter contains details on how to embed images into Jasper Reports documents. The same discussion applies here: it is critical to get the image dimensions correct on the report so that the image does not need to scale, or if it does, scales in a way that does not result in any loss of printing quality.

Dashboards and Views

One of the main uses of reports in OrderFlow is in the creation of dashboards. Dashboards are based on snippets of read-only information, called fragments, which aim to provide, in a very accessible and visual way, access to key items of information on the system. Unlike regular reports, dashboards are highly visible to users.

For example, a dashboard can provide information about courier selections and despatch delivery choices. In that case there would be fragments that display:

- shipments, packed at the day of viewing (essentially - today) - grouped by carrier
- outstanding shipments - grouped by carrier
- outstanding shipments - grouped by courier state
- outstanding shipments - grouped by destination country.

A custom dashboard fragment can be added, for instance, to display all shipments for the past month (grouped by courier) in the form of a graph. Alternatively, a new dashboard screen can be created to contain specific information relating to a particular courier.

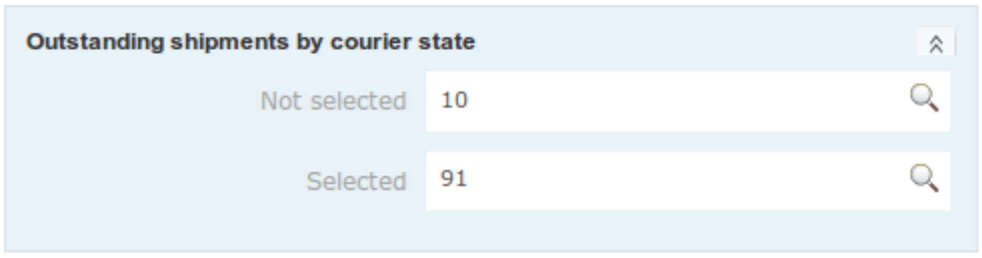
New dashboard screens, similarly to reports, can be added by:

- **creating** a new dashboard screen from scratch. This usually requires the most effort.
- **cloning** an existing dashboard - either a built-in or a custom screen.
- **overriding** a built-in dashboard screen. Essentially, this does not create a new dashboard, but rather modifies an existing one.

Existing dashboard screens can be edited, for example, by adding or removing fragments. Indeed, the system supports the creation of entirely new screens, which can be attached to menus (both new and existing), allowing for a highly customised view of data on the system without any development effort.

New or updated dashboard screens can be attached to existing or new *menu* entries. This greatly increases the capability that OrderFlow customers have to increase visibility of specific areas of the system that are important to their businesses.

An example of one of the dashboard fragments mentioned above is shown next:



Outstanding shipments by courier state

Not selected	10
Selected	91

This fragment displays all outstanding shipments, grouped by courier state. If there are none, it displays an information message, stating that there are no outstanding shipments on the system.

OrderFlow's dashboard management framework goes further by supporting the modification of dashboard fragments. Dashboard fragments are based on reports on the system. As with other reports, administrators have the tools at their disposal for overriding, cloning, exporting/importing or modifying dashboard fragments. More information on creating new reports can be found in the Report Administration section of the previous chapter.

OrderFlow support staff can be commissioned to perform these modifications. Alternatively, customer IT staff can be trained - with with a little help from this document - to do the same without the direct involvement of OrderFlow.

Dashboard Report Fragments

In order to configure dashboard fragments, it is worth familiarising yourself with the Custom Reports on the system. For authoring dashboard fragments and screens, the following additional skills are required:

- familiarity using HTML.
- knowledge of how to use the FreeMarker templating engine, which can be easily acquired from the FreeMarker web site.

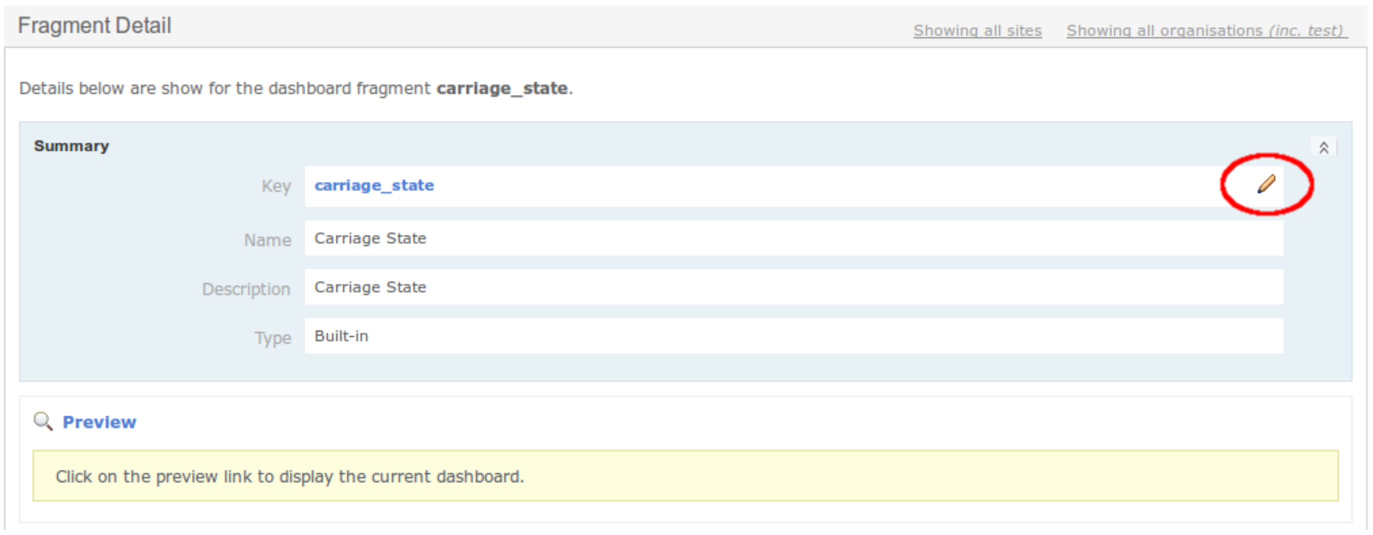
Editing Existing Dashboards

To configure the report design, navigate to the *Fragment Configuration* page from OrderFlow by clicking on the *Reports* tab, then *Dashboards* from the left-hand side menu. Select *Fragments* and filter by using the drop-down menu on the right-hand side of the page in order to find the dashboard you require. Clicking on it will take you to the *Fragment Detail* page, at the bottom of which you can find the code to be used in the dashboard screens.

An example code snippet for the fragment in the first section of this chapter is:

```
${dashboard.report(request,'carriage_state', 'right')}
```

From that page you can also navigate to the *Fragment Configuration* page by clicking on the *Edit* (pencil) icon at the top right-hand corner of the page, shown below.


Fragment Detail

Showing all sites   Showing all organisations *(inc. test)*

Details below are show for the dashboard fragment **carriage_state**.

Summary

Key	carriage_state
Name	Carriage State
Description	Carriage State
Type	Built-in

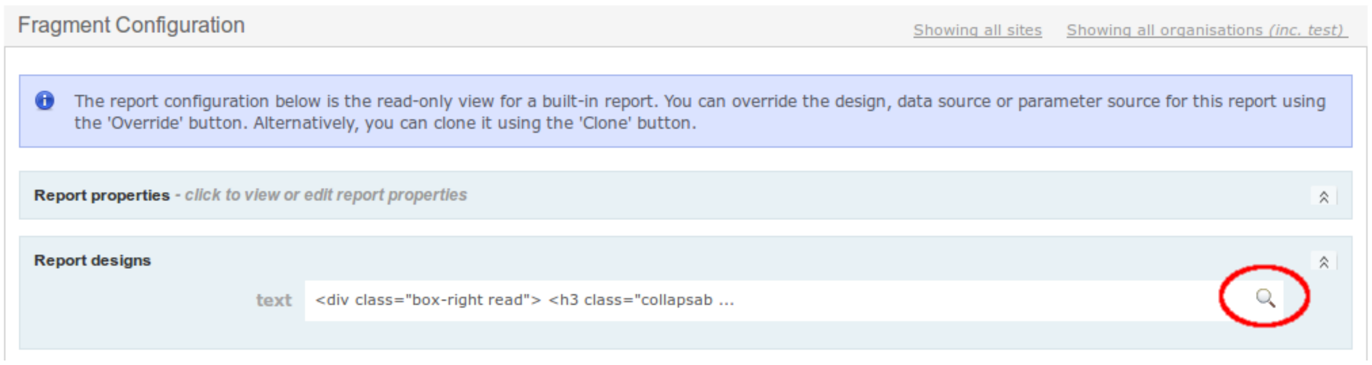
Preview

Click on the preview link to display the current dashboard.

Once in the *Fragment Configuration* page, you can edit all properties of the dashboard fragment. The *Report properties* menu is identical to the one for custom and overriden reports. However, one major difference here is the *Format* of the report, which for dashboards fragments must be **text**.

Fragment Design

To view the design of the report, click on the *Preview* (magnifying glass) icon on the right-hand corner of the *Report designs* menu:


Fragment Configuration

The report configuration below is the read-only view for a built-in report. You can override the design, data source or parameter source for this report using the 'Override' button. Alternatively, you can clone it using the 'Clone' button.

Report properties - *click to view or edit report properties*

Report designs

text `<div class="box-right read"> <h3 class="collapsab ...`

Note that the design for a dashboard fragment is based on a combination of HTML markup and FreeMarker templating elements. An example of design is shown below:

```
<div class="box-right read">

<h3 class="collapsable"><a href="#">Outstanding shipments by courier state <span>- click to view</span></a></
h3>

<#if results?size gt 0>

<dl class = "half">
<#list results as result>
    <#assign courierState=helper.makeLegible(result.courierState!'None')/>
    <dt>${courierState}</dt>
    <dd>${result.shipmentCount}

    <a title="Search for shipments in courier state '${courierState}'"
        href="${contextPath}/despatch/courier/shipment/searchnew.htm?courierState=${result.courierState!''}"
class="floatRight">
    <img src="${contextPath}/icons/inspect.gif">
    </a>

    </dd>
</#list>

<#else>

<div class="info message">
There are no outstanding shipments on the system.
</div>

</#if>

<!--[age]-->

</div>
```

You can preview the fragment by going back to the *Fragment Detail* page and clicking on *Preview*, located just below the *Summary*.

Data Providers

When working with dashboards, the manner in which data is retrieved is determined by it is useful how to use different data provider classes.

The data providers below are the main data providers used in custom SQL-based dashboards.

Name	Usage
`SqlDataProvider`	Display a single list of rows
`MapResultDataProvider`	Display just the named column values for a single row of data
`MultipleSqlDataProvider`	Allow multiple queries to be referenced in a single report

All of the providers above refer to Java class names for classes in the package `rtd.reports.dataprovider`. For example, the fully qualified name for `SqlDataProvider` is `rtd.reports.dataprovider.SqlDataProvider`.

Note that it is also possible for data providers to use functionality built-in to the system, but these data providers are typically not used in custom dashboards created by customer technical users or OrderFlow support staff.

SQLDataProvider

This data provider is used with SQL queries which return a list of rows from the database.

For example, the following SQL can be used to return the 10 orders most recently added to the system:

```
select externalReference, invoiceContactName from order_item order by id desc limit 10;
```

The FreeMarker markup which would be used to display the results of this report on a dashboard is shown below:

```
<#list results as result>
<p>${result.externalReference} ordered by ${result.invoiceContactName!'Unknown'}</p>
</#list>
```

A note on null values

Note the syntax `${result.invoiceContactName!'Unknown'}` allows for the value 'Unknown' to be used if the underlying value is null. FreeMarker is not very forgiving with null values; it will not allow the syntax

`${result.invoiceContactName}` to be used if the underlying data is null.

If no substitue value is required, then the syntax `${result.invoiceContactName!''}` can be used.

Alternatively, the underlying SQL can ensure that a non-null value is returned using a function call such as `ifnull(invoiceContactName,'Unknown')`.

MapResultDataProvider

This data provider is used with SQL which simply return a single row of data.

For example, the following SQL will give a count of shipments created in the last day by line count:

```
select
count(id) allShipments,
sum(if(shipment.multiLine,0,1)) singles,
sum(if(shipment.multiLine,1,0)) multis
from shipment
where created >= date(now());
```

Note that the query above will return a single row of data, as in the table below:

allShipments	singles	multis
115	84	31

When `MapResultDataProvider` is used, the results can be referenced in the following way:

```
<p>All Shipments: ${result.allShipments}</p>
<p>Single Line Shipments: ${result.singles}</p>
<p>Multiline Shipments: ${result.multis}</p>
```

MultipleSqlDataProvider

In some more complicated scenarios, it is useful to have the results of multiple SQL queries providing the data for a single dashboard fragment. For these situations, the `MultipleSqlDataProvider` data provider can be used.

An example data source value for this data provider might be the following:

```
all_open_orders:
select count(id) count from order_item where completed is null;

open_orders_by_channel:
select
channel.name channel,
count(orderItem.id) count
from order_item orderItem
join channel channel on orderItem.channelId = channel.id
where orderItem.completed is null
group by channel.id
order by channel.name;
```

The data returned from these queries can be displayed using the following:

```
<#assign allOpenOrders=results.all_open_orders[0]/>
<#assign openOrdersByChannel=results.open_orders_by_channel/>

<p>All Open Orders: ${allOpenOrders.count}</p>

<#list openOrdersByChannel as result>
<p>${result.channel} orders: ${result.count}</p>
</#list>
```

Notice that the results of all of the queries are grouped in a map called `results`, which allows the results of an individual report to be referenced using the syntax `results.identifier`.

Notice also how the identifier is defined using the `identifier:` syntax in the data source in the line prior to the associated SQL for that report identifier.

Macros

Unlike the example, described above, it is sometimes the case that macros are used in dashboard fragments. Macros (short for *macroinstruction*) are essentially rules that specify how a certain input should be mapped to a replacement output. A set of macros are available to make the task of using certain data in dashboard fragments less repetitive and easier to access.

Some of the more frequently used macros include:

Name	Parameters
`dateTime`	Displays date and time in locale-aware format.
`dateOnly`	Displays date only in locale-aware format.
`icon`	Used to display a named icon on the system.

A Note on Product Display

For different environments it is useful to display products in slightly different ways. For example, if the product external reference is a human readable SKU known to operators, then it is appropriate to use the product `externalReference` to represent products. However, if the product external reference is a system generated value without any obvious meaning, then the `description` might be more appropriate.

Similarly, when quantities need to be displayed, the system can be helpful in ensuring that the correct quantity type (e.g. units, weight, volume) is displayed, with the correct unit of measure.

The dashboards accommodate this using a set of macros, which are helpful in displaying products.

Name	Parameters
`productTitle`	Displays product title, applying configuration to determine whether this should be the SKU, the product description, or a combination of both.
`displayQuantityAndUnits`	Provides a formatted display of quantity together with they quantity units.
`initialiseProductMap`	Called just before iterating through a list of products in which the `productTitle` and `displayQuantityAndUnits` macros are used.

An example usage of these macros is shown below:

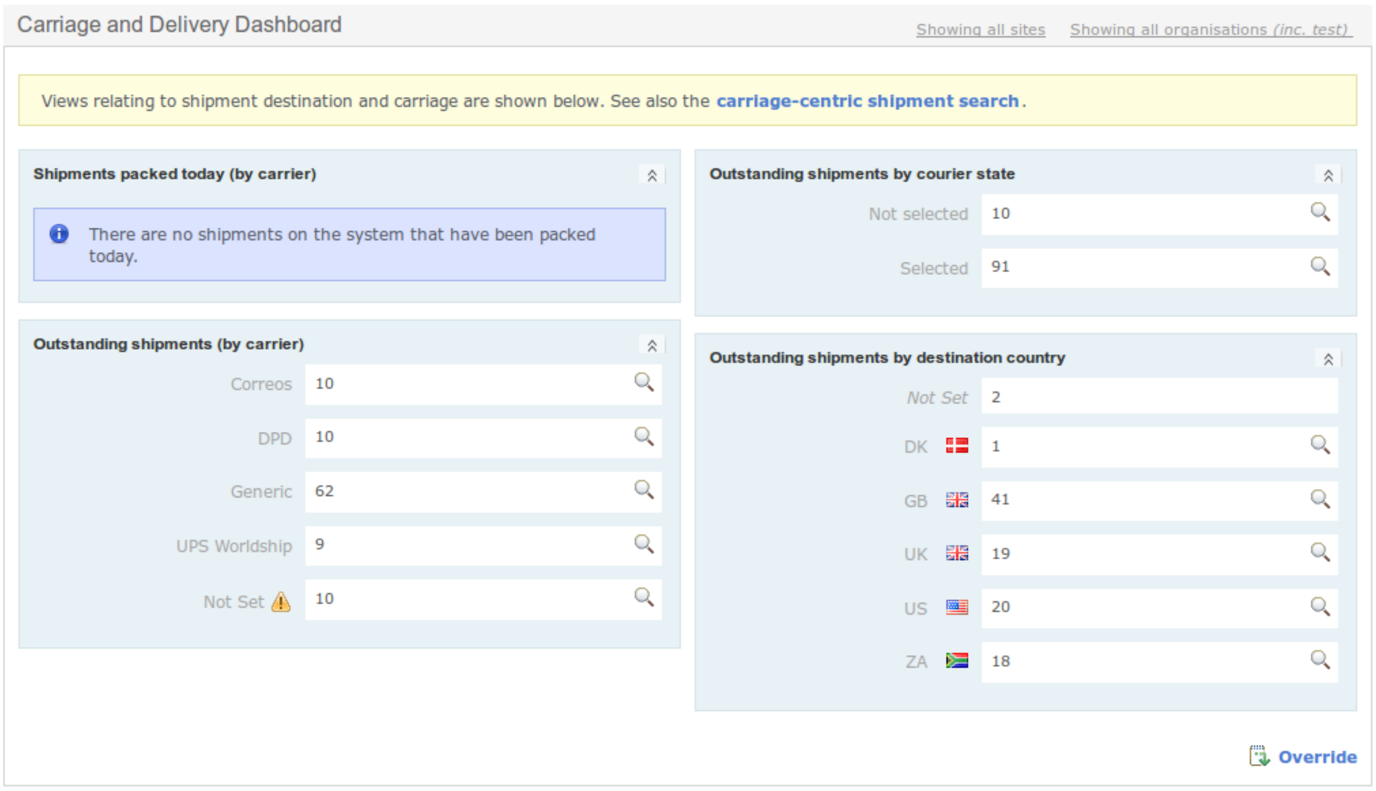
```
<table>
<thead>
    <tr>
       <th width="50%">Reference</th>
       <th width="25%" class="integer">Quantity</th>
    </tr>
</thead>

<tbody>
<@initialiseProductMap results=results productIdIdentifier="id"/>
<#list results as result>
    <tr>
    <td><@productTitle reference=result.reference description=result.description parameters=parameters/></td>
    <td class="integer">
    <@displayQuantityAndUnits productId=result.id quantity=result.count/>
    </td>
    </tr>
</#list>
</tbody>
</table>
```

Note that the `initialiseProductMap` macro is used to efficiently pull back to necessary configuration and data to apply the configured strategy for displaying product titles and quantities.

Note that for **custom dashboards** that needs to work in only one environment, a simpler approach will suffice, where the product title can be displayed using either `${product.externalReference}` or `${product.description}`, as the situation requires.

Dashboard Screens

Dashboard screens are essentially a group of dashboard fragments, displayed in a way that is most suitable for the user(s). Below is an example of a built-in dashboard screen with four fragments:



Carriage and Delivery Dashboard

Showing all sites   Showing all organisations *(inc. test)*

Views relating to shipment destination and carriage are shown below. See also the **carriage-centric shipment search**.

Shipments packed today (by carrier)

There are no shipments on the system that have been packed today.

Outstanding shipments (by carrier)

Correos	10
DPD	10
Generic	62
UPS Worldship	9
Not Set ⚠	10

Outstanding shipments by courier state

Not selected	10
Selected	91

Outstanding shipments by destination country

Not Set	2
DK	1
GB	41
UK	19
US	20
ZA	18

Any dashboard screen can be edited by clicking on the *Override* link at the bottom right-hand corner of it. This leads to the *Screen details* page, where you can see all properties of the dashboard. Clicking on *Edit dashboard screen* takes you to the *Edit Screen* page, where these properties can be edited.

The *Template* defines how the dashboard screen is structured. Below is an example of such a template:

```
<#assign pageHeader = "Carriage and Delivery Dashboard"/>
<#include "viewheader.txt"/>

<div class="textbox highlight">Views relating to shipment destination and carriage are shown below. See also
the
<a href="${contextPath}/despatch/courier/shipment/searchnew.htm" title="Carriage-centric shipment search">
    carriage-centric shipment search
</a>.</div>

<#if dashboard.userView(request,'order:status')>
<div class="container-left">
${dashboard.report(request,'carriage_packed_today', 'left')}
${dashboard.report(request,'carriage_outstanding', 'left')}
</div>
<div class="container-right">
${dashboard.report(request,'carriage_state', 'right')}
${dashboard.report(request,'carriage_destination_country', 'right')}
</div>
<@h.clearFloats/>
</#if>

<#include "viewfooter.txt"/>
```

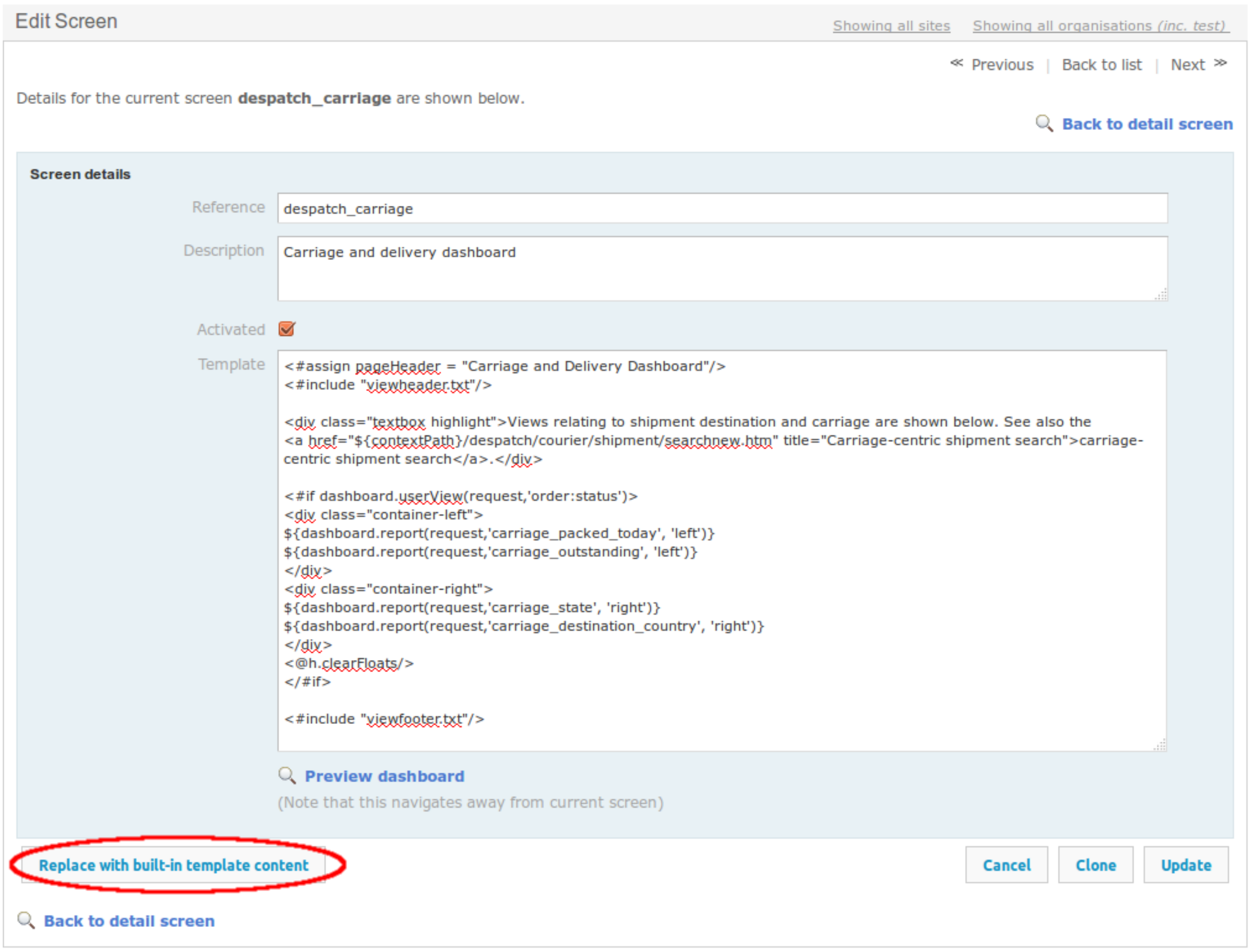
The parameters of the **dashboard views** are:

- **request**, which is a constant parameter, required to encapsulate the data
- **view reference**, where dashboard views essentially restrict access to certain dashboard fragments, so they are visible only to specific roles

The parameters of the **dashboard fragments** are as follows:

- **request**, which is a constant parameter, required to encapsulate the data
- **name of fragment**, where the name is defined upon the creation of the dashboard fragment.
- **position**, which can be: `left`, `right` or `full`. As the names suggest, `left` and `right` display the fragment in the respective half of the screen and `full` takes up the whole width - for fragments, which require more space to be displayed.
- **parameters** (optional), which are usually parsed in to apply filters. They follow the format `name=value;`, for example: `ignoreEmpty=true;maxAge=10`
- an optional **boolean** to control the exposure of helper methods available to the fragment (advanced use).

Should you need to go back to the default template of a built-in dashboard, this can be done from the *Edit Screen* page:


Edit Screen
Showing all sites
Showing all organisations (inc. test)
« Previous | Back to list | Next »

Details for the current screen **despatch_carriage** are shown below.

Screen details

Reference	despatch_carriage
Description	Carriage and delivery dashboard
Activated	✓

Template

```
<#assign pageHeader = "Carriage and Delivery Dashboard"/>
<#include "viewheader.txt"/>

<div class="textbox highlight">Views relating to shipment destination and carriage are shown below. See also the
<a href="${contextPath}/despatch/courier/shipment/searchnew.htm" title="Carriage-centric shipment search">carriage-
centric shipment search</a>.</div>

<#if dashboard.userView(request,'order:status')>
<div class="container-left">
${dashboard.report(request,'carriage_packed_today', 'left')}
${dashboard.report(request,'carriage_outstanding', 'left')}
</div>
<div class="container-right">
${dashboard.report(request,'carriage_state', 'right')}
${dashboard.report(request,'carriage_destination_country', 'right')}
</div>
<@h.clearFloats/>
</#if>

<#include "viewfooter.txt"/>
```

Preview dashboard
(Note that this navigates away from current screen)
Replace with built-in template content
Cancel
Clone
Update
Back to detail screen

However, this is only possible for built-in dashboard screens, so make sure you have a back-up of your custom dashboard templates before editing them - in case you want to go back to the previous version.

Dashboard Views

Dashboard views allow elements from within a dashboard screen to be exposed to users with different sets or roles, i.e. set restrictions on the visibility of views or groupings of dashboard elements.

Views can easily be edited by simply clicking on one when in the *Views* page - to navigate there, click on the *Reports* tab on the top, then *Dashboards* from the left-hand side menu and its submenu - *Views*. From there you can assign which roles have access to that particular view.

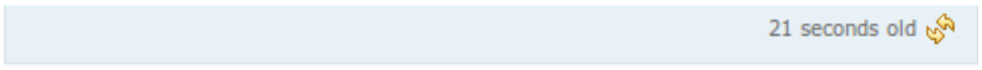
Similarly, when creating a new view (once in the *Views* page, click on the *New View* button at the bottom of the page), other than the view *reference* and *description*, you can set the visibility of the view to different roles simply by ticking (or unticking) the respective box.

Dashboard Caching

For performance reasons, most dashboard fragments are *cached*. Caching is the mechanism of temporarily storing computed data, so that future requests for that data can be served faster. This frees the system from having to recompute the data with every request, which is comparatively slower and more demanding in terms of system resources.

For most dashboard fragments, users are adequately served by a display of information which may be a few seconds old, rather than absolutely up to date in real time. However, for certain items of information it is essential to display the latest data data, so cacheing is not required.

When refreshing a page, a cached fragment will display how old the information is in its bottom right-hand corner:



By default dashboard fragments are cached for up to 90 seconds, at which point the data is computed again. However, if required, caching time can be changed for particular fragments by editing their to the maxAge desired amount of seconds. Also, the data can be manually recomputed by clicking on the refresh icon.

The following *age* escape sequence adds the refresh link to any dashboard fragment:

```
<div class="${boxClass!'box'} bordered">

... //fragments

<!--[age]-->

</div>
```

Periodic Reports

OrderFlow supports a powerful mechanism for reports to be run automatically on a scheduled basis. Reports of this kind are known on the system as **periodic reports**. They are highly customisable and can be run at different intervals - either hourly, daily, weekly or monthly.

Some examples of the kinds applications that periodic reports support include:

- Automatically running end of day reports such as stock level, deliveries and orders reports. For example, 3PL can use these reports to daily stock and order data to customers, removing the need to generate these reports manually.
- Management reports on warehouse operational performance.
- Automated reports to customer services on returns or stock adjustments.
- Running sanity checks in the background to identify anomalies or errors and sending a follow up notifications to the support team.

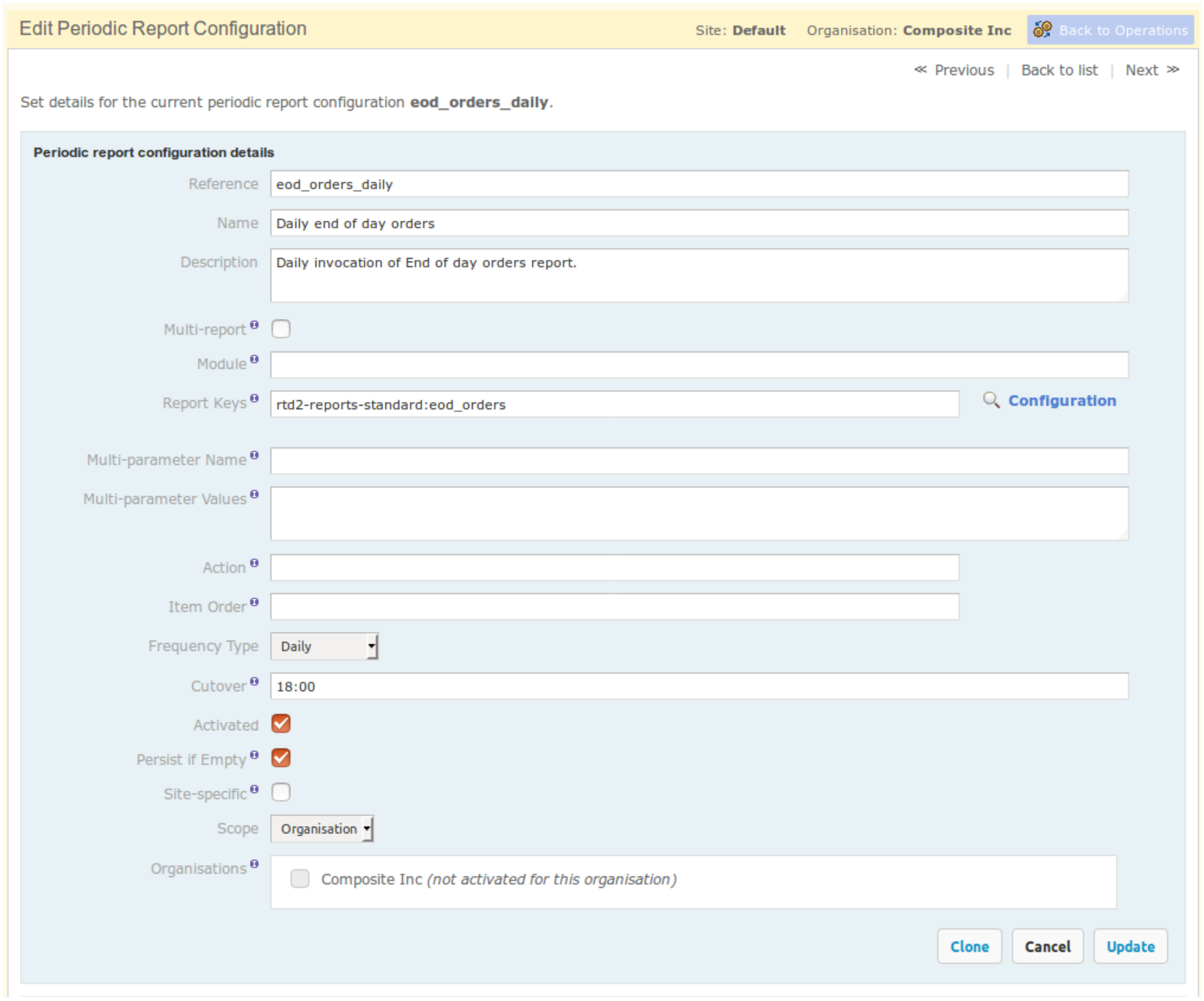
A key feature of periodic reports is their ability to be *exported*. Any periodic report can be set up with an arbitrary number of exports. These exports can be done using a number of mechanisms, including to a folder on the file system, an email to a configurable set of recipients, or a web service call to a remote system.

A periodic report definition can be configured to trigger the automatic execution of multiple reports. These reports can be both built-in reports as well as custom or bespoke reports, described in detail in the Custom Reports chapter.

Periodic Report Configuration

The starting point for setting up a periodic report is identifying an existing report, or creating a new custom report, which provides the data required.

You can view existing periodic reports on the system by navigating to *Reports* -> *Configurations* -> *Periodic*. As with dashboards and other reports, periodic reports can be created from scratch, or cloned from existing reports. You can create a new one from scratch by clicking on *New Periodic Report* at the bottom of the page. Below is an example of a periodic report configuration:


Edit Periodic Report Configuration
Site: **Default** Organisation: **Composite Inc**
Back to Operations
« Previous | Back to list | Next »

Set details for the current periodic report configuration **eod_orders_daily**.

Periodic report configuration details

Reference	eod_orders_daily
Name	Daily end of day orders
Description	Daily invocation of End of day orders report.
Multi-report	☐
Module	
Report Keys	rtd2-reports-standard:eod_orders
Multi-parameter Name	
Multi-parameter Values	
Action	
Item Order	
Frequency Type	Daily
Cutover	18:00
Activated	✓
Persist if Empty	✓
Site-specific	☐
Scope	Organisation
Organisations	☐ Composite Inc *(not activated for this organisation)*

In the next section we describe some of these fields in detail.

Reference

The name used in other parts of the application to identify the report. Must be a unique value.

Name

The human-readable name of the report. This differs from *Reference* in that the unique key is used by the application, whereas the report name is designed to be helpful to users.

Description

A description of what the report does, how it is used, its frequency and/or any other pertinent notes corcerning the report.

Multi-report

If this box is ticked, then a single periodic report definition can be backed up by multiple report configurations.

Report Keys

The *unique key(s)* of the report(s) to be executed on a schedule. These refer to the built-in or custom reports present on the system.

If *Multi-report* is unticked, the current definition backs only a single report.

Multi-Parameter Name and Multi-Parameter Values

If set, this allows the same report to be run multiple times as part of a single periodic report execution, with the individual values set using the Multi-Parameter Values field passed through with each successive invocation.

Action

This allows the report data to be passed back into the system for further processing. The action field value identifies the processor that is used.

You would not set a value for this field without prior knowledge of what the appropriate value would be.

Item Order

Controls the ordering of the reports run. Reports with a value set for this field are run first, in ascending order.

Frequency Type

Defines how often the report is to be executed - hourly, daily, weekly or monthly. Depending on the interval, the *cutover* would contain differently formatted values.

From OrderFlow 3.8.0.5, quarter hourly periodic reports are also possible.

A Note on Quarterly Hour Reports

If using quarter hourly reports, be sure that the report execution schedule is sufficiently frequent, for example, every five minutes.

We'd also recommend that you set the report Item Order value to ensure that quarter hour reports are run first.

Cutover

The point in time when the report is to be executed. The format depends on the *frequency type*:

- quarter hourly: `mm` , e.g. *12*
- hourly: `mm` , e.g. *45*
- daily: `HH:mm` , e.g. *17:30*
- weekly: `EEE HH:mm` , e.g. *Thu 08:55*
- monthly: `d HH:mm` , e.g. *7 12:20*

where the following format notation is used:

- `HH` - hours (0-23)
- `mm` - minutes (00-59)
- `EEE` - day of the week (e.g. Mon)
- `d` - date (depending on the month, 1-28/29/30/31).

If quarter hourly reporting is used, the cutover will need to be set to a value between 0 and 14.

Activated

Determines whether the report is active, i.e. if it is to be executed on a schedule. If the box is unticked, the report can be tested manually to see whether it produces the desired output.

Persist if Empty

Determines whether empty reports should be persisted on the system. For text, CSV and Excel based reports, empty reports typically occur when there are no data is retrieved when generating the report.

Site-specific

Denotes whether the periodic report definition is run once across all site or warehouse, or run potentially multiple times, for each selected site on the system. If ticked, then the definition will be run for each of the user-selected sites. Selecting no sites results in the definition not being invoked.

Scope

Specifies whether the report is run across certain organisation(s), channel(s) or is run globally. If *Organisation* is selected, periodic reports will be generated separately on a per organisation basis for selected organisations. Similarly, if *Channel* scope is used, periodic reports will be generated separately per channel, for selected channels.

A Note on Underlying Reports

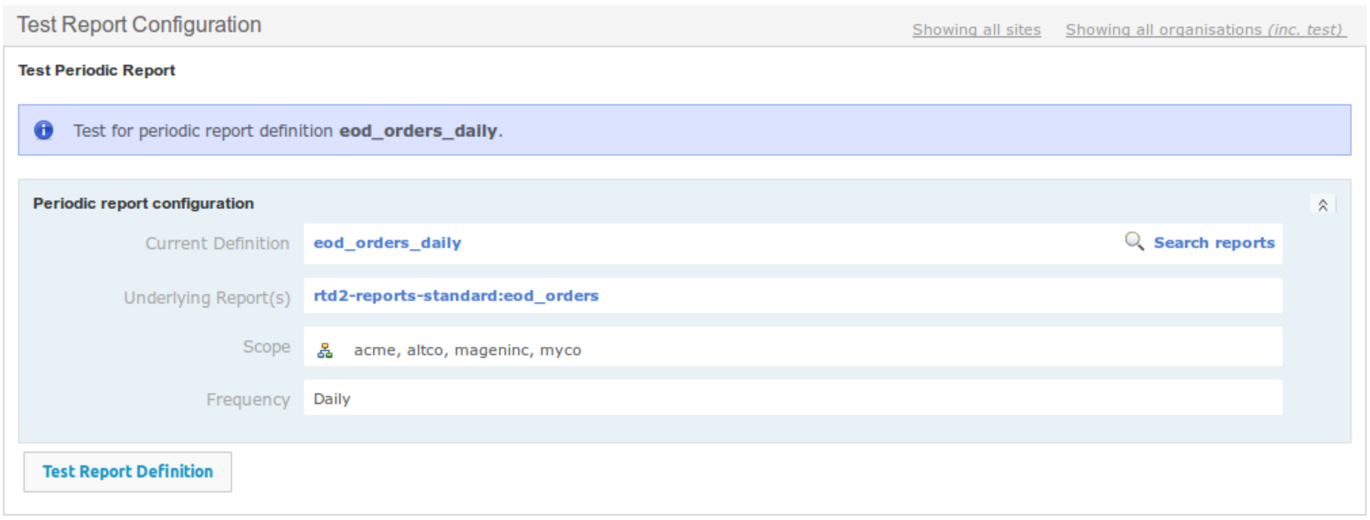
Note that the report that is used as an underlying report for a periodic report (as per the Report Keys field described in the previous section) needs to observe a couple of important requirements:

- if the data needs to be scoped, it should follow the reserved parameter conventions for the naming of the channel, organisation and site scoped parameters.
- it should have suitably defined date parameters to ensure that date-sensitive data is correctly filtered.

Testing a Periodic Report

Before activating a new periodic report, you can test it to check whether it produces the expected output. Note that in order to test a periodic report configuration, it must be *inactive*, as described in the previous section.

Once the *Activated* box has been unticked, a *Test configuration* link will appear at the bottom of the page. Clicking on it leads to the *Test Report Configuration* page, shown below.


Test Report Configuration
Showing all sites
Showing all organisations *(inc. test)*
Test Periodic Report

Test for periodic report definition **eod_orders_daily**.

Periodic report configuration

Current Definition	eod_orders_daily
Underlying Report(s)	rtd2-reports-standard:eod_orders
Scope	acme, altco, mageninc, myco
Frequency	Daily

Clicking on the *Test Report Definition* button at the bottom of the page will generate the report(s) for this definition. A small information window will then display at the top right corner of the screen, describing the results of the report generation action. Also, a *Delete Existing Reports* button will appear on the bottom of the page, which can be useful once the reports have been checked and are no longer wanted on the system.

Note that it is not possible to delete reports in this way once the periodic report has been activated.

If any reports were generated, you can view them by navigating to the periodic report search from *Reports* -> *Periodic* -> *Search*. From the *Periodic Report* detail screen, you can see all details of the generated report and download it and/or view the file content. From this screen you can also trigger the *export* of the periodic report Periodic report exports are described in the next section.

Activating a Periodic Report

In addition to checking the *Activated* checkbox whilst editing a periodic report definition, to ensure that a periodic report is run, a schedule handler definition that references the *generateReports* handler (and, for exports, one that references the *exportReports* handler) must be active. There is also a flag against every organisation and channel that controls whether schedules and periodic reports are active, so this also needs to be enabled for the required organisation(s) and channel(s).

Export Configuration

A key feature of periodic reports is their ability to be exported. Multiple export configurations can be set for a single periodic report definition. Export configurations can be set up by taking advantage of one of the *remote message types* enabled for periodic reports. Setup of remote messages is not covered in in this document.

The two most common message types for periodic reports are file system and email. However, HTTP- and FTP-based message types can also be used.

A *file system* export results in the output of the report into a directory on the file system, often exposed to remote users via FTP or RSync. The target directory and file names are configurable via a Groovy script. File system report exports allow, for example, end of day information on stock levels, orders, deliveries and other data to be made accessible to end users via a file syste based interface.

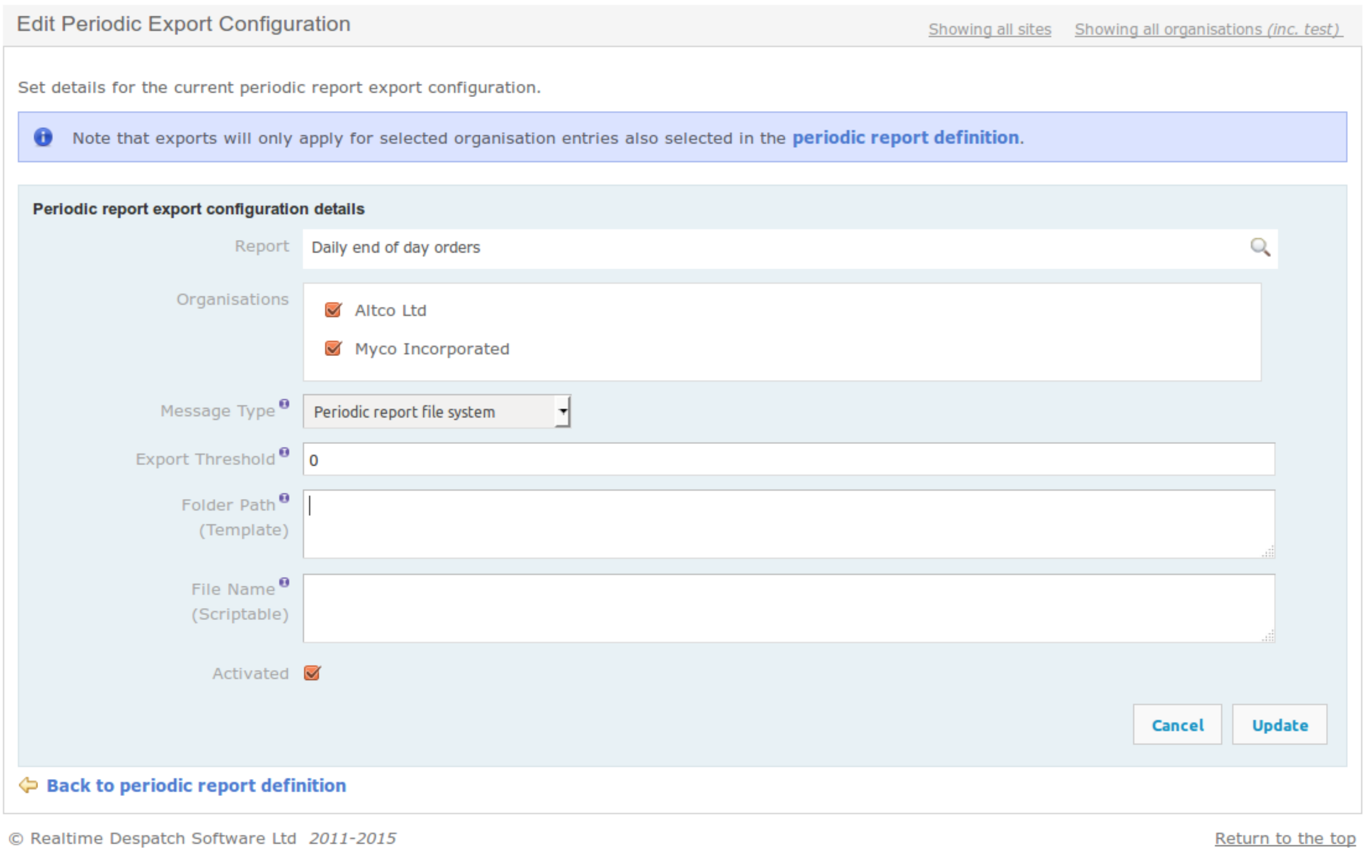
The *email* message type, as the name suggests, supports export of the reports in the given format through an email to a specific set of recipients.

Export Type

The mechanism for setting up periodic report export configurations is described next.

To add a new export configuration, from the *Edit Periodic Report Configuration* page, click on the *New Export Configuration* button at the bottom of the screen. Unlike reports and dashboards, export configurations cannot be cloned or exported. However, existng export configurations can be updated, and new ones can be created from scratch.

Below is an example of such a configuration:


Edit Periodic Export Configuration

Showing all sites   Showing all organisations *(inc. test)*

Set details for the current periodic report export configuration.

Note that exports will only apply for selected organisation entries also selected in the **periodic report definition**.

Periodic report export configuration details

Report	Daily end of day orders
Organisations	✓ Altco Ltd ✓ Myco Incorporated
Message Type	Periodic report file system
Export Threshold	0
Folder Path (Template)	
File Name (Scriptable)	
Activated	✓

© Realtime Despatch Software Ltd *2011-2015*   Return to the top

The details of the periodic export configuration fields are described next.

Report

An read only text field, containing the *Name* of the periodic report to be exported.

Organisations/Channels

Depending on the periodic report configuration, this field can either be a list of organisations, a list of channels, depending on the *Scope* of the report, described in the previous section.

If the periodic report's scope is set to *Global*, this field will be not be present, since the report runs across all channels and organisations. If the scope is either *Organisation*, or *Channel*, this field will contain only the entries which are selected in the periodic report configuration.

Message Type

The remote message type to be used for the report.

Export Threshold

Controls whether a generated report is actually exported.

Periodic reports whose result size is less than this threshold value will not be exported. A blank value or zero indicates that the export will always occur. This field is particularly useful for reports that notify for errors; in this case, a typical configuration is to only export the report if it has recorded at least one result.

Folder Path (Template)

A template can be used to define a folder path relative to the base directory for outputting periodic reports. If left blank, the default folder naming convention will be used.

This field provides a relatively simple way to control the output location of file based periodic reports data. An example is shown below which constructs a folder path based on the organisation, channel and formatted timestamp of the periodic report export.

```
/${organisation.externalReference}/${channel.externalReference}/${timestamp?string('yyyy-MM-dd')}
```

The second example below constructs a folder path using the site reference and the periodic report ID.

```
/${site.externalReference}/${periodicReport.id}
```

The syntax for the templates is the FreeMarker template language, which is used for example in CSV-based custom reports.

This field is currently only supported for file system-based periodic report exports.

File Name (Scriptable)

Optional field that identifies the target file name and path for file-based export reports. Both the path and the file name can be either literal, or scripted. This field is generally not required for HTTP and email based export message types.

If left blank, the default file name convention will be used, which includes the periodic report reference and the report invocation timestamp. Otherwise, the field can be populated with either a literal (fixed) value, or a Groovy script. In both cases, the value should return a file name, optionally prefixed with a relative path. The separator for directories is the character `/`.

Activated

Indicates whether the current export configuration is enabled.

Exporting Periodic Reports as emails

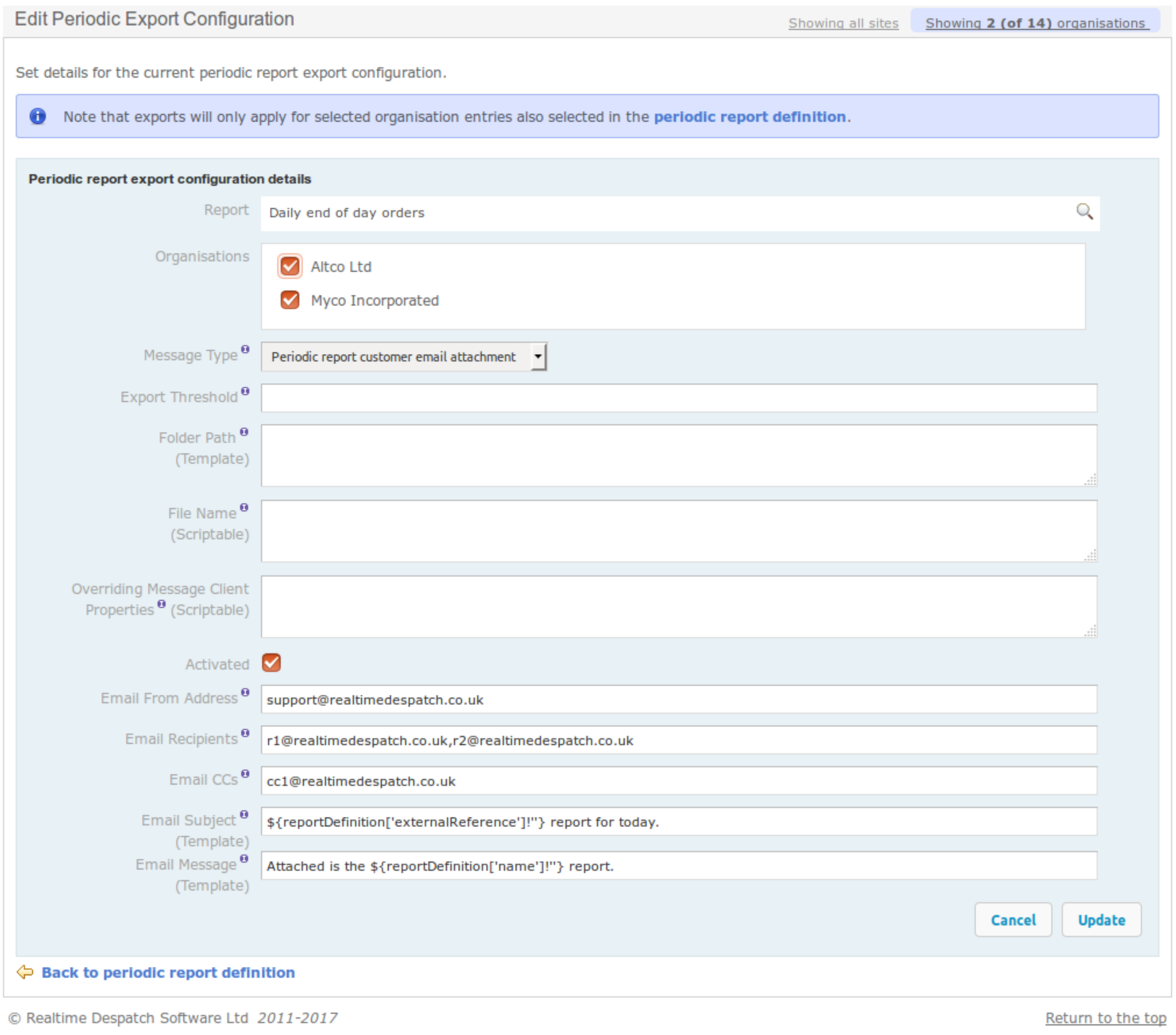
A periodic report can be exported as email text, or as an email attachment. The email settings for a periodic report can be configured in two ways.

System-wide settings

The OrderFlow *periodic.report.customer.email* properties are the default settings that will be used for all periodic reports to be exported as emails or email attachments, unless these are overridden by report-specific settings. Setup of these properties is not covered in this document.

Report-specific settings

These are specified on the export definition configuration. Below is an example of an export definition configuration for a periodic report to be exported as an email attachment:


Edit Periodic Export Configuration
Showing all sites
Showing 2 (of 14) organisations

Set details for the current periodic report export configuration.

Note that exports will only apply for selected organisation entries also selected in the **periodic report definition**.

Periodic report export configuration details

Report	Daily end of day orders
Organisations	✓ Altco Ltd ✓ Myco Incorporated
Message Type	Periodic report customer email attachment
Export Threshold	
Folder Path (Template)	
File Name (Scriptable)	
Overriding Message Client Properties (Scriptable)	
Activated	✓
Email From Address	support@realtimedespatch.co.uk
Email Recipients	r1@realtimedespatch.co.uk,r2@realtimedespatch.co.uk
Email CCs	cc1@realtimedespatch.co.uk
Email Subject (Template)	${reportDefinition['externalReference']!''} report for today.
Email Message (Template)	Attached is the ${reportDefinition['name']!''} report.

Cancel
Update
Back to periodic report definition
© Realtime Despatch Software Ltd 2011-2017
Return to the top

The details of the periodic export configuration email setting fields are described next.

Email From Address

The address to be displayed in the "From" field of the email.

Email Recipients

The addresses to be sent the email.

Email CCs

The addresses to be CC'd the email.

Email Subject (Template)

A template can be used to define the subject of the email. In the example above, the template evaluates to:

eod_orders_daily report for today.

Email Message (Template)

When a report is exported as an email, the email message text will be the contents of the report. However if the report is exported as an email **attachment**, a template can be used to define the message of the email. In the example above, the template evaluates to:

Attached is the Daily end of day orders report.