



# M3 Enterprise Collaborator Flat File Definition Tool User Guide

Version 11.4.1.0

Published August 2013

**Copyright © 2013 Infor. All rights reserved.**

## **Important Notices**

The material contained in this publication (including any supplementary information) constitutes and contains confidential and proprietary information of Infor.

By gaining access to the attached, you acknowledge and agree that the material (including any modification, translation or adaptation of the material) and all copyright, trade secrets and all other right, title and interest therein, are the sole property of Infor and that you shall not gain right, title or interest in the material (including any modification, translation or adaptation of the material) by virtue of your review thereof other than the non-exclusive right to use the material solely in connection with and the furtherance of your license and use of software made available to your company from Infor pursuant to a separate agreement, the terms of which separate agreement shall govern your use of this material and all supplemental related materials ("Purpose").

In addition, by accessing the enclosed material, you acknowledge and agree that you are required to maintain such material in strict confidence and that your use of such material is limited to the Purpose described above. Although Infor has taken due care to ensure that the material included in this publication is accurate and complete, Infor cannot warrant that the information contained in this publication is complete, does not contain typographical or other errors, or will meet your specific requirements. As such, Infor does not assume and hereby disclaims all liability, consequential or otherwise, for any loss or damage to any person or entity which is caused by or relates to errors or omissions in this publication (including any supplementary information), whether such errors or omissions result from negligence, accident or any other cause.

Without limitation, U.S. export control laws and other applicable export and import laws govern your use of this material and you will neither export or re-export, directly or indirectly, this material nor any related materials or supplemental information in violation of such laws, or use such materials for any purpose prohibited by such laws.

## **Trademark Acknowledgements**

The word and design marks set forth herein are trademarks and/or registered trademarks of Infor and/or related affiliates and subsidiaries. All rights reserved. All other company, product, trade or service names referenced may be registered trademarks or trademarks of their respective owners.

## **Publication Information**

Release: 11.4.1.0

Publication date: August 15, 2013

Document Number: MECFFDTUG\_11.4.1.0\_UWA\_01

---

## Version Log

The version log describes the changes between versions of this document.

Part Number	Release Date	Description
MECFFDTUG-10410W-01	201308	Upversion to 11.4.1.0
MECFFDTUG-11400W-01	201305	Upversion to 11.4.0.0
MECFFDTUG-103W-01	201208	Upversion to 10.3.0.0
MECFFDTUG-92W-01	201201	Upversion to 9.2.0.0
MECFFDTUG-91W-01	201105	Updated for 9.1.4.0
		Added Flat File Examples
MECFFDTUG-91W-01	201005	Updated for version 9.1.3.1

---

# Contents

<b>Chapter 1: Flat File Definition Tool Guide Overview.....</b>	<b>7</b>
About this Guide.....	7
Users of this Guide.....	8
Flat File Definition Tool Overview .....	8
Usage of Flat File Definition Tool.....	8
Managing Flat Files in Run Time.....	9
Managing Flat Files in Design Time.....	10
 <b>Chapter 2: Flat File Definitions .....</b>	<b>13</b>
Flat File Overview.....	13
Flat and XML File Formats.....	14
Flat File Message Structures.....	14
Flat File Definition Structure.....	15
Metadata in a Flat File Definition.....	16
Flat File Parser Behavior and Encoding.....	19
 <b>Chapter 3: Managing Flat File Definitions.....</b>	<b>21</b>
Flat File Repository Manager Overview.....	21
Starting Flat File Repository Manager.....	22
Creating a New Flat File Definition .....	23
Adding and Refreshing Local Folder .....	24
Importing a Flat File Definition.....	24
Exporting a Flat File Definition.....	25

---

Deleting a Flat File Definition .....	25
Editing a Flat File Definition.....	26
<b>Chapter 4: Describing Flat Files.....</b>	<b>28</b>
Flat File Descriptor Tool Overview .....	28
Flat File Descriptor User Interface.....	28
Usage of Flat File Descriptor.....	31
Defining Flat File Structures in Message Definition View .....	32
Creating a New Message Component .....	32
Changing the Message Component Property Values .....	32
Adding a Group to a Message.....	33
Adding a Record to a Group.....	33
Adding a Field to a Record.....	34
Defining Template Flat Files.....	34
Opening and Preparing a Template Flat File .....	35
Defining Position-Based Fields Using the Template Flat View.....	35
Defining Character-Separated Fields .....	36
Verifying Flat File Definition.....	36
Matching Considerations in Template Flat View.....	37
Verifying Flat File Template Against Definition .....	38
Validating a Flat File Definition .....	38
Sample Data Generation.....	39
Generating Sample Files.....	39
Sample Flat File Generation.....	40
Sample Schema Generation .....	40
Sample XML Generation.....	41

---

<b>Appendix A: Flat File Definition Format.....</b>	<b>43</b>
Example of a Flat File Definition in XML.....	43
Elements and Types.....	44
Elements in Flat File Definition.....	44
ComplexTypes in Flat File Definition.....	50
 <b>Appendix B: Flat File Examples.....</b>	 <b>55</b>
Record-Separated and Field-Separated Flat File to XML.....	55
XML to Record-Separated and Field-Separated Flat File.....	57
Record-Separated Fixed-Length Fields Flat File to XML.....	58
XML to Record-Separated Fixed-Length Fields Flat File.....	60
Fixed-Length Records and Fields Flat File to XML.....	61
XML to Fixed-Length Records and Fields.....	63
Default Empty Field Value.....	64
Padding and Alignment.....	67
Trim.....	70
Escape Character.....	72
Ignore Preceding and Trailing Field Separator.....	73
Record-Separated Semi-Fixed-Length Fields To and From XML.....	76
Variable Number of Occurrences, Groups, and Identifiers.....	79

---

# Flat File Definition Tool Guide Overview

# 1

This document provides a general information on the administration and uses of M3 Enterprise Collaborator (MEC) Flat File Definition tool. Following are the topics included in this chapter:

- "About this Guide" on page 7
- "Users of this Guide" on page 8
- "Flat File Definition Tool Overview " on page 8
- "Usage of Flat File Definition Tool" on page 8
- "Managing Flat Files in Run Time" on page 9
- "Managing Flat Files in Design Time" on page 10

## About this Guide

This guide provides you with a general understanding about all the aspects of the creation, management, and how flat file definition tool works in order to process flat file messages. This includes an overview of flat files, flat file messaging support in M3 Enterprise Collaborator (MEC), flat file definitions, Flat File Repository Manager, Flat File Descriptor, and Flat File Parser.

Appendix A describes the details of a flat file definition created manually using an XML editor of the user's choice.

Appendix B contains examples and exercises about the available features of the Flat File Parser.

This guide does not provide details on how to create and maintain partner agreements to enable MEC Server to detect and process flat messages in run time. Also, this document does not provide information about particular flat file formats, such as comma separated values (CSV).

For more information on the following topics, see their respective documentation in **MyLawson.com**:

- Installing M3 Enterprise Collaborator Tools, see *MEC Server and Client Tool Installation Guide*
- Creating and maintaining partner agreements, see *MEC Partner Administration Tool User Guide*
- Creating and maintaining MEC mappings, see *MEC Mapping Manager User Guide*
- SNA Communication Examples, see *MEC Administration Guide*

## Users of this Guide

The users of the M3 Enterprise Collaborator (MEC) Flat File Definition Tool User Guide are application engineers and business consultants who use Flat File Repository Manager and Flat File Descriptor. These users create, edit, or validate flat file definitions for their flat file messages.

The users of Partner Administration Tool can also refer to this guide for flat file definitions and similar topics. These users manage the partner agreement information models and the MEC server to send and receive messages between users and their partners. Also, these users can have multiple messaging partners or multiple installations of MEC on a single server.

## Prerequisite Knowledge

Users of this guide must be knowledgeable in the following topics:

- XML and XML schemas (XSD)
- Flat files and flat file messaging

## Flat File Definition Tool Overview

The Flat File Definition Tool is a set of design time tools used to create a flat file definition according to the specifications of a flat file message.

The Flat File Definition tool consists of the following components:

- Flat File Repository Manager: A tool for managing flat file definitions in the Flat File Definition Repository.
- Flat File Descriptor: A tool for creating, editing, and validating flat file definitions.

M3 Enterprise Collaborator (MEC) uses a flat file definition to process a message and transform a flat file to XML or XML to a flat file. To do this, the Flat File Definition tool provides support for flat file messaging in MEC.

Initially the flat file messaging support was delivered for the conversion of bank messages alone, consisting of the so called banking plug-in. This function was later extended to become the flat file parser that could manage a larger range of flat files in run time. Since MEC version 2.0, flat file messaging support was extended to include design time tooling and simplified administration and configuration of flat file messaging.

## Usage of Flat File Definition Tool

The Flat File Definition Tool provides flat file messaging support in M3 Enterprise Collaborator (MEC) for receiving, processing, and sending flat file messages from both a design time and a run time perspective.



MEC manages both inbound and outbound flat file messages using flat file definitions created in Flat File Definition tool. The following are two common scenarios:

**Inbound messages (to M3):**

- M3 Business Engine receives a flat file message containing data from a partner through MEC.
- Message will be processed and a reply can be sent back to the sender in XML or flat file format.

**Outbound messages (from M3):**

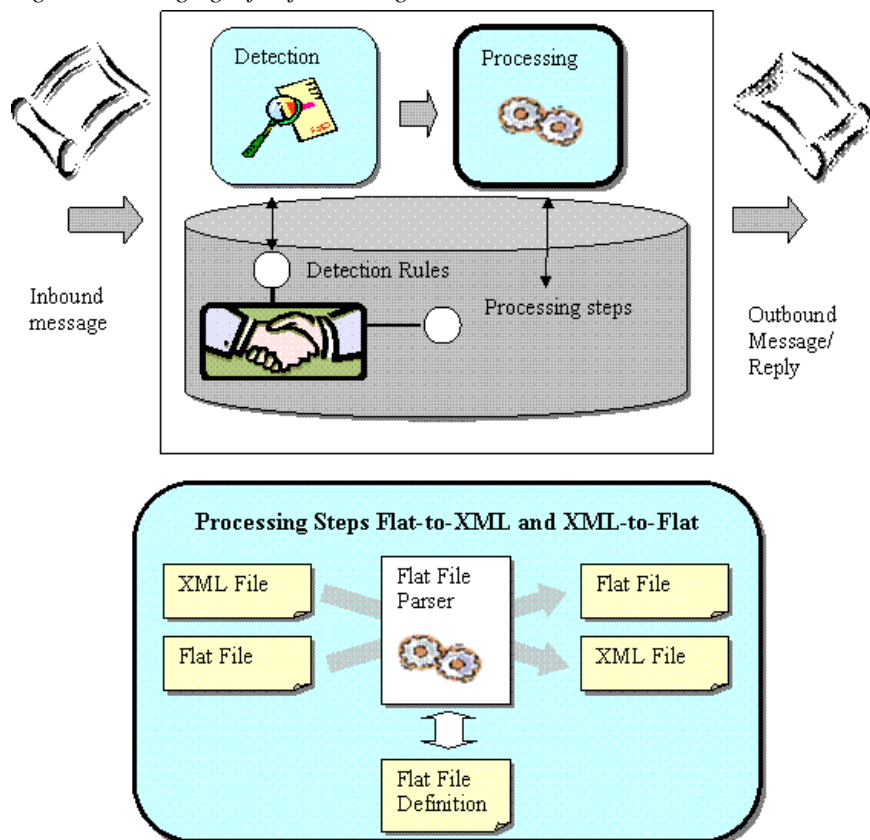
- M3 Business Engine generates a flat file message and sends it to a partner through MEC.
- Message is sent from MEC as a result of the execution of programs in M3.

## Managing Flat Files in Run Time

The following overview shows how M3 Enterprise Collaborator (MEC) manages a flat file in run time:

- 1 Detection of message:** For an inbound flat file message, the first step is to detect and identify the message that was received. The detection rules are defined using the Partner Administration Tool.
- 2 Processing a message:** After a message is detected, MEC can be set up to process the flat file. The processing steps are defined using the Partner Administration Tool. For inbound flat file messages, a typical first processing step is to convert the flat file to XML. MEC does this by using the flat file definition for the given flat file message. Another typical processing step is applying a MEC mapping for retrieving or storing data in M3.

Figure 1. Managing a flat file message in run time



**Note:** In earlier versions of MEC, conversion of flat files to XML files was a step performed prior to the detection step, since only XML files could be detected by MEC. Now, flat files and XML files are managed similarly. Flat files are detected and then processed just like XML messages. If a mapping is to be applied, the transformation to XML is still a necessary step.

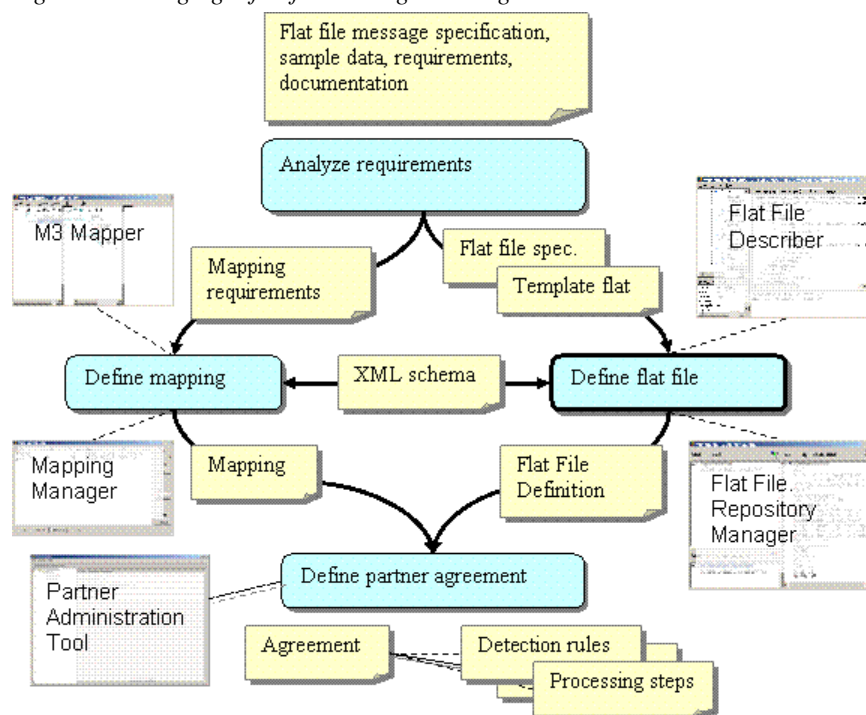
## Managing Flat Files in Design Time

The following overview shows how M3 Enterprise Collaborator (MEC) manages flat files for detecting and processing in design time. Typically, the format of the inbound or outbound flat file message is defined in advance and cannot be changed.

- 1 Analyze the requirements. You have to ensure that all necessary documentation for the flat file message and sample flat files exist.
- 2 Create a flat file definition according to the flat file message specification. You need this to transform a flat file to XML or XML to flat file.
- 3 If MEC mapping will be applied on the results of a transformation, you must generate the XML schema for the corresponding XML message from the Flat File Descriptor Tool in order to create a mapping.

- 4 Create the MEC mapping and publish it.
- 5 Publish the flat file definition using the Flat File Repository Manager. You need this to set up the transformation processing step using the Partner Administration Tool.
- 6 Publish the mapping using Mapping Manager. You need this to set up the mapping processing step using the Partner Administration Tool.
- 7 Define an agreement using the Partner Administration Tool. The agreement includes rules for detecting and processing flat file messages.
- 8 Generate sample data using the Flat File Descriptor. You can use these files as raw data for creating test data.

Figure 2. Managing a flat file message in design time



**Note:** The detection rules do not depend on a flat file definition being uploaded. The detection rules are set up separately from defining the flat file for the purpose of transforming it from, or to XML. The flat file definition is only used for transformation between XML and flat.

## Where to Go for Additional Information

For information on	See
Defining Flat Files	<a href="#">"Describing Flat Files"</a> on page 28
Defining Mapping	<i>MEC Mapping Manager User Guide</i>

For information on	See
Partner Agreement	<i>MEC Partner Administration Tool User Guide</i>

This chapter provides background information about Flat Files and Flat File Definitions. Following are the topics included in this chapter:

- ["Flat File Overview" on page 13](#)
- ["Flat and XML File Formats" on page 14](#)
- ["Flat File Message Structures" on page 14](#)
- ["Flat File Definition Structure" on page 15](#)
- ["Metadata in a Flat File Definition" on page 16](#)
- ["Flat File Parser Behavior and Encoding" on page 19](#)

## Flat File Overview

A flat file is a data file with the following characteristics:

- It is formally structured to enable a machine to parse it.
- It does not contain information about how the information is structured.
- It does not relate to or contain any links to other files.

A flat file is typically a standalone listing of data. Relations to other files, for example: describing the customers for an order or the vendors of the purchases, are described in other tools and are usually managed in a relational database or sometimes in a file manager. Flat files can be related, but only if the applications sending and receiving them are programmed to be aware of such relationships.

A flat file can be used for storing and transporting all types of application data in the form of flat file messages. Flat files are often used for exchanging messages such as purchase orders, invoices, and payments between two nodes in a network, for instance, between a supplier and a vendor. Flat file messages are therefore electronic business messages or simply business messages in flat file format.

## Flat and XML File Formats

Flat files and XML are two frequently used formats for data files. XML files and flat files are both formally structured. However, XML files contain constructs that flat files do not have. That is, each flat file can be stored in XML format, but the opposite is not true. Each flat file describes a message that has a corresponding XML schema that matches its contents. However, there is no corresponding flat file structure for each XML schema. An XML is a standard while flat files can be constructed in different ways.

Another important distinction between XML and flat files is that flat files do not contain metadata or information about the data itself. A receiver of a flat file must use a flat file definition to determine how data is organized in a flat file. The definition includes properties such as length of each item, type of data whether integer, float, string, or Boolean, and any relationships to other data items within the file.

## Flat File Message Structures

Message structures are defined by flat files containing records or group of records. Each record is specified in a single line and consists of a specified sequence of fields. The sequence of fields in a record is defined as having a number of occurrences.

Fields from each record may have a fixed length or may be terminated by a separator. There are field separators and record separators. In a comma separated value (CSV) file, a common type of flat file, the field separator is a comma “,” and records are separated with a line feed character (UNIX style), or a combination of line feed and carriage return character (Windows style).

### Example 1: Six records separated by a line break.

The fields in each record are separated by a comma. The third and fourth records are two occurrences of the same type of record. They are structured in exactly the same way.

```
DHDR,JSA EENMI567,Blueberryhills Technologies;222-555-1234EENMAM98329
A0176643002200011796V4000230,DAVID JONES CO;C011000209485456-145
RM IV23456,41796
RM IV2345,-30000
CM,CREDIT MEMO ADJUSTMENT
A0425577710000110316V40000004,SOFTBANK
```

### Example 2: A record that uses extra formatting to avoid separator confusion.

If you want to use a separator without its special meaning, use an escape sequence consisting of a string of characters. When it precedes a separator, an escape sequence has the effect of negating the separator's special meaning. The escape sequence, forward slash “/”, is consumed in the parsing of the field.

```
This field has a comma/,that needs to be escaped,this is another field,
```

### Example 3: A record that uses an identifier.

The first field in the record is defined as an identifier, which then contributes to making the enclosing records' signature possible to separate from other records in the message.

Order,39827483927,03982409238,02384902

## Flat File Definition Structure

A flat file definition contains detailed information about how the data is organized within the file and any relationships to other data items within the file. A flat file definition describes the structure of a flat file, indicating which records the message consists of and which fields are transferred within each type of record. It gives each field its tag name that then labels the field in the corresponding XML file.

A flat file definition enables M3 Enterprise Collaborator (MEC) to transform a business message in run time by capturing the metadata about an inbound or an outbound flat file. A flat file definition is needed when setting up a processing step for transforming a flat file to XML or XML to flat.

Each type of flat file that will be transformed, to or from XML, must have a flat file definition to enable MEC to process it. This flat file definition is stored in the Flat Definition Repository and an agreement to use this definition is set up in Partner Administration Tool.

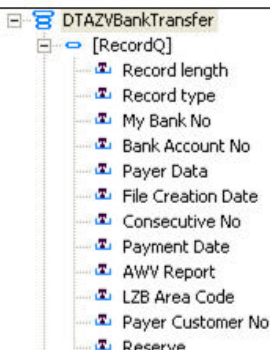
The following table lists the typical components in a flat file definition:

Component	Description
Message	A top-level component that represents the entire flat file message
Records	A sequence of groups
Group	A specified sequence of records. A group is used when two or more records repeat as a unit within a message
Record	A specified sequence of fields.
Field	A sequence of characters that forms an atomic piece of data

The following example illustrates how the flat file structure is defined from a simple flat file. In the following example, the flat file message structure has a top-level Message component "DTAZVBankTransfer." The fields in the flat file are structured as separate Fields in the structure under a Record component. XML tags in the corresponding XML file are also related to the records and fields of the flat file.

You can also use and add a Group component to define a nested structure of records and fields in the flat file structure and the XML file.

Figure 3. Example of defining the flat file structure

Flat File	Flat File Definition	XML
1111Q2222222211111111	 <p>DTAZVBankTransfer</p> <ul style="list-style-type: none"> <li>[RecordQ] <ul style="list-style-type: none"> <li>Record length</li> <li>Record type</li> <li>My Bank No</li> <li>Bank Account No</li> <li>Payer Data</li> <li>File Creation Date</li> <li>Consecutive No</li> <li>Payment Date</li> <li>AWV Report</li> <li>LZB Area Code</li> <li>Payer Customer No</li> <li>Reserve</li> </ul> </li> </ul>	<pre> &lt;DTAZVBankTransfer&gt;   &lt;RecordQ&gt;     &lt;RecordLength&gt;1111&lt;/RecordLength&gt;     &lt;RecordType&gt;Q&lt;/RecordType&gt;     &lt;MyBankNo&gt;22222222&lt;/MyBankNo&gt;     &lt;BankAccountNo&gt;1111111111&lt;/BankAccountNo&gt;     &lt;PayerData&gt;BBBBBBBBBBBBBBBBBBBBBBBBBBBB     &lt;FileCreationDate&gt;222222&lt;/FileCreationDate&gt;     &lt;ConsecutiveNo&gt;11&lt;/ConsecutiveNo&gt;     &lt;PaymentDate&gt;222222&lt;/PaymentDate&gt;     &lt;AWVReport&gt;A&lt;/AWVReport&gt;     &lt;LZBAreaCode&gt;11&lt;/LZBAreaCode&gt;     &lt;PayerCustomerNo&gt;22222222&lt;/PayerCustomerNo&gt;     &lt;Reserve&gt;BBBBBBBBBBBBBBBBBBBBBBBBBBBB   </pre>

## Metadata in a Flat File Definition

The metadata about a flat file is captured in a flat file definition. Some metadata are mandatory and some are optional. Mandatory metadata must be defined either by using a message default or by setting the metadata on the group, record, or field level. Examples of mandatory metadata for a field are Name, Tag Name, Data Type, and if applicable, Start Position.

The following tables provide a complete list of metadata and description for each component in a flat file definition.

### File

The following table lists the metadata for the File component.

Metadata	Details
ID	identifier for a message transformation definition
Messages	allows only one message per file

### Message Definition

The following table lists the metadata for the Message Definition component.

Metadata	Details
Name	set for documentation purposes
Tag name	tag identified in XML or a tag written to XML



Metadata	Details
Escape character	escape sequence for field and record separators
Record separators	common record separators are: carriage return - \r or <b>newline</b> and line feed - \n or <b>lf</b>
Fixed length separator	if set to true, this defines fixed lengths of records
Field separator fields	setting this value overrides any field length definitions. There can be only one separator in a message. A record separator is considered as a valid field separator.
Ignore preceding field separator	ignores preceding field separators
Default alignment	sets the alignment to use when a field does not have a defined alignment. Can be set for numeric and alpha fields
Default padding	sets the padding character to use when a field does not have a defined padding character. Can be set for numeric and alpha fields
Default trim	sets a trim behavior for XML to record if no trim is defined for a field. This will either remove a preceding and trailing white space or keep the data unchanged. Can be set for numeric and alpha fields
Default empty field value	value set here will be used if data is empty for a field or tag. By not setting the value it will cause an empty tag to be written or an empty field. Can be set for numeric and alpha fields
Add first field separator	adds an extra field separator (defined on the message level) in front of the first field of every record in the message
Add last field separator	adds an extra file separator after the last field of every record in the message

## Records

The following table lists the metadata for the Records component.

Metadata	Details
Group	a specified sequence of records. Groups are used to create an additional structure in the corresponding XML file.
Record	a group of fields in sequence

## Group

The following table lists the metadata for the Group component.

Metadata	Details
Name	set for documentation purposes
Tag name	tag to identify or write
Minimum occurrences	minimum required number of times this group must occur
Maximum occurrences	maximum number of times this group is allowed to occur. Unlimited occurrences is specified by 0 (null)

## Record

The following table lists the metadata for the Record component.

Metadata	Details
Maximum occurrences	maximum number of times this group is allowed to occur. Unlimited occurrences is specified by 0 (null)
Name	set for documentation purposes
Tag name	tag to identify or write
Minimum occurrences	minimum required number of times this group must occur
Maximum occurrences	maximum number of times this group is allowed to occur. Unlimited occurrences is specified by 0 (null)

## Field

The following table lists the metadata for the Field component.

Metadata	Details
Name	set for documentation purposes
Tag name	tag to identify or write
Data type	used to differentiate alignment, padding, trimming, and default empty value handling. Can be alpha or numeric. Default type is alpha.
Padding character	character used to pad this field

Metadata	Details
Alignment	alignment to use for this field
Empty field value	value to use if this field is empty
Trim	indicates whether or not to trim the field
Identifier	defines if this field is an identifier or not
Default value	identifier value that will be tested against or written to
Start position	field starting position if length-defined fields (no field separator) are defined
Length	If the field is shorter than this length it will be padded to fill the field length. If the length is specified for use as parsing fields without a field separator, then the last field may be defined as "-1" which will allow the field to have a variable length.

## Where to Go for Additional Information

For information on	See
Flat File Definition format	<a href="#">"Flat File Definition Format"</a> on page 43
Changing the properties of Flat File components	<a href="#">"Changing the Message Component Property Values"</a> on page 32

## Flat File Parser Behavior and Encoding

A Flat File Parser is a module used to break down a document into its component parts. It is the component in M3 Enterprise Collaborator (MEC) that performs the actual conversion between flat files and XML files using a defined record structure for the message. This detailed information about how the data is organized within the file and any relationships to other data items within the file is given by a flat file definition.

In addition to the message structure, fields are typically managed in the parsing according to the values specified in the following metadata:

Metadata	Description
Padding	Characters used when a field of a defined length does not contain enough data to occupy its entire length. For example, when a field has a length of seven digits but only three are used.
Alignment	Location of the padding characters' placement relative to the information padded.
White Space Trim	Removal of all formatting directives surrounding a string of characters, such as tabs, line feeds, and blanks.
Empty Value	Value used when a field lacks any strings of information, and this is signaled using a special string of information that is inserted by the parser. Also, records of variable number of occurrences and groups with variable number of occurrences place different limitations on what you can parse.
Record Separators	Characters commonly used as standard end of line separators.

The following table lists the characters commonly used as record separators:

#### Standard end of line separators

	Description	In Java	In MEC	Hex values
PC	[return][linefeed]	[\r][\n]	[\r][\n]	[0D][0A]
UNIX	[line feed]	[\n]	[\n]	[0A]
Mac	[return]	[\r]	[\r]	[0D]

The encoding of flat file messages is set using the Partner Administration Tool and is not part of the flat file definition. The same flat file definition can therefore be used for flat files that have the same structure but are encoded differently.

**Note:** The flat file definitions themselves are always encoded in Java Encoding (UTF-16) when imported to and exported from the Flat File Repository.

## Where to Go for Additional Information

For information on	See
Setting the encoding of flat file messages	<i>Partner Administration Tool User Guide</i>

This chapter describes how you can manage Flat File Definitions using the tools and features of the Flat File Repository Manager. Following are the topics included in this chapter:

- ["Flat File Repository Manager Overview" on page 21](#)
- ["Starting Flat File Repository Manager" on page 22](#)
- ["Creating a New Flat File Definition " on page 23](#)
- ["Adding and Refreshing Local Folder " on page 24](#)
- ["Importing a Flat File Definition" on page 24](#)
- ["Exporting a Flat File Definition" on page 25](#)
- ["Deleting a Flat File Definition " on page 25](#)
- ["Editing a Flat File Definition" on page 26](#)

## Flat File Repository Manager Overview

The Flat File Repository Manager is a tool used to import flat file definitions into the M3 Enterprise Collaborator (MEC) database and make it available to the Partner Administration Tool. To do this, you must import an existing flat file definition from a local disk to an available repository that is a MEC server.

The Flat File Repository Manager also serves as a single-point user interface (UI) for creating a new flat file definition on the local computer or editing an existing flat file definition from the repository. Flat File Repository Manager helps manage the physical locations of XML files of flat file definitions. Following is a list of tasks that you can perform to manage flat file definitions:

- [Starting Flat File Repository Manager](#)
- [Creating a New Flat File Definition](#)
- [Adding and Refreshing Local Folder](#)
- [Importing a Flat File Definition](#)
- [Exporting a Flat File Definition](#)

- [Deleting a Flat File Definition](#)
- [Editing a Flat File Definition](#)

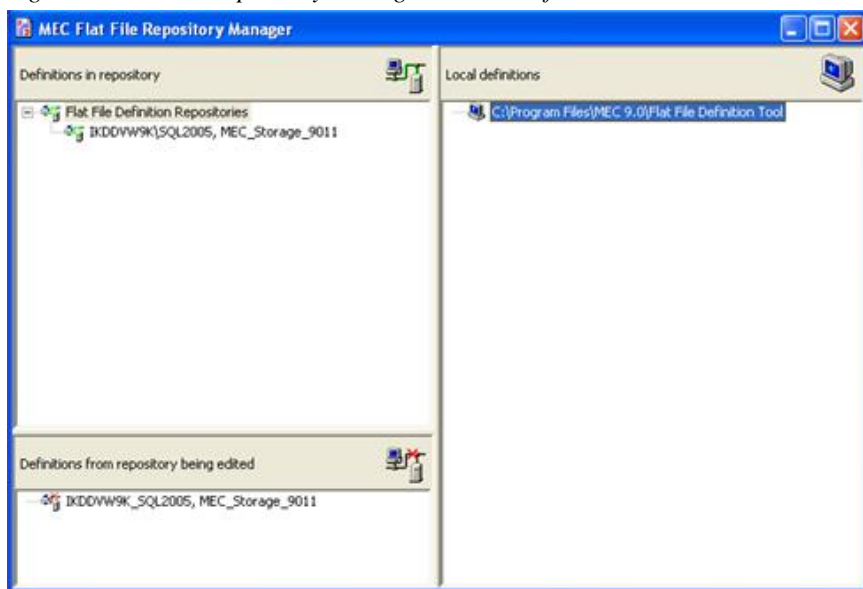
## Starting Flat File Repository Manager

- 1 Click the Windows Start menu.
- 2 Point to M3 Enterprise Collaborator 9.x.
- 3 Select Flat File Repository Manager.

The Flat File Repository Manager opens

The user interface of the Flat File Repository Manager has three panes providing three different views. The icon displayed in each pane indicates the environment in which the user is currently working.

Figure 4. Flat File Repository Manager User Interface



View panes	Description
Definitions in repository View	<p>Top left pane, this shows the flat file definitions that are uploaded in the repository. There can be multiple repositories, each representing a different M3 Enterprise Collaborator (MEC) Server.</p> <p>The green cable image in the icon indicates that the user is working in an online environment. If the connection to the database is lost, the green color changes to red.</p>

View panes	Description
Definitions from repository being edited View	<p>Bottom left pane, this shows all flat file definitions that are being edited. You can edit an existing version of a flat file definition in the repository instead of downloading a flat file definition to a local folder from the repository and then later import it as a new flat definition. Each repository has its own work folder.</p> <p>If working offline, the network icon is x-marked. The flat file editing is from the work folder.</p>
Local definitions View	<p>Right pane, this view shows folders on the local disk where local and locally downloaded flat file definitions are stored. The default folder displayed is the folder where the flat File Repository Manager was installed. Additional folders are added by browsing the local disk for definitions or by downloading definitions from a repository. Only flat file definitions found in the default folder are displayed.</p> <p>The computer icon in this pane indicates the local machine.</p>

## Where to Go for Additional Information

For information on	See
How to configure the working folder location	<i>MEC Server and Client Tool Install Guide</i>

## Creating a New Flat File Definition

- 1 Right-click in the Local definitions view and select New Flat File Definition.  
The Flat File Descriptor tool starts.
- 2 Create the flat file definition.
- 3 Save the flat file definition created, and then import the file to the appropriate repository.
- 4 Close and exit the Flat File Descriptor.  
The new flat file definition created displays in Repository Manager.

## Where to Go for Additional Information

For information on	See
Creating flat file definitions	<a href="#">"Describing Flat Files"</a> on page 28

## Adding and Refreshing Local Folder

- 1 Right-click in the Local definitions view and select Add Folder.
- 2 Select a flat file definition folder to add by browsing the local disk or by downloading definitions from a repository.
- 3 Right-click in the Local definitions view and select Refresh to list up the new folder in the view.
- 4 Verify that the new local folder is displayed in Local Definitions view.
- 5 Verify that existing flat file definitions in local folders display in the Repository Manager.
- 6 Close and exit the application

## Importing a Flat File Definition

A flat file definition must be imported into a Flat File Definition Repository before it can be used. Use this procedure to import a flat file definition into a flat file definition repository.

- 1 In the Local definitions view, right-click the definition you want to import and select Import.  
The Flat File Definition Meta Data window appears.

- 2 Select the name of repository in the Repository to import definition to field.
- 3 Type the name of the flat file definition to import In the Name field.
- 4 Type a short description of the flat file definition to import in the Description field.
- 5 Click import when you are finished.

The imported flat file definition displays in Definitions in Repository view.

- 6 Close and exit the application



## Exporting a Flat File Definition

Use this procedure to export a flat file definition. You can export a copy of a flat file definition from the Flat File Repository to the local disk. The copy can then be imported into another repository or be used as a draft for a modified version.

**Important:** It is not possible to overwrite a flat file definition on local disk. You must export to another folder that does not contain the same flat file definition.

- 1 In the Definitions in repository view, right-click the flat file definition you want to export and select Export.
- 2 Verify that the exported definition displays in Local definitions view

## Deleting a Flat File Definition

Use this procedure to delete a flat file definition from the repository or from a local disk.



**Caution:** You cannot undo a flat file definition delete action. The flat file definition is physically removed from the database or the local Definitions view.

- 1 In the Definitions in repository view, right-click the flat file definition you want to delete and do either of the following.

Task	Steps
To delete from the repository.	<p><b>Note:</b> Flat file definitions that are enabled and in use under the Definitions from the repository being edited view cannot be deleted from the Definitions in the repository view.</p> <ol style="list-style-type: none"> <li>a Select Delete.</li> <li>b Verify that the deleted definition is not listed in Definitions in repository view.</li> </ol>
To delete from a local disk.	<ol style="list-style-type: none"> <li>a Select Delete.</li> <li>b Verify that the deleted definition is not listed in Local definitions view.</li> </ol>

- 2 Close and exit the application.

## Editing a Flat File Definition

### To edit a definition in the Repository

- 1 In the Definitions in repository view, right-click the flat file definition you want to edit and select Check Out.

The Flat File Descriptor tool starts.

- 2 Edit the flat file definition.
- 3 Save the edited flat file definition.
- 4 In the Definitions from Repository being edited view , right-click the edited definition and select Check In.

The repository is updated with the edited flat file definition.

- 5 Close and exit the application.

### To edit a flat file definition in a local folder

- 1 In the Local definitions view, right-click the flat file definition you want to edit and select Edit.

The Flat File Descriptor tool starts.

- 2 Edit the flat file definition.
- 3 Save the edited flat file definition.

The Local definitions view is updated with the edited flat file definition.

### To continue editing a flat file definition

- 1 In the Definitions from Repository being edited view, right-click the flat file definition you want to continue editing and select Edit.

The Flat File Descriptor tool starts.

- 2 Continue editing the flat file definition.
- 3 Save the edited flat file definition.
- 4 In the Definitions from Repository being edited view, right-click the edited definition and select Check In.

The repository is updated with the edited flat file definition.

**To undo a flat file definition edit**

- 1 In the Definitions from Repository being edited view, right-click the edited flat file definition and select Undo.
- 2 Verify that the flat file definition reverts to the last version saved.
- 3 Close and exit the application.

**Where to Go for Additional Information**

For information on	See
Editing flat file definitions	<a href="#">"Describing Flat Files"</a> on page 28

This chapter provides information about how you can create, edit, and validate flat file definitions using the tools and features of the Flat File Descriptor. Following are the topics included in this chapter:

- ["Flat File Descriptor Tool Overview " on page 28](#)
- ["Flat File Descriptor User Interface" on page 28](#)
- ["Usage of Flat File Descriptor" on page 31](#)
- ["Defining Flat File Structures in Message Definition View " on page 32](#)
- ["Defining Template Flat Files" on page 34](#)
- ["Verifying Flat File Definition" on page 36](#)
- ["Sample Data Generation" on page 39](#)

## Flat File Descriptor Tool Overview

The Flat File Descriptor is a graphical design time tool packaged in M3 Enterprise Collaborator. This tool helps you create, verify, and validate flat file definitions.

The Flat File Descriptor can also be used to define flat files or as a complement tool to an XML editor of your choice. You can also generate sample files in XML, XSD, and flat file formats using this tool.

The Flat File Descriptor is launched from the Flat File Repository Manager.

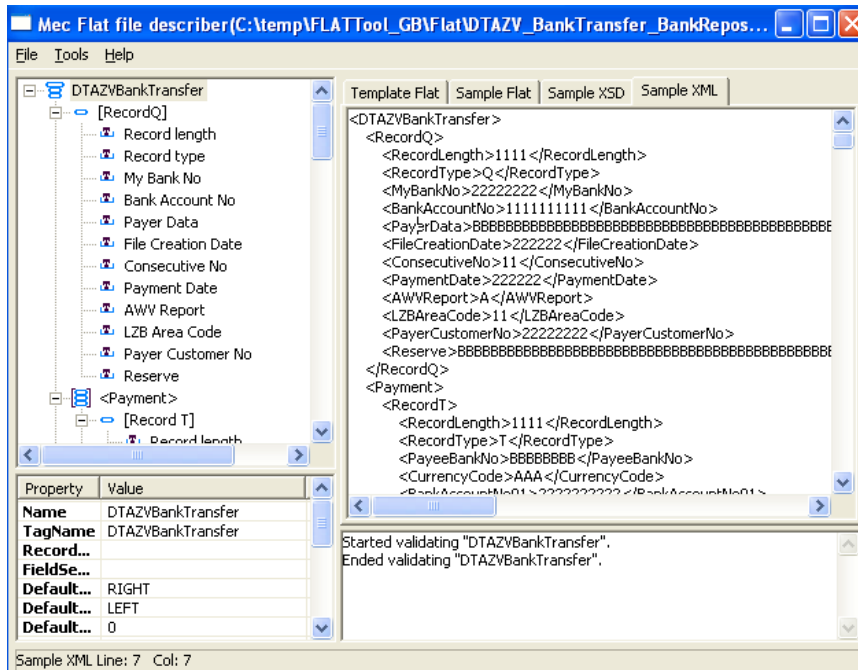
## Flat File Descriptor User Interface

The user interface of the Flat File Descriptor is composed of four different views:

- Message Definition in the top left pane
- Properties in the bottom left pane
- Output in the bottom right pane
- Template Flat, Sample Flat, Sample XSD, and Sample XML views in the top right pane

In addition to these views, File, Tools, and Help menus are also available on the menu bar.

Figure 5. Flat File Describer User Interface



Following is a topical listing of Flat File Describer User Interface components.

## Message Definition View

This view shows the message structure of a flat file. You can define the structure of a flat file using components such as Message, Group, and Record and Field.

## Properties view

This view shows the properties of components selected in the tree view. You can set a value for each property but some properties can have a default value defined on the message level. Some properties are mandatory and some are optional.

## Output view

This view shows status information, mismatches and other results when validating flat file definitions and verifying a flat file against a template flat file.

## Template Flat View

This view shows the structure of a template flat file from which you can create a flat file definition. You can create fields by dragging and dropping fields from the template flat file to the definition in the Message Definition view. You can also verify that the flat file definition matches a template flat file. Fields that have been defined are colored in the template flat file.

## Sample Flat View

This view shows a preview of the output from an XML-to-flat transformation. You can generate a sample flat file from the Tools menu. You can save the sample flat file to disk.

## Sample XSD View

This view shows a preview the XML schema that defines the format of the sample XML. You can generate a schema from the Tools menu. You can save the sample XSD to disk and use it when you define a mapping.

## Sample XML view

This view shows a preview of the output from a flat-to-XML transformation. You can generate a sample XML file from the Tools menu. You can save the sample XML file to disk.

## File Menu

The following table lists the options available in the File Menu.

Option	Description
Save Definition	Saves the current definition to disk. If the definition is new, you have to select a folder and provide a file name.
Save Definition as	Saves a copy of the current definition to disk and opens it for editing. You have to select a folder and provide a file name. The original definition closes without saving the changes.
Open Template File	Loads a template flat file and displays it in the Template Flat view. You can locate a template file using the browse dialog.
Save Template Flat File	Saves the contents in the Template Flat view to disk. You have to select a path and provide a file name.
Save Sample Files	Saves the content in the Template Flat, Sample XML, Sample Flat, and Sample Schema views to disk. You have to select a folder where the files will be saved. Existing sample files in the selected folder are replaced.
Exit	Exits the Flat File Descriptor Tool

## Tools Menu

The following table lists the options available in the Tools Menu.

Option	Description
Create Sample XML	Generates a sample XML and displays it in the Sample XML view.
Create Sample XSD	Generates an XML schema and displays it in the Sample Schema view.
Create Sample Flat	Generates a sample flat file and displays it in the Sample Flat view.
Validate	Validates the flat file definition.

## Help Menu

The following table lists the command available in the Help Menu.

Command	Description
About	Displays version information.

## Where to Go for Additional Information

For information on	See
How to use the Template Flat view	<a href="#">"Defining Template Flat Files"</a> on page 34
How to create mappings	<i>MEC Mapping Manager User Guide</i>

## Usage of Flat File Descriptor

The Flat File Descriptor enables you to create Flat File Definitions. You can use the Flat File Descriptor to do the following tasks:

- [Defining Flat File Structures in Message Definition View](#)
- [Defining Template Flat Files](#)
- [Verifying Flat File Definition](#)

- [Sample Data Generation](#)

## Defining Flat File Structures in Message Definition View

This section describes how you can define flat file structures in the Message Definition View. You can perform the following tasks:

- ["Creating a New Message Component " on page 32](#)
- ["Changing the Message Component Property Values " on page 32](#)
- ["Adding a Group to a Message" on page 33](#)
- ["Adding a Record to a Group" on page 33](#)
- ["Adding a Field to a Record" on page 34](#)

## Creating a New Message Component

Use this procedure to create a new message component.

- 1 Right-click in the Message Definition view and select Add Message as Sibling.  
A new message component displays in the Message Definition view.
- 2 In the Properties view, verify that default values are provided for mandatory properties.

## Changing the Message Component Property Values

Use this procedure to change the message component property values.

- 1 In the Message Definition view, select the component whose property you want to edit.  
The properties of the component display in the Properties view.
- 2 In the Properties view, select the value you want to edit and click the Value field.  
Current value is highlighted.
- 3 Type the new value in the Value field.



**Important:** When a property has a predefined set of values, you can only select a new value from a drop-down list.

- 4 Press Enter to confirm the new value.

## Where to Go for Additional Information

For information on	See
Description of properties and valid values for message components	<a href="#">"Metadata in a Flat File Definition"</a> on page 16

## Adding a Group to a Message

Use this procedure to add a group to a message.

- 1 In the Message Definition view, right-click a Message and select any of the following:

Option	Result
Add Group as Child	A new group is added to a message or a second group is added and placed above an existing group.
Add Group as Sibling	A group is added and placed below an existing group

**Note:** A Message can contain any number of Groups and Records.

- 2 Verify that the new group is added correctly.

## Adding a Record to a Group

Use this procedure to add a record to a group.

- 1 In the Message Definition view, right-click a Group component and select one of the following:

Option	Result
Add Record as Child	A new record is added to a group or a second record is added and placed above an existing record.
Add Record as Sibling	A record is added and placed below an existing record

**Note:** A Group can contain any number of Records or other Groups.

- 2 Verify that the new record is added correctly.

## Adding a Field to a Record

Use this procedure to add a field to a record.

- 1 In the Message Definition view, right-click a Record component and select one of the following:

Option	Result
Add Field as Child	A new field is added to a record or a second field is added and placed above an existing field.
Add Field as Sibling	A field is added and placed below an existing field.

- 2 Verify that the new record is added correctly.

## Defining Template Flat Files

This section describes how you can define template flat files in the Template Flat View. You can perform the following tasks:

- ["Opening and Preparing a Template Flat File " on page 35](#)
- ["Defining Position-Based Fields Using the Template Flat View" on page 35](#)
- ["Defining Character-Separated Fields " on page 36](#)

## Opening and Preparing a Template Flat File

A template flat file helps you create a flat file definition. Use this procedure to open and prepare a template flat file.

- 1 In the Flat File Describer File menu, select Template > Open Template file.
- 2 Select an existing template flat file and click Open.  
The template file displays in the Template Flat view.
- 3 If the template flat file is based on a sample flat file, prepare the template file in the following way:
  - Remove all duplicate records.
  - Add optional records to enable them to be defined.
- 4 In the Flat File Describer File Menu, select Template > Save Template file to save your work.



**Caution:** Editing parts of the template flat file that have already been defined might cause the flat file definition to become inconsistent with the template flat file.

## Defining Position-Based Fields Using the Template Flat View

Fields can be defined in the flat file definition using drag and drop as an alternative to right-clicking in the tree and selecting components to be added from a menu. Use this procedure to define position-based fields in a flat file definition using the template flat view.

- 1 Open an existing position-based flat file.
- 2 In the Template Flat view, select a field to be defined and drag it to the Message Definition view.

**Important:** It is not possible to drag and drop fields that span several records. Records have to be defined manually in the Message Definition View, one by one and in correct order.

- 3 If this is the first field to be defined, drop the field on the record. Otherwise, drop the field on the previous field.

A field is created in the Message Definition view with the correct start position and field length.

- 4 If the tool cannot automatically calculate the start position, set the start position manually to -1.

**Important:** If you attempt to set a start position and field length that would cause fields to overlap, a warning message is displayed and the user must provide a new value.

- 5 Once the structure is defined, set the metadata of the field such as Name and TagName and other mandatory properties in the Properties view.

## Where to Go for Additional Information

For information on	See
Setting properties and metadata	<a href="#">"Changing the Message Component Property Values "</a> on page 32

## Defining Character-Separated Fields

Fields in a character-separated flat file for example, in CSV, can be defined in the flat file definition by dragging and dropping in a similar way as with position-based flat files. With character-separated files, however, you can drag and drop multiple fields from a record. Use this procedure to define character-separated files in a flat file definition using the Template Flat View.

- 1 Open an existing character-separated flat file.
- 2 In the Template Flat view, select a field or multiple fields to be defined and drag it to the Message Definition view.
- 3 If this is the first field to be defined, drop the field on the record. Otherwise, drop the field on the previous field.  
The field or field components are created in the Message Definition view.
- 4 Once the structure is defined, set the metadata of the field such as Name and TagName and other mandatory properties in the Properties view.

## Where to Go for Additional Information

For information on	See
Setting properties and metadata	<a href="#">"Changing the Message Component Property Values "</a> on page 32

## Verifying Flat File Definition

The Template Flat view can be used to verify that the flat file definition matches a template flat file. This section discusses the following topics:

- ["Matching Considerations in Template Flat View" on page 37](#)

- "Verifying Flat File Template Against Definition " on page 38
- "Validating a Flat File Definition " on page 38

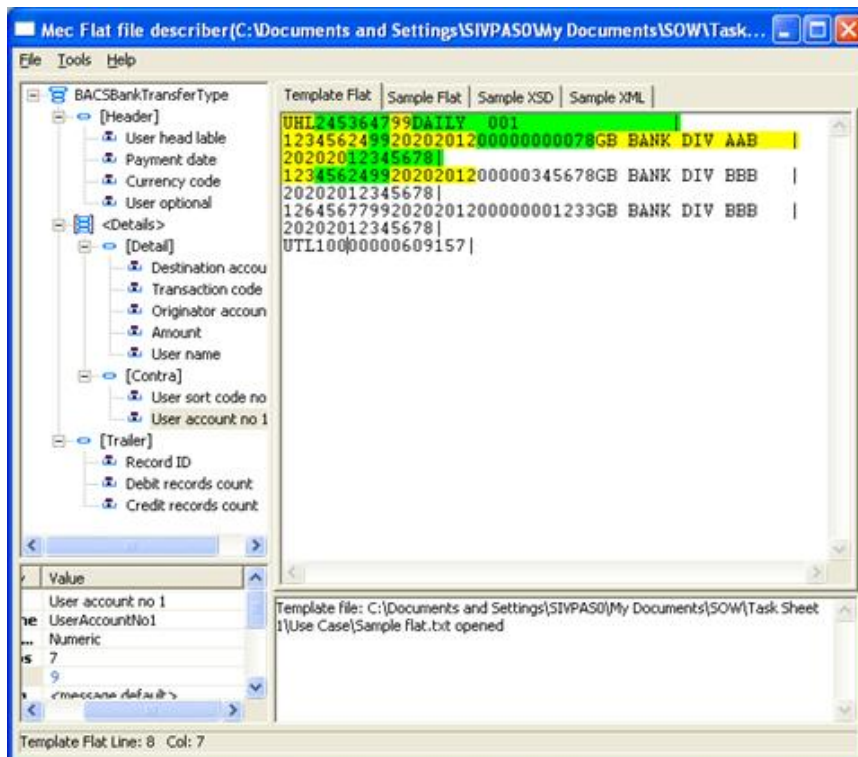
## Matching Considerations in Template Flat View

To match a template flat file and a flat file definition, the Template Flat view assumes that records are defined in order, from top to bottom, and that there are no repetitions of records in the template flat file. The marked fields are done record by record from top to bottom.

In the following example, only four records are colored in the flat template file and the fourth record seems incomplete, even though the definition in the Message Definition view seems correct. The cause of this mismatch is that the contents in the Template Flat view are not template flat files rather a sample file. There are repetitions of the Detail and Contra records, something that the Template Flat view does not consider.

The only way to have marked fields corresponding to the template flat file is to delete repetitive groups and records from the template file.

Figure 6. Example of a Template Flat File



## Where to Go for Additional Information

For information on	See
Opening and Preparing a template flat file	<a href="#">"Opening and Preparing a Template Flat File "</a> on page 35

## Verifying Flat File Template Against Definition

Use this procedure to verify a template flat file against a flat file definition. Fields that have been defined are colored in the template flat file.

- 1 Open an existing template flat file.
- 2 Right-click in the Template Flat view and select Refresh Field Markings.  
Fields that are defined appear colored.
- 3 Verify that all fields have color markings and no mismatch is detected.

**Note:** The Flat File Descriptor stops color marking the fields as soon as a mismatch is detected between the template flat file and flat file definition. The reason of the incompatibility between the flat definition and template flat file is displayed in the Output view.

## Validating a Flat File Definition

Validating a flat file definition ensures that your flat file definition is syntactically and semantically correct. The results should be considered as an indication of the validity of a definition. Validation in Flat File Descriptor covers the following tasks:

- Checking for overlapping fields in a position-based flat file.
- Checking the structure of the flat file definition itself. This is useful when the flat file definition is created manually using an XML editor and later imported into the tool.
- Initiating a verification of the template flat file against the definition when a template file is loaded.

**Important:** You still need to validate that a flat file definition is correct according to the specification of the Flat File Message.

Use this procedure to validate the syntax and semantics of a flat file definition.

**To validate a flat file definition**

- 1 Open an existing template flat file or a flat file definition.
- 2 In the Flat File Descriptor Tools Menu, select Validate.

The results of the validation process display in the Output View.

## Sample Data Generation

The Flat File Descriptor can be used to generate sample files. This section discusses the following topics:

- ["Generating Sample Files" on page 39](#)
- ["Sample Flat File Generation" on page 40](#)
- ["Sample Schema Generation " on page 40](#)
- ["Sample XML Generation" on page 41](#)

## Generating Sample Files

Use this procedure to generate sample files in XML, XSD, and flat format. These sample files are based on the message description that you have created.

- 1 Open an existing template flat file or a flat file definition.
- 2 In the Flat File Descriptor Tools Menu, select any of the following:

Option	Result
Create Sample XML	A sample XML is generated and displayed in the Sample XML view.
Create Sample XSD	A sample XML schema is generated and displayed in the Sample Schema view.
Create Sample Flat	A sample flat file is generated and displayed in the Sample Flat view.

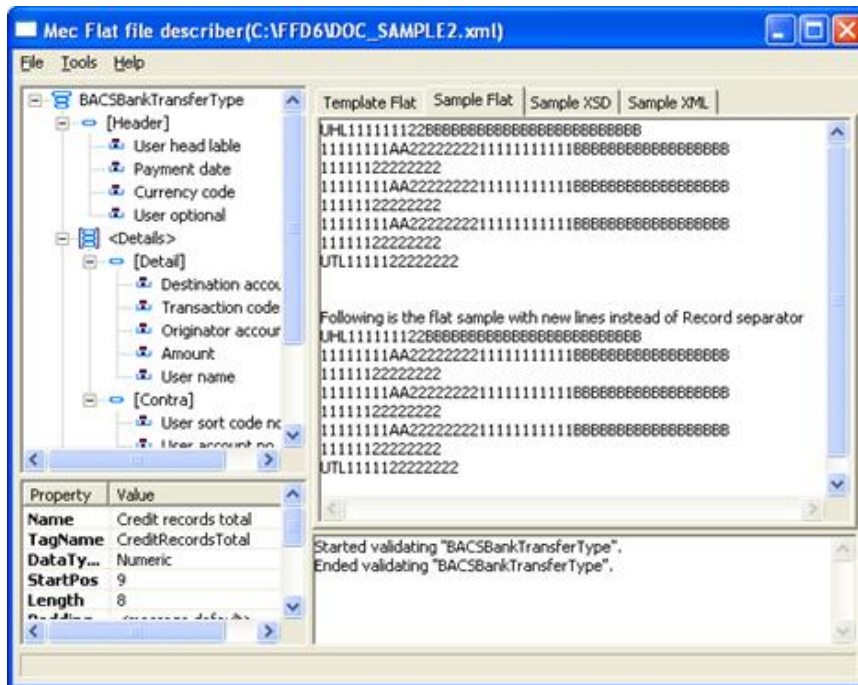
- 3 Close and exit the application.

## Sample Flat File Generation

In Flat File Descriptor, some properties in your flat file definition structure will affect the generated sample flat file. Consider the following:

- If a field has a specified length, it will be filled with data in the sample.
- If a field does not have a length, it will get a data and a separator.
- A record separator is always written after a record.
- Each occurrence of a sample record will be written three times or, if a minimum value is set, then the minimum occurrence is the set value.

Figure 7. Generated Sample Flat File data

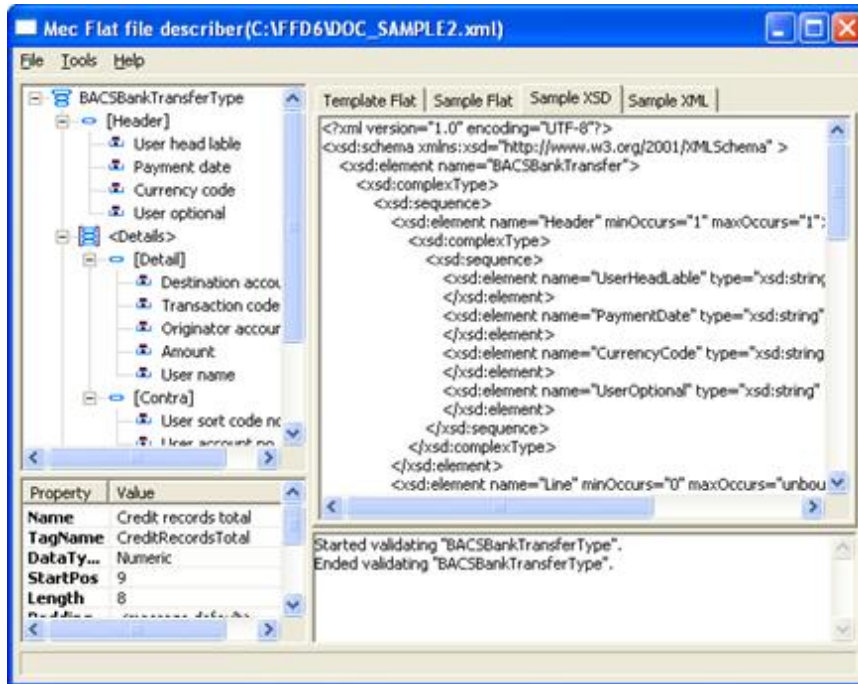


## Sample Schema Generation

In Flat File Describer, some properties in your flat file definition structure will affect the generated sample schema.



Figure 8. Generated Sample Schema

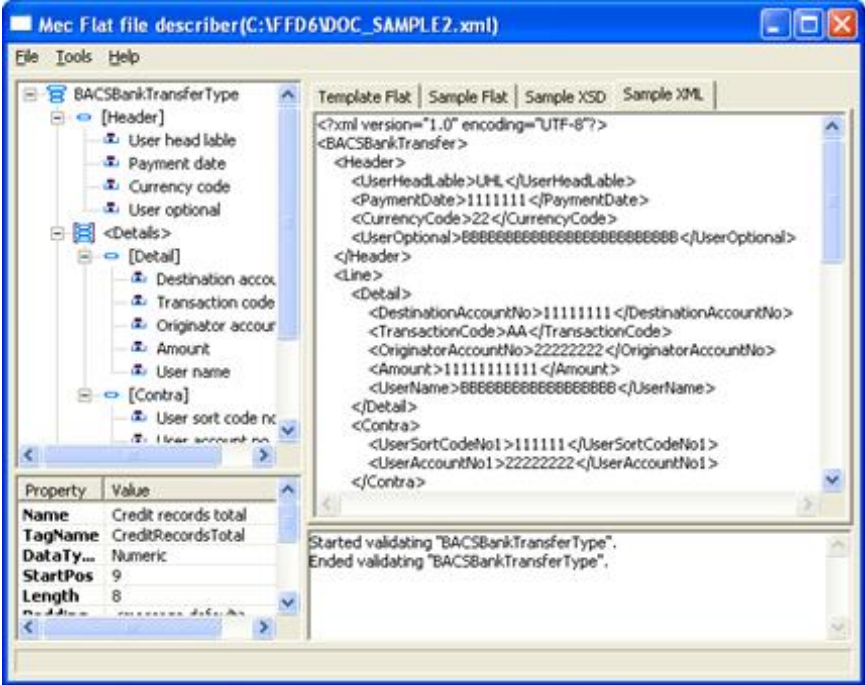


## Sample XML Generation

In Flat File Descriptor, some properties in your flat file definition structure will affect the generated sample XML file. Consider the following:

- Every unlimited occurrence will be written three times, unless a minimum number of occurrences are set.
- Alpha data type fields include "abc" (letters) as data.
- Numeric data type fields include "123" (numbers) as data.

Figure 9. Generated Sample XML



---

# Flat File Definition Format



This section details example flat file definition format that you can refer to when manually creating flat file definitions using an XML editor. The rest of this section provides the XML schema for flat file definitions.

- ["Example of a Flat File Definition in XML" on page 43](#)
- ["Elements and Types" on page 44](#)

## Example of a Flat File Definition in XML

The following example shows a flat file definition created manually using an XML editor.

```
<BankRepository>
  <Files>
    <File>
      <ID>flat_test_1</ID>
      <Messages>
        <Message>
          <Name>TopNode</Name>
          <TagName>TopNode</TagName>
          <DefaultAlignment>
            <Numeric>RIGHT</Numeric>
            <alpha>LEFT</alpha>
          </DefaultAlignment>
          <DefaultPadding>
            <Numeric>0</Numeric>
            <alpha></alpha>
          </DefaultPadding>
          <DefaultTrim>
            <Numeric>1</Numeric>
            <alpha></alpha>
          </DefaultTrim>
          <DefaultEmptyFieldValue>
            <Numeric>0</Numeric>
            <alpha></alpha>
          </DefaultEmptyFieldValue>
          <EscapeChar>/>
          <RecordSeparator>newline</RecordSeparator>
          <FieldSeparator>;</FieldSeparator>
          <IgnorePrecedingFieldSeparator>1</IgnorePrecedingFieldSeparator>
          <Records>
            <Group>
              <Name>Person</Name>
              <TagName>Person</TagName>
              <MinOccur>1</MinOccur>
              <MaxOccur>0</MaxOccur>
            </Group>
          </Records>
        </Message>
      </Messages>
    </File>
  </Files>
</BankRepository>
```

```
<Name>PersonDetails</Name>
<TagName>PersonDetails</TagName>
<MinOccur>1</MinOccur>
<MaxOccur>1</MaxOccur>
<Fields>
  <Field>
    <Name>FirstName</Name>
    <TagName>FirstName</TagName>
    <DataType>Alpha</DataType>
    <PaddingChar/>
    <DecimalLength/>
    <Alignment>LEFT</Alignment>
    <EmptyFieldValue></EmptyFieldValue>
    <Trim>0</Trim>
    <StartPos>0</StartPos>
    <Length>10</Length>
    <DefaultValue></DefaultValue>
    <Identifier>0</Identifier>
  </Field>
</Fields>
</Record>
</Records>
</Group>
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>
```

## Elements and Types

This section describes in detail the elements and complex types used in the example flat file definition in XML.

- ["Elements in Flat File Definition" on page 44](#)
- ["ComplexTypes in Flat File Definition" on page 50](#)

## Elements in Flat File Definition

The following table lists the elements used in the example flat file definition in XML. Details such as Type, Properties, Children, Usage, and Source are also provided in the following table.

Element	Type	Properties	Children	Used by	Source
Alpha	xs:string	content = simple		Default EmptyField ValueType Default Padding Type Default TrimType	<pre>&lt;xs:element name="Alpha" type="xs:string"/&gt;</pre>
Bank Repository		content = complex	Files		<pre>&lt;xs:element name="BankRepository"&gt; &lt;xs:complexType&gt; &lt;xs:sequence&gt; &lt;xs:element name="Files" type="FilesType"/&gt; &lt;/xs:sequence&gt; &lt;/xs:complexType&gt; &lt;/xs:element&gt;</pre>
Bank Repository /Files	FilesType	content = complex isRef = 0	File	FieldType	<pre>&lt;xs:element name="Files" type="FilesType"/&gt;</pre>
DataType	restriction of xs:string	content = simple			<pre>&lt;xs:element name="DataType"&gt; &lt;xs:simpleType&gt; &lt;xs:restriction base="xs:string"&gt; &lt;xs:enumeration value="Alpha"/&gt; &lt;xs:enumeration value="Numeric"/&gt; &lt;/xs:restriction&gt; &lt;/xs:simpleType&gt; &lt;/xs:element&gt;</pre>
ID	xs:string	content = simple		FileType	<pre>&lt;xs:element name="ID" type="xs:string"/&gt;</pre>
MaxOccur	xs:integer	content = simple		GroupType RecordType	<pre>&lt;xs:element name="MaxOccur" type="xs:integer"/&gt;</pre>
MinOccur	xs:integer	content = simple		GroupType RecordType	<pre>&lt;xs:element name="MinOccur" type="xs:integer"/&gt;</pre>
Name	xs:string	content = simple		FieldType GroupType MessageType RecordType	<pre>&lt;xs:element name="Name" type="xs:string"/&gt;</pre>

Element	Type	Properties	Children	Used by	Source
Numeric	xs:integer	content = simple		Default EmptyField ValueType Default Padding Type Default TrimType	<pre>&lt;xs:element name="Numeric" type="xs:integer"/&gt;</pre>
TagName	xs:string	content = simple		FieldType GroupType MessageType RecordType	<pre>&lt;xs:element name="TagName" type="xs:string"/&gt;</pre>
Default Alignment Type/ Numeric	restriction of xs:string	content = simple isRef = 0		Default EmptyField ValueType Default PaddingType Default TrimType	<pre>&lt;xs:element name="Numeric" minOccurs="0"&gt;   &lt;xs:simpleType&gt;     &lt;xs:restriction base="xs:string"&gt;       &lt;xs:enumeration value="RIGHT"/&gt;       &lt;xs:enumeration value="LEFT"/&gt;     &lt;/xs:restriction&gt;   &lt;/xs:simpleType&gt; &lt;/xs:element&gt;</pre>
Default Alignment Type/ Alpha	restriction of xs:string	content = simple isRef = 0		Default EmptyField ValueType Default PaddingType Default TrimType	<pre>&lt;xs:element name="Alpha" minOccurs="0"&gt;   &lt;xs:simpleType&gt;     &lt;xs:restriction base="xs:string"&gt;       &lt;xs:enumeration value="RIGHT"/&gt;       &lt;xs:enumeration value="LEFT"/&gt;     &lt;/xs:restriction&gt;   &lt;/xs:simpleType&gt; &lt;/xs:element&gt;</pre>
Fields Type /Field	FieldType	content = complex isRef = 0	Name, TagName, DataType, Padding Char, Length, StartPos, Alignment, Default Value, Trim, Empty FieldValue, Identifier		<pre>&lt;xs:element name="Field" type="FieldType" maxOccurs="unbounded"/&gt;</pre>

Element	Type	Properties	Children	Used by	Source
FieldType/ Padding Char	xs:string	content = simple isRef = 0			<pre>&lt;xs:element name="PaddingChar" type="xs:string" minOccurs="0"/&gt;</pre>
FieldType/ Length	xs:integer	content = simple isRef = 0			<pre>&lt;xs:element name="Length" type="xs:integer"/&gt;</pre>
FieldType/ StartPos	xs:integer	content = simple isRef = 0			<pre>&lt;xs:element name="StartPos" type="xs:integer"/&gt;</pre>
FieldType/ Alignment	restriction of xs:string	content = simple isRef = 0			<pre>&lt;xs:element name="Alignment" minOccurs="0"&gt; &lt;xs:simpleType&gt; &lt;xs:restriction base="xs:string"&gt; &lt;xs:enumeration value="RIGHT"/&gt; &lt;xs:enumeration value="LEFT"/&gt; &lt;/xs:restriction&gt; &lt;/xs:simpleType&gt; &lt;/xs:element&gt;</pre>
FieldType/ Default Value	xs:string	content = simple isRef = 0			<pre>&lt;xs:element name="DefaultValue" type="xs:string" minOccurs="0"/&gt;</pre>
FieldType/ Trim	xs:integer	content = simple isRef = 0			<pre>&lt;xs:element name="Trim" type="xs:integer" minOccurs="0"/&gt;</pre>
FieldType/ Empty Field Value	xs:string	content = simple isRef = 0			<pre>&lt;xs:element name="EmptyFieldValue" type="xs:string" minOccurs="0"/&gt;</pre>
FieldType/ Identifier	xs:string	content = simple isRef = 0			<pre>&lt;xs:element name="Identifier" type="xs:string" minOccurs="0"/&gt;</pre>
FileType/ File	FileType	content = complex isRef = 0	ID Messages		<pre>&lt;xs:element name="File" type="FileType"/&gt;</pre>
FileType/ Messages	Messages Type	content = complex isRef = 0	Message		<pre>&lt;xs:element name="Messages" type="MessagesType"/&gt;</pre>

Element	Type	Properties	Children	Used by	Source
Group Type/ Group	Group Type	content = complex isRef = 0	Name, TagName, MinOccur, MaxOccur, Group, Records		<pre>&lt;xs:element name="Group" type="GroupType" minOccurs="0" maxOccurs="unbounded"/&gt;</pre>
Group Type/ Records	Records Type	content = complex isRef = 0	Record		<pre>&lt;xs:element name="Records" type="RecordsType" minOccurs="0"/&gt;</pre>
Messages Type/ Message	extension of Message Type	content = complex isRef = 0	Name, TagName, Default Alignment, Default Padding, Default Trim, Default Empty FieldValue, Escape Char, Record Separator, Field Separator, Records, Group		<pre>&lt;xs:element name="Message"&gt; &lt;xs:complexType&gt; &lt;xs:complexContent&gt; &lt;xs:extension base="MessageType"/&gt; &lt;/xs:complexContent&gt; &lt;/xs:complexType&gt; &lt;/xs:element&gt;</pre>
Message Type/ Default Alignment	Default Alignment Type	content = complex isRef = 0	Numeric, Alpha		<pre>&lt;xs:element name="DefaultAlignment" type="DefaultAlignmentType"/&gt;</pre>
Message Type/ Default Padding	Default Padding Type	content = complex isRef = 0	Numeric, Alpha		<pre>&lt;xs:element name="DefaultPadding" type="DefaultPaddingType"/&gt;</pre>
Message Type/ Default Trim	Default Trim Type	content = complex isRef = 0	Numeric, Alpha		<pre>&lt;xs:element name="DefaultTrim" type="DefaultTrimType"/&gt;</pre>



Element	Type	Properties	Children	Used by	Source
Message Type/ Default Empty Field Value	Default Empty Field Value Type	content = complex isRef = 0	Numeric, Alpha		<pre>&lt;xs:element name="DefaultEmptyFieldValue" type="DefaultEmptyFieldValueType"/&gt;</pre>
Message Type/ Escape Char	xs:string	content = simple isRef = 0			<pre>&lt;xs:element name="EscapeChar" type="xs:string" minOccurs="0"/&gt;</pre>
Message Type/ Record Separator	xs:string	content = simple isRef = 0			<pre>&lt;xs:element name="RecordSeparator" type="xs:string"/&gt;</pre>
Message Type/ Field Separator	xs:string	content = simple isRef = 0			<pre>&lt;xs:element name="FieldSeparator" type="xs:string" minOccurs="0"/&gt;</pre>
Message Type/ Records	Records Type	content = complex isRef = 0	Record		<pre>&lt;xs:element name="Records" type="RecordsType" minOccurs="0"/&gt;</pre>
Message Type/ Group	Group Type	content = complex isRef = 0	Name, TagName, MinOccur, MaxOccur, Group, Records		<pre>&lt;xs:element name="Group" type="GroupType" minOccurs="0" maxOccurs="unbounded"/&gt;</pre>
Records Type/ Record	Record Type	content = complex isRef = 0	Name, TagName, MinOccur, MaxOccur, Fields		<pre>&lt;xs:element name="Record" type="RecordType" maxOccurs="unbounded"/&gt;</pre>
Record Type/ Fields	Fields Type	content = complex isRef = 0	Field		<pre>&lt;xs:element name="Fields" type="FieldsType" minOccurs="0"/&gt;</pre>

## ComplexTypes in Flat File Definition

The following table lists the ComplexType components in the example flat file definition in XML. Details such as ComplexType, Children, Usage, and Source are also provided in the table.

Details	Source
ComplexType - <b>DefaultAlignmentType</b> Children - <b>Numeric, Alpha</b> Used by - <b>MessageType/DefaultAlignment</b>	<pre>&lt;xs:complexType name="DefaultAlignmentType"&gt;</pre>
ComplexType - <b>DefaultEmptyFieldValueType</b> Children - <b>Numeric, Alpha</b> Used by - <b>MessageType/DefaultEmptyFieldValue</b>	<pre>&lt;xs:complexType name="DefaultEmptyFieldValueType"&gt;</pre>
ComplexType - <b>DefaultPaddingType</b> Children - <b>Numeric, Alpha</b> Used by - <b>MessageType/DefaultPadding</b>	<pre>&lt;xs:complexType name="DefaultPaddingType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element ref="Numeric" minOccurs="0"/&gt;     &lt;xs:element ref="Alpha" minOccurs="0"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt;</pre>
ComplexType - <b>DefaultTrimType</b> Children - <b>Numeric, Alpha</b> Used by - <b>MessageType/DefaultTrim</b>	<pre>&lt;xs:complexType name="DefaultTrimType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element ref="Numeric" minOccurs="0"/&gt;     &lt;xs:element ref="Alpha" minOccurs="0"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt;</pre>
ComplexType - <b>FieldsType</b> Children - <b>Field</b> Used by - <b>RecordType/Fields</b>	<pre>&lt;xs:complexType name="FieldsType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element name="Field" type="FieldType"       maxOccurs="unbounded"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt;</pre>
ComplexType - <b>FileType</b> Children - <b>File</b> Used by - <b>BankRepository/Files</b>	<pre>&lt;xs:complexType name="FileType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element name="File" type="FileType"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt;</pre>
ComplexType - <b>FileType</b> Children - <b>ID Messages</b> Used by - <b>FileType/File</b>	<pre>&lt;xs:complexType name="FileType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element ref="ID"/&gt;     &lt;xs:element name="Messages"       type="MessagesType"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt;</pre>

Details	Source
<b>ComplexType - GroupType</b> Children - <b>Name, TagName, MinOccur, MaxOccur, Group, Records</b> Used by - <b>MessageType/Group, GroupType/Group</b>	<pre> &lt;xs:complexType name="GroupType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element ref="Name"/&gt;     &lt;xs:element ref="TagName"/&gt;     &lt;xs:element ref="MinOccur"/&gt;     &lt;xs:element ref="MaxOccur"/&gt;     &lt;xs:element name="Group" type="GroupType"       minOccurs="0" maxOccurs="unbounded"/&gt;     &lt;xs:element name="Records" type="RecordsType"       minOccurs="0"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt; </pre>
<b>ComplexType - MessageType</b> Children - <b>Message</b> Used by - <b>FileType/Messages</b>	<pre> &lt;xs:complexType name="MessageType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element name="Message"&gt;       &lt;xs:complexType&gt;         &lt;xs:complexContent&gt;           &lt;xs:extension base="MessageType"/&gt;         &lt;/xs:complexContent&gt;       &lt;/xs:complexType&gt;     &lt;/xs:element&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt; </pre>
<b>ComplexType - RecordsType</b> Children - <b>Record</b> Used by - <b>MessageType/Records, GroupType/Records</b>	<pre> &lt;xs:complexType name="RecordsType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element name="Record" type="RecordType"       maxOccurs="unbounded"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt; </pre>
<b>ComplexType - RecordType</b> Children - <b>Name, TagName, MinOccur, MaxOccur, Fields</b> Used by - <b>RecordsType/Record</b>	<pre> &lt;xs:complexType name="RecordType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element ref="Name"/&gt;     &lt;xs:element ref="TagName"/&gt;     &lt;xs:element ref="MinOccur"/&gt;     &lt;xs:element ref="MaxOccur"/&gt;     &lt;xs:element name="Fields" type="FieldsType"       minOccurs="0"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt; </pre>
<b>ComplexType - RecordsType</b> Children - <b>Record</b> Used by - <b>MessageType/Records, GroupType/Records</b>	<pre> &lt;xs:complexType name="RecordsType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element name="Record" type="RecordType"       maxOccurs="unbounded"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt; </pre>

Details	Source
<b>ComplexType - RecordType</b> Children - <b>Name, TagName, MinOccur, MaxOccur, Fields</b> Used by - <b>RecordsType/Record</b>	<pre>&lt;xs:complexType name="RecordType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element ref="Name"/&gt;     &lt;xs:element ref="TagName"/&gt;     &lt;xs:element ref="MinOccur"/&gt;     &lt;xs:element ref="MaxOccur"/&gt;     &lt;xs:element name="Fields" type="FieldsType"       minOccurs="0"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt;</pre>
<b>ComplexType - RecordsType</b> Children - <b>Record</b> Used by - <b>MessageType/Records, GroupType/Records</b>	<pre>&lt;xs:complexType name="RecordsType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element name="Record" type="RecordType"       maxOccurs="unbounded"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt;</pre>
<b>ComplexType - RecordType</b> Children - <b>Name, TagName, MinOccur, MaxOccur, Fields</b> Used by - <b>RecordsType/Record</b>	<pre>&lt;xs:complexType name="RecordType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element ref="Name"/&gt;     &lt;xs:element ref="TagName"/&gt;     &lt;xs:element ref="MinOccur"/&gt;     &lt;xs:element ref="MaxOccur"/&gt;     &lt;xs:element name="Fields" type="FieldsType"       minOccurs="0"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt;</pre>
<b>ComplexType - RecordsType</b> Children - <b>Record</b> Used by - <b>MessageType/Records, GroupType/Records</b>	<pre>&lt;xs:complexType name="RecordsType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element name="Record" type="RecordType"       maxOccurs="unbounded"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt;</pre>
<b>ComplexType - RecordType</b> Children - <b>Name, TagName, MinOccur, MaxOccur, Fields</b> Used by - <b>RecordsType/Record</b>	<pre>&lt;xs:complexType name="RecordType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element ref="Name"/&gt;     &lt;xs:element ref="TagName"/&gt;     &lt;xs:element ref="MinOccur"/&gt;     &lt;xs:element ref="MaxOccur"/&gt;     &lt;xs:element name="Fields" type="FieldsType"       minOccurs="0"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt;</pre>
<b>ComplexType - RecordsType</b> Children - <b>Record</b> Used by - <b>MessageType/Records, GroupType/Records</b>	<pre>&lt;xs:complexType name="RecordsType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element name="Record" type="RecordType"       maxOccurs="unbounded"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt;</pre>

Details	Source
<b>ComplexType - RecordsType</b> Children - <b>Record</b> Used by - <b>MessageType/Records, GroupType/Records</b>	<pre> &lt;xs:complexType name="RecordsType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element name="Record" type="RecordType"       maxOccurs="unbounded"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt; </pre>
<b>ComplexType - RecordType</b> Children - <b>Name, TagName, MinOccur, MaxOccur, Fields</b> Used by - <b>RecordsType/Record</b>	<pre> &lt;xs:complexType name="RecordType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element ref="Name"/&gt;     &lt;xs:element ref="TagName"/&gt;     &lt;xs:element ref="MinOccur"/&gt;     &lt;xs:element ref="MaxOccur"/&gt;     &lt;xs:element name="Fields" type="FieldsType"       minOccurs="0"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt; </pre>
<b>ComplexType - MessageType</b> Children - <b>Name, TagName, DefaultAlignment, DefaultPadding, DefaultTrim, DefaultEmptyFieldValue, EscapeChar, RecordSeparator, FieldSeparator, Records, Group</b> Used by - <b>Message</b>	<pre> &lt;xs:complexType name="MessageType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element ref="Name"/&gt;     &lt;xs:element ref="TagName"/&gt;     &lt;xs:element name="DefaultAlignment"       type="DefaultAlignmentType"/&gt;     &lt;xs:element name="DefaultPadding"       type="DefaultPaddingType"/&gt;     &lt;xs:element name="DefaultTrim"       type="DefaultTrimType"/&gt;     &lt;xs:element name="DefaultEmptyFieldValue"       type="DefaultEmptyFieldValueType"/&gt;     &lt;xs:element name="EscapeChar"       type="xs:string" minOccurs="0"/&gt;     &lt;xs:element name="RecordSeparator"       type="xs:string"/&gt;     &lt;xs:element name="FieldSeparator"       type="xs:string" minOccurs="0"/&gt;     &lt;xs:sequence&gt;       &lt;xs:element name="Records" type="RecordsType"         minOccurs="0"/&gt;       &lt;xs:element name="Group" type="GroupType"         minOccurs="0" maxOccurs="unbounded"/&gt;     &lt;/xs:sequence&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt; </pre>

Details	Source
<b>ComplexType - FieldType</b> <b>Children - Name, TagName, DataType, PaddingChar, Length, StartPos, Alignment, DefaultValue, Trim, EmptyFieldValue, Identifier</b> <b>Used by - FieldType/Field</b>	<pre>&lt;xs:complexType name="FieldType"&gt;   &lt;xs:sequence&gt;     &lt;xs:element ref="Name"/&gt;     &lt;xs:element ref="TagName"/&gt;     &lt;xs:element ref="DataType"/&gt;     &lt;xs:element name="PaddingChar"       type="xs:string" minOccurs="0"/&gt;     &lt;xs:element name="Length"       type="xs:integer"/&gt;     &lt;xs:element name="StartPos"       type="xs:integer"/&gt;     &lt;xs:element name="Alignment" minOccurs="0"&gt;       &lt;xs:simpleType&gt;         &lt;xs:restriction base="xs:string"&gt;           &lt;xs:enumeration value="RIGHT"/&gt;           &lt;xs:enumeration value="LEFT"/&gt;         &lt;/xs:restriction&gt;       &lt;/xs:simpleType&gt;     &lt;/xs:element&gt;     &lt;xs:element name="DefaultValue"       type="xs:string" minOccurs="0"/&gt;     &lt;xs:element name="Trim"       type="xs:integer" minOccurs="0"/&gt;     &lt;xs:element name="EmptyFieldValue"       type="xs:string" minOccurs="0"/&gt;     &lt;xs:element name="Identifier"       type="xs:string" minOccurs="0"/&gt;   &lt;/xs:sequence&gt; &lt;/xs:complexType&gt;</pre>

# Flat File Examples



This section details examples and exercises for reference to the available features of the Flat File Parser.



**Warning:** When uncertain about a feature, Lawson recommends that you create a test scenario similar to the given examples and verify that you have correctly interpreted the feature. However, if your required feature is not available, you can also use a small example and the desired output as basis of a feature request.

## Record-Separated and Field-Separated Flat File to XML

This example shows one of the most straightforward types of transformations to perform with a parser: a flat file with semicolon-separated fields and end-of-line-separated records.

Flat File Definition:

```
<BankRepository>
<Files>
<File>
<ID>SampleOne</ID>
<Messages>
<Message>
  <IgnorePrecedingFieldSeparator>0</IgnorePrecedingFieldSeparator>
  <DefaultEmptyFieldValue>
    <Numeric>0</Numeric>
    <Alpha></Alpha>
  </DefaultEmptyFieldValue>
  <DefaultAlignment>
    <Numeric>RIGHT</Numeric>
    <Alpha>LEFT</Alpha>
  </DefaultAlignment>
  <DefaultPadding>
    <Numeric>0</Numeric>
    <Alpha> </Alpha>
  </DefaultPadding>
  <DefaultTrim>
    <Numeric>0</Numeric>
    <Alpha>0</Alpha>
  </DefaultTrim>
  <Name>Sample</Name>
  <TagName>Sample</TagName>
```

```
<EscapeChar/>
<RecordSeparator>\n</RecordSeparator>
<FieldSeparator>;</FieldSeparator>
<Records>
  <Record>
    <Name>SampleRecord</Name>
    <TagName>SampleRecord</TagName>
    <MinOccur>1</MinOccur>
    <MaxOccur>0</MaxOccur>
    <Fields>
      <Field>
        <Name>SampleFieldOne</Name>
        <TagName>SampleFieldOne</TagName>
        <DataType>Alpha</DataType>
        <PaddingChar/>
        <DecimalLength/>
        <Alignment>LEFT</Alignment>
        <StartPos>0</StartPos>
        <Length>0</Length>
        <DefaultValue></DefaultValue>
        <Identifier>0</Identifier>
      </Field>
      <Field>
        <Name>SampleFieldTwo</Name>
        <TagName>SampleFieldTwo</TagName>
        <DataType>Alpha</DataType>
        <PaddingChar/>
        <DecimalLength/>
        <Alignment>LEFT</Alignment>
        <StartPos>0</StartPos>
        <Length>0</Length>
        <DefaultValue></DefaultValue>
        <Identifier>0</Identifier>
      </Field>
    </Fields>
  </Record>
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>
```

### Input File:

```
Record_One_Field_Data_One;Record_One_Field_Data_Two;
Record_Two_Field_Data_One;Record_Two_Field_Data_Two;
```

The expected result without the surrounding envelope will be:

```
<SampleMessage>
  <SampleRecord>
    <SampleFieldOne>Record_One_Field_Data_One</SampleFieldOne>
    <SampleFieldTwo>Record_One_Field_Data_Two</SampleFieldTwo>
  </SampleRecord>
  <SampleRecord>
    <SampleFieldOne>Record_Two_Field_Data_One</SampleFieldOne>
    <SampleFieldTwo>Record_Two_Field_Data_Two</SampleFieldTwo>
  </SampleRecord>
</SampleMessage>
```



# XML to Record-Separated and Field-Separated Flat File

This example is another type of transformations to perform with a parser. An XML file will be transformed into a flat file with semicolon-separated fields and end-of-line-separated records.

Flat File Definition:

```
<BankRepository>
<Files>
<File>
<ID>SampleTwo</ID>
<Messages>
<Message>
<IgnorePrecedingFieldSeparator>0</IgnorePrecedingFieldSeparator>
<DefaultEmptyFieldValue>
<Numeric>0</Numeric>
<Alpha></Alpha>
</DefaultEmptyFieldValue>
<DefaultAlignment>
<Numeric>RIGHT</Numeric>
<Alpha>LEFT</Alpha>
</DefaultAlignment>
<DefaultPadding>
<Numeric>0</Numeric>
<Alpha> </Alpha>
</DefaultPadding>
<DefaultTrim>
<Numeric>0</Numeric>
<Alpha>0</Alpha>
</DefaultTrim>
<Name>Sample</Name>
<TagName>Sample</TagName>
<EscapeChar/>
<RecordSeparator>\n</RecordSeparator>
<FieldSeparator>;</FieldSeparator>
<Records>
<Record>
<Name>SampleRecord</Name>
<TagName>SampleRecord</TagName>
<MinOccur>1</MinOccur>
<MaxOccur>0</MaxOccur>
<Fields>
<Field>
<Name>SampleFieldOne</Name>
<TagName>SampleFieldOne</TagName>
<DataType>Alpha</DataType>
<PaddingChar/>
<DecimalLength/>
<Alignment>LEFT</Alignment>
<StartPos>0</StartPos>
<Length>0</Length>
<DefaultValue></DefaultValue>
<Identifier>0</Identifier>
</Field>
<Field>
<Name>SampleFieldTwo</Name>
<TagName>SampleFieldTwo</TagName>
<DataType>Alpha</DataType>
<PaddingChar/>
<DecimalLength/>
<Alignment>LEFT</Alignment>
<StartPos>0</StartPos>
<Length>0</Length>
<DefaultValue></DefaultValue>
<Identifier>0</Identifier>
</Field>
</Fields>
</Record>
```

```
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>
```

Input file:

```
<SampleMessage>
  <SampleRecord>
    <SampleFieldOne>Record_One_Field_Data_One</SampleFieldOne>
    <SampleFieldTwo>Record_One_Field_Data_Two</SampleFieldTwo>
  </SampleRecord>
  <SampleRecord>
    <SampleFieldOne>Record_Two_Field_Data_One</SampleFieldOne>
    <SampleFieldTwo>Record_Two_Field_Data_Two</SampleFieldTwo>
  </SampleRecord>
</SampleMessage>
```

The expected result will be:

```
Record_One_Field_Data_One;Record_One_Field_Data_Two;
Record_Two_Field_Data_One;Record_Two_Field_Data_Two;
```

## Record-Separated Fixed-Length Fields Flat File to XML

This example illustrates a transformation for fields that are not separated by a sequence of characters. These fields must have defined length.

Flat File Definition:

```
<BankRepository>
  <Files>
    <File>
      <ID>SampleThree</ID>
      <Messages>
        <Message>
          <IgnorePrecedingFieldSeparator>0</IgnorePrecedingFieldSeparator>
          <DefaultEmptyFieldValue>
            <Numeric>0</Numeric>
            <Alpha></Alpha>
          </DefaultEmptyFieldValue>
          <DefaultAlignment>
            <Numeric>RIGHT</Numeric>
            <Alpha>LEFT</Alpha>
          </DefaultAlignment>
          <DefaultPadding>
            <Numeric>0</Numeric>
            <Alpha></Alpha>
          </DefaultPadding>
          <DefaultTrim>
            <Numeric>0</Numeric>
            <Alpha>0</Alpha>
          </DefaultTrim>
          <Name>Sample</Name>
          <TagName>Sample</TagName>
          <EscapeChar/>
          <RecordSeparator>\n</RecordSeparator>
          <FieldSeparator></FieldSeparator>
```

```

<Records>
  <Record>
    <Name>SampleRecord</Name>
    <TagName>SampleRecord</TagName>
    <MinOccur>1</MinOccur>
    <MaxOccur>0</MaxOccur>
    <Fields>
      <Field>
        <Name>SampleFieldOne</Name>
        <TagName>SampleFieldOne</TagName>
        <DataType>Alpha</DataType>
        <PaddingChar/>
        <DecimalLength/>
        <Alignment>LEFT</Alignment>
        <StartPos>1</StartPos>
        <Length>3</Length>
        <DefaultValue></DefaultValue>
        <Identifier>0</Identifier>
      </Field>
      <Field>
        <Name>SampleFieldTwo</Name>
        <TagName>SampleFieldTwo</TagName>
        <DataType>Alpha</DataType>
        <PaddingChar/>
        <DecimalLength/>
        <Alignment>LEFT</Alignment>
        <StartPos>4</StartPos>
        <Length>3</Length>
        <DefaultValue></DefaultValue>
        <Identifier>0</Identifier>
      </Field>
    </Fields>
  </Record>
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>

```

Input file:

```

AAABBB
CCDDDD

```

The expected result without the surrounding envelope will be:

```

<Sample>
  <SampleRecord>
    <SampleFieldOne>AAA</SampleFieldOne>
    <SampleFieldTwo>BBB</SampleFieldTwo>
  </SampleRecord>
  <SampleRecord>
    <SampleFieldOne>CCC</SampleFieldOne>
    <SampleFieldTwo>DDD</SampleFieldTwo>
  </SampleRecord>
</Sample>

```

# XML to Record-Separated Fixed-Length Fields Flat File

This example is another type of transformation to perform with a parser. An XML file will be transformed into a record-separated fixed-length fields flat file.

Input File:

```
<BankRepository>
<Files>
  <File>
    <ID>SampleFour</ID>
    <Messages>
      <Message>
        <IgnorePrecedingFieldSeparator>0</IgnorePrecedingFieldSeparator>
        <DefaultEmptyFieldValue>
          <Numeric>0</Numeric>
          <Alpha></Alpha>
        </DefaultEmptyFieldValue>
        <DefaultAlignment>
          <Numeric>RIGHT</Numeric>
          <Alpha>LEFT</Alpha>
        </DefaultAlignment>
        <DefaultPadding>
          <Numeric>0</Numeric>
          <Alpha></Alpha>
        </DefaultPadding>
        <DefaultTrim>
          <Numeric>0</Numeric>
          <Alpha>0</Alpha>
        </DefaultTrim>
      <Name>Sample</Name>
      <TagName>Sample</TagName>
      <EscapeChar/>
      <RecordSeparator>\n</RecordSeparator>
      <FieldSeparator></FieldSeparator>
    <Records>
      <Record>
        <Name>SampleRecord</Name>
        <TagName>SampleRecord</TagName>
        <MinOccur>1</MinOccur>
        <MaxOccur>0</MaxOccur>
        <Fields>
          <Field>
            <Name>SampleFieldOne</Name>
            <TagName>SampleFieldOne</TagName>
            <DataType>Alpha</DataType>
            <PaddingChar/>
            <DecimalLength/>
            <Alignment>LEFT</Alignment>
            <StartPos>1</StartPos>
            <Length>3</Length>
            <DefaultValue></DefaultValue>
            <Identifier>0</Identifier>
          </Field>
          <Field>
            <Name>SampleFieldTwo</Name>
            <TagName>SampleFieldTwo</TagName>
            <DataType>Alpha</DataType>
            <PaddingChar/>
            <DecimalLength/>
            <Alignment>LEFT</Alignment>
            <StartPos>4</StartPos>
            <Length>3</Length>
            <DefaultValue></DefaultValue>
            <Identifier>0</Identifier>
          </Field>
        </Fields>
      </Record>
    </Records>
  </File>
</Files>
</BankRepository>
```

```

</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>

```

Input file:

```

<SampleMessage>
  <SampleRecord>
    <SampleFieldOne>AAA_One</SampleFieldOne>
    <SampleFieldTwo>BBB</SampleFieldTwo>
  </SampleRecord>
  <SampleRecord>
    <SampleFieldOne>CCC</SampleFieldOne>
    <SampleFieldTwo>DDD</SampleFieldTwo>
  </SampleRecord>
</SampleMessage>

```

The expected result will be:

```

AAABBB
CCDDDD

```

## Fixed-Length Records and Fields Flat File to XML

When no record separators are available, the records must consist of all fixed length fields. The following example illustrates a transformation from fixed-length record and Field Flat File to XML.

Flat File Definition:

```

<BankRepository>
  <Files>
    <File>
      <ID>SampleFive</ID>
      <Messages>
        <Message>
          <IgnorePrecedingFieldSeparator>1</IgnorePrecedingFieldSeparator>
          <DefaultEmptyFieldValue>
            <Numeric>0</Numeric>
            <Alpha></Alpha>
          </DefaultEmptyFieldValue>
          <DefaultAlignment>
            <Numeric>RIGHT</Numeric>
            <Alpha>LEFT</Alpha>
          </DefaultAlignment>
          <DefaultPadding>
            <Numeric>0</Numeric>
            <Alpha></Alpha>
          </DefaultPadding>
          <DefaultTrim>
            <Numeric>1</Numeric>
            <Alpha>0</Alpha>
          </DefaultTrim>
          <Name>TopNode</Name>
          <TagName>TopNode</TagName>
          <EscapeChar/>
          <RecordSeparator></RecordSeparator>
          <FieldSeparator></FieldSeparator>
          <FixedLengthSeparator>1</FixedLengthSeparator>
        </Message>
      </Messages>
    </File>
  </Files>
</BankRepository>

```

```
<Name>RecordOne</Name>
<TagName>RecordOne</TagName>
<MinOccur>1</MinOccur>
<MaxOccur>0</MaxOccur>
<Fields>
  <Field>
    <Name>Field1</Name>
    <TagName>Field1</TagName>
    <DataType>Alpha</DataType>
    <PaddingChar/>
    <DecimalLength/>
    <Alignment>LEFT</Alignment>
    <StartPos>1</StartPos>
    <Length>1</Length>
    <DefaultValue></DefaultValue>
    <Identifier>0</Identifier>
  </Field>
  <Field>
    <Name>Field2</Name>
    <TagName>Field2</TagName>
    <DataType>Alpha</DataType>
    <PaddingChar/>
    <DecimalLength/>
    <Alignment>LEFT</Alignment>
    <StartPos>2</StartPos>
    <Length>1</Length>
    <DefaultValue></DefaultValue>
    <Identifier>0</Identifier>
  </Field>
</Fields>
</Record>
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>
```

Input file content:

```
abCDeFGh
```

The expected result without the surrounding envelope will be:

```
<TopNode>
  <RecordOne>
    <Field1>a</Field1>
    <Field2>b</Field2>
  </RecordOne>
  <RecordOne>
    <Field1>C</Field1>
    <Field2>D</Field2>
  </RecordOne>
  <RecordOne>
    <Field1>e</Field1>
    <Field2>F</Field2>
  </RecordOne>
  <RecordOne>
    <Field1>G</Field1>
    <Field2>h</Field2>
  </RecordOne>
</TopNode>
```

# XML to Fixed-Length Records and Fields

This example is another type of transformation to perform with a parser. An XML file is transformed into a flat file with fixed-length records and fields.

Input File:

```
<BankRepository>
<Files>
<File>
<ID>SampleSix</ID>
<Messages>
<Message>
  <IgnorePrecedingFieldSeparator>1</IgnorePrecedingFieldSeparator>
  <DefaultEmptyFieldValue>
    <Numeric>0</Numeric>
    <Alpha></Alpha>
  </DefaultEmptyFieldValue>
  <DefaultAlignment>
    <Numeric>RIGHT</Numeric>
    <Alpha>LEFT</Alpha>
  </DefaultAlignment>
  <DefaultPadding>
    <Numeric>0</Numeric>
    <Alpha> </Alpha>
  </DefaultPadding>
  <DefaultTrim>
    <Numeric>1</Numeric>
    <Alpha>0</Alpha>
  </DefaultTrim>
  <Name>TopNode</Name>
  <TagName>TopNode</TagName>
  <EscapeChar/>
  <RecordSeparator></RecordSeparator>
  <FieldSeparator></FieldSeparator>
  <FixedLengthSeparator>1</FixedLengthSeparator>
</Records>
<Record>
  <Name>RecordOne</Name>
  <TagName>RecordOne</TagName>
  <MinOccur>1</MinOccur>
  <MaxOccur>0</MaxOccur>
  <Fields>
  <Field>
    <Name>Field1</Name>
    <TagName>Field1</TagName>
    <DataType>Alpha</DataType>
    <PaddingChar/>
    <DecimalLength/>
    <Alignment>LEFT</Alignment>
    <StartPos>1</StartPos>
    <Length>1</Length>
    <DefaultValue></DefaultValue>
    <Identifier>0</Identifier>
  </Field>
  <Field>
    <Name>Field2</Name>
    <TagName>Field2</TagName>
    <DataType>Alpha</DataType>
    <PaddingChar/>
    <DecimalLength/>
    <Alignment>LEFT</Alignment>
    <StartPos>2</StartPos>
    <Length>1</Length>
    <DefaultValue></DefaultValue>
    <Identifier>0</Identifier>
  </Field>
  </Fields>
</Record>
</Records>
```

```
</Message>
</Messages>
</File>
</Files>
</BankRepository>
```

Input file:

```
<TopNode>
  <RecordOne>
    <Field1>a</Field1>
    <Field2>b</Field2>
  </RecordOne>
  <RecordOne>
    <Field1>C</Field1>
    <Field2>D</Field2>
  </RecordOne>
  <RecordOne>
    <Field1>e</Field1>
    <Field2>F</Field2>
  </RecordOne>
  <RecordOne>
    <Field1>G</Field1>
    <Field2>h</Field2>
  </RecordOne>
</TopNode>
```

The expected result will be:

```
abCDeFGh
```

## Default Empty Field Value

The following examples show transformations that use empty field values. When a field is empty, the value to write is defined using the default empty field value.

### Flat to XML with Empty Fields

This example shows an input flat file that is field separated with some empty fields. The generated XML output will be substituted with the default empty value.

Flat File Definition:

```
<BankRepository>
  <Files>
    <File>
      <ID>SampleSeven</ID>
      <Messages>
        <Message>
          <IgnorePrecedingFieldSeparator>0</IgnorePrecedingFieldSeparator>
          <DefaultEmptyFieldValue>
            <Numeric>0</Numeric>
            <Alpha>X</Alpha>
          </DefaultEmptyFieldValue>
          <DefaultAlignment>
            <Numeric>RIGHT</Numeric>
            <Alpha>LEFT</Alpha>
          </DefaultAlignment>
          <DefaultPadding>
            <Numeric>0</Numeric>
```



```

        <Alpha> </Alpha>
      </DefaultPadding>
    </DefaultTrim>
    <Numeric>1</Numeric>
    <Alpha></Alpha>
  </DefaultTrim>
<Name>TopNode</Name>
<TagName>TopNode</TagName>
<EscapeChar/>
<RecordSeparator>\n</RecordSeparator>
<FieldSeparator>:</FieldSeparator>
<Records>
  <Record>
    <Name>RecordOne</Name>
    <TagName>RecordOne</TagName>
    <MinOccur>1</MinOccur>
    <MaxOccur>0</MaxOccur>
    <Fields>
      <Field>
        <Name>Field1</Name>
        <TagName>Field1</TagName>
        <DataType>Alpha</DataType>
        <PaddingChar/>
        <DecimalLength/>
        <Alignment>LEFT</Alignment>
        <StartPos>0</StartPos>
        <Length>0</Length>
        <DefaultValue></DefaultValue>
        <Identifier>0</Identifier>
      </Field>
      <Field>
        <Name>Field2</Name>
        <TagName>Field2</TagName>
        <DataType>Alpha</DataType>
        <PaddingChar/>
        <DecimalLength/>
        <Alignment>LEFT</Alignment>
        <StartPos>0</StartPos>
        <Length>0</Length>
        <DefaultValue></DefaultValue>
        <Identifier>0</Identifier>
      </Field>
    </Fields>
  </Record>
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>

```

Input file:

```

a:b:
c::
::
:d:
e:f:

```

The expected result will be:

```

<TopNode>
  <RecordOne>
    <Field1>a</Field1>
    <Field2>b</Field2>
  </RecordOne>
  <RecordOne>
    <Field1>c</Field1>
    <Field2>X</Field2>
  </RecordOne>
  <RecordOne>
    <Field1>X</Field1>
  </RecordOne>

```

```
<Field2>X</Field2>
</RecordOne>
<RecordOne>
  <Field1>X</Field1>
  <Field2>d</Field2>
</RecordOne>
<RecordOne>
  <Field1>e</Field1>
  <Field2>f</Field2>
</RecordOne>
</TopNode>
```

## XML to Flat with Empty Fields

This example shows an input XML file with empty content for some fields. The generated flat file output will be substituted with the default empty value. When writing fixed-length flat files, the alignment and padding must be defined since the empty field value will be aligned and padded to fill the specified length.

Flat File Definition:

```
<BankRepository>
<Files>
<File>
  <ID>SampleSeven</ID>
  <Messages>
  <Message>
    <IgnorePrecedingFieldSeparator>0</IgnorePrecedingFieldSeparator>
    <DefaultEmptyFieldValue>
      <Numeric>0</Numeric>
      <Alpha>X</Alpha>
    </DefaultEmptyFieldValue>
    <DefaultAlignment>
      <Numeric>RIGHT</Numeric>
      <Alpha>LEFT</Alpha>
    </DefaultAlignment>
    <DefaultPadding>
      <Numeric>0</Numeric>
      <Alpha> </Alpha>
    </DefaultPadding>
    <DefaultTrim>
      <Numeric>1</Numeric>
      <Alpha></Alpha>
    </DefaultTrim>
  <Name>TopNode</Name>
  <TagName>TopNode</TagName>
  <EscapeChar/>
  <RecordSeparator>\n</RecordSeparator>
  <FieldSeparator>:</FieldSeparator>
  <Records>
  <Record>
    <Name>RecordOne</Name>
    <TagName>RecordOne</TagName>
    <MinOccur>1</MinOccur>
    <MaxOccur>0</MaxOccur>
    <Fields>
    <Field>
      <Name>Field1</Name>
      <TagName>Field1</TagName>
      <DataType>Alpha</DataType>
      <PaddingChar/>
      <DecimalLength/>
      <Alignment>LEFT</Alignment>
      <StartPos>0</StartPos>
      <Length>0</Length>
      <DefaultValue></DefaultValue>
      <Identifier>0</Identifier>
    </Field>
  </Fields>
  </Record>
</Records>
</Message>
</Files>
</BankRepository>
```

```

<Name>Field2</Name>
<TagName>Field2</TagName>
<DataType>Alpha</DataType>
<PaddingChar/>
<DecimalLength/>
<Alignment>LEFT</Alignment>
<StartPos>0</StartPos>
<Length>0</Length>
<DefaultValue></DefaultValue>
<Identifier>0</Identifier>
</Field>
</Fields>
</Record>
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>

```

Input File:

```

<Record>
  <FieldOne>a</FieldOne>
  <FieldTwo>b</FieldTwo>
</Record>
<Record>
  <FieldOne/>
  <FieldTwo/>
</Record>
<Record>
  <FieldOne>c</FieldOne>
  <FieldTwo/>
</Record>
<Record>
  <FieldOne/>
  <FieldTwo>d</FieldTwo>
</Record>
<Record>
  <FieldOne>e</FieldOne>
  <FieldTwo>f</FieldTwo>
</Record>

```

The expected result will be:

```

a:b:
X:X:
c:X:
X:d:
e:f:

```

## Padding and Alignment

When a field has fixed length and the data is shorter than the specified length, the data must be padded with additional characters for it to reach the specified length. The following examples show transformations that use padding and alignment.

## Flat File with Padding to XML

This example shows a transformation where the padding of the flat file and left alignment is removed in the generated XML output.

Flat File Definition:

```
<BankRepository>
<Files>
<File>
  <ID>SampleEight</ID>
  <Messages>
  <Message>
    <IgnorePrecedingFieldSeparator>1</IgnorePrecedingFieldSeparator>
    <DefaultEmptyFieldValue>
      <Numeric>0</Numeric>
      <Alpha></Alpha>
    </DefaultEmptyFieldValue>
    <DefaultAlignment>
      <Numeric>RIGHT</Numeric>
      <Alpha>LEFT</Alpha>
    </DefaultAlignment>
    <DefaultPadding>
      <Numeric>0</Numeric>
      <Alpha>X</Alpha>
    </DefaultPadding>
    <DefaultTrim>
      <Numeric>1</Numeric>
      <Alpha>1</Alpha>
    </DefaultTrim>
    <Name>TopNode</Name>
    <TagName>TopNode</TagName>
    <EscapeChar/>
    <RecordSeparator>\n</RecordSeparator>
    <FieldSeparator></FieldSeparator>
    <Records>
    <Record>
      <Name>RecordOne</Name>
      <TagName>RecordOne</TagName>
      <MinOccur>1</MinOccur>
      <MaxOccur>0</MaxOccur>
      <Fields>
      <Field>
        <Name>Field1</Name>
        <TagName>Field1</TagName>
        <DataType>Alpha</DataType>
        <StartPos>1</StartPos>
        <Length>3</Length>
        <DefaultValue></DefaultValue>
        <Identifier>0</Identifier>
      </Field>
      <Field>
        <Name>Field2</Name>
        <TagName>Field2</TagName>
        <DataType>Alpha</DataType>
        <StartPos>4</StartPos>
        <Length>3</Length>
        <DefaultValue></DefaultValue>
        <Identifier>0</Identifier>
      </Field>
      </Fields>
    </Record>
    </Records>
  </Message>
</Messages>
</File>
</Files>
</BankRepository>
```

Input file:

```
XXAXXB
XCXDXX
```

The expected result will be:

```
<TopNode>
<RecordOne>
  <Field1>XXA</Field1>
  <Field2>XXB</Field2>
</RecordOne>
<RecordOne>
  <Field1>XC</Field1>
  <Field2>D</Field2>
</RecordOne>
</TopNode>
```

## XML to Flat File with Padding

This example shows a transformation of an XML file to a flat file output with padding. Fields are padded when they are too short. Inverse of previous example.

Flat File Definition:

```
<BankRepository>
<Files>
<File>
  <ID>SampleEight</ID>
  <Messages>
  <Message>
    <IgnorePrecedingFieldSeparator>1</IgnorePrecedingFieldSeparator>
    <DefaultEmptyFieldValue>
      <Numeric>0</Numeric>
      <Alpha></Alpha>
    </DefaultEmptyFieldValue>
    <DefaultAlignment>
      <Numeric>RIGHT</Numeric>
      <Alpha>LEFT</Alpha>
    </DefaultAlignment>
    <DefaultPadding>
      <Numeric>0</Numeric>
      <Alpha>X</Alpha>
    </DefaultPadding>
    <DefaultTrim>
      <Numeric>1</Numeric>
      <Alpha>1</Alpha>
    </DefaultTrim>
    <Name>TopNode</Name>
    <TagName>TopNode</TagName>
    <EscapeChar/>
    <RecordSeparator>\n</RecordSeparator>
    <FieldSeparator></FieldSeparator>
  </Message>
  <Records>
  <Record>
    <Name>RecordOne</Name>
    <TagName>RecordOne</TagName>
    <MinOccur>1</MinOccur>
    <MaxOccur>0</MaxOccur>
    <Fields>
    <Field>
      <Name>Field1</Name>
      <TagName>Field1</TagName>
      <DataType>Alpha</DataType>
      <StartPos>1</StartPos>
      <Length>3</Length>
      <DefaultValue></DefaultValue>
      <Identifier>0</Identifier>
    </Field>
    <Field>
      <Name>Field2</Name>
```

```
<TagName>Field2</TagName>
<DataType>Alpha</DataType>
<StartPos>4</StartPos>
<Length>3</Length>
<DefaultValue></DefaultValue>
<Identifier>0</Identifier>
</Field>
</Fields>
</Record>
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>
```

### Input File:

```
<TopNode>
<RecordOne>
<Field1>XXA</Field1>
<Field2>XXB</Field2>
</RecordOne>
<RecordOne>
<Field1>XC</Field1>
<Field2>D</Field2>
</RecordOne>
</TopNode>
```

The expected result will be:

```
XXAXXB
XCXDXX
```

## Trim

This example shows XML to flat file transformation where white space surrounding the data in XML is removed. Trimming is not applicable in the flat file to XML transformation.

### Flat File Definition:

```
<BankRepository>
<Files>
<File>
<ID>SampleNine</ID>
<Messages>
<Message>
<IgnorePrecedingFieldSeparator>0</IgnorePrecedingFieldSeparator>
<DefaultEmptyFieldValue>
<Numeric></Numeric>
<Alpha></Alpha>
</DefaultEmptyFieldValue>
<DefaultAlignment>
<Numeric>RIGHT</Numeric>
<Alpha>LEFT</Alpha>
</DefaultAlignment>
<DefaultPadding>
<Numeric>0</Numeric>
<Alpha> </Alpha>
</DefaultPadding>
<DefaultTrim>
<Numeric>1</Numeric>
<Alpha>1</Alpha>
```

```

    </DefaultTrim>
    <Name>TopNode</Name>
    <TagName>TopNode</TagName>
    <EscapeChar/>
    <RecordSeparator>\n</RecordSeparator>
    <FieldSeparator>:</FieldSeparator>
    <Records>
    <Record>
    <Name>Record</Name>
    <TagName>Record</TagName>
    <MinOccur>1</MinOccur>
    <MaxOccur>0</MaxOccur>
    <Fields>
    <Field>
    <Name>FieldOne</Name>
    <TagName>FieldOne</TagName>
    <DataType>Alpha</DataType>
    <PaddingChar/>
    <DecimalLength/>
    <Alignment>LEFT</Alignment>
    <StartPos>0</StartPos>
    <Length>0</Length>
    <DefaultValue></DefaultValue>
    <Identifier>0</Identifier>
    </Field>
    <Field>
    <Name>FieldTwo</Name>
    <TagName>FieldTwo</TagName>
    <DataType>Alpha</DataType>
    <PaddingChar/>
    <DecimalLength/>
    <Alignment>LEFT</Alignment>
    <StartPos>0</StartPos>
    <Length>0</Length>
    <DefaultValue></DefaultValue>
    <Identifier>0</Identifier>
    </Field>
    </Fields>
    </Record>
    </Records>
    </Message>
    </Messages>
    </File>
    </Files>
    </BankRepository>

```

#### Input File:

```

<TopNode>
  <Record>
    <FieldOne> a </FieldOne>
    <FieldTwo> b </FieldTwo>
  </Record>
  <Record>
    <FieldOne> e</FieldOne>
    <FieldTwo>f </FieldTwo>
  </Record>
</TopNode>

```

The expected result will be:

```

a:b:
e:f:

```

## Escape Character

This example shows flat file to XML transformation where the effect of a field or record separator that occurs as data in the input flat file is negated using an escape character prefix. The escape sequence will be included in the data.

The following are typical issues encountered with escape sequences:

- Escapewithafixed length is not tested.
- If XML data contains a separator as data, the parser will not escape the separator and an incorrect flat file will be generated.

Flat File Definition:

```
<BankRepository>
<Files>
<File>
<ID>Sample Ten</ID>
<Messages>
<Message>
<IgnorePrecedingFieldSeparator>0</IgnorePrecedingFieldSeparator>
<DefaultEmptyFieldValue>
<Numeric></Numeric>
<Alpha></Alpha>
</DefaultEmptyFieldValue>
<DefaultAlignment>
<Numeric>RIGHT</Numeric>
<Alpha>LEFT</Alpha>
</DefaultAlignment>
<DefaultPadding>
<Numeric></Numeric>
<Alpha></Alpha>
</DefaultPadding>
<DefaultTrim>
<Numeric>0</Numeric>
<Alpha>0</Alpha>
</DefaultTrim>
<Name>TopNode</Name>
<TagName>TopNode</TagName>
<EscapeChar>++</EscapeChar>
<RecordSeparator>\n</RecordSeparator>
<FieldSeparator>:</FieldSeparator>
<Records>
<Record>
<Name>RecordOne</Name>
<TagName>RecordOne</TagName>
<MinOccur>1</MinOccur>
<MaxOccur>0</MaxOccur>
<Fields>
<Field>
<Name>Field1</Name>
<TagName>Field1</TagName>
<DataType>Alpha</DataType>
<PaddingChar/>
<DecimalLength/>
<Alignment>LEFT</Alignment>
<StartPos>0</StartPos>
<Length>0</Length>
<DefaultValue></DefaultValue>
<Identifier>0</Identifier>
</Field>
<Field>
<Name>Field2</Name>
<TagName>Field2</TagName>
<DataType>Alpha</DataType>
<PaddingChar/>
<DecimalLength/>
<Alignment>LEFT</Alignment>
```



```

    <StartPos>0</StartPos>
    <Length>0</Length>
    <DefaultValue></DefaultValue>
    <Identifier>0</Identifier>
  </Field>
</Fields>
</Record>
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>

```

Input File:

```

aa++:aa:bb++:bb:
cc++::dd++::
++:ee:++:ff:
g:h:
i:++
j:

```

The expected result will be:

```

<TopNode>
  <RecordOne>
    <Field1>aa:aa</Field1>
    <Field2>bb:bb</Field2>
  </RecordOne>
  <RecordOne>
    <Field1>cc:</Field1>
    <Field2>dd:</Field2>
  </RecordOne>
  <RecordOne>
    <Field1>:ee</Field1>
    <Field2>:ff</Field2>
  </RecordOne>
  <RecordOne>
    <Field1>g</Field1>
    <Field2>h</Field2>
  </RecordOne>
  <RecordOne>
    <Field1>i</Field1>
    <Field2>++j:</Field2>
  </RecordOne>
</TopNode>

```

## Ignore Preceding and Trailing Field Separator

The following examples show transformations that use ignore preceding and trailing field separator.

### Ignore Preceding

If this option is used, a record that starts with a field separator ignores this first field. Otherwise, considered an empty field.

Flat File Definition:

```
<BankRepository>
<Files>
<File>
<ID>SampleEleven</ID>
<Messages>
<Message>
<IgnorePrecedingFieldSeparator>1</IgnorePrecedingFieldSeparator>
<DefaultEmptyFieldValue>
<Numeric>0</Numeric>
<Alpha></Alpha>
</DefaultEmptyFieldValue>
<DefaultAlignment>
<Numeric>RIGHT</Numeric>
<Alpha>LEFT</Alpha>
</DefaultAlignment>
<DefaultPadding>
<Numeric>0</Numeric>
<Alpha> </Alpha>
</DefaultPadding>
<DefaultTrim>
<Numeric>1</Numeric>
<Alpha>0</Alpha>
</DefaultTrim>
<Name>TopNode</Name>
<TagName>TopNode</TagName>
<EscapeChar/>
<RecordSeparator>\n</RecordSeparator>
<FieldSeparator>:</FieldSeparator>
<Records>
<Record>
<Name>RecordOne</Name>
<TagName>RecordOne</TagName>
<MinOccur>1</MinOccur>
<MaxOccur>0</MaxOccur>
<Fields>
<Field>
<Name>Field1</Name>
<TagName>Field1</TagName>
<DataType>Alpha</DataType>
<PaddingChar/>
<DecimalLength/>
<Alignment>LEFT</Alignment>
<StartPos>0</StartPos>
<Length>0</Length>
<DefaultValue></DefaultValue>
<Identifier>0</Identifier>
</Field>
<Field>
<Name>Field2</Name>
<TagName>Field2</TagName>
<DataType>Alpha</DataType>
<PaddingChar/>
<DecimalLength/>
<Alignment>LEFT</Alignment>
<StartPos>0</StartPos>
<Length>0</Length>
<DefaultValue></DefaultValue>
<Identifier>0</Identifier>
</Field>
</Fields>
</Record>
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>
```

Input File:

Input without preceding field separator (This input is not used, only showed as a reference)

```
a:b:
c:d
```

Input with preceding field separator will be:

```
a:b:
c:d
```

## Ignore Trailing

If a field separator is the last field in a record, it is not mandatory. And, this is not a configurable behavior.

Flat File Definition:

```
<BankRepository>
<Files>
<File>
<ID>SampleEleven</ID>
<Messages>
<Message>
  <IgnorePrecedingFieldSeparator>1</IgnorePrecedingFieldSeparator>
  <DefaultEmptyFieldValue>
    <Numeric>0</Numeric>
    <Alpha></Alpha>
  </DefaultEmptyFieldValue>
  <DefaultAlignment>
    <Numeric>RIGHT</Numeric>
    <Alpha>LEFT</Alpha>
  </DefaultAlignment>
  <DefaultPadding>
    <Numeric>0</Numeric>
    <Alpha> </Alpha>
  </DefaultPadding>
  <DefaultTrim>
    <Numeric>1</Numeric>
    <Alpha>0</Alpha>
  </DefaultTrim>
  <Name>TopNode</Name>
  <TagName>TopNode</TagName>
  <EscapeChar/>
  <RecordSeparator>\n</RecordSeparator>
  <FieldSeparator>:</FieldSeparator>
  <Records>
  <Record>
    <Name>RecordOne</Name>
    <TagName>RecordOne</TagName>
    <MinOccur>1</MinOccur>
    <MaxOccur>0</MaxOccur>
    <Fields>
    <Field>
      <Name>Field1</Name>
      <TagName>Field1</TagName>
      <DataType>Alpha</DataType>
      <PaddingChar/>
      <DecimalLength/>
      <Alignment>LEFT</Alignment>
      <StartPos>0</StartPos>
      <Length>0</Length>
      <DefaultValue></DefaultValue>
      <Identifier>0</Identifier>
    </Field>
    <Field>
      <Name>Field2</Name>
      <TagName>Field2</TagName>
      <DataType>Alpha</DataType>
      <PaddingChar/>
      <DecimalLength/>
      <Alignment>LEFT</Alignment>
      <StartPos>0</StartPos>
```

```
<Length>0</Length>
<DefaultValue></DefaultValue>
<Identifier>0</Identifier>
</Field>
</Fields>
</Record>
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>
```

Input File:

Input without trailing field separator and with preceding separator:

```
:a:b
:c:d
```

## Record-Separated Semi-Fixed-Length Fields To and From XML

Using "-1" as the length of last field in a record means that the field is of undefined length and that the end of the record will be used to signal the end of the field.

The following are two transformation examples.

### Record-Separated Semi-Fixed Length Fields to XML

Example showing how a record-separated semi-fixed length fields in a flat file is transformed to XML.

Flat File Definition:

```
<BankRepository>
<Files>
<File>
<ID>SampleTwelve</ID>
<Messages>
<Message>
  <IgnorePrecedingFieldSeparator>1</IgnorePrecedingFieldSeparator>
  <DefaultEmptyFieldValue>
    <Numeric>0</Numeric>
    <Alpha></Alpha>
  </DefaultEmptyFieldValue>
  <DefaultAlignment>
    <Numeric>RIGHT</Numeric>
    <Alpha>LEFT</Alpha>
  </DefaultAlignment>
  <DefaultPadding>
    <Numeric>0</Numeric>
    <Alpha></Alpha>
  </DefaultPadding>
  <DefaultTrim>
    <Numeric>1</Numeric>
    <Alpha>0</Alpha>
  </DefaultTrim>
  <Name>TopNode</Name>
  <TagName>TopNode</TagName>
```

```

<EscapeChar/>
<RecordSeparator>CRLF</RecordSeparator>
<FieldSeparator></FieldSeparator>
<!--<FixedLengthSeparator>1</FixedLengthSeparator>-->
<Records>
  <Record>
    <Name>RecordOne</Name>
    <TagName>RecordOne</TagName>
    <MinOccur>1</MinOccur>
    <MaxOccur>0</MaxOccur>
    <Fields>
      <Field>
        <Name>Field1</Name>
        <TagName>Field1</TagName>
        <DataType>Alpha</DataType>
        <PaddingChar/>
        <DecimalLength/>
        <Alignment>LEFT</Alignment>
        <StartPos>1</StartPos>
        <Length>3</Length>
        <DefaultValue></DefaultValue>
        <Identifier>0</Identifier>
      </Field>
      <Field>
        <Name>Field2</Name>
        <TagName>Field2</TagName>
        <DataType>Alpha</DataType>
        <PaddingChar/>
        <DecimalLength/>
        <Alignment>LEFT</Alignment>
        <StartPos>4</StartPos>
        <Length>3</Length>
        <DefaultValue></DefaultValue>
        <Identifier>0</Identifier>
      </Field>
      <Field>
        <Name>Field3</Name>
        <TagName>Field3</TagName>
        <DataType>Alpha</DataType>
        <PaddingChar/>
        <DecimalLength/>
        <Alignment>LEFT</Alignment>
        <StartPos>7</StartPos>
        <Length>-1</Length>
        <DefaultValue></DefaultValue>
        <Identifier>0</Identifier>
      </Field>
    </Fields>
  </Record>
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>

```

The input with variable length last field is:

```

A B CDEFG
1 4 789

```

The expected result will be:

```

<TopNode>
  <RecordOne>
    <Field1>A</Field1>
    <Field2>B</Field2>
    <Field3>CDEFG</Field3>
  </RecordOne>
  <RecordOne>
    <Field1>1</Field1>
    <Field2>4</Field2>

```

```
<Field3>789</Field3>
</RecordOne>
</TopNode>
```

## XML to Record-Separated Semi-Fixed-Length Fields

This is the opposite of the previous example. The output and repositories in the previous example are used here to generate new sample output.

Flat File Definition:

```
<BankRepository>
<Files>
<File>
<ID>SampleTwelve</ID>
<Messages>
<Message>
<IgnorePrecedingFieldSeparator>1</IgnorePrecedingFieldSeparator>
<DefaultEmptyFieldValue>
<Numeric>0</Numeric>
<Alpha></Alpha>
</DefaultEmptyFieldValue>
<DefaultAlignment>
<Numeric>RIGHT</Numeric>
<Alpha>LEFT</Alpha>
</DefaultAlignment>
<DefaultPadding>
<Numeric>0</Numeric>
<Alpha> </Alpha>
</DefaultPadding>
<DefaultTrim>
<Numeric>1</Numeric>
<Alpha>0</Alpha>
</DefaultTrim>
<Name>TopNode</Name>
<TagName>TopNode</TagName>
<EscapeChar/>
<RecordSeparator>CRLF</RecordSeparator>
<FieldSeparator></FieldSeparator>
<!--<FixedLengthSeparator>1</FixedLengthSeparator>-->
<Records>
<Record>
<Name>RecordOne</Name>
<TagName>RecordOne</TagName>
<MinOccur>1</MinOccur>
<MaxOccur>0</MaxOccur>
<Fields>
<Field>
<Name>Field1</Name>
<TagName>Field1</TagName>
<DataType>Alpha</DataType>
<PaddingChar/>
<DecimalLength/>
<Alignment>LEFT</Alignment>
<StartPos>1</StartPos>
<Length>3</Length>
<DefaultValue></DefaultValue>
<Identifier>0</Identifier>
</Field>
<Field>
<Name>Field2</Name>
<TagName>Field2</TagName>
<DataType>Alpha</DataType>
<PaddingChar/>
<DecimalLength/>
<Alignment>LEFT</Alignment>
<StartPos>4</StartPos>
<Length>3</Length>
<DefaultValue></DefaultValue>
<Identifier>0</Identifier>
```

```

</Field>
<Field>
  <Name>Field3</Name>
  <TagName>Field3</TagName>
  <DataType>Alpha</DataType>
  <PaddingChar/>
  <DecimalLength/>
  <Alignment>LEFT</Alignment>
  <StartPos>7</StartPos>
  <Length>-1</Length>
  <DefaultValue></DefaultValue>
  <Identifier>0</Identifier>
</Field>
</Fields>
</Record>
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>

```

Input file:

```

<TopNode>
  <RecordOne>
    <Field1>A</Field1>
    <Field2>B</Field2>
    <Field3>CDEFG</Field3>
  </RecordOne>
  <RecordOne>
    <Field1>1</Field1>
    <Field2>4</Field2>
    <Field3>789</Field3>
  </RecordOne>
</TopNode>

```

The expected result will be:

```

A B CDEFG
1 4 789

```

## Variable Number of Occurrences, Groups, and Identifiers

The following sections show two transformation examples that use variable number of occurrences, groups, and identifiers. The number of occurrences of a record can be controlled. Groups are containers used to manage a group of records and other groups, often with a variable number of occurrences. To identify a record an identifier can be used.

### Example 13 A

This example shows how a field and record separated flat file with identifiers is transformed to XML.

Flat File Definition:

```
<BankRepository>
<Files>
  <File>
    <ID>SampleThirteenA</ID>
  <Messages>
    <Message>
      <IgnorePrecedingFieldSeparator>1</IgnorePrecedingFieldSeparator>
      <DefaultEmptyFieldValue><Numeric>0</Numeric><Alpha></Alpha></DefaultEmptyFieldValue>
      <DefaultAlignment><Numeric>RIGHT</Numeric><Alpha>LEFT</Alpha></DefaultAlignment>
      <DefaultPadding><Numeric>0</Numeric><Alpha> </Alpha></DefaultPadding>
      <DefaultTrim><Numeric>1</Numeric><Alpha>0</Alpha></DefaultTrim>
      <Name>Purchases</Name>
      <TagName>PurchasedItems</TagName>
      <EscapeChar/>
      <RecordSeparator>\n</RecordSeparator>
      <FieldSeparator>:</FieldSeparator>
      <Records>
        <Group>
          <Name>Purchase</Name>
          <TagName>Purchase</TagName>
          <MinOccur>0</MinOccur>
          <MaxOccur>U</MaxOccur>
          <Records>
            <Record>
              <Name>CustomerDetails</Name>
              <TagName>CustomerDetails</TagName>
              <MinOccur>1</MinOccur>
              <MaxOccur>1</MaxOccur>
              <Fields>
                <Field>
                  <Name>ID</Name>
                  <TagName>Identifier</TagName>
                  <DataType>Alpha</DataType>
                  <PaddingChar/>
                  <DecimalLength/>
                  <Alignment>LEFT</Alignment>
                  <StartPos>0</StartPos>
                  <Length>0</Length>
                  <DefaultValue>CUSTOMER</DefaultValue>
                  <Identifier>1</Identifier>
                </Field>
                <Field>
                  <Name>Name</Name>
                  <TagName>Name</TagName>
                  <DataType>Alpha</DataType>
                  <PaddingChar/>
                  <DecimalLength/>
                  <Alignment>LEFT</Alignment>
                  <StartPos>0</StartPos>
                  <Length>0</Length>
                  <DefaultValue></DefaultValue>
                  <Identifier>0</Identifier>
                </Field>
                <Field>
                  <Name>Number</Name>
                  <TagName>Number</TagName>
                  <DataType>Alpha</DataType>
                  <PaddingChar/>
                  <DecimalLength/>
                  <Alignment>LEFT</Alignment>
                  <StartPos>0</StartPos>
                  <Length>0</Length>
                  <DefaultValue></DefaultValue>
                  <Identifier>0</Identifier>
                </Field>
              </Fields>
            </Record>
          </Group>
          <Name>Payment</Name>
          <TagName>Payment</TagName>
          <MinOccur>1</MinOccur>
          <MaxOccur>1</MaxOccur>
          <Records>
            <Record>
```



```

<Name>PaymentDetails</Name>
<TagName>PaymentDetails</TagName>
<MinOccur>0</MinOccur>
<MaxOccur>1</MaxOccur>
<Fields>
  <Field>
    <Name>ID</Name>
    <TagName>Identifier</TagName>
    <DataType>Alpha</DataType>
    <PaddingChar/>
    <DecimalLength/>
    <Alignment>LEFT</Alignment>
    <StartPos>0</StartPos>
    <Length>0</Length>
    <DefaultValue>PAYMENT</DefaultValue>
    <Identifier>1</Identifier>
  </Field>
  <Field>
    <Name>ID</Name>
    <TagName>Identifier</TagName>
    <DataType>Alpha</DataType>
    <PaddingChar/>
    <DecimalLength/>
    <Alignment>LEFT</Alignment>
    <StartPos>0</StartPos>
    <Length>0</Length>
    <DefaultValue>TYPE</DefaultValue>
    <Identifier>1</Identifier>
  </Field>
  <Field>
    <Name>ID</Name>
    <TagName>Identifier</TagName>
    <DataType>Alpha</DataType>
    <PaddingChar/>
    <DecimalLength/>
    <Alignment>LEFT</Alignment>
    <StartPos>0</StartPos>
    <Length>0</Length>
    <DefaultValue>CARD</DefaultValue>
    <Identifier>1</Identifier>
  </Field>
  <Field>
    <Name>CardType</Name>
    <TagName>CardType</TagName>
    <DataType>Alpha</DataType>
    <PaddingChar/>
    <DecimalLength/>
    <Alignment>LEFT</Alignment>
    <StartPos>0</StartPos>
    <Length>0</Length>
    <DefaultValue></DefaultValue>
    <Identifier>0</Identifier>
  </Field>
  <Field>
    <Name>CardNumber</Name>
    <TagName>CardNumber</TagName>
    <DataType>Alpha</DataType>
    <PaddingChar/>
    <DecimalLength/>
    <Alignment>LEFT</Alignment>
    <StartPos>0</StartPos>
    <Length>0</Length>
    <DefaultValue></DefaultValue>
    <Identifier>0</Identifier>
  </Field>
</Fields>
</Record>
<Record>
  <Name>PaymentDetails</Name>
  <TagName>PaymentDetails</TagName>
  <MinOccur>0</MinOccur>
  <MaxOccur>1</MaxOccur>
  <Fields>
    <Field>

```

```
<Name>ID</Name>
<TagName>Identifier</TagName>
<DataType>Alpha</DataType>
<PaddingChar/>
<DecimalLength/>
<Alignment>LEFT</Alignment>
<StartPos>0</StartPos>
<Length>0</Length>
<DefaultValue>PAYMENT</DefaultValue>
<Identifier>1</Identifier>
</Field>
<Field>
  <Name>ID</Name>
  <TagName>Identifier</TagName>
  <DataType>Alpha</DataType>
  <PaddingChar/>
  <DecimalLength/>
  <Alignment>LEFT</Alignment>
  <StartPos>0</StartPos>
  <Length>0</Length>
  <DefaultValue>TYPE</DefaultValue>
  <Identifier>1</Identifier>
</Field>
<Field>
  <Name>ID</Name>
  <TagName>Identifier</TagName>
  <DataType>Alpha</DataType>
  <PaddingChar/>
  <DecimalLength/>
  <Alignment>LEFT</Alignment>
  <StartPos>0</StartPos>
  <Length>0</Length>
  <DefaultValue>CASH</DefaultValue>
  <Identifier>1</Identifier>
</Field>
</Fields>
</Record>
</Records>
</Group>
<Record>
  <Name>Date</Name>
  <TagName>Date</TagName>
  <MinOccur>1</MinOccur>
  <MaxOccur>1</MaxOccur>
  <Fields>
    <Field>
      <Name>ID</Name>
      <TagName>Identifier</TagName>
      <DataType>Alpha</DataType>
      <PaddingChar/>
      <DecimalLength/>
      <Alignment>LEFT</Alignment>
      <StartPos>0</StartPos>
      <Length>0</Length>
      <DefaultValue>DATE</DefaultValue>
      <Identifier>1</Identifier>
    </Field>
    <Field>
      <Name>Day</Name>
      <TagName>Day</TagName>
      <DataType>Alpha</DataType>
      <PaddingChar/>
      <DecimalLength/>
      <Alignment>LEFT</Alignment>
      <StartPos>0</StartPos>
      <Length>0</Length>
      <DefaultValue></DefaultValue>
      <Identifier>0</Identifier>
    </Field>
    <Field>
      <Name>Month</Name>
      <TagName>Month</TagName>
      <DataType>Alpha</DataType>
      <PaddingChar/>
```

```

        <DecimalLength/>
        <Alignment>LEFT</Alignment>
        <StartPos>0</StartPos>
        <Length>0</Length>
        <DefaultValue></DefaultValue>
        <Identifier>0</Identifier>
    </Field>
    <Field>
        <Name>Year</Name>
        <TagName>Year</TagName>
        <DataType>Alpha</DataType>
        <PaddingChar/>
        <DecimalLength/>
        <Alignment>LEFT</Alignment>
        <StartPos>0</StartPos>
        <Length>0</Length>
        <DefaultValue></DefaultValue>
        <Identifier>0</Identifier>
    </Field>
</Fields>
</Record>
</Records>
<Group>
    <Name>Items</Name>
    <TagName>Items</TagName>
    <MinOccur>1</MinOccur>
    <MaxOccur>1</MaxOccur>
    <Records>
        <Group>
            <Name>Item</Name>
            <TagName>Item</TagName>
            <MinOccur>1</MinOccur>
            <MaxOccur>U</MaxOccur>
            <Records>
                <Record>
                    <Name>ItemDetails</Name>
                    <TagName>ItemDetails</TagName>
                    <MinOccur>1</MinOccur>
                    <MaxOccur>1</MaxOccur>
                    <Fields>
                        <Field>
                            <Name>ID</Name>
                            <TagName>Identifier</TagName>
                            <DataType>Alpha</DataType>
                            <PaddingChar/>
                            <DecimalLength/>
                            <Alignment>LEFT</Alignment>
                            <StartPos>0</StartPos>
                            <Length>0</Length>
                            <DefaultValue>ITEM</DefaultValue>
                            <Identifier>1</Identifier>
                        </Field>
                        <Field>
                            <Name>Name</Name>
                            <TagName>Name</TagName>
                            <DataType>Alpha</DataType>
                            <PaddingChar/>
                            <DecimalLength/>
                            <Alignment>LEFT</Alignment>
                            <StartPos>0</StartPos>
                            <Length>0</Length>
                            <DefaultValue></DefaultValue>
                            <Identifier>0</Identifier>
                        </Field>
                        <Field>
                            <Name>Quantity</Name>
                            <TagName>Quantity</TagName>
                            <DataType>Alpha</DataType>
                            <PaddingChar/>
                            <DecimalLength/>
                            <Alignment>LEFT</Alignment>
                            <StartPos>0</StartPos>
                            <Length>0</Length>
                            <DefaultValue></DefaultValue>

```

```

        <Identifier>0</Identifier>
      </Field>
    </Field>
    <Name>Price</Name>
    <TagName>Price</TagName>
    <DataType>Alpha</DataType>
    <PaddingChar/>
    <DecimalLength/>
    <Alignment>LEFT</Alignment>
    <StartPos>0</StartPos>
    <Length>0</Length>
    <DefaultValue></DefaultValue>
    <Identifier>0</Identifier>
  </Field>
</Fields>
</Record>
</Records>
</Group>
</Records>
</Group>
</Group>
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>

```

Input file:

```

CUSTOMER:Jonas:0123456:
PAYMENT:TYPE:CARD:VISA:98765432:
DATE:09:10:2002:
ITEM:Hub cap:4:199.00:
ITEM:Tyre:4:679.00:
ITEM:Fuzzy dice:1:37.50:
CUSTOMER:Magnus:1234567:
PAYMENT:TYPE:CASH:
DATE:10:10:2002:
ITEM:SAAB Oil pump:1:1229.00:
ITEM:SAAB Oil filter:1:56.00:

```

The expected result will be:

```

<PurchasedItems>
  <Purchase>
    <CustomerDetails>
      <Identifier>CUSTOMER</Identifier>
      <Name>Jonas</Name>
      <Number>0123456</Number>
    </CustomerDetails>
    <Payment>
      <PaymentDetails>
        <Identifier>PAYMENT</Identifier>
        <Identifier>TYPE</Identifier>
        <Identifier>CARD</Identifier>
        <CardType>VISA</CardType>
        <CardNumber>98765432</CardNumber>
      </PaymentDetails>
    </Payment>
    <Date>
      <Identifier>DATE</Identifier>
      <Day>09</Day>
      <Month>10</Month>
      <Year>2002</Year>
    </Date>
    <Items>
      <Item>
        <ItemDetails>
          <Identifier>ITEM</Identifier>
          <Name>Hub cap</Name>
          <Quantity>4</Quantity>

```

```

    <Price>199.00</Price>
  </ItemDetails>
</Item>
<Item>
  <ItemDetails>
    <Identifier>ITEM</Identifier>
    <Name>Tyre</Name>
    <Quantity>4</Quantity>
    <Price>679.00</Price>
  </ItemDetails>
</Item>
<Item>
  <ItemDetails>
    <Identifier>ITEM</Identifier>
    <Name>Fuzzy dice</Name>
    <Quantity>1</Quantity>
    <Price>37.50</Price>
  </ItemDetails>
</Item>
</Items>
</Purchase>
<Purchase>
  <CustomerDetails>
    <Identifier>CUSTOMER</Identifier>
    <Name>Magnus</Name>
    <Number>1234567</Number>
  </CustomerDetails>
  <Payment>
    <PaymentDetails>
      <Identifier>PAYMENT</Identifier>
      <Identifier>TYPE</Identifier>
      <Identifier>CASH</Identifier>
    </PaymentDetails>
  </Payment>
  <Date>
    <Identifier>DATE</Identifier>
    <Day>10</Day>
    <Month>10</Month>
    <Year>2002</Year>
  </Date>
</Items>
<Item>
  <ItemDetails>
    <Identifier>ITEM</Identifier>
    <Name>SAAB Oil pump</Name>
    <Quantity>1</Quantity>
    <Price>1229.00</Price>
  </ItemDetails>
</Item>
<Item>
  <ItemDetails>
    <Identifier>ITEM</Identifier>
    <Name>SAAB Oil filter</Name>
    <Quantity>1</Quantity>
    <Price>56.00</Price></ItemDetails>
  </Item>
</Items>
</Purchase>
</PurchasedItems>

```

## Example 13 B

This example shows how a field and record separated flat file without identifiers is transformed to XML.

Link Flat File Definition:

```

<BankRepository>
  <Files>
    <File>
      <ID>SampleThirteenB</ID>
    </File>
  </Files>
  <Messages>

```

```

<Msg><Type>FlatFile</Type><RecordSeparator><DefaultTrim><Numeric>1</Numeric><Alpha>0</Alpha></DefaultTrim>
</Alpha></DefaultPadding><DefaultTrim><Numeric>1</Numeric><Alpha>0</Alpha></DefaultTrim>
  <Name>TopNode</Name>
  <TagName>TopNode</TagName>

  <EscapeChar/>
  <RecordSeparator>crLf</RecordSeparator>
  <FieldSeparator></FieldSeparator>
  <Records>
    <Group>
      <Name>People</Name>
      <TagName>People</TagName>
      <MinOccur>1</MinOccur>
      <MaxOccur>1</MaxOccur>
      <Records>
        <Group>
          <Name>Person</Name>
          <TagName>Person</TagName>
          <MinOccur>1</MinOccur>
          <MaxOccur>0</MaxOccur>
          <Records>
            <Record>
              <Name>PersonDetails</Name>
              <TagName>PersonDetails</TagName>
              <MinOccur>1</MinOccur>
              <MaxOccur>1</MaxOccur>
              <Fields>
                <Field>
                  <Name>FirstName</Name>
                  <TagName>FirstName</TagName>
                  <DataType>Alpha</DataType>
                  <PaddingChar/>
                  <DecimalLength/>
                  <Alignment>LEFT</Alignment>
                  <StartPos>1</StartPos>
                  <Length>10</Length>
                  <DefaultValue></DefaultValue>
                  <Identifier>0</Identifier>
                </Field>
                <Field>
                  <Name>LastName</Name>
                  <TagName>LastName</TagName>
                  <DataType>Alpha</DataType>
                  <PaddingChar/>
                  <DecimalLength/>
                  <Alignment>LEFT</Alignment>
                  <StartPos>11</StartPos>
                  <Length>10</Length>
                  <DefaultValue></DefaultValue>
                  <Identifier>0</Identifier>
                </Field>
                <Field>
                  <Name>Organization</Name>
                  <TagName>Organization</TagName>
                  <DataType>Alpha</DataType>
                  <PaddingChar/>
                  <DecimalLength/>
                  <Alignment>LEFT</Alignment>
                  <StartPos>21</StartPos>
                  <Length>10</Length>
                  <DefaultValue></DefaultValue>
                  <Identifier>0</Identifier>
                </Field>
                <Field>
                  <Name>E-Mail</Name>
                  <TagName>EMail</TagName>
                  <DataType>Alpha</DataType>
                  <PaddingChar/>
                  <DecimalLength/>
                  <Alignment>LEFT</Alignment>
                  <StartPos>31</StartPos>
                  <Length>40</Length>

```

```

    <DefaultValue></DefaultValue>
    <Identifier>0</Identifier>
  </Field>
</Fields>
</Record>
<Group>
  <Name>Address</Name>
  <TagName>Address</TagName>
  <MinOccur>0</MinOccur>
  <MaxOccur>1</MaxOccur>
  <Records>
    <Record>
      <Name>AddressLine</Name>
      <TagName>AddressLine</TagName>
      <MinOccur>1</MinOccur>
      <MaxOccur>0</MaxOccur>
      <Fields>
        <Field>
          <Name>Text</Name>
          <TagName>Text</TagName>
          <DataType>Alpha</DataType>
          <PaddingChar/>
          <DecimalLength/>
          <Alignment>LEFT</Alignment>
          <StartPos>0</StartPos>
          <Length>40</Length>
          <DefaultValue></DefaultValue>
          <Identifier>0</Identifier>
        </Field>
      </Fields>
    </Record>
  </Records>
</Group>
</Records>
</Group>
</Records>
</Group>
</Records>
</Message>
</Messages>
</File>
</Files>
</BankRepository>

```

The input is a field and record separated file without identifiers:

```

123456789012345678901234567890123456789012345678901234567890
Jonas Fügedi IRD jonas.fugedi@lawson.se
Vendevägen 89
181 82 Danderyd
Joakim Hemligt HEM joakim.hemligt@lawson.se
Vendevägen 89
Magnus Persson ICS magnus.persson@lawson.se
Vendevägen 89

```

The expected result will be:

```

<TopNode>
  <People>
    <Person>
      <PersonDetails>
        <FirstName>1234567890</FirstName>
        <LastName>1234567890</LastName>
        <Organization>1234567890</Organization>
        <EMail>1234567890123456789012345678901234567890</EMail>
      </PersonDetails>
    </Person>
    <Person>
      <PersonDetails>
        <FirstName>Jonas</FirstName>
        <LastName>Fügedi</LastName>
        <Organization>IRD</Organization>

```

```
<Email>jonas.fugedi@intentia.se</Email>
</PersonDetails>
<Address>
  <AddressLine>
    <Text>Vendevägen 89</Text>
  </AddressLine>
  <AddressLine>
    <Text>181 82 Danderyd</Text>
  </AddressLine>
</Address>
</Person>
<Person>
  <PersonDetails>
    <FirstName>Joakim</FirstName>
    <LastName>Hemligt</LastName>
    <Organization>HEM</Organization>
    <Email>joakim.hemligt@intentia.se</Email>
  </PersonDetails>
  <Address>
    <AddressLine>
      <Text>Vendevägen 89</Text>
    </AddressLine>
  </Address>
</Person>
<Person>
  <PersonDetails>
    <FirstName>Magnus</FirstName>
    <LastName>Persson</LastName>
    <Organization>ICS</Organization>
    <Email>magnus.persson@intentia.se</Email>
  </PersonDetails>
  <Address>
    <AddressLine>
      <Text>Vendevägen 89</Text>
    </AddressLine>
  </Address>
</Person>
</People>
</TopNode>
```