StreamServe™ Persuasion™

# StreamServe Persuasion SP5 Document Sorting and Bundling

## User Guide

**Rev B**

StreamServe Persuasion SP5 Document Sorting and Bundling User Guide
Rev B
© OPEN TEXT CORPORATION
ALL RIGHTS RESERVED
United States and other international patents pending

Use of this software program is protected by copyright law, patent law, and international treaties. No part of this software product, associated documentation (including online help tools) may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Open Text Corporation. Information in this documentation is subject to change without notice. Open Text Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this software program. All brands, product names and trademarks of other companies mentioned in this software program are used for identification purposes only and are acknowledged as property of the respective company. Companies, names and data used in examples in this software program are fictitious unless otherwise noted.

Open Text Corporation offers no guarantees and assumes no responsibility or liability of any type with respect to third party products and services, including any liability resulting from incompatibility between the third party products and services and the products and services offered by Open Text Corporation and its direct/indirect subsidiaries. By using Open Text Corporation software products and the third party products or services mentioned in this software product, you agree that you will not hold Open Text Corporation and its direct/indirect subsidiaries responsible or liable with respect to use of such third party products or services.

The trademarks, logos, brands, and service marks found in this software program are the property of Open Text Corporation or other third parties. You are not permitted to use such marks without the prior written consent of Open Text Corporation or the third party that owns the marks.

Use of any Open Text Corporation products or services with any third party products or services not mentioned in this documentation is entirely at your own risk.

# Contents

# About document sorting and bundling

Document sorting and bundling includes the features shown in the table below.

| Feature | Description |
|---|---|
| **Document sorting** | Sort documents. See *Sorting documents* on page 7. |
| **Enveloping** | Bundle documents and prepare for enveloping. See *Bundling documents* on page 11. |
| **OMR marks** | Define OMR marks to be printed on the sheets in the documents. See *OMR* on page 23. |
| **Labels** | Define labels to be printed on the sheets in the documents. See *Labels* on page 31. |
| **Script functions** | Use document sorting and bundling script functions. See *Sorting and bundling scripts* on page 35. |

### Page formatted output

Document sorting and bundling can only be applied to page formatted output. This means you must use the appropriate output connector and driver for the output, for example a File output connector and AFP driver.

# Documents retrieved from repository or defined by document trigger

Document sorting and bundling can be applied to documents retrieved from a document repository as well as to page formatted output documents from standard runtime jobs.

### In this section



## Documents retrieved from document repository

Output documents can be stored in a document repository, and then be retrieved using a post-processor and a Post Processor Query (PPQ). All document sorting and bundling features are enabled by default, and can be applied to the documents retrieved from the document repository (via the post-processor).

Metadata is associated with the documents stored in the document repository. When sorting and enveloping documents retrieved from a document repository, metadata is used as keys.

## Documents not retrieved from document repository

Document sorting and bundling features can be applied to page formatted output documents from standard runtime jobs. In this case you must create a Document Trigger, i.e. define what to include in each document, and enable document and sorting on the output connector.

When sorting and enveloping documents, you can use variables defined in the Project, e.g. the document trigger variable, as keys.

### To create a document trigger

1. Open the Runtime Output Connector Settings dialog box (generic layer).
2. Click the **Document Trigger** icon.
3. Enter the **Document trigger variable**.

### To enable document sorting and bundling

1. Open the Runtime Output Connector Settings dialog box (generic layer).
2. Click the **Global Settings** icon.
3. On the **Post-processing** tab, select **Enable Document Broker**.

# Sorting documents

You can sort page formatted documents retrieved from a document repository as well as page formatted documents from standard runtime jobs.

The documents must be sorted to make sure they arrive in the right order before enveloping is applied. You can, for example, sort the documents using the zip code as the first sort criterion and the customer number as the second sort criterion. This will first group the documents per zip code, and then, within each zip code group, group the documents per customer number.

### Sort keys

Sorting is applied using one or more sort keys, for example zip code and customer number as described in the example above. When you define a sort key, you define which key to use, the type of key (numeric or string), and the sort order (ascending or descending).

How to define the sort keys depends on the source of the documents, i.e. the type of document repository from which the documents are retrieved. See the table below.

| Document repository | Description |
|---|---|
| FastObjects database | If the documents are retrieved from a FastObjects database, you must use metadata as sort keys. You define the sort keys as variables (`$<metadata>`).<br>See *Sort keys for documents in FastObjects database* on page 8. |
| Relational database | If the documents are retrieved from a relational database (Document Broker Plus), you must use metadata as sort keys. You define the sort keys as metadata names.<br>See *Sort keys for documents in relational database (Document Broker Plus)* on page 9. |
| No document repository | If the documents are defined by a document trigger, you can use any variable defined in the Project, for example the document trigger variable, as sort key.<br>See *Sort keys when no document repository is used* on page 10. |

# Defining sort keys

### In this section



## Sort keys for documents in FastObjects database

If the documents are retrieved from a FastObjects database, you must use metadata associated with the stored documents as sort keys.

### To define a sort key

1. Open the Runtime Output Connector Settings dialog box (generic layer).
2. Click the **Job End** icon and select the **Document Sort** tab.
3. Click  . The Add Sort Key dialog box opens.
4. Configure the sort key:
   - **Key** – the metadata (`$<metadata>`) to use as sort key.
   - **Type** – Numeric or String. Must be the same type as defined for the metadata in the document repository.
   - **Sort order** – Ascending or Descending.
5. Click **OK**.

### Defining sort keys using scripts

The *DeclareMetadata* script function specifies a metadata name to be used for sorting or bundling of documents. The metadata name can then be associated with a variable by for example using the following functions:

- *SetCurrMetadata* – sets the metadata value on the current document.
- *SetSegMetadata* – sets the metadata value on a specified document.

You can then sort the documents in the current segment by using the *SortSegDoc* script function:

**Note:** If you do not use enveloping the whole input job is one segment.

*Example 1* *Job begin script*

```
$ret = DeclareMetadata("zip", "N"; "A")
```

*Example 2* *Post-processor Job begin script*

```
$doc_id = getFirstSegDoc();
```

```
while (num($doc_id) > 0)
{$zip = "1234"
$setSegMetaData($doc_id, "zip", $zip);
$doc_id = GetNextSegDoc($doc_id);}
```

# Sort keys for documents in relational database (Document Broker Plus)

If the documents are retrieved from a relational database (Document Broker Plus), you must use metadata associated with the stored documents as sort keys.

### Prerequisites

Metadata is defined using document type resources in the Project that stored the documents. To be able to define sort keys in your Project, you must connect the resource set that contains the document type resources to your runtime configuration.

### To define a sort key

1. Open the Runtime Output Connector Settings dialog box (generic layer).
2. Click the **Job End** icon and select the **Document Sort** tab.
3. Click **Browse**. The Browse for Resources dialog box opens.
4. Browse to and open the document type resource that contains the metadata to use as key.
5. Click  . The Add Sort Key dialog box opens.
6. Configure the sort key:
   - **Metadata name** – the metadata name to use as sort key.
   - **Sort order** – Ascending or Descending.
7. Click **OK**.

### Defining sort keys using scripts

The *DeclareMetadata* script function specifies a metadata name to be used for sorting or bundling of documents. The metadata name can then be associated with a variable by for example using the following functions:

- *SetCurrMetadata* – sets the metadata value on the current document.
- *SetSegMetadata* – sets the metadata value on a specified document.

You can then sort the documents in the current segment by using the *SortSegDoc* script function:

**Note:** If you do not use enveloping the whole input job is one segment.

*Example 3* *Job begin script*

```
$ret = DeclareMetadata("zip", "N"; "A")
```

*Example 4* *Post-processor Job begin script*

```
$doc_id = getFirstSegDoc();
while (num($doc_id) > 0)
{$zip = "1234"
$setSegMetaData($doc_id, "zip", $zip);
$doc_id = GetNextSegDoc($doc_id);}
```

# Sort keys when no document repository is used

If the documents are defined by a document trigger, you can use any variable defined in the Project, for example the document trigger variable, as sort key.

### Prerequisites

Sorting must be enabled. See *To enable document sorting and bundling* on page 6.

### To define a sort key

1. Open the Runtime Output Connector Settings dialog box (generic layer).
2. Click the **Job End** icon and select the **Document Sort** tab.
3. Click . The Add Sort Key dialog box opens.
4. Configure the sort key:
   - **Key** – the variable to use as sort key.
   - **Type** – Numeric or String. Must correspond to the key variable value.
   - **Sort order** – Ascending or Descending.
5. Click **OK**.

# Bundling documents

**In this section**

# Defining mailing machines

Enveloping can be handled by one or more mailing machines. For example:

- One mailing machine for documents containing up to seven sheets.
- One mailing machine for documents containing 8-14 sheets.
- One mailing machine for documents containing more than 14 sheets.

In the example above, the output job is divided into one unit per mailing machine. If only one mailing machine is used, the entire output job is sent to this mailing machine.

When you define a mailing machine, you must specify the name of the mailing machine, and the maximum number of sheets per envelope for the mailing machine. The name of the mailing machine must be the same as the target folder used by the mailing machine.

*Example 1* *Using two mailing machines*

---

In this example, there are two mailing machines:

- MM1 for documents containing up to seven sheets.
- MM2 for documents containing more than seven sheets.

The output file path (see *Defining output paths to mailing machines* on page 13) is specified as:

```
C:\STRS_Data\Out\%{TARGET}\document.pdf
```

When StreamServer runs the job, the output job is divided into two units:

- One unit for all documents that contain less than eight sheets. These documents are sent in a pdf file to the target folder for MM1:

  ```
  C:\STRS_Data\Out\MM1\document.pdf
  ```

- One unit for all documents that contain more than eight sheets. These documents are sent in a pdf file to the target folder for MM2:

  ```
  C:\STRS_Data\Out\MM2\document.pdf
  ```

---

### Handling large documents

You must make sure the mailing machines can handle large documents, i.e. documents containing a large number of sheets. For example, if you only have one mailing machine, and maximum number of sheets per envelope for this mailing machine is set to 10, it cannot handle documents containing more than 10 sheets. To solve this you can do the following:

- Define a new mailing machine with unlimited number of sheets per envelope. This will require manual handling of the envelopes.
- Enable splitting of documents into several envelopes. See *Splitting documents* on page 18.

### To define a mailing machine

1 Open the Runtime Output Connector Settings dialog box (generic layer).

2 Click the **Job End** icon and select the **Enveloping** tab.

3 Click . The Add mailing machine definition dialog box opens.

4 Configure the settings:

- **Mailing machine name** – the name of the mailing machine. Must be the same as the name of the target folder used by the mailing machine.
- **Maximum sheets per envelope** – the maximum number of sheets per envelope allowed for the mailing machine.
- **Define maximum sheets per segment** – used if you need to divide large output files into several smaller files. See *Segmenting output files* on page 13.

5 Click **OK**.

### Segmenting output files

If large output jobs are sent to a mailing machine, you can divide the output file into segments, i.e. divide a large output file into several smaller files. This means a printer can start printing as soon as a segment file is ready instead of waiting until the entire output job is ready for printing.

You use the **Maximum** setting to specify the maximum number of sheets to include in a segment file. For example, if you set this limit to 1000, a segment file can include up to 1000 sheets. Note that segments are always split between documents, which means all sheets in a document are always contained in the same segment file.

You use the **Last segment max** setting to specify the maximum number of sheets to include in the last segment file. For example, if **Maximum** is set to 1000, and there are 1010 sheets left in the job, there would be 10 sheets in the last segment file if there was no option to set a separate limit for the last segment. If you set **Last segment max** to a higher value than **Maximum**, for example 1100, all 1010 sheets are included in the last segment.

# Defining output paths to mailing machines

You must specify the appropriate path and file name for the output file that contains the documents to envelope. If you are using a File or FTP output connector, you specify the path using the **File** property in the Output Connector Settings dialog box (physical layer). The path must be set to the target folders used by the mailing machines (see *Defining mailing machines* on page 12).

The target folder for the output file and the output file must be specified as `<TargetFolder>/<FileName>` where `<TargetFolder>` must include the `%{TARGET}` variable if several mailing machines are used, and `<FileName>` must include the `%{SEGMENT}` variable if you want to enable segmentation of the output files.

*Example 2* *Example of an output path*

```
C:\STRS_Data\Out\%{TARGET}\document_%{SEGMENT}.afp
```

When the output file is created, the `%{TARGET}` variable is replaced by the name of the target folder for the mailing machine. The `%{SEGMENT}` variable is replaced by a number that increases for each new segment file.

# Defining envelope keys

By default, a mailing machine handles each document in an output job as the amount of sheets to include in an envelope. For example, if an output job contains invoices only, all sheets in an invoice are included in the same envelope.

If the output job contains several types of documents, and you want to include all documents to a recipient in the same envelope, you must define how to bundle documents and prepare them for enveloping.

### Envelope keys

Bundling is applied using envelope keys, for example “customer number” if you want to include all documents to each customer in the same envelope. When you define an envelope key, you define which key to use, the type of key (numeric or string), and the sort order (ascending or descending).

How to define the envelope keys depends on the source of the documents, i.e. the type of document repository from which the documents are retrieved. See the table below.

| Document repository | Description |
|---|---|
| FastObjects database | If the documents are retrieved from a FastObjects database, you must use metadata as envelope keys. You define the envelope keys as variables (`$<metadata>`).<br>See *Envelope keys for documents in FastObjects database* on page 15. |
| Relational database | If the documents are retrieved from a relational database (Document Broker Plus), you must use metadata as envelope keys. You define the envelope keys as metadata names.<br>See *Envelope keys for documents in relational database (Document Broker Plus)* on page 16. |
| No document repository | If the documents are defined by a document trigger, you can use any variable defined in the Project, for example the document trigger variable, as envelope key.<br>See *Envelope keys when no document repository is used* on page 16. |

# Envelope keys for documents in FastObjects database

If the documents are retrieved from a FastObjects database, you must use metadata associated with the stored documents as envelope keys.

### To define an envelope key

1. Open the Runtime Output Connector Settings dialog box (generic layer).
2. Click the **Job End** icon and select the **Enveloping** tab.
3. Click  . The Add Envelope Key dialog box opens.

4 Configure the envelope key:
   - **Key** – the metadata (`$<metadata>`) to use as envelope key.
   - **Type** – Numeric or String. Must be the same type as defined for the metadata in the document repository.
   - **Sort order** – Ascending or Descending.

5 Click **OK**.

# Envelope keys for documents in relational database (Document Broker Plus)

If the documents are retrieved from a relational database (Document Broker Plus), you must use metadata associated with the stored documents as envelope keys.

### Prerequisites

Metadata is defined using document type resources in the Project that stored the documents. To be able to define envelope keys in your Project, you must connect the resource set that contains the document type resources to your runtime configuration.

### To define an envelope key

1 Open the Runtime Output Connector Settings dialog box (generic layer).
2 Click the **Job End** icon and select the **Enveloping** tab.
3 Click **Browse**. The Browse for Resources dialog box opens.
4 Browse to and open the document type resource that contains the metadata to use as key.
5 Click . The Add Envelope Key dialog box opens.
6 Configure the envelope key:
   - **Metadata name** – the metadata name to use as envelope key.
   - **Sort order** – Ascending or Descending.
7 Click **OK**.

# Envelope keys when no document repository is used

If the documents are defined by a document trigger, you can use any variable defined in the Project, for example the document trigger variable, as envelope key.

### Prerequisites

Enveloping must be enabled. See *To enable document sorting and bundling* on page 6.

### To define an envelope key

1. Open the Runtime Output Connector Settings dialog box (generic layer).
2. Click the **Job End** icon and select the **Enveloping** tab.
3. Click  . The Add Envelope Key dialog box opens.
4. Configure the envelope key:
   - **Key** – the variable to use as envelope key.
   - **Type** – Numeric or String. Must correspond to the key variable value.
   - **Sort order** – Ascending or Descending.
5. Click **OK**.

# Splitting documents

**In this section**



## Splitting a document set across several envelopes

By default, all documents that match an envelope definition are bundled together as a document set, which means they should be included in the same envelope. For example, if you have the following mailing machines:

- MM1 for envelopes containing up to seven sheets.
- MM2 for envelopes containing 8-14 sheets.
- MM3 for envelopes containing more than 14 sheets (manual handling of envelopes).

Then all documents containing more than 14 sheets are sent to MM3, and these documents must be enveloped manually.

To minimize manual handling of envelopes, you can enable **Minimize manual handling** when you define envelopes. If you do, StreamServer tries to sort the documents in the same document set (i.e. documents with the same envelope key) across several envelopes, which reduces manual handling.

*Example 3* *Minimizing manual handling*

In this example you have the following mailing machines:

- MM1 for envelopes containing up to seven sheets.
- MM2 for envelopes containing 8-14 sheets.
- MM3 for envelopes containing more than 14 sheets (manual handling of envelopes).

The envelope definition uses “customer ID” as key, which means all documents with the same “customer ID” should be included in the same envelope.

Suppose that you have a three documents, each containing six sheets, where all documents have the same “customer ID”.

- If you select **Minimize manual handling**, MM2 will handle the enveloping and create two envelopes – one envelope with two documents (12 sheets) and one envelope with one document (six sheets).
- If you do not select **Minimize manual handling**, all 18 sheets are sent to MM3 (manual handling).

Note that if any of the three documents contains more than 14 sheets, all sheets are sent to MM3 even if you select **Minimize manual handling**.

## Splitting a document across several envelopes

By default, all documents with the same envelope key are bundled together as a document set. The **Minimize manual handling** option (see *Splitting a document set across several envelopes* on page 18) enables splitting of a document set with the same envelope key across several envelopes.

If you want to enable splitting of a document in a document set, i.e. enable splitting of a separate document across several envelopes, you must use advanced enveloping.

If a document is split across several envelopes, the address must be printed on the first sheet in each envelope. You can achieve this by either:

- Producing page-formatted data where all pages contain the address.
- Repeating the page that contains the address for each envelope.
- Inserting a pre-defined address page for each envelope.

### To enable splitting of documents

1. Open the Runtime Output Connector Settings dialog box (generic layer).
2. Click the **Job End** icon and select the **Enveloping** tab.
3. Click **Advanced**. The Advanced Enveloping dialog box opens.
4. Select **Allow document splitting**.
5. Enter the appropriate settings and click **OK**.

| Setting | Description |
|---|---|
| **Simple** | No address page is added to the envelopes.The address must be included on all logical pages in the output. |
| **Copy address page** | The first page of the document is duplicated and put into each envelope. |
| **Insert address page** | Inserts a new address sheet in each envelope, except in the first envelope. The address sheet must be available as an overlay in a resource set connected to the runtime configuration. Select the overlay to be used for the address sheet. |
| **Define address sheet layout** | If you select **Copy address page** or **Insert address page**, you can define the layout to use for the address pages. |

# Limiting the number of envelopes to recipients

If you have enabled **Minimize manual handling** (see *Splitting a document set across several envelopes* on page 18) or **Allow document splitting** (see *Splitting a document across several envelopes* on page 19) some recipients may receive to many envelopes.

You can set a limit defining the maximum number of envelopes to send to a recipient. If this limit is exceeded, the corresponding documents are sent to a mailing machine for manual handling.

### To limit the number of envelopes

1. Open the Runtime Output Connector Settings dialog box (generic layer).
2. Click the **Job End** icon and select the **Enveloping** tab.
3. Click **Advanced**. The Advanced Enveloping dialog box opens.
4. Select **Limit envelope count**.
5. Specify **Max envelopes per recipient** and click **OK**.

# Managing inserts

Pre-printed sheets, for example marketing material, can be included in the envelopes. You can take these inserts into account when calculating the total number of sheets in an envelope.

### Weight equivalents table

If the inserts are printed on different paper quality than the sheets used for the output documents, you must specify weight equivalents for the inserts to count the total number of sheets correctly.

A weight equivalent is the weight of the insert divided by weight of the standard sheet. For example, if the weight of the standard sheet is 15 grams, and the weight of the insert is 30 grams, the weight equivalent is 2.

Weight equivalents are defined in a weight equivalents table, which is a table resource in a resource set connected to the runtime configuration. The table must contain two tab separated columns. The first column lists insert trays. The trays are defined by **Insert mask** in the OMR settings, see *Defining OMR codes* on page 24. The second column lists the weight equivalents for each insert. The maximum value is 16.

**Note:** You do not have to include weight equivalents equal to 1 in the table.

*Example 4* *Weight equivalents table*

```
//!CodePage UTF8!
1    2
3    0.5
5    1.5
```

### To enable counting of inserts

1. Open the Runtime Output Connector Settings dialog box (generic layer).
2. Click the **Job End** icon and select the **Enveloping** tab.
3. Click **Advanced**. The Advanced Enveloping dialog box opens.
4. Select **Count inserts**.
5. Enter the appropriate settings and click **OK**.

| Setting | Description |
|---|---|
| **Define insert equivalents** | Enables selection of a weight equivalents table. Browse to and select the table resource that contains the weight equivalents table. |
| **Optimize inserts** | Makes sure a recipient does not receive more than one insert of the same type. This could happen if documents are split across several envelopes. |

# OMR

OMR (Optical Mark Reader) codes are a set of strokes that can be used by an enveloping machine to, for example, identify the first and last pages in a document.

Depending on the requirements of the enveloping machine, you can define:

- Standard OMR codes (STD)
- Generic OMR codes
- Bar codes (where the strokes in an OMR code definition can have different thicknesses)

### In this section

# Defining OMR codes

You can add one or more OMR codes to a document. You specify OMR codes at Job Begin, Document Begin, Process Begin, or Page Begin.

**Note:** If you use constant values for the OMR code, add the OMR code at Job begin only, for performance reasons.

### Standard strokes

STD OMR codes use standard strokes, i.e. each stroke position in the code has a predefined meaning. See *STD strokes* on page 26.

### Bar functions

By defining a generic OMR code or a bar code, you can use bar functions at any stroke position in the OMR code. See *Bar functions* on page 27.

### To define an OMR code at Job Begin

This procedure describes how to define an OMR code at Job Begin. To define OMR codes at other levels (Document Begin etc.) you must select the corresponding icon.

1. Open the Runtime Output Connector Settings dialog box (generic layer).
2. Click the **Job Begin** icon and select the **OMR** tab.
3. Click  . The Define Options dialog box opens.
4. Configure the OMR settings (see *OMR settings* on page 25 and click **OK**.

### OMR settings

| Setting | Description |
|---|---|
| **Name** | The OMR code name. This is only used as an internal reference.<br>Only the first OMR code of two or more OMR codes with identical names is printed. An OMR code defined on both Job begin and Document begin will be printed only on Job begin. |
| **Type** | The OMR code type.<br>For **STD, STDH, and STDE** OMR types, see *STD strokes* on page 26.<br>For all barcod types, you can define the OMR by using bar functions, see *Bar functions* on page 27. |
| **X (mm)** | The absolute horizontal position of the OMR code (`0` is left). |
| **Y (mm)** | The absolute vertical position of the OMR code (`0` is top). |
| **Rotation (degrees)** | The rotation of the OMR code. |
| **Side** | The side of the sheet where the OMR code is placed. |
| **Bar functions** | Functions representing bar strokes.<br>See *Bar functions* on page 27. |
| **Low sequence** | The sequence start value. |
| **High sequence** | The highest value in sequence. |
| **Sequence order** | The sequence order. **None** is **Ascending**. |
| **Insert mask** | By setting this option, inserts can be included in envelopes depending on the OMR type. You can use a maximum of 31 insert trays, so you can enter an integer from 0 to ($2^{32}$-1) since the value is used as a binary mask. For example, 13=1101 means that inserts 1,3 and 4 are used for the documents within the job. |
| **ON value** | The value that indicates that a specific stroke is printed, normally `1`. |
| **Bar height (mm)** | The height of the OMR code in millimeters. |
| **Font size** | The font size for the barcode text. |

For the OMR type specific attributes, refer to the corresponding barcode specification.

# STD strokes

STD OMR codes use standard strokes, i.e. each stroke position in the code has a predefined meaning.

| Stroke position | Value | Explanation |
|---|---|---|
| 1 | 1 | Synchronization |
| 2 | 0 | Space, double row-distance |
| 3 | 1 | Read control |
| 4 | Seq1 | 1st bit of the sequence |
| 5 | Seq2 | 2nd bit of the sequence |
| 6 | Seq4 | 3rd bit of the sequence |
| 7 | Seq8 | 4th bit of the sequence |
| 8 | notlast | 1 – if the sheet is not the last one.<br>0 – if the sheet is the last one. |
| 9 | last | 1 – if the sheet is the last one.<br>0 – if the sheet is not the last one. |
| 10 | 0 | Call slave – this stroke is not supported by StreamServe, always 0. |
| 11 | Ins1 | 1st bit of insert mask |
| 12 | Ins2 | 2nd bit of the insert mask |
| 13 | Ins3 | 3rd bit of the insert mask |
| 14 | Ins4 | 4th bit of the insert mask |
| 15 | Parity | 1 – if the sum of strokes with value 1 is even<br>0 – if the sum of strokes with value 1 is odd. |
| **Only for STDE and STDHE** | | |
| 16 | 1 | Extra bit at the end. |

# Bar functions

The table below list the bar functions that can be used.

### Bit and byte sequence

By default, a bar function defines a bit in a binary sequence. You can switch back and forth to a byte sequence with < and > characters. The < character switches to byte sequence, and the > character switches to bit sequence.

For example

```
1/1/0/<1/0/>1/0/0/0
```

### Byte translation

Bytes are compiled from the bit/byte sequence in the same order as the bit sequence is entered.

For example:

```
1/1/1/0/0/0/1/1 = 11100011 = 227
```

To compile bytes using old INT2OF5 and Code 39 OMR behavior, add a # character in front of the bit sequence. For example:

```
#1/1/0/1/0/1/0/0/0
```

**Note:** For the old INT2OF5 and Code 39 OMR behavior, this is not recommended when byte sequences are included.

### String within the bar function sequence

You can add strings to the bar function sequence by surrounding the string with single quotes. For example

```
1/1/0/<'25'>/1/0
```

| Barfunction | Description |
|---|---|
| `1` | Bar is printed. |
| `0` | Bar is not printed. |
| `seqN` | *N*=1,2,3,4,... A bar is printed depending on the binary/byte sequence.<br>See *Example 2* on page 30.<br>**Note:** In byte representation, this barfunction must be specified as `<seq/>`. If for example `seq=25`, the result is the same as `1/<'25'>/1/0=1252` |

| Barfunction | Description |
|---|---|
| ins*N* | *N*=1,2,3,4... A bar is printed depending on binary or byte representation of the insert mask.<br>For example, if *Insert mask* is set to 10, which is 1010 in binary representation, it means that ins2 and ins4 are available to use as inserts for the job. The maximum value of *N* is 31.<br>**Note:** In byte representation, the value is a byte instead of a bit. |
| firstpage | A bar is printed if it is the first page in the document. |
| notfirstpage | A bar is printed if it is not the first page in the document. |
| lastpage | A bar is printed if it is the last page in the document. |
| notlastpage | A bar is printed if it is not the last page in the document. |
| $*variable* | A bar is printed depending on value of *$variable* |
| $*variable*[*N*] | A bar is printed depending on value of *$variable* and the bit or byte representation. *N*=1,2,3,4...<br>**Bit representation**<br>The maximum value of *N* is 31.<br>For example, if the following bar functions are specified:<br>$page_no[3]/$page_no[2]/$page_no[1]<br>then for $page_no = 3, bars are printed corresponding to 0/1/1.<br>**Byte representation**<br>*N* represents the *N*:th character of the variable.<br>For example:<br>$variable="abcd"<br>=> $variable[0]="a", $variable[3]="c"<br>**Variable arrays**<br>the *N*:th bit or byte (character) is specified as the second pair square brackets, e.g. $var[0][1], where [1] is the first bit or the second character in the first element of the $variable array. |

| Barfunction | Description |
|---|---|
| `evenparity` | A bar is printed if the number of the other bars is odd. This bar function is used as a security check. If this function is set, the numbers of bars on the sheet must be even. This means that if the number of bars printed to the sheet is odd, the parity bar must be added. If the OMR reads an odd number, there is a misreading. Therefore, security is doubled. |
| `oddparity` | A bar is printed if the number of the other bars is even. This bar function is used as a security check. If this function is set, the numbers of bars on the sheet must be odd. This means that if the number of bars printed to the sheet is even, the parity bar must be added. If the OMR will reads an even number, there is a misreading. Therefore, security is doubled. |
| `firstsheetinenvelope` | A bar is printed if it is the first sheet in the envelope. |
| `notfirstsheetinenvelope` | A bar is printed if it is not the first sheet in the envelope. |
| `lastsheetinenvelope` | A bar is printed if it is the last sheet in the envelope. |
| `notlastsheetinenvelope` | A bar is printed if it is not the last sheet in the envelope. |
| **Combinations of bar functions** | |
| *`func1`*`&`*`func2`* | A bar is printed if both *`func1`* and *`func2`* print a bar.<br>**Note:** The `evenparity` and `oddparity` functions cannot be combined. |
| *`func1`*`\|`*`func2`* | A bar is printed if *`func1`* or *`func2`* print a bar.<br>**Note:** The `evenparity` and `oddparity` functions cannot be combined. |

# OMR examples

*Example 1* *OMR example with envelope bar functions*

OMR codes are printed from left to right. The following bar functions print an OMR code with two bars on every sheet, one bar if the sheet is first in the envelope, one bar if the sheet is last in the envelope, and another two bars on every sheet:

**Bar functions** `1/1/firstsheetinenvelope/lastsheetinenvelope/ /1/1`

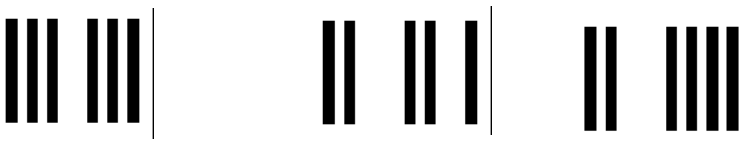
First sheet of envelope | Intermediate sheets | Last sheet of envelope

*Example 2* *OMR example with sequence bars*

This example is identical to the example above, but sequence bars are added to the OMR code. The sequence bars have binary representation.

**Bar functions** `1/1/firstsheetinenvelope/lastsheetinenvelope/ /1/1/seq1/seq2/seq3`

First sheet | Second sheet | Third sheet



# Labels

You can add one or more labels to an output document. The label can contain variables that for example specify the sheet number or the enveloping machine that handled the sheet.

### In this section

# Defining labels

You can add one or more labels to a document. You specify labels at Job Begin, Document Begin, Process Begin, or Page Begin.

**Note:** If you use constant values for the label, for performance reasons, add the label at Job begin only.

### To define an OMR code at Job Begin

This procedure describes how to define a label at Job Begin. To define labels at other levels (Document Begin etc.) you must select the corresponding icon.

1. Open the Runtime Output Connector Settings dialog box (generic layer).
2. Click the **Job Begin** icon and select the **Labels** tab.
3. Click . The Define Options dialog box opens.
4. Configure the label settings (see *Label settings* on page 33 and click **OK**.

### Label settings

| Setting | Description |
|---|---|
| **Name** | The label name. This is only used internally as a reference.<br>Only the first label of two or more labels with identical names is printed. A label defined on both Job begin and Document begin is printed only on Job begin. |
| **X (mm)** | The X position of the label in millimeters (0 is left). |
| **Y (mm)** | The Y position of the label in millimeters (0 is bottom). |
| **Rotation (degrees)** | The clock-wise rotation of the label. |
| **Side** | The side of the sheet where the label is placed. |
| **Place on sheets** | The sheet(s) in a document where the label is placed. Select **Between** to put the label on all sheets except the first and the last sheet. |
| **Font** | The font to use for the label text. |
| **Font size** | The size in number of points of the font used for the label text. |
| **Text** | The printed text. You can use the following variables (case sensitive) in your text:<br><%NumSheets> – The number of sheets in the document.<br><%MailMach> – The name of the mailing machine.<br><%EnvelNr> – The number of the current envelope. Every envelope gets a unique number when it is created.<br><DOCUMENT> – The number of the current document. Documents are numbered per segment (if defined) and mailing machine.<br><PAGEINJOB> – The number of the current logical page. Pages are numbered per segment (if defined) and mailing machine.<br><SHEETINJOB> – The number of the current sheet. Sheets are numbered per segment (if defined) and mailing machine.<br><PAGESINDOCUMENT> – The number of pages in the document.<br><PAGEINDOCUMENT> – The number of the current page in the document.<br><SHEETINDOCUMENT> – The number of the current sheet in the document. |

# Sorting and bundling scripts

You can retrieve data about the documents that will be sorted and bundled. For example you can retrieve metadata for a document, the number of sheets in a document, or the number of documents in an envelope. This information can be used as criteria for different processing options for the documents in a specific segment. For example, you can add or skip a specific page depending on the invoice number, or split the input job into two output files, where each file contain documents with a specific range of number of pages.

For an example of how to divide output depending on number of pages in the documents, see *Example 1* on page 36

### Retrieving information about documents

You retrieve data about the documents by using document sorting and bundling scripting functions. For example:

- *GetPPMetadata* – returns the value of a metadata for a specific document, for example the invoice number.
- *GetSegMetadata* – returns the value of a metadata for a specific document within the segment, for example the invoice number.
- *GetPPDocProperty* – returns the value of a specified property for a document, for example the number of logical pages in the document.
- *GetEnvelNr* – returns the envelope number that a document is included in.
- *GetSegment* – returns the segment number that a document and an envelope are included in.
- *IsFirstPageInEnvelope* – returns information about whether or not the current page is the first page in the envelope.

### Taking action based on the retrieved information

You can perform actions on the documents or parts of the documents using scripting functions. For example:

- *NextPage* – skips the rest of the current page and processes next page.
- *NextDoc* – skips the rest of the current document and processes the next document.
- *NextSegment* – skips the rest of the current segment and processes the next segment.
- *InsertPage* – inserts a page in the current document.
- *FlushOutput* – sends parts of the job to the output connector even if the job is not completed.
- *SetDestPath* – sets a destination path, and optionally, a file name.

To add a document sorting and bundling script, see *Adding a document sorting and bundling script* on page 38.

For more information on document sorting and bundling scripting functions, see the *Scripting reference*.

*Example 1* *Document sorting and bundling script on Job begin for splitting output*

```
$doc_index = GetFirstDoc();
$PageCount = 0;
while(num($doc_index) > 0)
{$PageCount = num($pageCount) + num(GetPPDocProperty($doc_index,
"pages"));
$doc_index=GetNextPPDoc($doc_index);}
if(num($PageCount) > 5)
{$filename = "./out/big.pcl";}
else
{$filename = "./out/small.pcl";}
```

# Generating statistics reports

You can retrieve and report information about documents to be post-processed. The information can be written to a separate file or a repository. You must use scripting to retrieve and report the information. For example, you can use the following script functions to retrieve statistics information:

- *GetEnvelNr*
- *GetMailMachine*
- *ConnectorPageActual*

**Note:** The *GetDocSheetCount* function can only be used if you have specified a mailing machine. See *Defining mailing machines* on page 12.

To write or store the retrieved information you can, for example, use the following script functions:

- *FileOpen*
- *FileWrite*
- *OdbcConnect*
- *OdbcExecute*

**Note:** To use Post-processor scripting to insert or skip pages, you must specify a sort variable or a mailing machine.

*Example 2* *Script that generates number of sheets per mailing machine*

```
$mailmachine = "Sheets to mailingmachine" + "<20>" +
GetMailMachine() + "<3a><20>";
$pagecount = ConnectorPageActual();
fileopen("Statistics\pagecount.txt","a");
filewrite("Statistics\pagecount.txt",$mailmachine + $pagecount  +
"<0d><0a>" + "<0d><0a>");
```

This can result in the following rows in the statistics file:

```
Sheets to mailingmachine Invoice: 24
Sheets to mailingmachine Overflow: 16
```

# Adding a document sorting and bundling script

1. Open the Runtime Output Connector Settings dialog box (generic layer).
2. Right-click the level where you want to add the script and select **Script**. The Script editor opens.
3. If you use scripting on Process begin or Page begin, select whether to run the script before or after the Process or Page is processed.
4. Edit the script and click **OK**.