

# SAP Exchange Infrastructure 3.0 - Graphical Mapping Introduction and New Features

**William Li**

**SAP Labs, LLC**

# Agenda

## Concepts / Overview

## Test/Debugging Environment

## Standard Functions

- n Function Usage
- n Simple Function Examples
- n Node Function Examples

## Introduction to Most-Common Java Usage

## User-defined Functions

- n Usage

## Advanced User-Defined Functions

- n Usage
- n ResultList, Container Object, GlobalContainer Object, MappingTrace Object

## Message Mapping “Patterns”

- n Summarization
- n Sequence Number Generation
- n Duplicating Subtrees

## Multi-Mappings

## Mapping Templates

## Summary

# Agenda

## Concepts / Overview

### Test/Debugging Environment

### Standard Functions

- n Function Usage
- n Simple Function Examples
- n Node Function Examples

### Introduction to Most-Common Java Usage

### User-defined Functions

- n Usage

### Advanced User-Defined Functions

- n Usage
- n ResultList, Container Object, GlobalContainer Object, MappingTrace Object

### Message Mapping “Patterns”

- n Summarization
- n Sequence Number Generation
- n Duplicating Subtrees

### Multi-Mappings

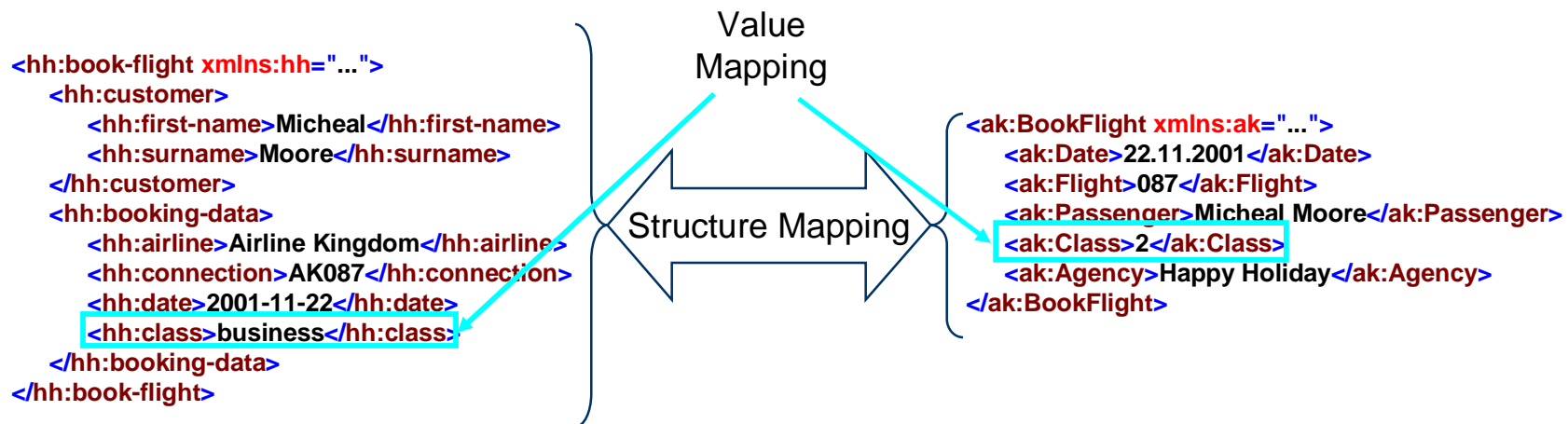
### Mapping Templates

### Summary

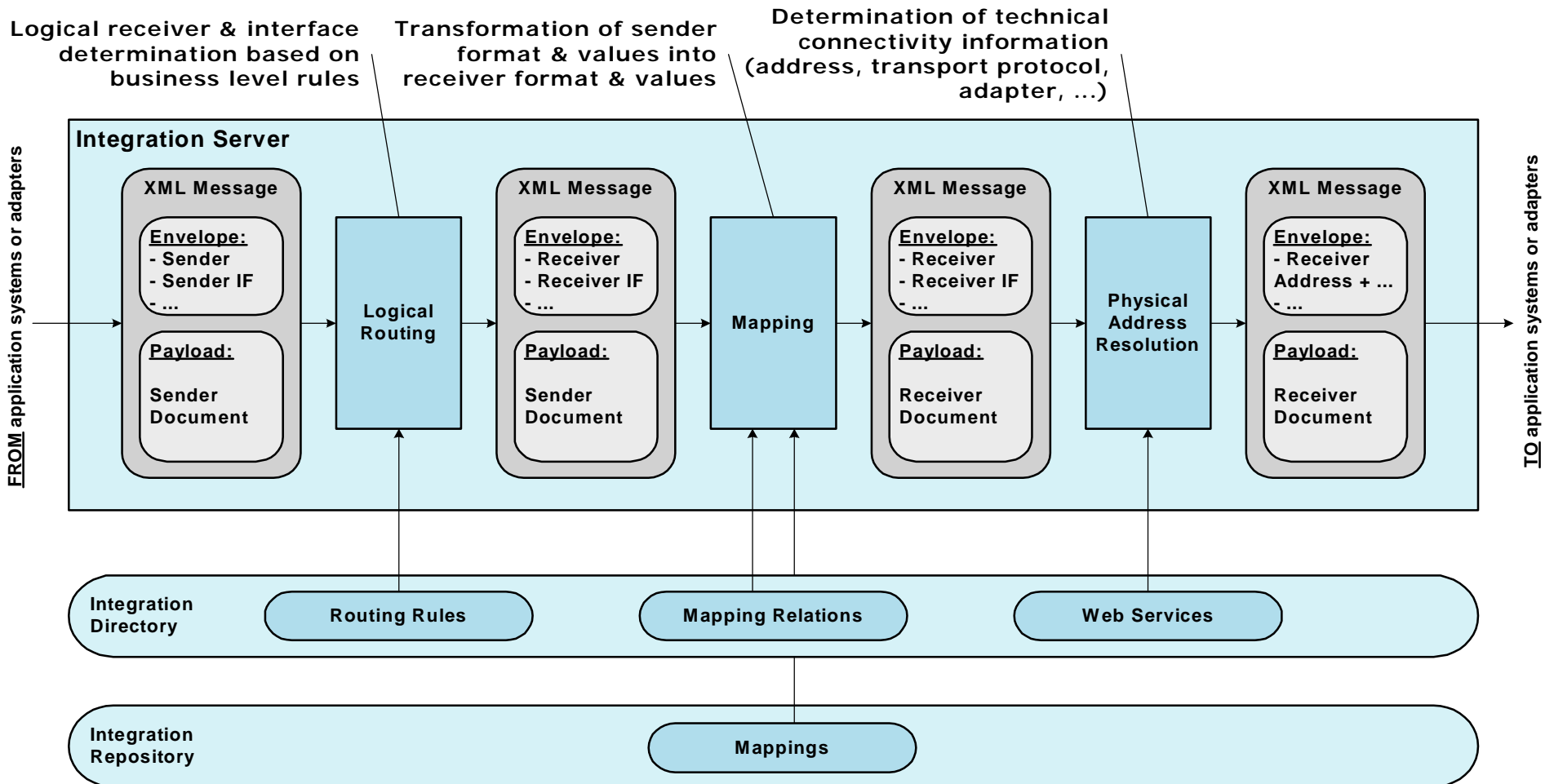
## Mapping

n Transformation from one message structure to another

n Transformation of one key value to another

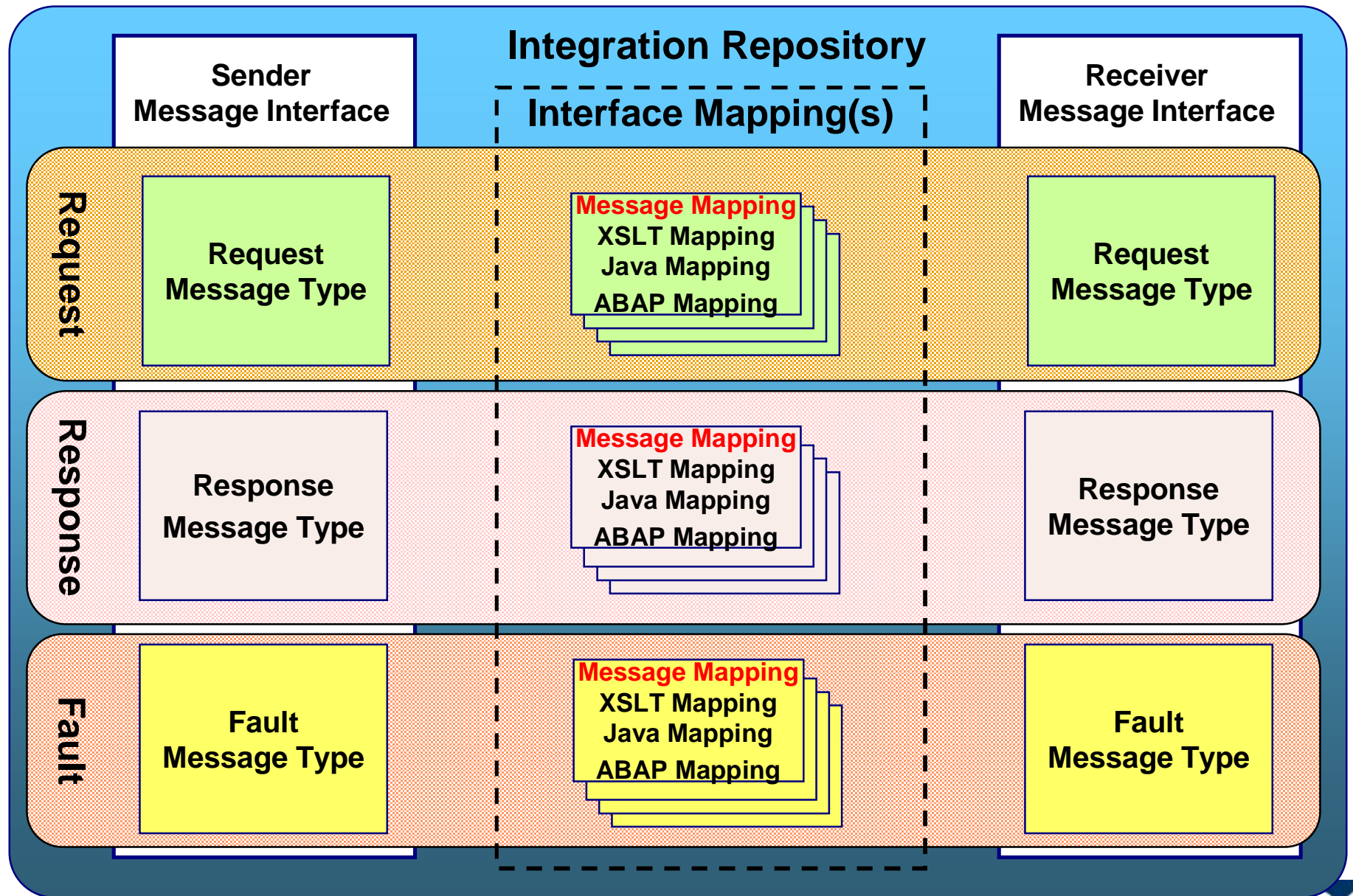


# Stateless Processing Model - Logical View



Technical activities (sending, receiving, queuing,...) omitted

# Concepts - Relate Mapping Programs to Interfaces



## Message Mapping

- n Designed by using the graphical mapping editor of Integration Builder
- n Result: Generated Java Code.
- n NOT memory-intensive, can “theoretically” handle GB-sized messages

## Imported Archives

- n Import externally defined mapping programs into repository
- n Java mapping:  
Implemented by using a specific interface
- n XSLT mapping:
  - u Runtime supports XSLT processor
  - u Java methods can be called from within a Style Sheet

## ABAP Workbench

- n ABAP Mappings in ABAP objects
- n XSLT Mappings (ABAP Engine)

**You can execute mapping programs in a sequence  
(Only for a request or a response message)**

## Use

- n Design structure mapping between any 2 XML structures
- n Connect to “value” mapping
- n Generates java source code and jar files to be used during runtime.
- n Can load the following into the structure overview:
  - u A schema from the Integration Repository
    - I Message Types
    - I External Definitions
    - I Imported Objects (RFCs and IDOCs)
  - u XML or XSD files from a local file (*import* and *include* are ignored in XSD)



## Standard Functions

n E.g. string processing, number/date formatting, boolean, ...

## Value Mapping




## Node functions

## User-defined functions

n Java method

# Icon Status Display

## Icon Types

Icon	Meaning
	Attribute
	Element
	Element with more than 1 occurrence

## Icon Colors

Color	Meaning
White	Optional attribute or element not assigned
Red	Attribute or element must be assigned to complete the mapping
Yellow	Attribute or element has already been assigned but the corresponding mapping in the data-flow editor is not complete
Green	Mapping to target field complete

# Overview - Mapping Editor

The screenshot displays the SAP Mapping Editor interface for the message mapping 'POReq\_ZBAPI\_PO\_CREATE\_req'. The interface is divided into several key sections:

- Structure Overview:** Located at the top left, it provides metadata for the message mapping, including its name, namespace, software component version, and description.
- Object Toolbar:** Located at the top center, it contains various icons for navigating and editing the mapping.
- Source Structure:** A table on the left side showing the structure of the source message. It includes columns for 'type' and 'Occurrence'.
- Target Structure:** A table on the right side showing the structure of the target message. It also includes columns for 'type' and 'Occurrence'.
- Data-Flow Editor:** Located at the bottom, it shows a visual representation of the data flow between the source and target structures. In this case, a data flow is shown from 'DocType' in the source to 'DOC\_TYPE' in the target.

**Source Structure Table:**

	type	Occurrence
PurchaseOrderRequest	PurchaseOrder...	1..1
bypass_bapi	xsd:string	0..1
Recordset		1..unbounded
header		1..1
item		1..unbounded

**Target Structure Table:**

	type	Occurrence
ZBAPI_PO_CREATE		1..1
BYPASS_BAPI	xsd:string	0..1
PO_HEADER	ZBAPI_PO_HEA...	1..1
DOC_DATE	date	0..1
DOC_TYPE	xsd:string	0..1
PURCH_ORG	xsd:string	0..1
PUR_GROUP	xsd:string	0..1

**Data-Flow Editor:**

```
graph LR; DocType[DocType] --> DOC_TYPE[DOC_TYPE]
```

# Overview - Assignment of Fields

The screenshot shows the 'Edit Message Mapping' interface for the mapping 'Customer\_to\_DEBCOR'. It features two data tables and a data-flow editor at the bottom.

**Customer Table:**

Field	type	Occurrences
customerid	xsd:string	1..1
name	xsd:string	1..1
address	xsd:string	1..1
city	xsd:string	1..1
state	xsd:string	1..1
zip	xsd:string	1..1
country	xsd:string	1..1

**DEBCOR01 Table:**

Field	type	Occurrences
BEGIN	xsd:string	required
EDI_DC40	EDI_DC40.DEB...	1..1
E1KUNNR	DEBCOR01.E1...	1..9999
SEGMENT	xsd:string	required
MSGFN	xsd:string	0..1
KUNNR	xsd:string	0..1
ANRED	xsd:string	0..1
KTOKD	xsd:string	0..1
LOEVM	xsd:string	0..1

**Data-Flow Editor:** Shows a mapping from 'customerid' to 'KUNNR'.

**Annotations:**

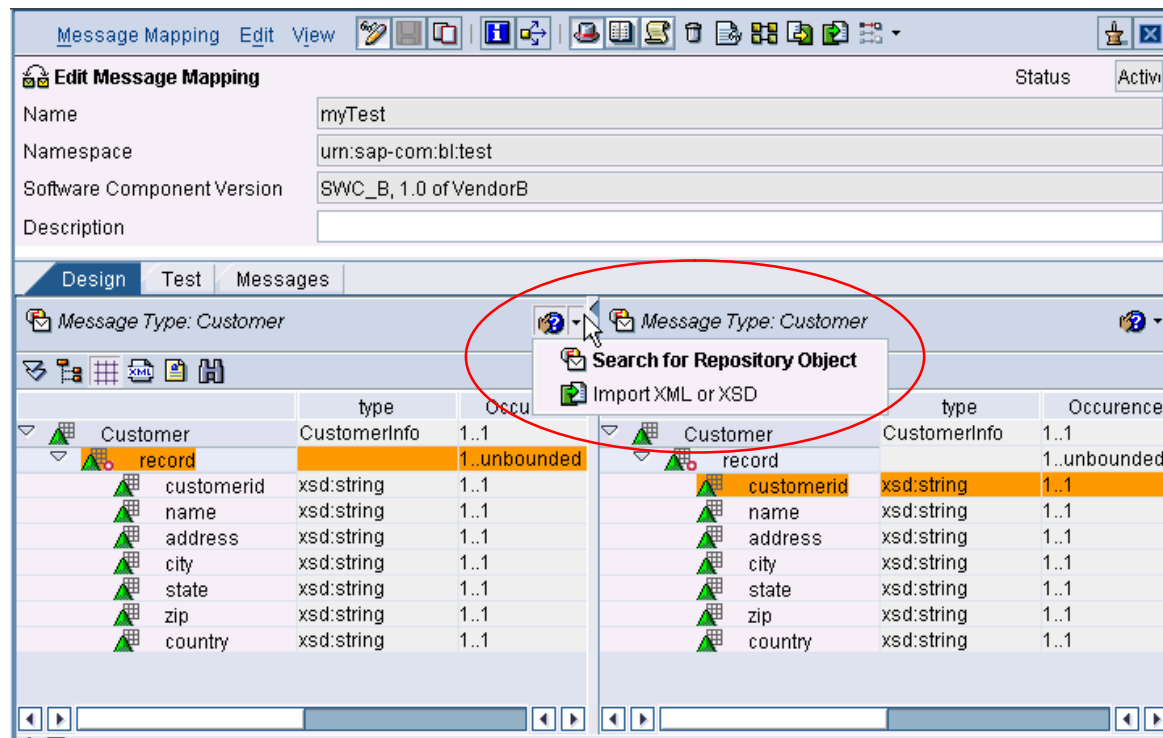
- Double click on an any field = insert field into data-flow editor** (points to 'customerid')
- Double click on a green/yellow field = navigate to target field mapping** (points to 'KUNNR')
- Double click on a white/red field = insert field into data-flow editor** (points to 'KUNNR')

**Actions:** 'drag&drop' labels indicate the movement of fields between the tables and into the data-flow editor.

# Overview – Assign Source/Target Message Types

There are 3 ways message types can be assigned to the Message Mapping.

1. By using an existing object already created in the Integration Repository (e.g. Message Type, External Definitions)
2. By using an RFC or IDoc meta data imported from SAP
3. By importing XML or XSD files from local file system





# Overview - Data-Flow Editor

The screenshot displays the SAP Data-Flow Editor interface for editing a message mapping. The main window shows two message types: 'Customer' and 'DEBCOR01'. The 'Customer' message type has fields like 'customerid', 'name', 'address', 'city', 'state', 'zip', and 'country'. The 'DEBCOR01' message type has fields like 'BEGIN', 'EDI\_DC40', 'E1KNA1C', 'SEGMENT', 'MSGFN', 'KUNNR', 'ANRED', 'KTOKD', and 'LOEVM'. A mapping is shown between 'customerid' and 'KUNNR'. The 'Functions' list at the bottom includes 'substring', 'concat', 'equalsS', 'indexOf', 'lastIndexOf', 'compare', and 'replaceString'. Callouts point to various features: 'Delete Floating Objects' (trash icon), 'Lay Out Mapping' (grid icon), 'Delete Current Mapping' (X icon), 'Define User Function' (plus icon), 'Select Function Category' (dropdown arrow), and 'Select Function' (function list).

Field	Type	Occurrences
customerid	xsd:string	1..1
name	xsd:string	1..1
address	xsd:string	1..1
city	xsd:string	1..1
state	xsd:string	1..1
zip	xsd:string	1..1
country	xsd:string	1..1
BEGIN	xsd:string	required
EDI_DC40	EDI_DC40.DEB...	1..1
E1KNA1C	DEBCOR01.E1...	1..9999
SEGMENT	xsd:string	required
MSGFN	xsd:string	0..1
KUNNR	xsd:string	0..1
ANRED	xsd:string	0..1
KTOKD	xsd:string	0..1
LOEVM	xsd:string	0..1

# Overview - Data-Flow Editor - Example

Message Mapping Navigation Edit View

**Edit Message Mapping** Status

Name: Customer\_to\_DEBCOR

Namespace: urn:sap-com:bl:test

Software Component Version: SWC\_B, 1.0 of VendorB

Description:

Design Test Messages


Message Type: Customer IDoc: DEBCOR.DEBCOR01

	type	Occu
Customer	CustomerInfo	1..1
record		1..unbo
customerid	xsd:string	1..1
<b>lastname</b>	<b>xsd:string</b>	<b>1..1</b>
firstname	xsd:string	1..1
address	xsd:string	1..1
city	xsd:string	1..1
state	xsd:string	1..1

	type	Oc
MSGFN	xsd:string	0..1
KUNNR	xsd:string	0..1
ANRED	xsd:string	0..1
KTOKD	xsd:string	0..1
LOEVM	xsd:string	0..1
<b>NAME1</b>	<b>xsd:string</b>	<b>0..1</b>
NAME2	xsd:string	0..1
ORT01	xsd:string	0..1

Functions: Text substring **concat** equalsS indexOf indexOf lastIndexOf lastIndexOf

# Overview - Text Preview

Message Mapping Navigation Edit View 

**Display Message Mapping** Text Preview status Active

Name: POReq\_\_ZBAPI\_PO\_CREATE\_req  
 Namespace: urn:xiworkshop:groupXX:webapp  
 Software Component Version: XI RIG US workshop XX, 1.0 of SAP XI RIG US  
 Description:

Design Test Messages

External Message: PurchaseOrderRequest RFC Message: ZBAPI\_PO\_CREATE

	type	Occurrence
PurchaseOrderRequest	PurchaseOrder...	1..1
bypass_bapi	xsd:string	0..1
Recordset		1..unboundec
header		1..1
item		1..unboundec

	type	Occurrences	De
ZBAPI_PO_CREATE		1..1	
BYPASS_BAPI	xsd:string	0..1	
PO_HEADER	ZBAPI_PO_HEA...	1..1	
DOC_DATE	date	0..1	
DOC_TYPE	xsd:string	0..1	

```

/ins0:ZBAPI_PO_CREATE/BYPASS_BAPI=/ns:PurchaseOrderRequest/bypass_bapi
/ins0:ZBAPI_PO_CREATE/PO_HEADER=/ns:PurchaseOrderRequest/Recordset/header
/ins0:ZBAPI_PO_CREATE/PO_HEADER/DOC_DATE=/ns:PurchaseOrderRequest/Recordset/header/DocDate
/ins0:ZBAPI_PO_CREATE/PO_HEADER/DOC_TYPE=/ns:PurchaseOrderRequest/Recordset/header/DocType
/ins0:ZBAPI_PO_CREATE/PO_HEADER/PURCH_ORG=/ns:PurchaseOrderRequest/Recordset/header/PurchOrg
/ins0:ZBAPI_PO_CREATE/PO_HEADER/PUR_GROUP=/ns:PurchaseOrderRequest/Recordset/header/PurGroup
/ins0:ZBAPI_PO_CREATE/PO_HEADER/VENDOR=/ns:PurchaseOrderRequest/Recordset/header/Vendor
/ins0:ZBAPI_PO_CREATE/PO_ITEMS/item=/ns:PurchaseOrderRequest/Recordset/item
/ins0:ZBAPI_PO_CREATE/PO_ITEMS/item/PO_ITEM=/ns:PurchaseOrderRequest/Recordset/item/ItemNo
/ins0:ZBAPI_PO_CREATE/PO_ITEMS/item/PUR_MAT=/ns:PurchaseOrderRequest/Recordset/item/Material
/ins0:ZBAPI_PO_CREATE/PO_ITEMS/item/PLANT=/ns:PurchaseOrderRequest/Recordset/item/Plant
/ins0:ZBAPI_PO_CREATE/PO_ITEMS/item/DELIV_DATE=/ns:PurchaseOrderRequest/Recordset/item/DelivDate
/ins0:ZBAPI_PO_CREATE/PO_ITEMS/item/QUANTITY=/ns:PurchaseOrderRequest/Recordset/item/Quantity
  
```

**All target field mappings**



# Overview - Dependencies

The screenshot displays the SAP Message Mapping interface. At the top, the 'Where-Used List' menu is circled in red, with a callout box labeled 'Dependencies'. Below this, the 'Display Message Mapping' section shows details for the mapping 'ContactToCustomer' in the namespace 'http://sap.com/xi/tutorial/mapping'. The main area is split into two panels: 'Message Type: Contact' on the left and 'Message Type: Customer' on the right. Both panels show a tree view of nodes and a table with columns for 'type', 'Occurrences', and 'Description'. Red lines connect the 'Name' node in the Contact tree to the 'CustomerName' node in the Customer tree. A callout box points to these connections with the text: 'All mappings or mappings of two selected subnodes are displayed'. The bottom of the interface shows a 'Functions' list including 'Text', 'substring', 'concat', 'equalsS', 'indexOf', 'lastIndexOf', 'compare', 'replaceString', 'length', and 'endsWith'.

Node	type	Occurrences	Description
Contact	Contact	1..1	
RefNo	xsd:string	1..1	
Name	Name	1..1	
Title	xsd:string	1..1	
Fistname	xsd:string	1..1	
Surname	xsd:string	1..1	
Address	Address	1..1	
@Type	xsd:string	optional	
Street	xsd:string	1..1	
ZIPCode	xsd:string	1..1	
City	xsd:string	1..1	

Node	type	Occurrences	Description
Customer	Customer	1..1	
CustomerID	xsd:string	1..1	
CustomerName	CustomerName	1..1	
Fullname	xsd:string	1..1	
Surname	xsd:string	1..1	
CustomerAddress	Address	1..1	
@Type	xsd:string	optional	
Street	xsd:string	1..1	
ZIPCode	xsd:string	1..1	
City	xsd:string	1..1	

# Agenda

Concepts / Overview

**Test/Debugging Environment**

Standard Functions

- n Function Usage
- n Simple Function Examples
- n Node Function Examples

Introduction to Most-Common Java Usage

User-defined Functions

- n Usage

Advanced User-Defined Functions

- n Usage
- n ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”

- n Summarization
- n Duplicating Subtrees
- n Table/Value Lookup (within source, not external)
- n Tree-Reversal

Multi-Mappings

Mapping Templates

Summary

# Test/Debugging Environment

A completed Message Mapping can be tested by using the “Test” tab.

The screenshot shows the SAP Test/Debugging Environment interface. The 'Test' tab is active. The interface includes a toolbar with icons for loading XML, executing mapping, and saving test cases. A search box is present for finding nodes in the tabular tree view. The main area displays a message mapping for 'Customer' with a 'Result' column. Callouts provide the following information:

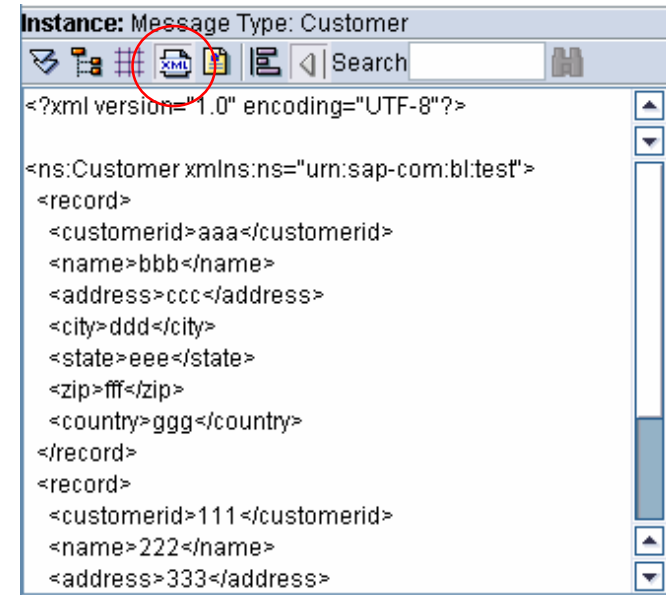
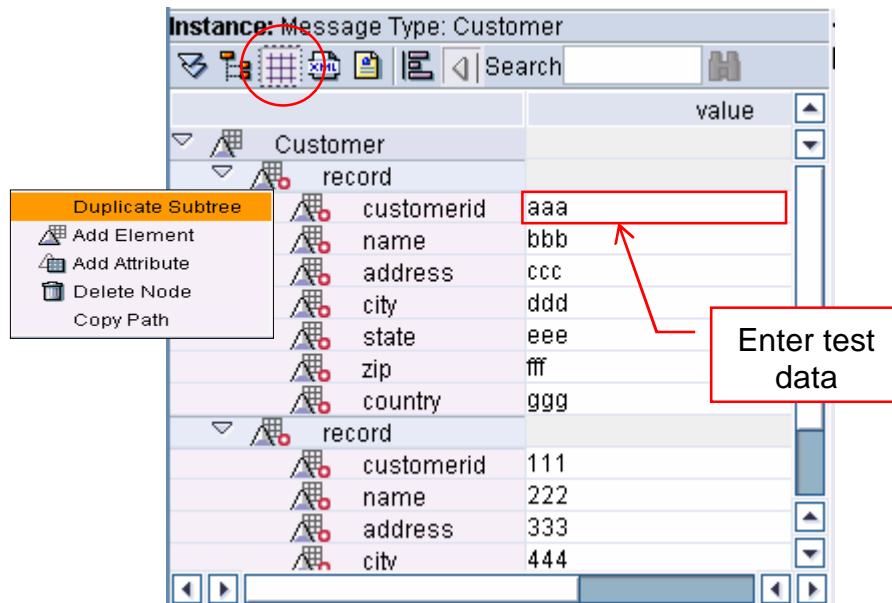
- Loads an XML instance from a local file.** (Points to the XML file icon in the toolbar)
- Search node name in the Tabular Tree View** (Points to the search box in the toolbar)
- Execute Mapping** (Points to the execute icon in the toolbar)
- Tabular Tree View** (Points to the grid icon in the toolbar)
- XML document view** (Points to the XML icon in the toolbar)
- Dropdown provides the creation and saving of test cases** (Points to the dropdown menu in the toolbar)

## Prerequisites

1. The defined message mapping must be *complete*. This means that each mandatory target field must be assigned to one or more source fields.
2. There must not be any unassigned arguments when using functions.

# Test/Debugging Environment

Editing the XML instances to use in test cases:



Using the context menu in the *(Tabular) Tree View*, you can copy sub-trees, delete nodes, and add elements and attributes. In the *Value* column in the tabular tree view you can also enter values for fields.

Using the editor for the XML view, you can manually edit elements and attributes or their values here. Use **CTRL C**, **CTRL X**, and **CTRL V** respectively to copy, cut, and paste parts of an XML instance.

# Test/Debugging Environment

Using the “Display Queue” to examine each step of the mapping:

The screenshot shows the SAP Message Mapping Editor interface. At the top, the 'Edit Message Mapping' window displays metadata for 'myTest' with namespace 'urn:sap-com:bl:test' and version 'SWC\_B, 1.0 of VendorB'. Below this, the 'Design' view shows a message flow. A 'concat' function is applied to the 'customerid' field. Two 'Display Queue' windows are open, showing the data at different stages: the first shows '[aaa]' and '[111]', and the second shows the concatenated result '[ABC-aaa]' and '[ABC-111]'. A context menu is open over the 'concat' function, with 'Display Queue' selected. Another context menu is open over the 'customerid' field, also with 'Display Queue' selected. Red arrows point from the 'Display Queue' windows to the corresponding 'customerid' field and 'concat' function in the design view.

type	Occurrences
CustomerInfo	1..1
customerid	1..1
name	xsd:string 1..1
address	xsd:string 1..1
city	xsd:string 1..1
state	xsd:string 1..1
country	xsd:string 1..1

**Note:**  
The “context” display and usage in the queues will be discussed in later slides.

# Agenda

Concepts / Overview

Test/Debugging Environment

**Standard Functions**

- n **Function Usage**
- n **Simple Function Examples**
- n **Node Function Examples**

Introduction to Most-Common Java Usage

User-defined Functions

- n Usage

Advanced User-Defined Functions

- n Usage
- n ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”

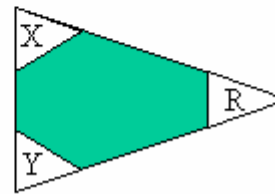
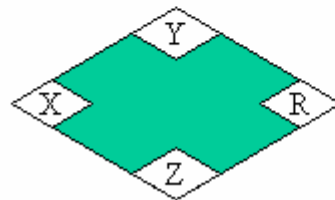
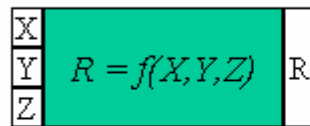
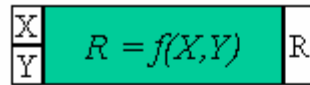
- n Summarization
- n Duplicating Subtrees
- n Table/Value Lookup (within source, not external)
- n Tree-Reversal

Multi-Mappings

Mapping Templates

Summary

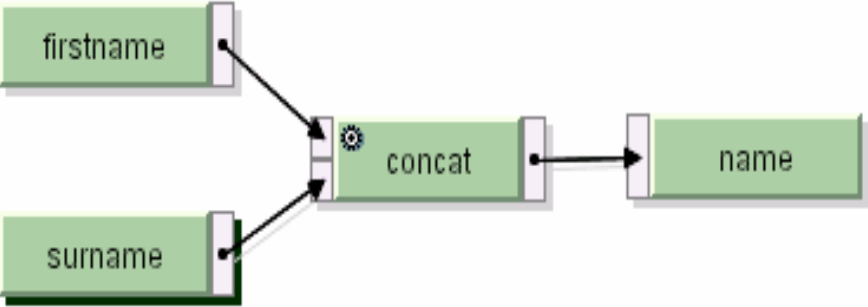
# Standard Functions: Introduction



- n Most data-flow objects have two or three inbound channels on the left-hand side, and one outbound channel on the right-hand side.
- n If functions are shaped as rhombuses or triangles.
- n There are also conversion functions with one inbound and outbound channel and functions that get a value without making any entries.
- n Standard functions that require additional specifications are indicated with a star (⊛). To call the dialog for *Function Properties*, double click the data-flow object.

# Simple Standard Functions: concat

## Function category: Text

Source Message	Mapping	Target Message
<pre>&lt;?xml version="1.0" encoding="UTF-8" ?&gt;  &lt;Customer&gt;   &lt;firstname&gt;     Harry   &lt;/firstname&gt;   &lt;surname&gt;     Potter   &lt;/surname&gt; &lt;/Customer&gt;</pre>	 <p><b><u>concat properties</u></b> Delimiter=" "</p>	<pre>&lt;?xml version="1.0" encoding="UTF-8" ?&gt;  &lt;Contact&gt;   &lt;name&gt;     Harry Potter   &lt;/name&gt; &lt;/Contact&gt;</pre>



# Simple Standard Functions: substring


## Function category: Text

Source Message	Mapping	Target Message
<pre>&lt;?xml version="1.0" encoding="UTF-8" ?&gt;  &lt;Customer&gt;   &lt;customerID&gt;     0123456789-181170   &lt;/customerID&gt; &lt;/Customer&gt;</pre>	<p><b>substring properties</b> start from: 0 count: 10</p>	<pre>&lt;?xml version="1.0" encoding="UTF-8" ?&gt;  &lt;Contact&gt;   &lt;customerNo&gt;     0123456789   &lt;/customerNo&gt; &lt;/Contact&gt;</pre>

**Position count for all text functions start at 0!**

# Simple Standard Functions: DateTrans

## Function category: Date

Source Message	Mapping	Target Message
<pre>&lt;?xml version="1.0" encoding="UTF-8" ?&gt;  &lt;Customer&gt;   &lt;custBirthday&gt;     181170   &lt;/custBirthday&gt; &lt;/Customer&gt;</pre>	 <p><b><u>DateTrans properties</u></b> src Format: ddMMyy dst Format: MM/dd/yyyy</p>	<pre>&lt;?xml version="1.0" encoding="UTF-8" ?&gt;  &lt;Contact&gt;   &lt;birthday&gt;     11/18/1970   &lt;/birthday&gt; &lt;/Contact&gt;</pre>

- uses standard Java date template

# Function Categories

**Arithmetic:** add, subtract, equalsA, abs, sqrt, sign, sqrt, etc.

**Boolean:** And, Or, Not, Equals, notEquals, if, ifWithoutElse

**Constants:** Constant, CopyValue, sender, receiver

**Conversions:** FixValues, Value mapping

**Date:** currentDate, DateTrans, DateBefore, DateAfter, CompareDate

**Node Functions:** createlf, removeContexts, replaceValue, exists, SplitByValue, collapseContexts

**Statistic:** sum, average, count

**Text:** substring, concat, equalsS, indexOf, lastIndexOf, compare, etc.

(The functions marked in blue will be explored in detail.)

# Constant Function – CopyValue

Use CopyValue() to copy a repeating value's position in the source structure and assign it to a target field.

Example:

The screenshot shows the SAP mapping tool interface. On the left, the source structure 'PartnerMsg' is expanded to show a repeating 'addrDat' field (type xsd:string, occurrence 3..3). On the right, the target structure 'CustomerMsg' is expanded to show a 'street' field (type xsd:string, occurrence 1..1). A 'CopyValue' function is connected between 'addrDat' and 'street'. A 'CopyValue properties' dialog box is open, showing 'Position' set to 0 and 'Notation' set to 'Any string value'.

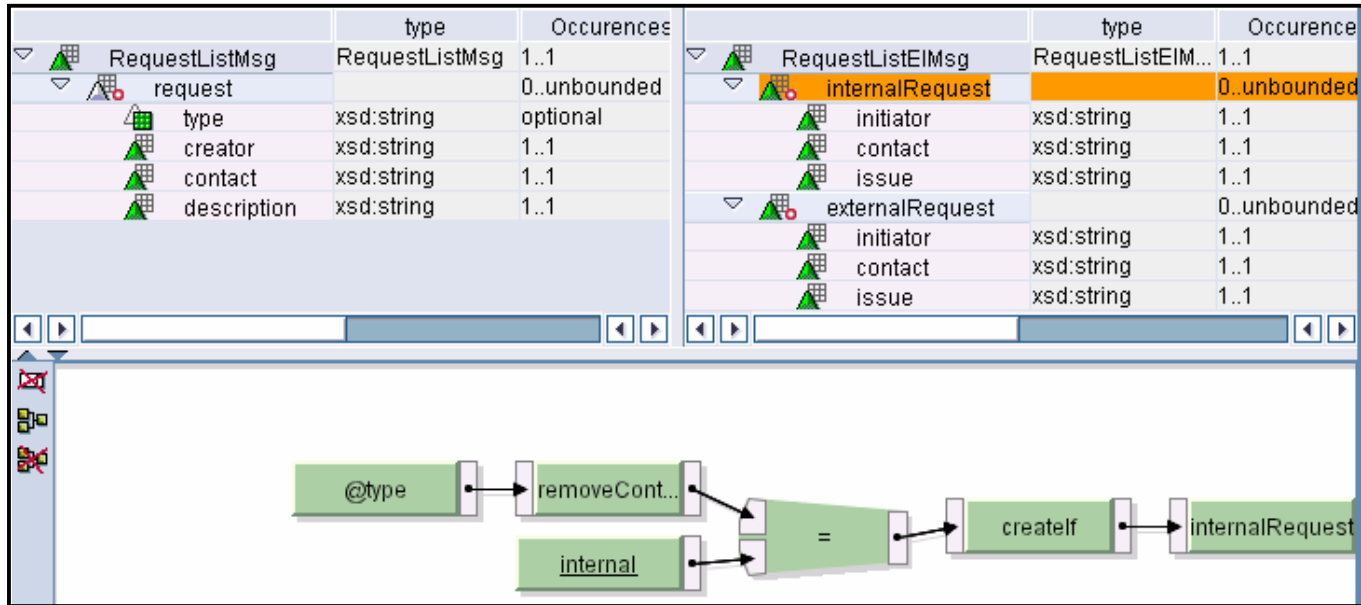
Result:

	value		value
PartnerMsg		CustomerMsg	
partner		customer	
name	Joe	name	Joe
addrDat	123 Main Street	street	123 Main Street
addrDat	Smallville	city	Smallville
addrDat	12345	zipCode	12345

# Node Function - createlf

Use createlf() to create a tag in the target structure using a condition.

Example: Depending on the value of type (internal or external), the target tag either internalRequest or externalRequest will be created.



RequestListMsg	value
RequestListMsg	
request	
type	internal
creator	Creator_In_1
contact	Contact_In_1
description	Descrip_In_1
request	
type	external
creator	Creator_Ex_2
contact	Contact_Ex_2
description	Descrip_Ex_2

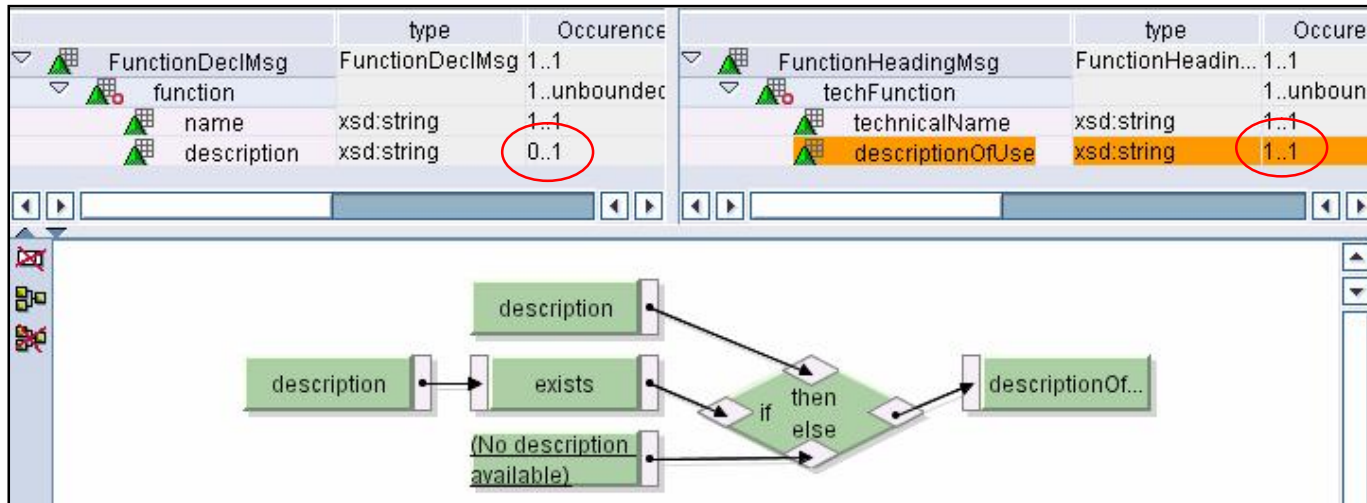
  

RequestListEIMsg	value
RequestListEIMsg	
internalRequest	
initiator	Creator_In_1
contact	Contact_In_1
issue	Descrip_In_1
externalRequest	
initiator	Creator_Ex_2
contact	Contact_Ex_2
issue	Descrip_Ex_2

# Node Function- exists

Use `exists()` to determine whether a particular source field exists in the XML instance to be processed. If does, `exists()` returns *true*, otherwise, *false*.

**Example:** In the source, description is optional. However, in the target, description is mandatory.

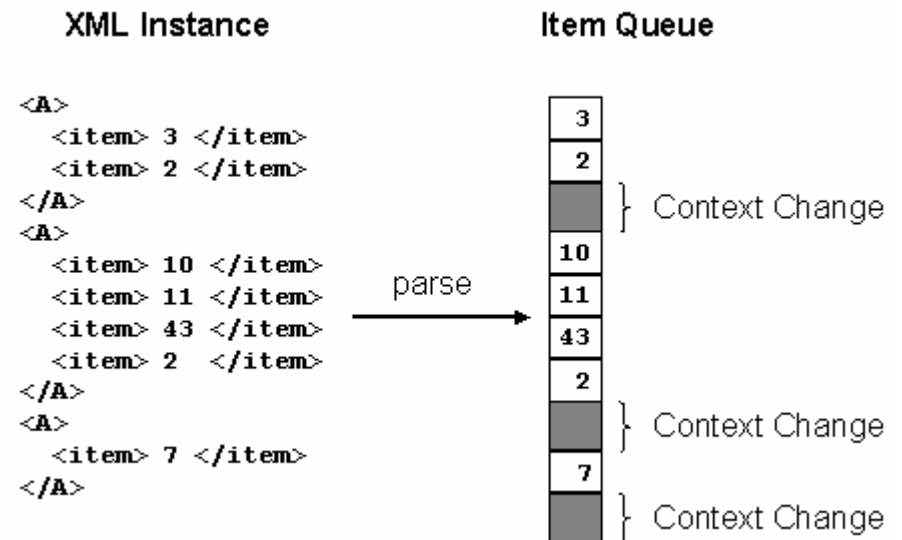


Source		Target	
	value		value
FunctionDeclMsg		FunctionHeadingMsg	
function		techFunction	
name	name1	technicalName	name1
description	description2	descriptionOfUse	(No description available)
function		techFunction	
name	name2	technicalName	name2
description	description2	descriptionOfUse	description2

# Node Functions – Contexts

## Context Change

- n Message mapping works internally by using queues
- n If no further elements are imported at a particular hierarchy level, a *Context Change* is inserted in the queue
- n Use node functions to handle changes in the message hierarchy.



## removeContexts

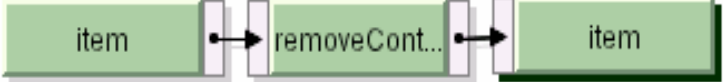
- n deletes all context changes of a queue

## SplitByValue

- n insert additional context changes in a queue

# Node Functions - removeContexts (I)

With *removeContexts*


Source Message	Mapping	Target Message
<pre> &lt;?xml version="1.0" encoding="UTF-8" ?&gt;  &lt;Test_Out_Remove&gt;   &lt;header name="A"&gt;     &lt;item&gt;A.one&lt;/item&gt;     &lt;item&gt;A.two&lt;/item&gt;     &lt;item&gt;A.three&lt;/item&gt;   &lt;/header&gt;   &lt;header name="B"&gt;     &lt;item&gt;B.one&lt;/item&gt;     &lt;item&gt;B.two&lt;/item&gt;   &lt;/header&gt; &lt;/Test_Out_Remove&gt;           </pre>		<pre> &lt;?xml version="1.0" encoding="UTF-8" ?&gt;  &lt;Test_In_Remove&gt;   &lt;item&gt;A.one&lt;/item&gt;   &lt;item&gt;A.two&lt;/item&gt;   &lt;item&gt;A.three&lt;/item&gt;   &lt;item&gt;B.one&lt;/item&gt;   &lt;item&gt;B.two&lt;/item&gt; &lt;/Test_In_Remove&gt;           </pre>

§use `removeContext ( )` to delete the parent context of an element.



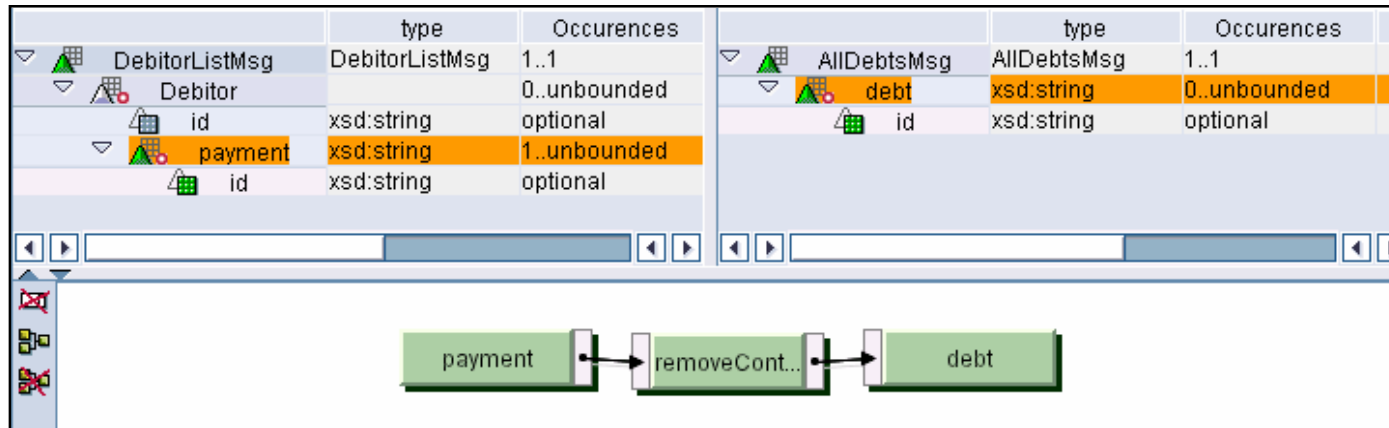
# Node Functions - removeContexts (II)

## Without *removeContexts*

Source Message	Mapping	Target Message
<pre>&lt;?xml version="1.0" encoding="UTF-8" ?&gt;  &lt;Test_Out_Remove&gt;   &lt;header name="A"&gt;     &lt;item&gt;A.one&lt;/item&gt;     &lt;item&gt;A.two&lt;/item&gt;     &lt;item&gt;A.three&lt;/item&gt;   &lt;/header&gt;   &lt;header name="B"&gt;     &lt;item&gt;B.one&lt;/item&gt;     &lt;item&gt;B.two&lt;/item&gt;   &lt;/header&gt; &lt;/Test_Out_Remove&gt;</pre>		<pre>&lt;?xml version="1.0" encoding="UTF-8" ?&gt;  &lt;Test_In_Remove&gt;   &lt;item&gt;A.one&lt;/item&gt;   &lt;item&gt;A.two&lt;/item&gt;   &lt;item&gt;A.three&lt;/item&gt; &lt;/Test_In_Remove&gt;</pre>

# Node Function - removeContext

Use `removeContext()` to delete all the higher contexts for an element. This deletes all higher hierarchy levels, so that all elements of the target queue are assigned to a root element of the source queue.



DebitorListMsg		value	AllDebtsMsg		value
DebitorListMsg	Debitor		AllDebtsMsg	debt	50
	id	1		id	1
	payment	50		debt	90
	id	1		id	2
	payment	90		debt	2200
	id	2		id	3
	Debitor			debt	450
	id	2		id	4
	payment	2200		debt	1500
	id	3		id	5
	payment	450			
	id	4			
	payment	1500			
	id	5			

# Node Function- SplitByValue

SplitByValue() is the counterpart to removeContexts(): Instead of deleting a context, you can insert a context change in the source value queue.

	type	Occurrences
ManagerMsg	ManagerMsg	1..1
managers		1..1
personalld	xsd:string	0..unbounded

	type	Occurrences
RoomsMsg	RoomsMsg	1..1
room		3..3
personalld	xsd:string	1..1

SplitByValue properties

Insert context change after

Each value

OK Cancel

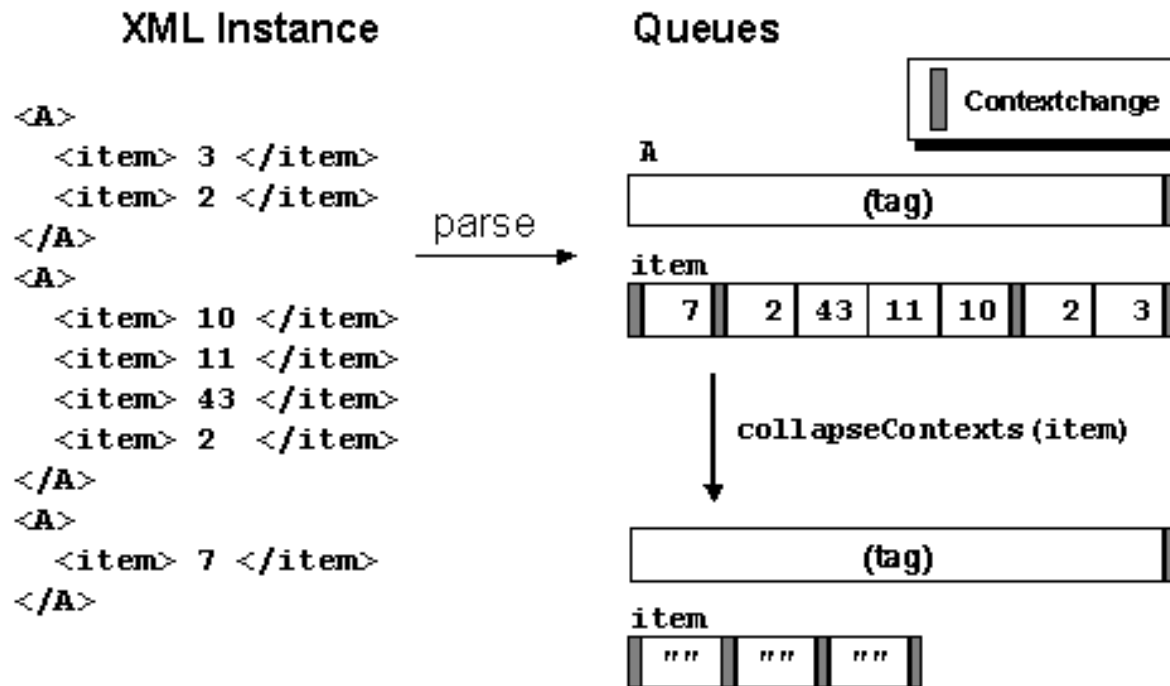
Only this number of elements will be created. If the number in the source is less, error exception will occur.

	value
ManagerMsg	
managers	
personalld	1111
personalld	2222
personalld	3333
personalld	4444
personalld	5555

	value
RoomsMsg	
room	
personalld	1111
room	
personalld	2222
room	
personalld	3333

# Node Function - collapseContexts

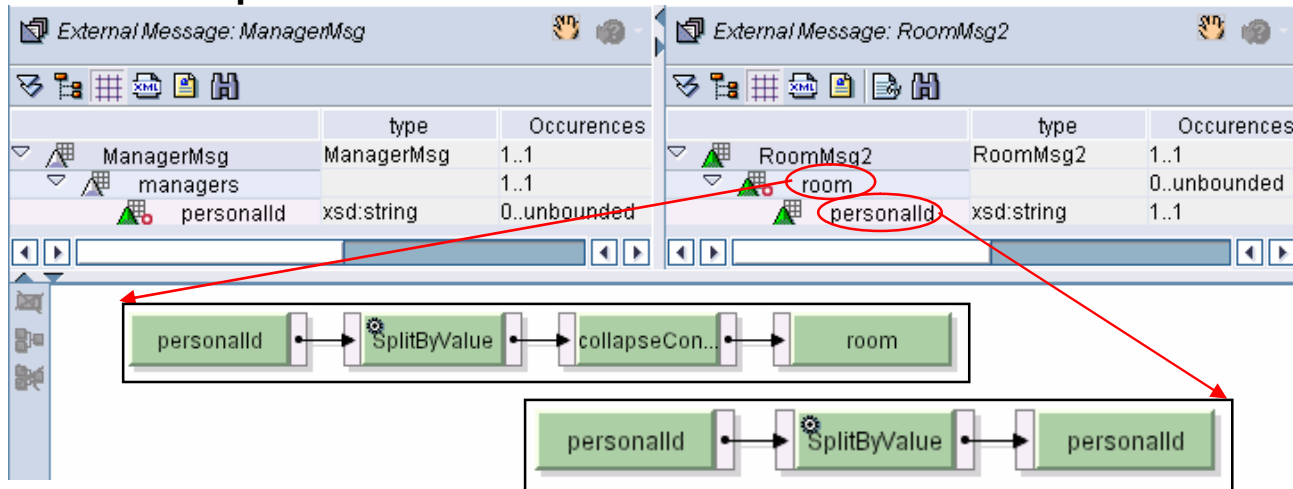
Using `collapseContexts()` replaces all values of all contexts with an empty string:



This can be useful if each time an upper-level node is to be created if a lower-level node exists, for example. The function is useful in combination with the function [SplitByValue\(\)](#).

# Node Function – collapseContexts (cont.)

To avoid previous example's limitation of just using `SplitByValue()`, when exact number of elements had to be provided.



	value		value
ManagerMsg		RoomsMsg2	
managers		room	
personald	1111	personald	1111

**Example when source has only 1 element.**

	value		value
ManagerMsg		RoomsMsg2	
managers		room	
personald	1111	personald	1111
personald	2222	room	
personald	3333	personald	2222
personald	4444	room	
personald	5555	personald	3333
		room	
		personald	4444
		room	
		personald	5555

**Example when source has multiple elements.**

# Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions

- n Function Usage
- n Simple Function Examples
- n Node Function Examples

**Introduction to Most-Common Java Usage**

User-defined Functions

- n Usage

Advanced User-Defined Functions

- n Usage
- n ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”

- n Summarization
- n Duplicating Subtrees
- n Table/Value Lookup (within source, not external)
- n Tree-Reversal

Multi-Mappings

Mapping Templates

Summary

Using Java in user functions will extend the capability of Message Mapping and, many time, will be necessary.

To effectively use Java in Message Mapping:

- n Do not need to be an expert Java programmer
- n Should be familiar with classes in packages:
  - u `java.lang.*` (e.g. String)
  - u `java.util.*` (e.g. Vector, Collections)
- n Should be familiar with String and array operations:
  - u `length()` and `length`
  - u compare (do not use `==`, instead, use `equals()`)
- n Should be familiar with `for...next` and `while` loops.

## String

- n All parameters passed in user functions are as **String** or array of **String**:
  - u Single **String** variable is passed in simple user function
  - u **String** array is passed in advanced user function
- n **String** has the following useful methods:
  - u **substring**
  - u **concat**
  - u **equals** and **equalsIgnoreCase**
  - u **length**
- n **String** comparisons:
  - u **DO NOT** use:
    - String a;
    - if (a == "PRODUCT") ...
  - u **DO** use:
    - String a;
    - if (a.equals("PRODUCT")) ...
    - or
    - if (a.equalsIgnoreCase("PRODUCT")) ...



## Arrays

- n Parameters in advanced user functions are passed as arrays.
- n Most common processing of arrays is the use of *for...next*:

```
String[ ] a;  
for (int i=0; i<a.length; i++) {  
    if (a[i].equals("PRODUCT") ...  
        .....  
}
```

## Integer

- n Use Integer class to convert integer to String

```
int i = 100;  
String a = Integer.toString(i);
```

# Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions

- n Function Usage
- n Simple Function Examples
- n Node Function Examples

Introduction to Most-Common Java Usage

**User-defined Functions**

- n **Usage**

Advanced User-Defined Functions

- n Usage
- n ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”

- n Summarization
- n Duplicating Subtrees
- n Table/Value Lookup (within source, not external)
- n Tree-Reversal

Multi-Mappings

Mapping Templates

Summary

### User–Define Functions:

- n Functional enhancements: if standard functions do not fulfill requirements**
- n Is only visible in the message mapping in which you created it**
- n Can use Java programs from imported archives of the same software component version**
- n Use just like standard functions**

### User-Defined Function types:

- n *Simple functions***, which can process individual field input values for each function call. Simple functions, therefore, expect strings as input values and return a string.
- n *Advanced functions***, which can process non-single string field input values (or arrays) for each function call. You can pass either all field values of a context or the whole queue for the field in an array when calling the function. Each input field is passed as an array of String. Returned values are stored in a String array, ResultList.

### Objects available to Simple and Advanced Functions

Object	Use
<b>Container</b>	This object enables you to cache the values that you want to read again when you next call the same user-defined function.
<b>GlobalContainer</b>	This object enables you to cache the values that you want to read again when you next call any user-defined function that is in the same message mapping.
<b>MappingTrace</b>	This object enables you to transfer information for the mapping trace during mapping to a container that can be viewed by users in the message monitoring.

# User-Defined Functions - Editor

The screenshot displays the SAP Message Mapping Editor interface. At the top, the menu bar includes 'Message Mapping', 'Navigation', 'Edit', and 'View'. Below the menu, the 'Edit Message Mapping' section shows the following details:

- Name: Manager\_to\_Room2
- Namespace: urn:sap-com:bl:test
- Software Component Version: SWC\_B, 1.0 of VendorB
- Status: Active

The main workspace is divided into two message type editors:

- Message Type: ManagerMsg**

	type	Occurrences
ManagerMsg	ManagerMsg	1..1
managers		1..1
personalId	xsd:string	0..unbounded
- Message Type: RoomsMsg2**

	type	Occurrences	Description
RoomsMsg2	RoomsMsg2	1..1	
room		0..unbounded	
personalId	xsd:string	1..1	

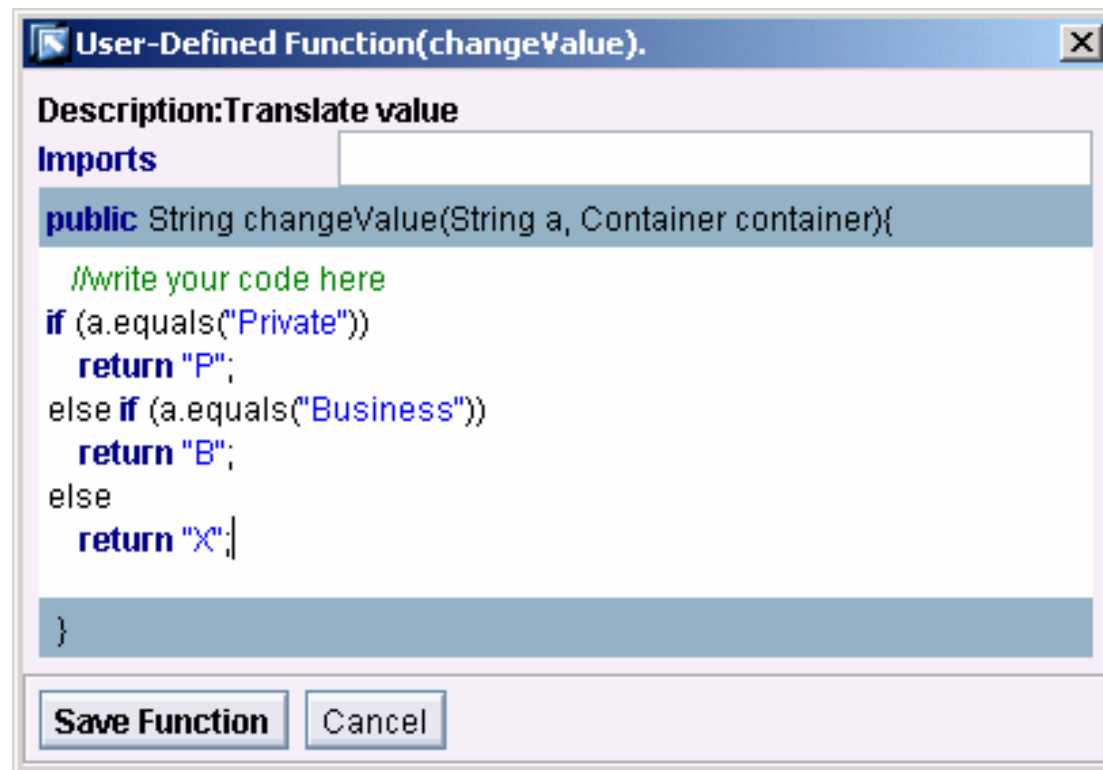
A 'New Function' dialog box is open in the foreground, containing the following fields:

- Label: changeValue
- Argument Count: 1
- Description: Translate value

Buttons for 'Create Function' and 'Cancel' are visible at the bottom of the dialog. In the background, a 'Simple Function' button is circled in red, with a callout bubble pointing to it that says 'Define User Function'. Other buttons like 'Advanced Function' and various function names (substring, concat, etc.) are also visible in the background.

# User-Defined Simple Function

- n Select Simple Function
- n Single return value is of type `java.lang.String`



- n Import Java packages if required,  
e.g. `com.company.sap.xi.mytools.*`; `com.company.sap.xi.myutils.*`;

# Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions

- n Function Usage
- n Simple Function Examples
- n Node Function Examples

Introduction to Most-Common Java Usage

User-defined Functions

- n Usage

**Advanced User-Defined Functions**

- n Usage
- n **ResultList, Container Object, GlobalContainer Object, MappingTrace Object**

Message Mapping “Patterns”

- n Summarization
- n Duplicating Subtrees
- n Table/Value Lookup (within source, not external)
- n Tree-Reversal

Multi-Mappings

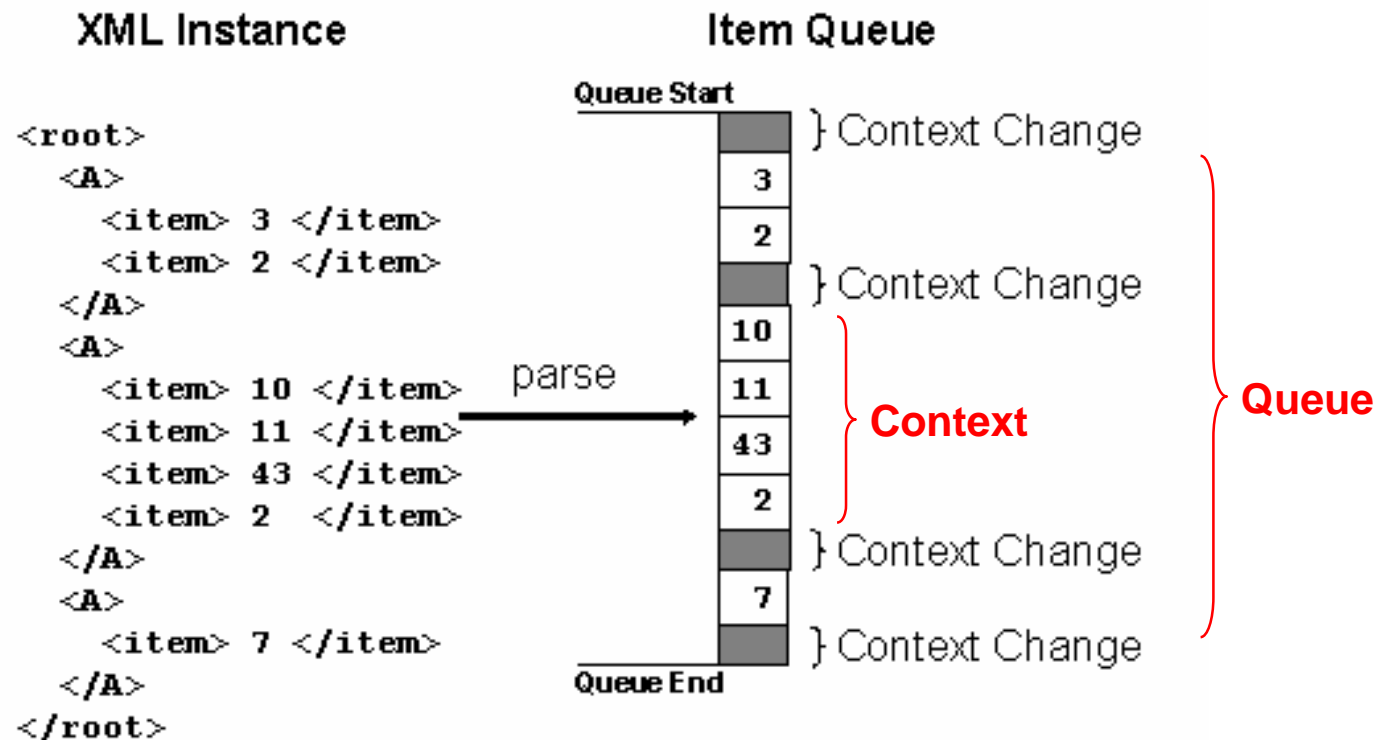
Mapping Templates

Summary



# Advanced User-Defined Functions

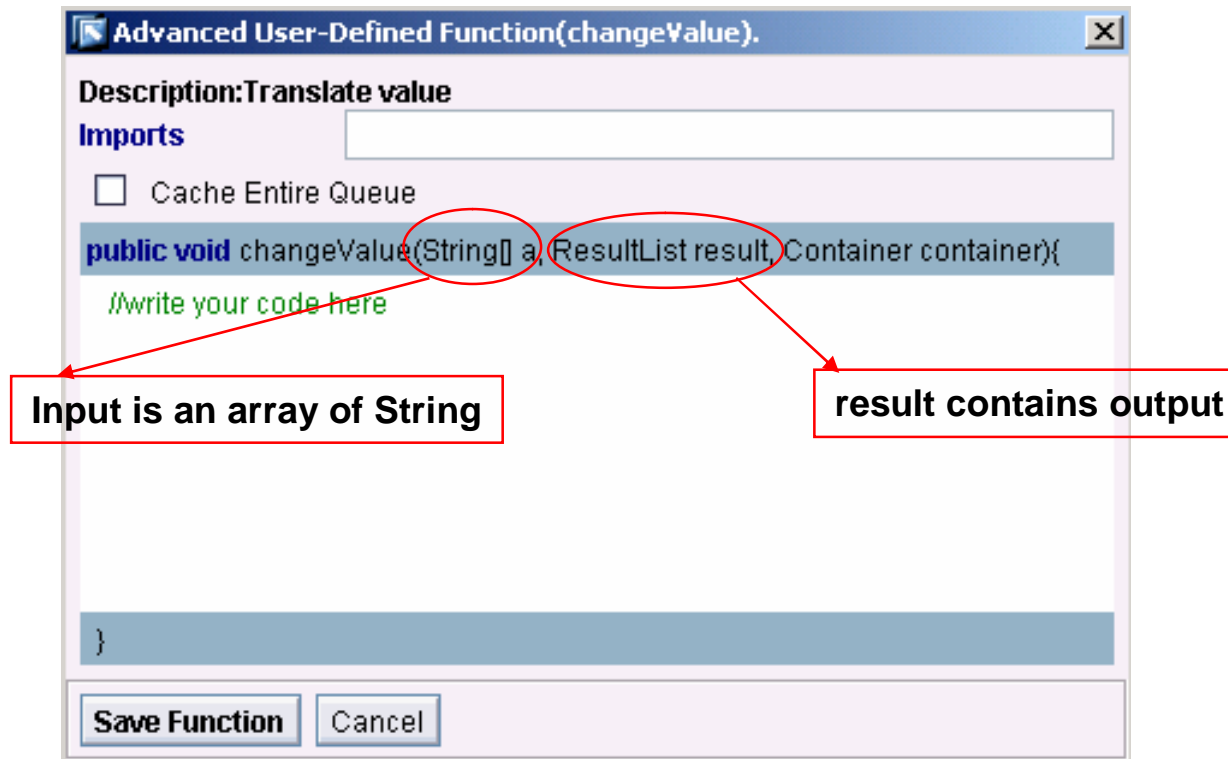
Advanced user-defined functions can access more than just individual values. A complete context or an entire queue can be accessed.



Message Mapping works by using queues. There is a queue for each element.

# User-Defined Advanced Function

- n Select Advanced Function
- n Values to be returned are in the object result



- n Import Java packages if required,  
e.g. com.company.sap.xi.mytools.\*; com.company.sap.xi.myutils.\*;

# Advanced User-Defined Functions

Advanced user-functions can import either just one context or the complete queue into input String arrays.

To import the complete queue, check the “Cache Entire Queue” checkbox in the function editor.

## Working with Contexts or Queues

Information in Cache	Implications
Context	Do not have identifiable context change.
Queue	Contains context change indicator. Much more memory intensive.

Context changes at the beginning and end of the context or queue are implicit and cannot be identified.

# Advanced User-Defined Functions

A queue can have the following entries:

<b>Value</b>	<b>Meaning</b>
<b>(empty string)</b>	In the queue, there is an entry with an empty string each time the field appears in the XML instance, e.g. <A />, <root />.
<b>(string)</b>	Value of a value field, e.g. <item>2</item>
<b>ResultList.CC</b>	Constant that shows a context change.
<b>ResultList.SUPPRESS</b>	Constant that causes a field and its sub node to be ignored during processing.

Special structures can be created by adding or removing context changes, e.g. by using functions, `SplitByValue()` or `removeContexts()` or manually inserting the constant.

# Advanced User-Defined Functions – ResultList Object

This object is used in advanced user-defined functions to return the result of the function. It can be used to either return a queue or to return the context of a queue.

## Methods of the ResultList Object

Method	Use
<code>void addValue(String value);</code>	Append a value to the results list.
<code>void addContextChange();</code>	Appends a context change to the list. This is the same as append ResultList.CC by using addValue().
<code>void addSuppress();</code>	Appends the constant ResultList.SUPPRESS to the list.
<code>void clear();</code>	Deletes all previously appended values from the list.

**The ResultList.CC constant can be used to find context changes in the input array, when the “Cache Entire Queue” is checked.**

**For example:**

```
if ( a[index].equals(ResultList.cc) ) ...
```

### **Note:**

- n When the input array contains only context, no context change exists.**
- n When the input array contains the entire queue, context changes can exist, but not at the beginning or end of the queue.**

# Advanced User-Defined Functions – Container Object

**Use Container object to:**

**Cache (or save) data which can be accessed again when the user-function is called again.**

<b>Methods</b>	<b>Use</b>
<code>void setParameter(String parName, object obj);</code>	Saves the <i>obj</i> object under the name <i>parName</i> in a container for a user-specific function.
<code>Object getParameter (String parName);</code>	Returns the parameter that was saved under the <i>parName</i> name in this user-specific function by using the <b>setParameter</b> method. If no such parameter exists, <i>null</i> is returned.
<code>MappingTrace getTrace();</code>	Returns a <i>MappingTrace</i> object with which messages can be written to the mapping trace in the monitor. (to be discussed)
<code>GlobalContainer getGlobalContainer();</code>	Returns a <i>GlobalContainer</i> object with which data can be saved and read by any user-defined functions in the same message mapping. (to be discussed)
<code>Java.util.Map getTransformationParameters();</code>	Returns a map with the mapping runtime constants. (to be discussed)

# Advanced User-Defined Functions – Container Object

## Use Container object:

To access mapping runtime constants, e.g. sender or receiver.

- n The constants are attributes of `com.sap.aii.mapping.api.StreamTransformationConstants`. Therefore, it must be imported in the user-defined function.
- n The following values can be obtained: `PROCESSING_MODE`, `MESSAGE_ID`, `REF_TO_MESSAGE_ID`, `TIME_SENT`, `INTERFACE`, `INTERFACE_NAMESPACE`, `SENDER_PARTY`, `SENDER_SERVICE`, `RECEIVER_NAME`, `RECEIVER_NAMESPACE`, `RECEIVER_PARTY`, `RECEIVER_SERVICE`, etc. (additional ones can be found in the Java Mapping documentation)
- n Usage example:

```
String constant;  
java.util.Map map;  
// get constant map  
map = container.getTransformationParameters();  
constant = (String) map.get(  
    StreamTransformationConstants.INTERFACE_NAMESPACE);  
return constant;
```

*(during design time, test string is used.)*



## Advanced User-Defined Functions – GlobalContainer Object

This object enables you to cache the values that you want to read again when you next call any user-defined function that is in the same message mapping.

Note that the sequence in which user-defined functions are called is predefined. It depends on the position of the target fields that the function was assigned to.

To get a global container object, call the *getGlobalContainer()* method for the *Container* Object.

Methods	Use
void setParameter(String parName, object obj);	Saves the <i>obj</i> object under the name <i>parName</i> in a global container for a user-specific function.
Object getParameter (String parName);	Returns the parameter that was saved under the <i>parName</i> name in this user-specific function by using the <b>setParameter</b> method. If no such parameter exists, <i>null</i> is returned.

## Advanced User-Defined Functions – MappingTrace Object

This object enables the writing of messages to the mapping trace in the message monitoring (SXMB\_MONI).

Trace Level	Method	Use
0		No trace
1	<code>void addInfo(String message);</code>	Add message with trace level <i>info</i> .
2	<code>void addWarning(String message);</code>	Add message with trace level <i>warning</i> .
3	<code>void addDebugMessage(String message);</code>	Add message with trace level <i>debug</i> .

Trace level can be set in the Pipeline configuration. Each trace level will include messages from lesser trace levels, e.g. *debug* will include messages from *warning* and *info*.

Usage example:

```
MappingTrace trace = container.getTrace();  
trace.addInfo("This is a trace message.");
```

# Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions

- n Function Usage
- n Simple Function Examples
- n Node Function Examples

Introduction to Most-Common Java Usage

User-defined Functions

- n Usage

Advanced User-Defined Functions

- n Usage
- n ResultList, Container Object, GlobalContainer Object, MappingTrace Object

**Message Mapping “Patterns”**

- n Summarization**
- n Sequence-Number Generation**
- n Duplicating Subtrees**

Multi-Mappings

Mapping Templates

Summary

## Mapping “Patterns”

**Mapping “patterns” are some of the common mapping requirements we found in the past which XSLT was used, because we did not think that Message Mapping provided the functionality**

- n Summarization**
- n Sequence–Number Generation**
- n Duplicating Subtrees**

# Mapping "Patterns" - Summarization

Summarization is when we try to consolidate detailed information into total/subtotals and counts.

The count and sum functions will act on the content of a context. Therefore, the context of the group to be summarized must be selected appropriately.

Example:

**Source: Order details containing customer, order numbers and order amount**

Structure	Category	Type	Occurrence
▼ SalesOrderDetail	Complex Type		
▼ Customer	Element		1..unbounded
Name	Element	xsd:string	1
▼ Order	Element		1..unbounded
ID	Element	xsd:string	1
Amount	Element	xsd:deci...	1

**Target: Total and count the order by each customer and a summary of all orders**

Structure	Category	Type	Occurrence
▼ SalesOrderSummary	Complex Type		
▼ Buyer	Element		1..unbounded
BuyerName	Element	xsd:string	1
BuyerOrderCount	Element	xsd:integ...	1
BuyerTotal	Element	xsd:deci...	1
▼ Summary	Element		1
NumberOfBuyers	Element	xsd:integ...	1
TotalOrders	Element	xsd:integ...	1
TotalAmount	Element	xsd:deci...	1

# Mapping "Patterns" - Summarization

Source	Target	Source Type	Target Type
SalesOrderDetail	SalesOrderSummary	1..1	1..1
Customer	Buyer	1..unbounded	1..unbounded
Name	BuyerName	xsd:string	1..1
Order	BuyerOrderCount	1..unbounded	xsd:integer
ID	BuyerTotal	xsd:string	xsd:decimal
Amount	Summary	xsd:decimal	1..1
	NumberOfBuyers		xsd:integer
	TotalOrders		xsd:integer
	TotalAmount		xsd:decimal

# Mapping "Patterns" - Summarization

## Mapping Result:

SalesOrderDetail		SalesOrderSummary	
Customer		Buyer	
Name	Company_A	BuyerName	Company_A
Order		BuyerOrderCount	3
ID	100	BuyerTotal	300
Amount	100		
Order		Buyer	
ID	150	BuyerName	Company_B
Amount	150	BuyerOrderCount	1
Order		BuyerTotal	200
ID	200		
Amount	50	Buyer	
Customer		BuyerName	Company_C
Name	Company_B	BuyerOrderCount	2
Order		BuyerTotal	550
ID	300		
Amount	200	Summary	
Customer		NumberOfBuyers	3
Name	Company_C	TotalOrders	6
Order		TotalAmount	1050
ID	400		
Amount	250		
Order			
ID	500		
Amount	300		

# Mapping "Patterns" - Sequence-Number Generation

There are situations when sequence numbers had to be generated based on the number of occurrences of the source data. This is frequently used during mapping to SAP IDocs or BAPIs when item number, which is no available from the source, had to be added.

**Example:**

**Source: Order information.**

Structure	Category	Type	Occurrence
ItemSeqNoIn	Complex Type		
row	Element		1..unbounded
name	Element	xsd:string	1
ordertype	Element	xsd:string	1

**Target: Two complex elements had to be populated from the source. Each one contains a sequence number which matches the position of the data in the source.**

Structure	Category	Type	Occurrence
ItemSeqNoOut	Complex Type		
item	Element		1..unbounded
itemNo	Element	xsd:integer	1
name	Element	xsd:string	1
row	Element		1..unbounded
seqNo	Element	xsd:integer	1
type	Element	xsd:string	1

Sequence number to be generated.



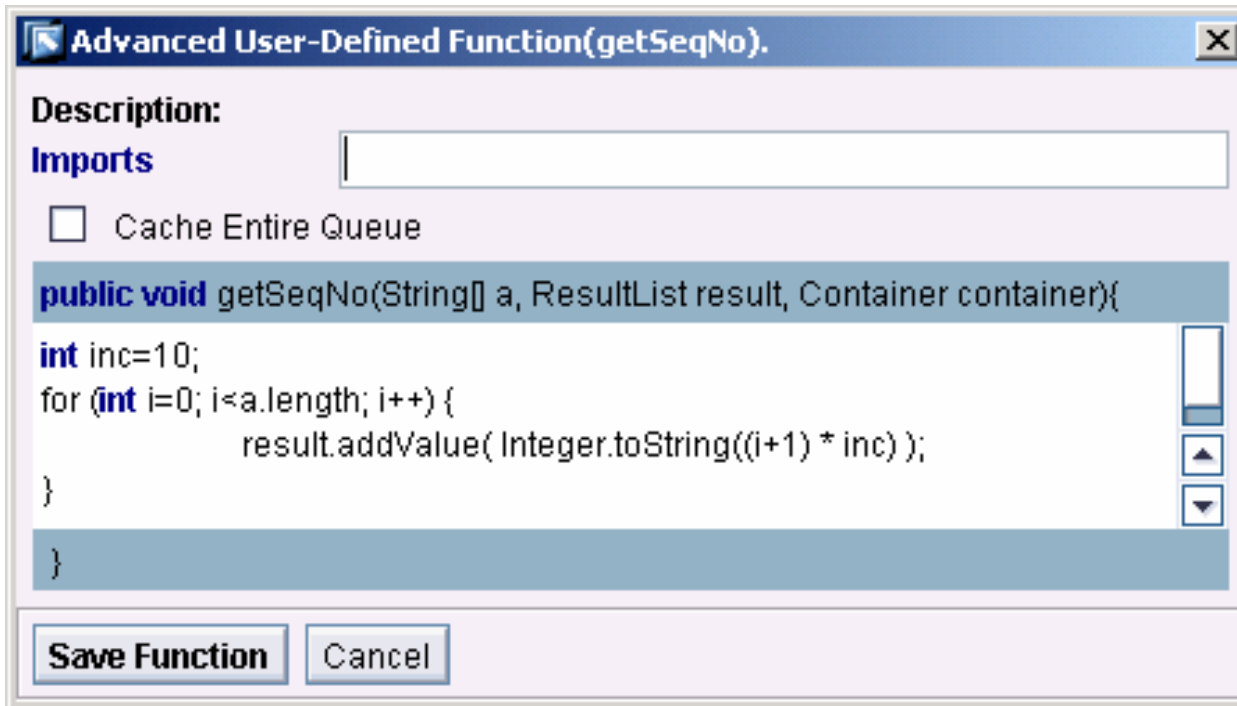
## Mapping “Patterns” – Sequence-Number Generation

**There are two ways the sequence number can be generated.**

- 1. Use Advanced User-Defined function to generate the sequence number all at once.**
- 2. Use Simple User-Defined function to generate the sequence number for each occurrence separately, keeping track of the sequence number value previously generated.**

# Mapping "Patterns" - Sequence-Number Generation

1. Use Advanced User-Defined function to generate the sequence number all at once.



The java function creates the sequence number all at once based on the number of elements in the source.

The sequence numbers are written to the ResultList object.

# Mapping "Patterns" - Sequence-Number Generation

1. Use Advanced User-Defined function to generate the sequence number all at once.

The screenshot displays the SAP Studio interface with two message type panels and a data flow diagram. The left panel shows the input message type 'ItemSeqNoIn' with a table structure. The right panel shows the output message type 'ItemSeqNoOut' with a table structure. Below these panels is a data flow diagram with three 'Display Queue' windows showing the results of the mapping process.

Message Type	Element	Type	Occurrences
ItemSeqNoIn	ItemSeqNoIn	ItemSeqNoIn	1..1
	row		1..unbounded
	name	xsd:string	1..1
	ordertype	xsd:string	1..1

Message Type	Element	Type	Occurrences
ItemSeqNoOut	ItemSeqNoOut	ItemSeqNoOut	1..1
	item		1..unbounded
	itemNo	xsd:integer	1..1
	name	xsd:string	1..1
	row		1..unbounded
	seqNo	xsd:integer	1..1

The data flow diagram shows the following sequence of operations:

```
graph LR; row --> getSeqNo; getSeqNo --> SplitByValue; SplitByValue --> seqNo;
```

The 'Display Queue' windows show the following data:

- Display Queue 1: /ns:ItemSeqNoIn/row
- Display Queue 2: getSeqNo
- Display Queue 3: SplitByValue

# Mapping "Patterns" - Sequence-Number Generation

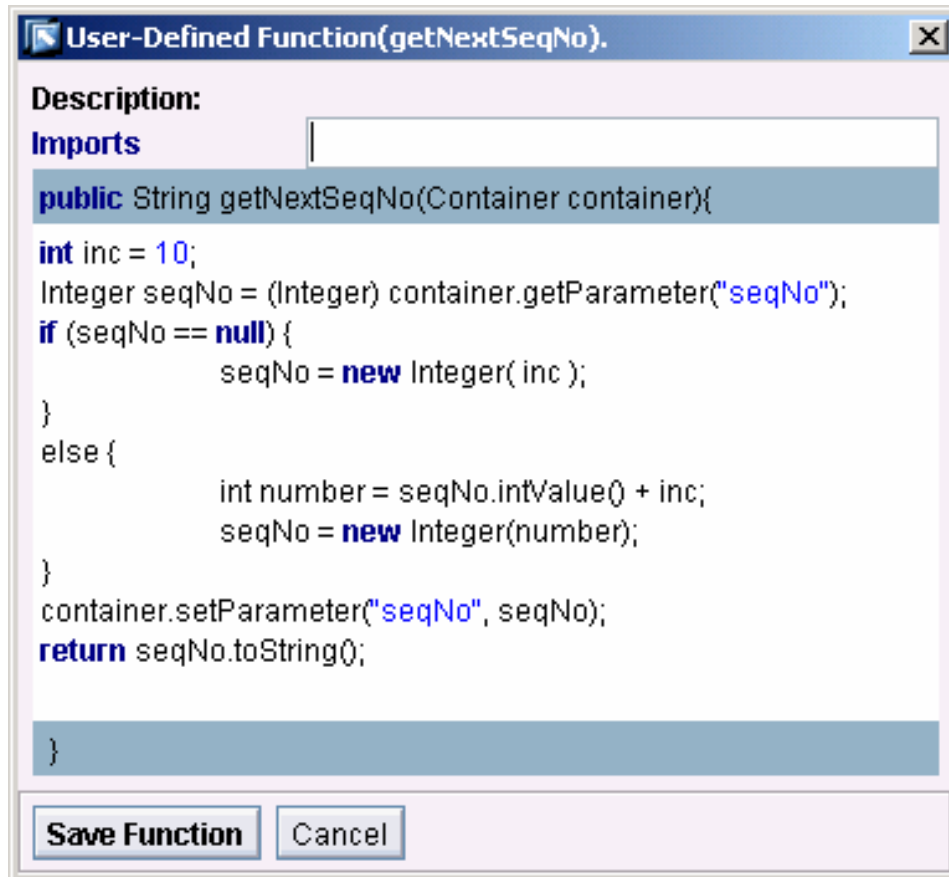
## 1. Use Advanced User-Defined function to generate the sequence number all at once.

### Test Results:

	value		value
ItemSeqNoIn		ItemSeqNoOut	
row		item	
name	aaa	itemNo	10
ordertype	111	name	aaa
row		item	
name	bbb	itemNo	20
ordertype	222	name	bbb
row		item	
name	ccc	itemNo	30
ordertype	333	name	ccc
		row	
		seqNo	10
		type	111
		row	
		seqNo	20
		type	222
		row	
		seqNo	30
		type	333

## Mapping "Patterns" - Sequence-Number Generation

2. Use Simple User-Defined function to generate the sequence number for each occurrence separately, keeping track of the sequence number value previously generated.



```
public String getNextSeqNo(Container container){
    int inc = 10;
    Integer seqNo = (Integer) container.getParameter("seqNo");
    if (seqNo == null) {
        seqNo = new Integer(inc);
    }
    else {
        int number = seqNo.intValue() + inc;
        seqNo = new Integer(number);
    }
    container.setParameter("seqNo", seqNo);
    return seqNo.toString();
}
```

The sequence number is stored in the container object. Each time it is retrieved, incremented and saved.

**Note:** There is no input required.

# Mapping "Patterns" - Sequence-Number Generation

2. Use Simple User-Defined function to generate the sequence number for each occurrence separately, keeping track of the sequence number value previously generated.

The screenshot displays the SAP Message Designer interface with two message types side-by-side: *ItemSeqNoIn* and *ItemSeqNoOut*. The *ItemSeqNoIn* message type has a root element *ItemSeqNoIn* (type *ItemSeqNoIn*, 1..1) containing a *row* element (type *ItemSeqNoIn*, 1..unbounded), which in turn contains *name* (type *xsd:string*, 1..1) and *ordertype* (type *xsd:string*, 1..1). The *ItemSeqNoOut* message type has a root element *ItemSeqNoOut* (type *ItemSeqNoOut*, 1..1) containing an *item* element (type *ItemSeqNoOut*, 1..unbounded), which contains *itemNo* (type *xsd:integer*, 1..1), *name* (type *xsd:string*, 1..1), and a *row* element (type *ItemSeqNoOut*, 1..unbounded) containing *seqNo* (type *xsd:integer*, 1..1) and *type* (type *xsd:string*, 1..1). Below the message type definitions, a mapping diagram shows a green box labeled *getNextSeqNo* connected by an arrow to a green box labeled *itemNo*, indicating that the *getNextSeqNo* function is used to generate the value for the *itemNo* field.

We cannot display the queue to examine the result of the mapping.

# Mapping "Patterns" - Sequence-Number Generation

2. Use Simple User-Defined function to generate the sequence number for each occurrence separately, keeping track of the sequence number value previously generated.

	value		value
ItemSeqNoIn		ItemSeqNoOut	
row		item	
name	aaa	itemNo	10
ordertype	111	name	aaa
row		item	
name	bbb	itemNo	20
ordertype	222	name	bbb
row		item	
name	ccc	itemNo	30
ordertype	333	name	ccc
		row	
		seqNo	10
		type	111
		row	
		seqNo	20
		type	222
		row	
		seqNo	30
		type	333

# Mapping "Patterns" - Duplicating Subtrees

Even if elements are shown to occur more than once in the XML instance according to XML Schema Definition, they are only displayed once in the structure overview.

To assign source field(s) to multiple positions of an element in the target structure, the element or the entire subtree can be duplicated (copied) using the context menu in the target structure.

**Example:**

**Source: Accounting information with adjustment type and amount.**

Structure	Category	Type	Occurrence
AcctSource	Complex Type		
Register	Element		1..unbounded
ADDBY	Element	xsd:string	1
ADDDTTM	Element	xsd:string	1
ADJTYPE	Element	xsd:string	1
AMOUNT	Element	xsd:decimal	1
JOURNALKEY	Element	xsd:string	1

**Target: Two CurrencyAmt's are to be created. One is to credit one account. The other is to debit another account. The amount will either be positive or negative depending on the ADJTYPE.**

Structure	Category	Type	Occurrence
AcctTarget	Complex Type		
CurrencyAmt	Element		1..unbounded
ITEMNO_ACC	Element	xsd:string	1
ACCOUNT	Element	xsd:string	1
CURRENCY	Element	xsd:string	1
AMT_DOCCUR	Element	xsd:deci...	1



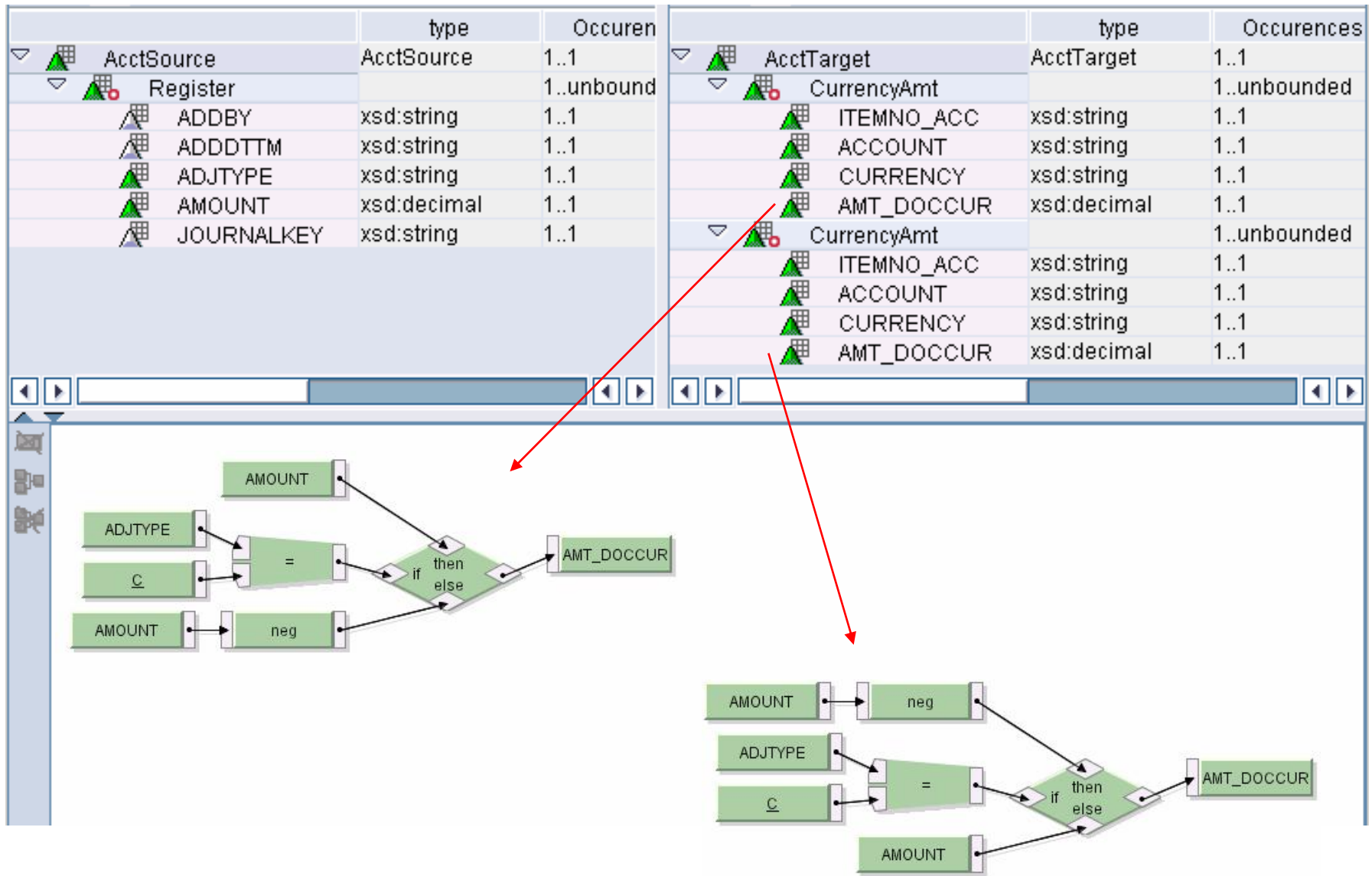
# Mapping "Patterns" - Duplicating Subtrees

The subtree "CurrencyAmt" must be duplicated first. This will create 2 CurrencyAmt's. One for credit, and one for debit.

The screenshot shows two message type editors side-by-side. The left editor is for 'Message Type: AcctSource' and the right is for 'Message Type: AcctTarget'. In the 'AcctTarget' editor, the 'CurrencyAmt' subtree is selected, and a context menu is open with 'Duplicate Subtree' highlighted. The menu also includes options like 'Disable Field', 'Copy Path', 'Load Mapping Template', 'Save Mapping Template', and 'Dependencies'.

The screenshot shows the result of the duplication. The 'AcctTarget' message type now contains two 'CurrencyAmt' subtrees. The first 'CurrencyAmt' subtree has children: ITEMNO\_ACC (xsd:string, 1..1), ACCOUNT (xsd:string, 1..1), CURRENCY (xsd:string, 1..1), and AMT\_DOCCUR (xsd.decimal, 1..1). The second 'CurrencyAmt' subtree is identical to the first.

# Mapping "Patterns" - Duplicating Subtrees



# Mapping "Patterns" - Duplicating Subtrees

## Mapping Results:

	value		value
AcctSource		AcctTarget	
Register		CurrencyAmt	
ADDBY	aaaa	ITEMNO_ACC	1
ADDDTTM	20040301	ACCOUNT	13579
ADJTYPE	D	CURRENCY	USD
AMOUNT	1000	AMT_DOCCUR	-1000
JOURNALKEY	yyyy	CurrencyAmt	
Register		ITEMNO_ACC	1
ADDBY	aaaa	ACCOUNT	13579
ADDDTTM	20040401	CURRENCY	USD
ADJTYPE	C	AMT_DOCCUR	2000
AMOUNT	2000	CurrencyAmt	
JOURNALKEY	xxxxx	ITEMNO_ACC	2
		ACCOUNT	24680
		CURRENCY	USD
		AMT_DOCCUR	1000
		CurrencyAmt	
		ITEMNO_ACC	2
		ACCOUNT	24680
		CURRENCY	USD
		AMT_DOCCUR	-2000

# Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions

- n Function Usage
- n Simple Function Examples
- n Node Function Examples

Introduction to Most-Common Java Usage

User-defined Functions

- n Usage

Advanced User-Defined Functions

- n Usage
- n ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”

- n Summarization
- n Duplicating Subtrees
- n Table/Value Lookup (within source, not external)
- n Tree-Reversal

**Multi-Mappings**

Mapping Templates

Summary

# Multi-Mappings

## **Multi-Mapping can only be used in ccBPM:**

### **n n:1 Transformation**

**Bundles multiple messages into one message, for example, individual purchase order items into one purchase order.**

### **n 1:n Transformation**

**Splits a message into multiple messages, for example, a purchase order into the individual purchase order items.**

### **n n:m Transformation**

**Converts a message into another message, for example, a message that is defined by interface A is converted to message that is defined by interface B.**

## **Multi-Mappings reference multiple message structures:**

**n All source message structures are combined into 1 source structure.**

**n All target message structures are combined into 1 target structure.**

**n Therefore, there is only one source structure mapped to one target.**

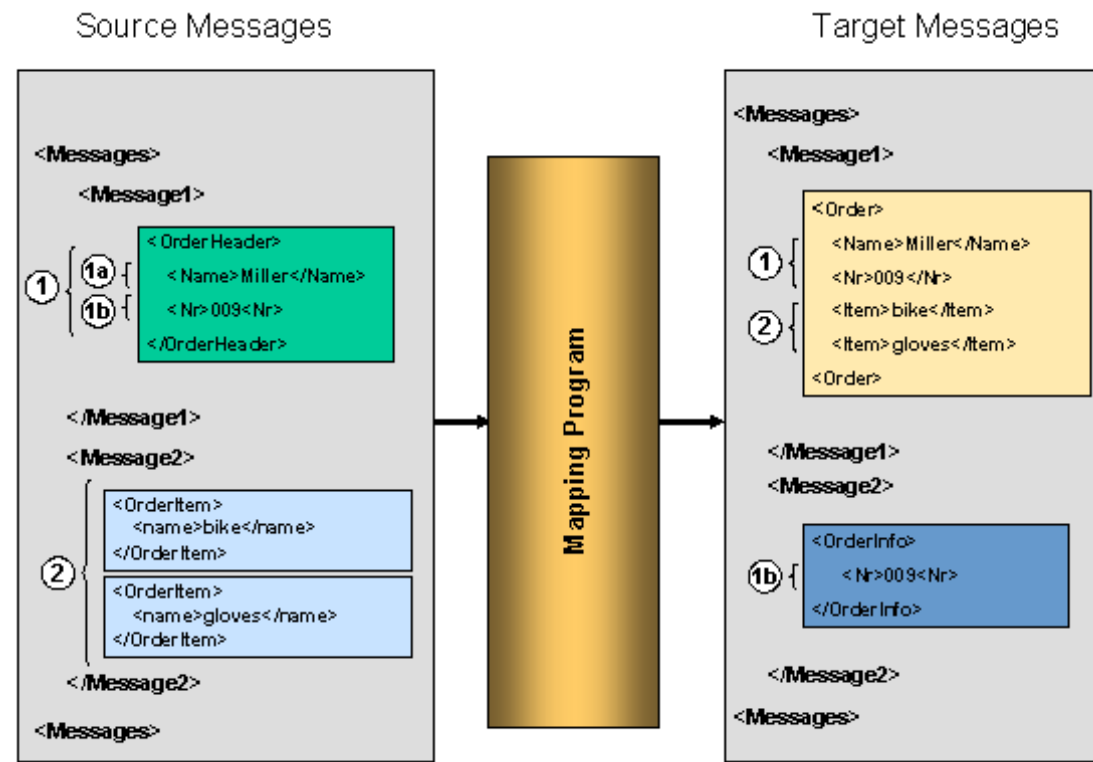
# Mapping "Patterns" - Multi-Mappings

The Message Types are entered using the Messages tab in the mapping editor.

- n The root element is always <Messages>.
- n The Mapping Editor inserts an element <MessageN> for each source or target message, where N is the position of the message.

Example:

Message Type (Source)	Message Type (Target)
OrderHeader(1)	Order(1)
OrderItem(0...unbounded)	OrderInfo(1)



# Mapping "Patterns" - Multi-Mappings

Design Test **Messages**

**Source Message(s)**

Object Type	Name	Namespace	Occurrence
Message Type	OrderHeader	urn:sap-com:bl:test	1
Message Type	OrderItem	urn:sap-com:bl:test	0..unbounded

**Target Message(s)**

Object Type	Name	Namespace	Occurrence
Message Type	Order	urn:sap-com:bl:test	1
Message Type	OrderInfo	urn:sap-com:bl:test	1

**Insert more messages**

**Change Occurrences**

Design Test Messages

**Source Messages**

type	Occurrence
Messages	1..1
Message1	1..1
OrderHeader	1..1
Name	xsd:string 1..1
Nr	xsd:string 1..1
Message2	1..1
OrderItem	OrderItem 0..unbounded
Name	xsd:string 1..1

**Target Messages**

type	Occurrences
Messages	1..1
Message1	1..1
Order	Order 1..1
Name	xsd:string 1..1
Nr	xsd:string 1..1
Item	xsd:string 1..unbounded
Message2	1..1
OrderInfo	OrderInfo 1..1
Nr	xsd:string 1..1

**No msg type name**

Mapping rules are the same as for previously discussed Message Mappings.

# Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions

- n Function Usage
- n Simple Function Examples
- n Node Function Examples

Introduction to Most-Common Java Usage

User-defined Functions

- n Usage

Advanced User-Defined Functions

- n Usage
- n ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”

- n Summarization
- n Duplicating Subtrees
- n Table/Value Lookup (within source, not external)
- n Tree-Reversal

Multi-Mappings

**Mapping Templates**

Summary



# Mapping Templates

**Message Mappings can be saved as *Mapping Templates*.**

***Mapping Templates* can be reused (or loaded) in other Message Mappings or Mapping Templates.**

**Features:**

**Mapping Templates can be defined for structure mappings of the following structures:**

- n Data Types**
- n Complex types in IDocs and RFCs**
- n Complex types in External Definitions**
- n The referenced types used in mapping templates can be located in any software component versions.**

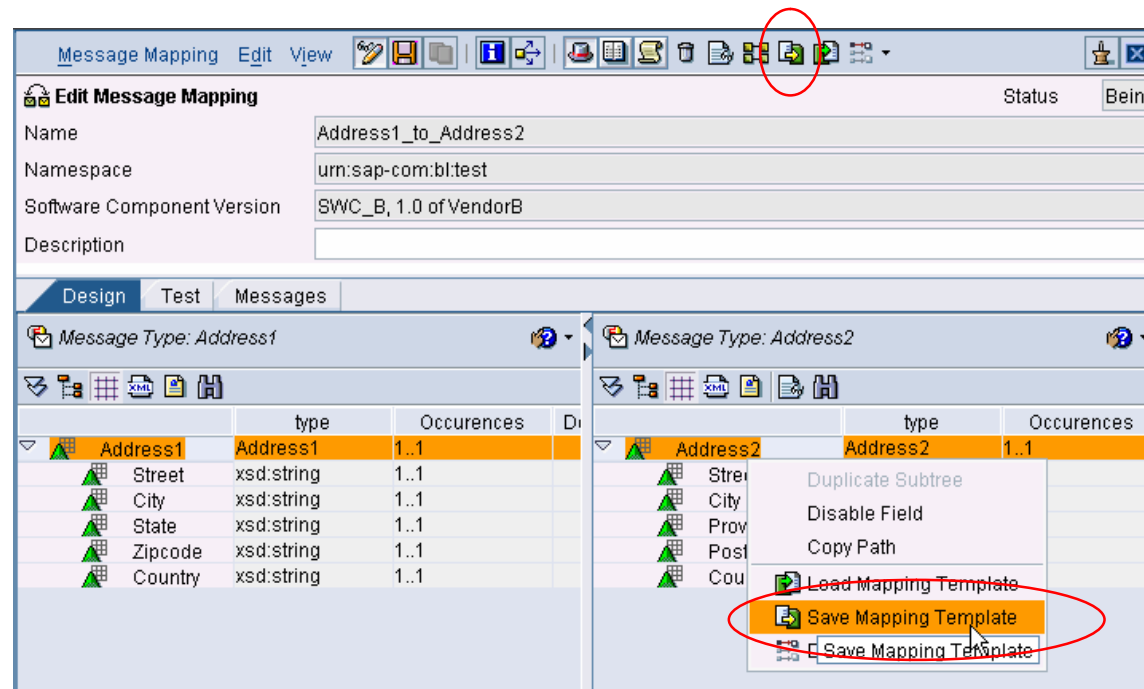
# Mapping Templates

The Mapping Editor is used to create and load the structures used by the Mapping Templates.

## Defining Mapping Templates

n The Mapping Template can be saved by:

1. Selecting the Source and Target source in the Mapping Editor
2. Choose Save Mapping Template from either the target structure context menu or from the object toolbar
3. Enter a unique name for the new Mapping Template.

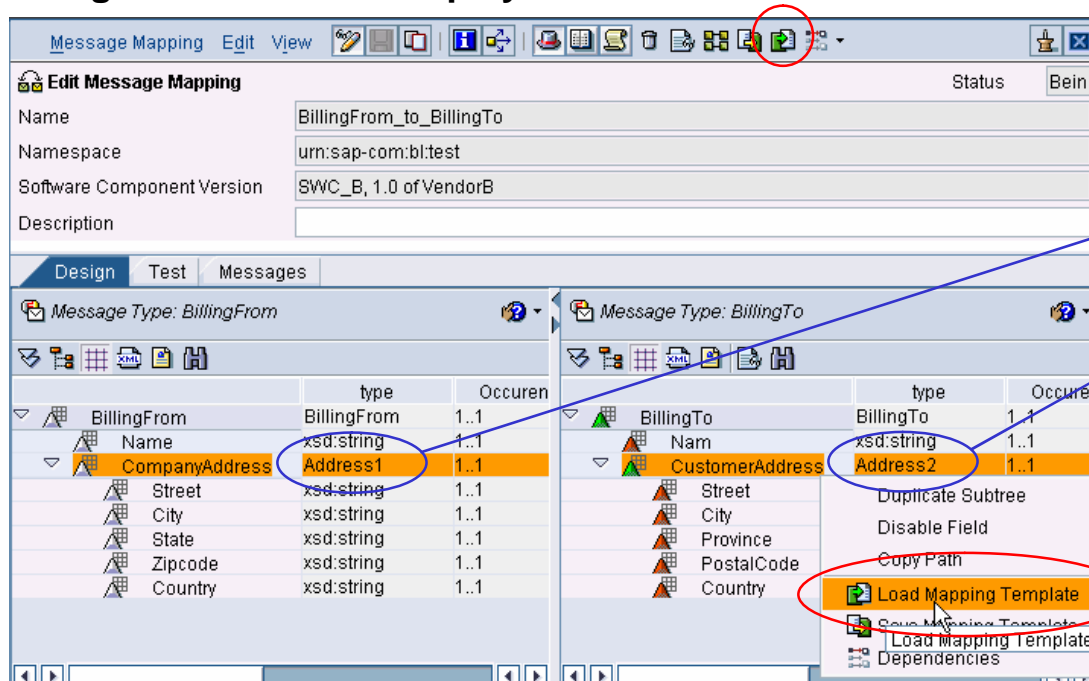


# Mapping Templates

## Using Mapping Templates:

Mapping Templates can be used in Message Mappings from any software component version:

1. In the Mapping Editor, select a type element in the source and target structure.
2. Choose Load Mapping Templates from either the target structure context menu or from the object toolbar.
3. If mapping templates are available for the types in the source and target fields, a select dialog window will be displayed.



The Types must match what were defined in the existing Mapping Templates.

# Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions

- n Function Usage
- n Simple Function Examples
- n Node Function Examples

Introduction to Most-Common Java Usage

User-defined Functions

- n Usage

Advanced User-Defined Functions

- n Usage
- n ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”

- n Summarization
- n Duplicating Subtrees
- n Table/Value Lookup (within source, not external)
- n Tree-Reversal

Multi-Mappings

Mapping Templates

**Summary**

## Further Information

### è Public Web:

<http://help.sap.com>

<http://sdn.sap.com>

<http://service.sap.com>

SAP Customer Services Network: <http://www.sap.com/services/>

### è Related XI 3.0 Workshop / Training Opportunities

NetWeaver04 Overview

XI 3.0 Implementation Workshop

Introduction to XML and Technical Standards

Advanced Integration Builder

Advanced BPM

Adapter Framework

B2B and Industry Standards

Proxy Development and Deployment

Questions?

# Q&A



- n No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.
- n Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.
- n Microsoft®, WINDOWS®, NT®, EXCEL®, Word®, PowerPoint® and SQL Server® are registered trademarks of Microsoft Corporation.
- n IBM®, DB2®, DB2 Universal Database, OS/2®, Parallel Sysplex®, MVS/ESA, AIX®, S/390®, AS/400®, OS/390®, OS/400®, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere®, Netfinity®, Tivoli®, Informix and Informix® Dynamic Server™ are trademarks of IBM Corporation in USA and/or other countries.
- n ORACLE® is a registered trademark of ORACLE Corporation.
- n UNIX®, X/Open®, OSF/1®, and Motif® are registered trademarks of the Open Group.
- n Citrix®, the Citrix logo, ICA®, Program Neighborhood®, MetaFrame®, WinFrame®, VideoFrame®, MultiWin® and other Citrix product names referenced herein are trademarks of Citrix Systems, Inc.
- n HTML, DHTML, XML, XHTML are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.
- n JAVA® is a registered trademark of Sun Microsystems, Inc.
- n JAVASCRIPT® is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.
- n MarketSet and Enterprise Buyer are jointly owned trademarks of SAP AG and Commerce One.
- n SAP, SAP Logo, R/2, R/3, mySAP, mySAP.com and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are trademarks of their respective companies.

- n Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, ohne die ausdrückliche schriftliche Genehmigung durch SAP AG nicht gestattet. In dieser Publikation enthaltene Informationen können ohne vorherige Ankündigung geändert werden.
- n Die von SAP AG oder deren Vertriebsfirmen angebotenen Softwareprodukte können Softwarekomponenten auch anderer Softwarehersteller enthalten.
- n Microsoft®, WINDOWS®, NT®, EXCEL®, Word®, PowerPoint® und SQL Server® sind eingetragene Marken der Microsoft Corporation.
- n IBM®, DB2®, DB2 Universal Database, OS/2®, Parallel Sysplex®, MVS/ESA, AIX®, S/390®, AS/400®, OS/390®, OS/400®, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere®, Netfinity®, Tivoli®, Informix und Informix® Dynamic Server™ sind Marken der IBM Corporation in den USA und/oder anderen Ländern.
- n ORACLE® ist eine eingetragene Marke der ORACLE Corporation.
- n UNIX®, X/Open®, OSF/1® und Motif® sind eingetragene Marken der Open Group.
- n Citrix®, das Citrix-Logo, ICA®, Program Neighborhood®, MetaFrame®, WinFrame®, VideoFrame®, MultiWin® und andere hier erwähnte Namen von Citrix-Produkten sind Marken von Citrix Systems, Inc.
- n HTML, DHTML, XML, XHTML sind Marken oder eingetragene Marken des W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.
- n JAVA® ist eine eingetragene Marke der Sun Microsystems, Inc.
- n JAVASCRIPT® ist eine eingetragene Marke der Sun Microsystems, Inc., verwendet unter der Lizenz der von Netscape entwickelten und implementierten Technologie.
- n MarketSet und Enterprise Buyer sind gemeinsame Marken von SAP AG und Commerce One.
- n SAP, SAP Logo, R/2, R/3, mySAP, mySAP.com und weitere im Text erwähnte SAP-Produkte und -Dienstleistungen sowie die entsprechenden Logos sind Marken oder eingetragene Marken der SAP AG in Deutschland und anderen Ländern weltweit. Alle anderen Namen von Produkten und Dienstleistungen sind Marken der jeweiligen Firmen.