



# **StreamServe Persuasion SP4 StreamServe Connect *for SAP* - Business Processes**

## **User Guide**

Rev A

StreamServe Persuasion SP4StreamServe Connect *for* SAP - Business Processes User Guide  
Rev A

SAP, mySAP.com, and all other names of SAP solutions, products, and services are trademarks of SAP AG.

© 2001-2009 STREAMSERVE, INC.  
ALL RIGHTS RESERVED  
United States patent #7,127,520

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of StreamServe, Inc. Information in this document is subject to change without notice. StreamServe Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this book. All registered names, product names and trademarks of other companies mentioned in this documentation are used for identification purposes only and are acknowledged as property of the respective company. Companies, names and data used in examples in this document are fictitious unless otherwise noted.

StreamServe, Inc. offers no guarantees and assumes no responsibility or liability of any type with respect to third party products and services, including any liability resulting from incompatibility between the third party products and services and the products and services offered by StreamServe, Inc. By using StreamServe and the third party products mentioned in this document, you agree that you will not hold StreamServe, Inc. responsible or liable with respect to the third party products and services or seek to do so.

The trademarks, logos, and service marks in this document are the property of StreamServe, Inc. or other third parties. You are not permitted to use the marks without the prior written consent of StreamServe, Inc. or the third party that owns the marks.

Use of the StreamServe product with third party products not mentioned in this document is entirely at your own risk, also as regards the StreamServe products.

StreamServe Web Site  
<http://www.streamserve.com>

# Contents

---

<b>Business Processes .....</b>	<b>7</b>
<b>Introduction .....</b>	<b>7</b>
IDocs (Intermediate Documents) .....	9
BAPI functions.....	9
Supported releases of the SAP system .....	10
<b>The IDoc file interface.....</b>	<b>11</b>
Receiving IDoc data from SAP using the IDoc file interface .....	11
The SAP IDoc Reader.....	12
<b>The IDoc ALE interface.....</b>	<b>13</b>
SAP and the ALE interface .....	13
ALE Certification and StreamServe .....	13
SAP and the ALE Message Handler and ALE Converter .....	14
StreamServe and the ALE interface.....	14
StreamServe and the ALE Converter .....	15
Sending and receiving IDoc data using the ALE interface.....	16
Configuring Messages for the IDoc ALE interface.....	17
The SAP IDoc Converter.....	18
The SAP IDoc Extractor.....	18
The SAPBP repository .....	18
Meta data file .....	18
Template file .....	19
<b>The BAPI interface .....</b>	<b>20</b>
<b>Configuring the IDoc file interface .....</b>	<b>21</b>
<b>Configuring SAP 4.6/4.7 for the IDoc file interface .....</b>	<b>22</b>
Creating a port definition .....	22
Creating a partner profile .....	23
Generating an IDoc structure file .....	23
<b>Generating IDoc documentation .....</b>	<b>26</b>
HTML files containing IDoc details .....	26
<b>Sending IDoc file data from SAP to StreamServe.....</b>	<b>27</b>
<b>Configuring StreamServe for IDoc file data .....</b>	<b>28</b>
<b>Creating a StreamIN Event for IDoc file data.....</b>	<b>29</b>
Naming a StreamIN Event for IDoc data .....	29
Generating the description file for the IDoc .....	30
Importing an IDoc structure file into StreamIN .....	31
Building the StreamIN Event for IDoc file data.....	32
Multiple Messages for an IDoc basic type .....	33
Comparing Messages.....	33
Specifying the description file as an Event resource.....	34

<b>Configuring the IDoc ALE interface (StreamServe inbound).....</b>	<b>35</b>
<b>Configuring SAP 4.6/4.7 for the IDoc ALE interface (SAP outbound) .....</b>	<b>36</b>
Creating an RFC server destination .....	36
Creating a port definition.....	37
Creating a partner profile .....	37
<b>Configuring StreamServe to receive IDoc data .....</b>	<b>39</b>
Configuring an input connector to receive IDoc data .....	39
Creating a StreamIN Event to receive IDoc data .....	42
Naming a StreamIN Event for IDoc data.....	42
Importing IDoc data into StreamIN .....	43
Importing IDoc data using StreamServe meta-data files .....	44
Retrieving an IDoc meta-data file from SAP .....	44
Exporting a meta file for an IDoc.....	46
Configuring the StreamIN Event for IDoc data .....	47
<b>Configuring the IDoc ALE interface (StreamServe outbound) .....</b>	<b>49</b>
<b>Configuring SAP for the IDoc ALE interface (SAP inbound).....</b>	<b>50</b>
Creating a partner profile .....	50
<b>Configuring StreamServe to send IDoc data .....</b>	<b>51</b>
Creating an XMLOUT Process to send IDoc data .....	51
Configuring an XMLOUT Process to send IDoc data .....	52
IDoc structure and hierarchy.....	52
Sending multiple IDocs to SAP .....	55
<b>Configuring the IDoc Converter.....</b>	<b>57</b>
Viewing and editing IDoc Converter properties .....	61
<b>Running the IDoc Converter on UNIX.....</b>	<b>63</b>
<b>StreamServe and IDoc format.....</b>	<b>65</b>
<b>Control records.....</b>	<b>66</b>
<b>Data records.....</b>	<b>67</b>
Fields in the EDI_DD dictionary object for SAP 3.1 .....	67
Fields in the EDI_DD40 dictionary object for SAP 4.x.....	67
<b>Status records.....</b>	<b>68</b>
<b>Header records .....</b>	<b>69</b>
Variables available for SAP 3.1 .....	69
Variables available for SAP 4.x .....	70
<b>Configuring the BAPI interface.....</b>	<b>73</b>
<b>The BAPI interface and StreamServe .....</b>	<b>74</b>
Licensing StreamServe BAPI scripting .....	75
Enabling BAPI scripting in StreamServe .....	75
<b>The BAPI interface.....</b>	<b>76</b>
BAPI function components .....	76
Rules for using script functions.....	77
BAPI script functions and StreamServe processing phases.....	78
Using SAPInvokeFunction or SAPInvokeFunction2 .....	79
Using SAPInvokeFunction with CallProc and CallBlock .....	79

<b>StreamServe BAPI script functions .....</b>	<b>80</b>
BAPI scripting arguments.....	81
BAPI scripting return values.....	81
SAPConnect.....	81
SAPCreateFunction .....	83
SAPDisconnect .....	84
SAPGetComplexParameter .....	84
SAPGetSimpleParameter .....	85
SAPGetTableParameter .....	86
SAPGetTableRowCount .....	86
SAPInvokeFunction.....	87
SAPInvokeFunction2.....	88
SAPLastError .....	89
SAPSetComplexParameter.....	90
SAPSetSimpleParameter.....	90
SAPSetTableParameter.....	91
Examples of BAPI script functions .....	92
Retrieving data from the SAP system.....	92
Updating data in the SAP system .....	95
Creating a Sales Order in StreamServe .....	98
<b>Useful SAP transaction codes.....</b>	<b>101</b>
<b>IDoc Converter argument file examples .....</b>	<b>103</b>



# Business Processes

---

This guide describes how to configure StreamServe Connect *for SAP* - Business Processes with your SAP system. Business Processes is an add-on module to StreamServer.

**Note:** This guide only contains StreamServe information specific to the Business Processes Connect solution. For general StreamServe information, see the standard StreamServe documentation.

Business Processes is one of four StreamServe Connect solutions available for use with SAP. For information on the other solutions, see the following documentation:

- StreamServe Connect *for SAP* - Output+
- StreamServe Connect *for SAP* - E-docs
- StreamServe Connect *for SAP* - Delivery Manager

## Installation

For information on how to install the StreamServe Connect solutions, see the *StreamServe Connect for SAP - Installation Guide*.

## Introduction

The Business Processes Connect solution enables you to receive and send data between your SAP system and StreamServe. There are three different interfaces you can use:

- The IDoc file interface
- The IDoc ALE interface
- The BAPI interface

### The IDoc file interface

The IDoc file interface enables you to send an IDoc (Intermediate Document) file from your SAP system to StreamServe for processing. See [The IDoc file interface](#) on page 11.



Figure 1 The IDoc file interface

### The IDoc ALE interface

The IDoc ALE interface enables you to use StreamServe to both send and receive IDoc data.

Typically, the IDoc ALE interface is used to convert application data into IDoc format, before sending the data to a SAP system. The IDoc ALE interface can also be used to convert IDoc format from a SAP system to another format, such as EDI or XML, before output.

See [The IDoc ALE interface](#) on page 13.

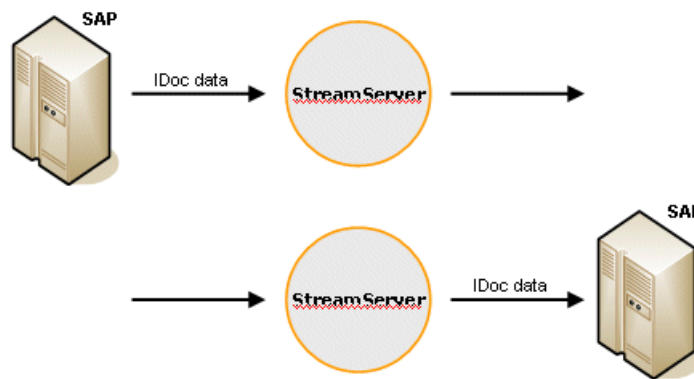


Figure 2 The IDoc ALE interface

## The BAPI interface

The BAPI interface provides a number of script functions which enable you to call any BAPI (Business Application Programming Interface) function available in the SAP system from StreamServe. See [The BAPI interface](#) on page 20.

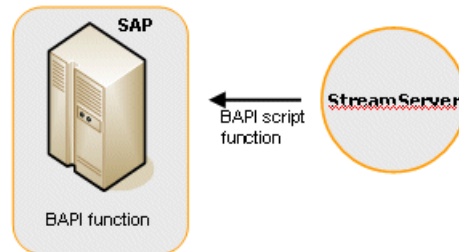


Figure 3 The BAPI interface

## IDocs (Intermediate Documents)

IDocs provide a clearly defined container to send and receive data to, and from a SAP system. IDocs consist of a control record and actual data records that form the structure of the IDoc.

The control record provides all information needed to identify the business document type, in combination with the routing information.

The control record consists of:

- the IDocType, such as a purchase order
- the MessageType (create)
- sender and receiver information.

Data segments of IDocs consist of various segment types organized in a hierarchical structure. These segments hold all actual data contained in the business document, such as customer address and order line items.

For more information about the format of IDocs, see [StreamServe and IDoc format](#) on page 65.

## BAPI functions

BAPI (Business Application Programming Interface) is a SAP standard interface technology, which enables you to retrieve and update data in SAP systems using RFC (Remote Function Calls) functionality from external data sources. The Business Processes Connect solution supports all BAPI functions using the RFC interface, providing system independent access to all business objects used in the SAP system.

See [The BAPI interface](#) on page 20.

## Supported releases of the SAP system

The Business Processes Connect solution supports all releases of SAP from 3.1h onwards.

# The IDoc file interface

The IDoc file interface enables you to send IDoc data as a file from your SAP system to StreamServe for processing.

StreamServer receives the IDoc file as field-based data in a StreamIN Event. The Event is configured with a Business Processes agent, which StreamServer uses to recognize and process the data correctly.

StreamServer uses a description file during runtime to interpret the IDoc data, and process the file according to the specified output process.

See *Configuring the IDoc file interface* on page 21.

## Receiving IDoc data from SAP using the IDoc file interface

The following diagram illustrates how StreamServe receives IDoc data from the SAP system using the IDoc file interface.

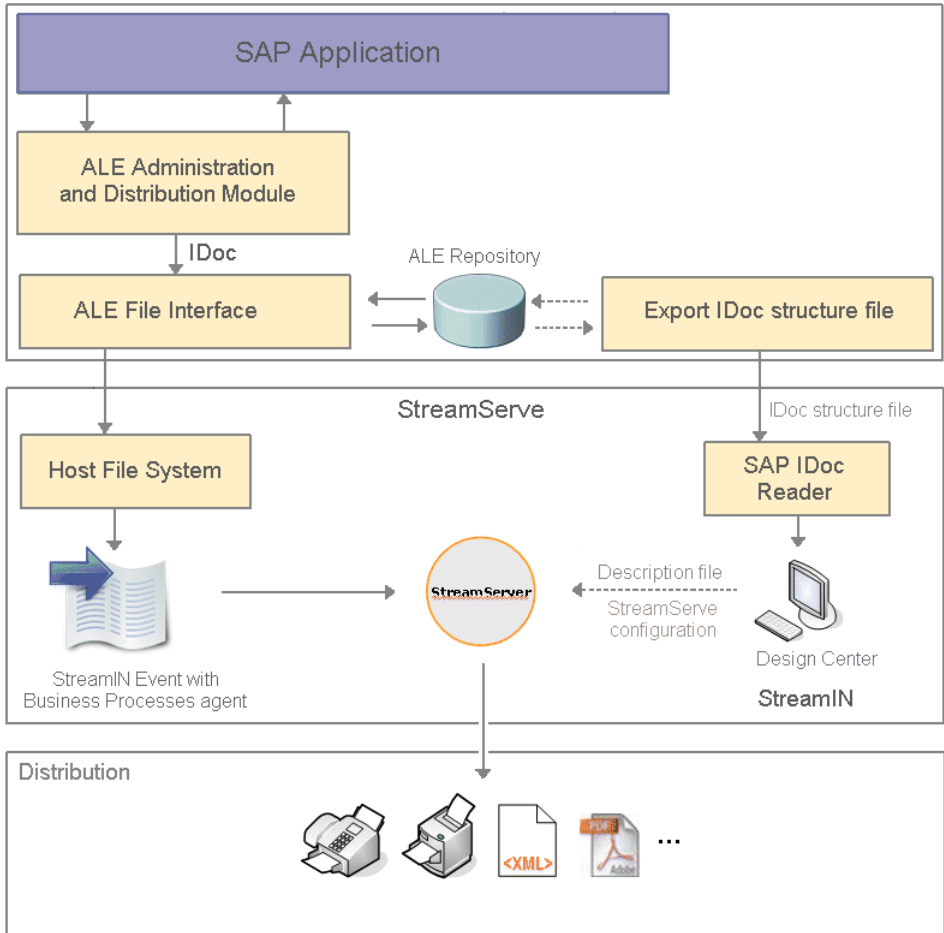


Figure 4 Receiving IDoc data from SAP using the IDoc file interface

## The SAP IDoc Reader

The SAP IDoc Reader is an add-on component to the StreamIN module, that enables you to build a StreamIN Event for an IDoc file. To use the SAP IDoc Reader, you load an IDoc structure file into the SAP IDoc Reader, then use the loaded data as a base to configure the Event in StreamIN.

When you close the StreamIN tool, the SAP IDoc Reader automatically creates a description file (\*.dsc) of the IDoc StreamIN Event. StreamServer uses the description file to interpret the incoming IDoc data during runtime.

## The IDoc ALE interface

The IDoc ALE interface enables you to use StreamServe to receive IDoc data from your SAP system, and send IDoc data from StreamServe to the same or different SAP system.

### SAP and the ALE interface

SAP provides functionality to enable distribution of data between different systems, and enables communication with specific partners for documents in electronic format.

This functionality is known as the ALE (Application Linking and Embedding) interface. The business documents interchanged using ALE are called *IDocs (Intermediate Documents)*. Routing IDocs to an external system is done in two ways - through an ALE Message Handler or ALE Converter. SAP communicates with both the ALE Message Handler and the ALE Converter using the transactional Remote Function Call (tRFC) interface.

#### **ALE Message Handler**

The ALE Message Handler acts as a bridge between SAP and other applications, with its main purpose being the handling and guaranteeing of IDoc delivery.

#### **ALE Converter**

The ALE Converter transforms either outgoing IDoc to a format suitable for the target application, or incoming non-IDoc format to IDoc format.

### ALE Certification and StreamServe

In October 2002, StreamServe was awarded SAP ALE Converter Certification for the suite of StreamServe Connect *for SAP* solutions. This means that the StreamServe Connect solutions are certified to exchange data with SAP systems, and to connect non-SAP systems to SAP solutions.

## SAP and the ALE Message Handler and ALE Converter

The following diagram illustrates how SAP communicates with both the ALE Message Handler and the ALE Converter using the transactional Remote Function Call (tRFC) interface.

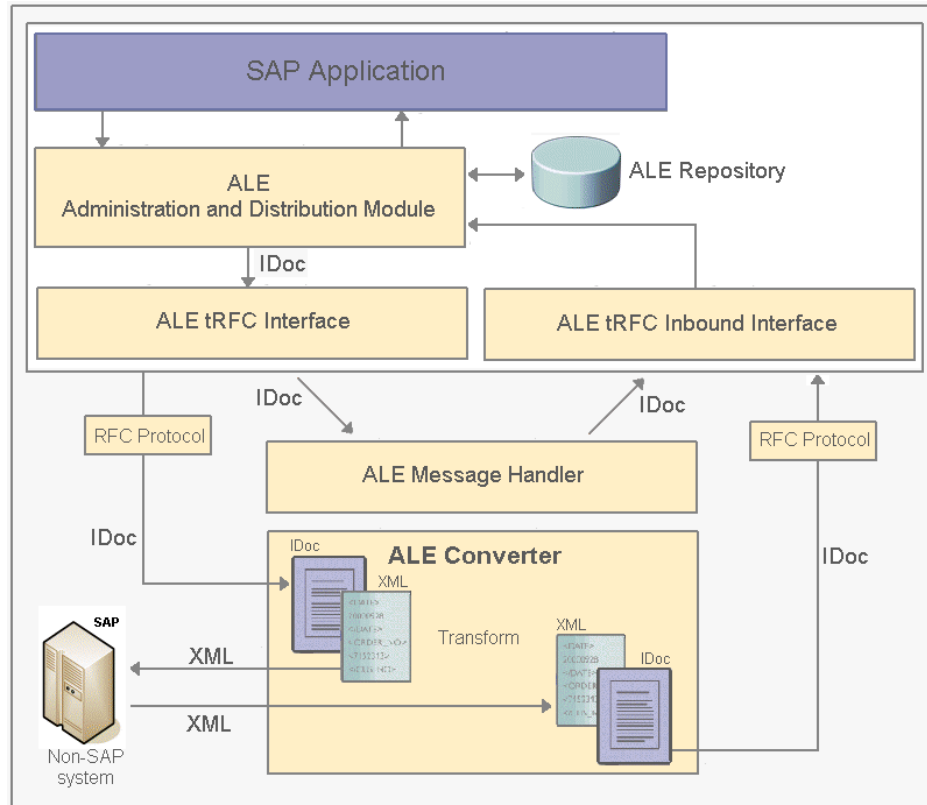


Figure 5 The ALE Message Handler and ALE Converter

**Note:** The Business Processes Connect solution only incorporates the ALE Converter handling of IDoc data, see [StreamServe and the ALE interface](#) on page 14.

## StreamServe and the ALE interface

There are two ways you can transfer IDoc data between StreamServe and your SAP system using the ALE interface:

- SAP IDoc outbound - StreamServe IDoc inbound
- SAP IDoc inbound - StreamServe IDoc outbound

### SAP IDoc outbound - StreamServe IDoc inbound

StreamServe receives IDoc data from the SAP system via the SAP IDoc Converter, and processes the IDoc data as a StreamIN Event. The data output is configured using any Process tool.

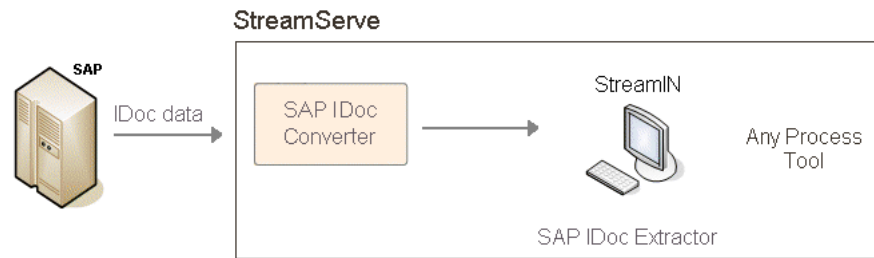


Figure 6 SAP IDoc outbound - StreamServe IDoc inbound

See [Configuring the IDoc ALE interface \(StreamServe inbound\)](#) on page 35.

### SAP IDoc inbound - StreamServe IDoc outbound

StreamServe receives data from any data source, and after processing, distributes the data in IDoc format to a SAP system. The data output is configured as an XMLOUT Process, and is sent to the SAP system via the SAP IDoc Converter that transforms the data to an IDoc format.

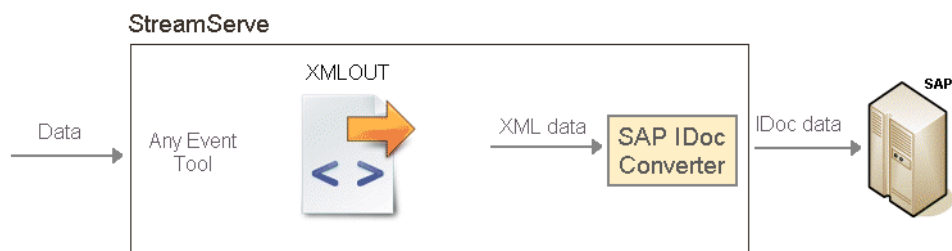


Figure 7 SAP IDoc inbound - StreamServe IDoc outbound

See [Configuring the IDoc ALE interface \(StreamServe outbound\)](#) on page 49.

### StreamServe and the ALE Converter

The Business Processes Connect solution incorporates the ALE Converter handling of IDoc data. This enables you to convert IDoc data from your SAP system to a format suitable for a target application, in this case, StreamServer, and then send the data to StreamServer for processing. The ALE Converter handling also enables you to take data from an external application, in this case, StreamServe, and convert the data to an IDoc format, which is then sent to the required SAP system.

**Note:** The Business Processes Connect solution does not support the ALE Message Handler component of the SAP ALE concept.

## Sending and receiving IDoc data using the ALE interface

The following diagram illustrates both the StreamServe inbound and StreamServe outbound scenarios, showing how StreamServe sends and receives IDoc data to and from the SAP system using the ALE interface.

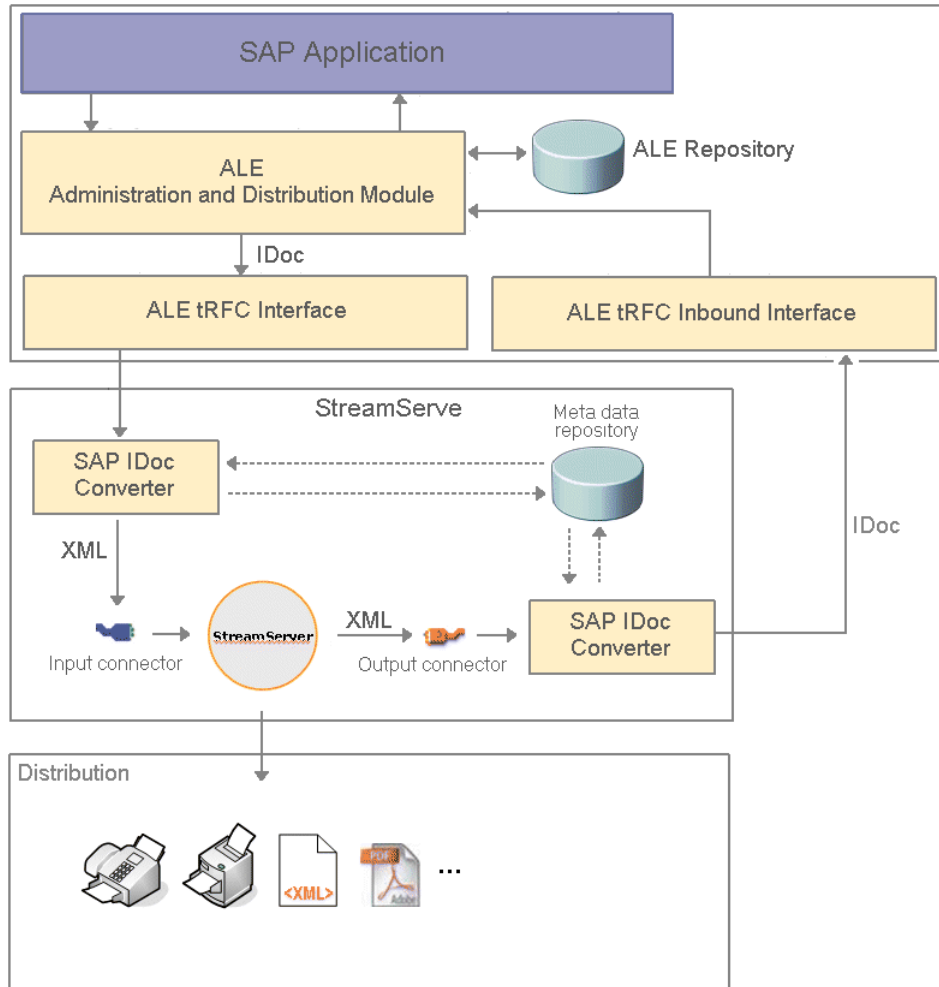


Figure 8 Sending and receiving IDoc data using the ALE Interface

### SAP IDoc outbound - StreamServe IDoc inbound

- 1 In this scenario, the ALE interface handles the IDoc transfer to the specified destination, sending the IDoc data to the SAP IDoc Converter. The SAP IDoc Converter acts as a link between the SAP system and StreamServer.
- 2 The SAP IDoc Converter transforms the data to an internal StreamServe format (XML) using meta data from the SAPBP repository.
- 3 The SAP IDoc Converter sends the XML file to StreamServer where the XML file is processed as a StreamIN Event. You can use a Directory, HTTP, or Service Channel (HTTP) input connector to receive the XML file.

### SAP IDoc inbound - StreamServe IDoc outbound

- 1 In this scenario, StreamServe receives data from any data source, and processes the data as an XMLOUT Process.
- 2 StreamServer outputs the IDoc data in XML format, and sends it to the SAP IDoc Converter via:
  - a directory that you specify
  - HTTP
  - the Service Broker Service Channel connector.
- 3 The SAP IDoc Converter receives the StreamServe XML, and uses the meta data from the SAPBP repository to convert the data from XML format to IDoc format, before sending the data to the SAP system.

### Configuring Messages for the IDoc ALE interface

The following diagram illustrates the design mode in StreamServe, showing how you can use the ALE repository to create a StreamIN Event to receive IDoc data, or an XMLOUT Process to send IDoc data to the SAP system.

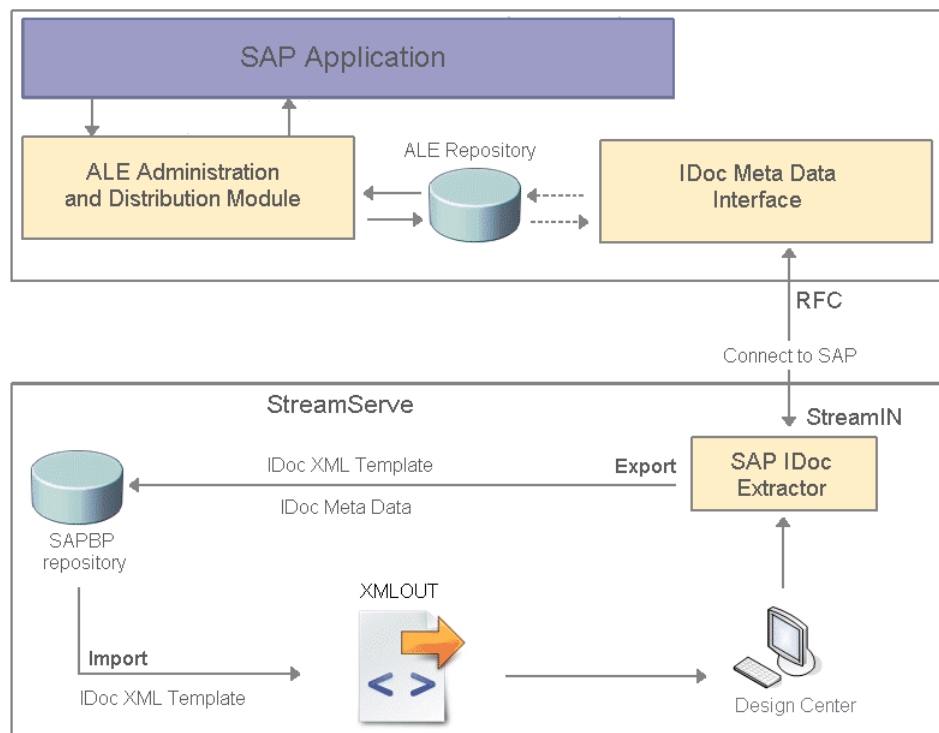


Figure 9 Configuring Messages for the IDoc ALE interface

## The SAP IDoc Converter

The SAP IDoc Converter is an add-on component included in the StreamServe Business Processes installation.

The main purposes of the SAP IDoc Converter are to:

- Receive incoming IDoc data sent from the source SAP system, and convert the IDoc data into a format that StreamServer can receive - StreamServe XML format. The SAP IDoc Converter can receive an incoming file that contains one or more IDocs.
- Transform the XML data from the XMLOUT Process to the IDoc format, and send it back to the SAP system.

The SAP IDoc Converter converts data between IDoc and StreamServe XML format using IDoc meta-data stored in the following repository directory:

`<StreamServe installation>\Applications\SAP connect\sapbp\meta`

**Note:** On UNIX, the meta template files are located in the `$STRS_HOME/meta` directory.

The default path to the meta directory can be overridden by setting a path in the argument file.

The SAP IDoc Converter is set up and run as a service in the Control Center. See [Configuring the IDoc Converter](#) on page 57.

## The SAP IDoc Extractor

The SAP IDoc Extractor is an add-on component to the StreamIN tool, that enables you to build a StreamIN Event for IDoc data from your SAP system.

In the StreamIN tool, you can use the SAP IDoc Extractor to connect to the SAP system to retrieve the IDoc, or you can load an IDoc meta file from the SAPBP repository into the Event. You can then use the IDoc data to build the Event.

See [Creating a StreamIN Event for IDoc file data](#) on page 29.

## The SAPBP repository

StreamServer uses the SAPBP repository when IDoc data is transferred between SAP and StreamServe using the ALE interface. The SAPBP repository contains two types of files - meta data files and template files.

### Meta data file

The meta data file is used by the StreamServe SAP IDoc Converter, and the SAP IDoc Extractor (a component of StreamIN).

#### SAP IDoc Converter

The SAP IDoc Converter uses the meta data file to transform:

- The in-coming IDoc data to a StreamServe XML format. The transformed file is then sent to the input connector for further StreamServe processing.
- The XML data from the XMLOUT Process to the IDoc format.

### **SAP IDoc Extractor**

The SAP IDoc Extractor component of StreamIN uses the meta data file to assist in creating the StreamServe Message. When you import the meta file into the SAP IDoc Extractor, you can use drag and drop functionality to create the Message from the meta file.

## **Template file**

The template file is imported into the XMLOUT tool, and used to create the IDoc that is sent to the SAP system. When you load the template file into XMLOUT, you can build the IDoc using drag and drop functionality.

## The BAPI interface

The BAPI interface comprises a number of script functions, which enable you to call any BAPI (Business Application Programming Interface) function available in the SAP system from StreamServe. With the Business Processes Connect solution, the BAPI script functions are integrated into the standard StreamServe scripting language.

BAPI is a SAP standard interface technology, which enables you to retrieve and update data in SAP systems using RFC (Remote Function Calls) functionality from external data sources. The Business Processes Connect solution supports all BAPIs using the RFC interface, providing system independent access to all business objects used in the SAP system.

See [Configuring the BAPI interface](#) on page 73.

# Configuring the IDoc file interface

---

This section describes how to configure the Business Processes Connect solution with your SAP system to use the IDoc file interface.

The IDoc File Interface enables you to send IDoc (Intermediate Document) data as a file from your SAP system to StreamServe for processing.

## Prerequisites

To use this section, your SAP system should already be configured to use IDocs, and you should have assistance from a person with SAP knowledge during the configuration phase.

## Required activities

To configure the Business Processes Connect solution to process IDoc data using the IDoc file interface, complete the following steps:

- **Configuring your SAP system for the IDoc file interface**  
See *Configuring SAP 4.6/4.7 for the IDoc file interface* on page 22
- **Sending IDoc file data to StreamServe**  
See *Sending IDoc file data from SAP to StreamServe* on page 27
- **Configuring StreamServe for IDoc file data**  
See *Configuring StreamServe for IDoc file data* on page 28

# Configuring SAP 4.6/4.7 for the IDoc file interface

## Required activities

- *Creating a port definition* on page 22
- *Creating a partner profile* on page 23
- *Generating an IDoc structure file* on page 23

## Creating a port definition

To send IDoc data from your SAP system, you need to create a port definition that acts as a channel for communication between SAP and StreamServe. To transfer IDoc data, you need to create at least one port definition.

### To create a port definition

- 1 Log on to your SAP system as a user with administrative permissions.
- 2 In the transaction box, enter `/nwe21`. The Ports in IDoc processing window opens.
- 3 From the Ports folder, select the **File** folder, then click **Create**. The Creating a file port window opens.
- 4 Specify the port settings.

Port settings	
<b>Port</b>	A name for the StreamServe port definition, such as ZSTRS.
<b>Description</b>	A description for the port definition, such as StreamServe Port.
<b>Version</b>	Ensure <b>Doc record types SAP Release 4.x</b> is selected.

- 5 On the Outbound file tab, select **physical directory**, and enter the directory where you want the file to be sent. You specify this directory for the StreamServe input connector. StreamServer will look for the IDoc file in this directory.
- 6 Click **Access test** to test whether the SAP application server is accessible.
- 7 In the Function module box, select the function module you want to use.
- 8 Save the port definition.

## Creating a partner profile

You need to create profiles of partners with whom you want to establish communication through IDoc data. For example, if IDoc data needs to be transmitted to a customer, you need to create a partner profile of ‘Customer’ type.

### Prerequisites

This section assumes the partner you want to communicate with is already configured in your SAP system. If the partner is not configured, you will need to create the partner first. See your SAP system documentation for information on creating partners.

### To create a partner profile

- 1 In the transaction box, enter `/nwe20`. The Partner profiles window opens.
- 2 In the Partner profiles folder, browse to the folder containing the partner you want to communicate with, and select the partner number. The partner’s details are displayed.
- 3 In the Outbound parameters table, select the type of messages that will be communicated with the specified partner, for example ORDERS.
- 4 Click the **Create Outbound Parameters** toolbar button. The Partner profiles window opens.
- 5 Specify the partner profile settings.

Partner profile settings	
<b>Partner function</b>	Enter <code>2C</code> for the IDoc File interface.
<b>Receiver port</b>	The name of the StreamServe port definition you created in <i>Creating a port definition</i> on page 22. For example, <code>ZSTRS</code> .
<b>Output mode</b>	Select <b>Transfer IDoc immed.</b>
<b>Basic type</b>	The IDoc basic type you want to use, for example, <code>ORDERS01</code> .

- 6 Save the partner profile.

## Generating an IDoc structure file

To use StreamServe to process IDoc data, you need to generate an IDoc structure file from your SAP system that you will use to build the StreamServe Message.

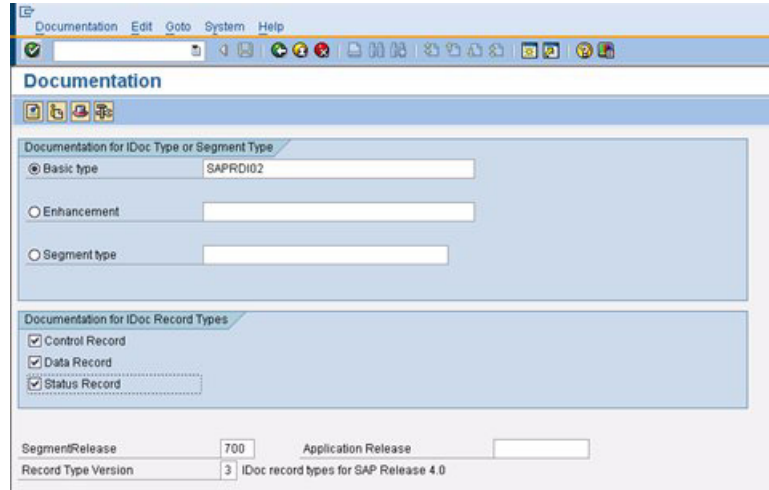
### To generate an IDoc structure file

- 1 In the transaction box, enter `/nwe60`. The Documentation IDoc Record Types window opens.
- 2 Ensure all Documentation for IDoc record types check boxes are selected.

## 24 | Configuring SAP 4.6/4.7 for the IDoc file interface

### Configuring the IDoc file interface

- 3 Select the **Basic types** check box, and select the IDoc type you want to generate a structure file for. For example, SAPRDI02.

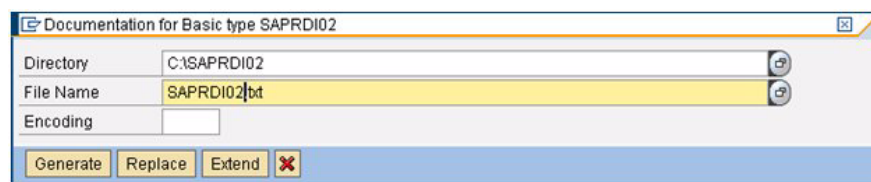


- 4 Click **Parse**.



The system displays the structure of the IDoc type you selected. The structure shown is the meta data of the IDoc. You will use this structure file to build the StreamIN Event.

- 5 Select **System > List > Save > Local File**. The Save list in file dialog box opens.
- 6 Select **unconverted** to specify the data is saved in unconverted format, and click **Enter**. The Transfer List to a Local File dialog box opens.
- 7 Specify the path and filename for the structure file. For example:



- 8 Click **Generate**. The structure of the IDoc is downloaded to the text file.
- 9 Import the file to the resource set as a **Sample** resource type.

You can import this file into the StreamIN tool to build the Event for the IDoc, see [Importing an IDoc structure file into StreamIN](#) on page 31.

---



#### **Generating IDoc documentation**

If you are unfamiliar with fields and elements in the IDoc file, you can generate HTML documentation of the IDoc data from your SAP system, which contains a description of all fields.

See [Generating IDoc documentation](#) on page 26.

---

## Generating IDoc documentation

If you are unfamiliar with the fields and elements in the IDoc file, you can generate HTML documentation of the IDoc data from your SAP system that contains a description of all fields.

**Note:** The following instructions are taken from SAP system release 4.6 - the transaction code is the same for all releases.

### To generate documentation for the IDoc file

- 1 Log on to your SAP system, as a user with administrative permissions.
- 2 In the transaction box, enter `/nwe60`. The Documentation for IDoc types window opens.
- 3 In the Obj name box, enter the object name that is the same as the IDoc basic type, for example, `ORDERS01`.
- 4 Click the **HTML** toolbar button. The Basic name for HTML files dialog box opens.
- 5 Save the IDoc documentation file in the same directory as that containing the IDoc file you downloaded in previous steps.

The SAP system saves the IDoc documentation file as three HTML files:

- `idoc_name_f.htm`
- `idoc_name_d.htm`
- `idoc_name_i.htm`

where `idoc_name` is the name of the IDoc, for example:

```
ORDERS01_d.htm  
ORDERS01_f.htm  
ORDERS01_i.htm
```

## HTML files containing IDoc details

The SAP system saves the IDoc documentation file as three HTML files.

### ***idoc\_name\_f.htm***

Contains the index file (`idoc_name_i.htm`) and the documentation file (`idoc_name_d.htm`). You can use this file to navigate to all field descriptions.

### ***idoc\_name\_d.htm***

Contains the actual description of the IDoc file, with a description of each segment and its fields (type, description and position), and the hierarchal structure of the segments.

### ***idoc\_name\_i.htm***

The index file for the IDoc on-line documentation. You can use this file to navigate to each segment and structure within the IDoc file.

## Sending IDoc file data from SAP to StreamServe

To send IDoc data as a file from your SAP system, you need to configure a port definition that acts as a channel between your SAP system and StreamServer, and create profiles of partners with whom you want to communicate IDoc data.

You send IDoc data from your SAP system as an output file defined for the port definition and the Business Processes fixed agent. StreamServer receives the data via a Message configured with the Business Processes agent.

StreamServer converts the IDoc data to an internal StreamServe format using a description file (\*.dsc) created from the StreamIN Event for the IDoc. StreamServer processes the data according to the configured StreamIN Event and Processes.

## Configuring StreamServe for IDoc file data

You can use the IDoc file interface to send an IDoc file from your SAP system to StreamServe for processing.

StreamServer receives the IDoc file as field-based data in a StreamIN Event. The Event is configured with a Business Processes agent which StreamServer uses to recognize and process the data correctly.

StreamServer uses a description file during runtime to interpret the IDoc data and process the file according to the specified output process.

**Note:** This guide only contains instructions specific for configuring the Business Processes Connect solution. For general information, see the standard StreamServe documentation.

### Required activities

To configure StreamServe to receive IDoc file data, complete the following steps:

- [Creating a StreamIN Event for IDoc file data](#) on page 29
- [Importing an IDoc structure file into StreamIN](#) on page 31
- [Building the StreamIN Event for IDoc file data](#) on page 32
- [Specifying the description file as an Event resource](#) on page 34

### Message for IDoc file data

To process IDoc file data from your SAP system, you use a Message containing a StreamIN Event and one or more Processes. The Processes can be of any type.

### StreamIN Event for IDoc file data

The StreamIN Event is configured with the Business Processes agent, which enables StreamServer to recognize and process the data correctly.

You build the StreamIN Event using a structure file (\*.txt) of the IDoc data. You generate the structure file from the IDoc data within the SAP system. You import the structure file into the StreamIN Event using the SAP IDoc Reader, an add-on component included in the StreamServe Business Processes installation.

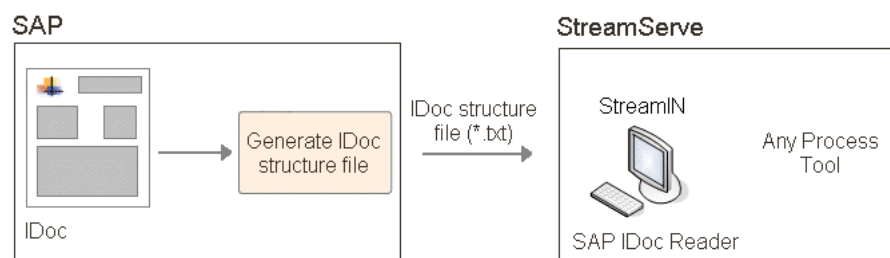


Figure 1 StreamIN Event for IDoc file data

## Creating a StreamIN Event for IDoc file data

To process IDoc file data from your SAP system, you use a StreamServe Message containing a StreamIN Event and one or more Processes. The StreamIN Event is configured with a Business Processes agent, which enables StreamServer to recognize and process the data correctly.

### To create a StreamIN Event for IDoc file data

- 1 In Design Center, create a new Message definition.
- 2 Add a StreamIN Event to the Message.
- 3 Name the new Event according to the IDoc data sent from your SAP system. See [Naming a StreamIN Event for IDoc data](#) on page 29.
- 4 Right-click the StreamIN Event and select **Settings**. The Event Settings dialog box opens.
- 5 On the Agent Settings tab, from the Input type list, select **StreamServe Connect for SAP - Business Processes**.
- 6 Click **OK**.

### Required activities

To configure the StreamIN Event, you need to complete the following steps:

- [Importing an IDoc structure file into StreamIN](#) on page 31
- [Building the StreamIN Event for IDoc file data](#) on page 32
- [Specifying the description file as an Event resource](#) on page 34

## Naming a StreamIN Event for IDoc data

StreamServer must be able to recognize different types of incoming documents and match them with the appropriate Event configuration. For IDoc data, the Event takes its name from the values of the DOCUMENTTYPE and MESSAGETYPE objects in the data.

Use the following syntax:

DOCUMENTTYPE\_MESSAGETYPE

For example:

orders01\_ordrsp



You must name the Event *before* you close StreamIN and save the Event, see [Generating the description file for the IDoc](#) on page 30.

---

The values of the objects are located in the first row of the IDoc data.

**SAP 3.1**

- DOCUMENTTYPE object starts at position 35 and is 8 characters long
- MESSAGETYPE object starts at position 417 and is 6 characters long.

**Note:** If the MESSAGETYPE object is not shown at the position, look for the object at position 164 (6 characters long).

**SAP 4.x**

- DOCUMENTTYPE object starts at position 39 and is 30 characters long
- MESSAGETYPE object starts at position 99 and is 30 characters long.

*Example 1 Naming a StreamIN Event for IDoc data*

From SAP release 4.x, where the Event name would be ORDERS01\_ORDRSP.

EDI_DC40	1000000000000019905446B	3012	ORDERS01	ORDRSP
E2EDK01005	10000000000000000000	199054000		
E2EDK14	10000000000000000000	199054000		
E2EDK14	10000000000000000000	199054000		
E2EDK14	10000000000000000000	199054000		
E2EDK14	10000000000000000000	199054000		
E2EDK03	10000000000000000000	199054000		
E2EDK03	10000000000000000000	199054000		
E2EDK03	10000000000000000000	199054000		
E2EDK03	10000000000000000000	199054000		
E2EDK03	10000000000000000000	199054000		
E2EDK04001	10000000000000000000	199054000	6.75	
E2EDKA1003	10000000000000000000	199054000		Sam
E2EDKA1003	10000000000000000000	199054000		Sam
E2EDKA1003	10000000000000000000	199054000		-

## Generating the description file for the IDoc

When you close the StreamIN tool and save the Event for an IDoc file, StreamServe automatically creates a description file (\*.dsc) based on and named after the Event. StreamServer uses this description file to convert the IDoc data from the SAP system to an internal StreamServe format.

The description file is located in the same directory the IDoc structure file was imported from.

When the description file has been generated, you need to specify the file as a resource for the Event, see [Specifying the description file as an Event resource](#) on page 34.



### Naming the Event before closing StreamIN

You must name the Event correctly *before* closing StreamIN, otherwise the description file will not have the correct name for the Event, and you will have to update the name of the file manually.

### To update the description file manually

- 1 Open the description file generated for the Event in a text editor.
- 2 Add the correct DOCUMENTTYPE and MESSAGETYPE data (with an underscore separator). See [Naming a StreamIN Event for IDoc data](#) on page 29.

*Example 2* Description file for IDoc data

---

```

IDOCSEVENT "ORDERS01_ORDRSP"
RECORD "E2OCLFM001" 1 ""
  FIELD "MSGFN" 000064 000066
  FIELD "OBTAB" 000067 000076
  FIELD "OBJEK" 000077 000126
  FIELD "KLART" 000127 000129
  FIELD "MAFID" 000130 000130
  FIELD "OBJECT_TABLE" 000131 000160
END
RECORD "E2KSSKM" 1 ""
  FIELD "MSGFN" 000064 000066
  FIELD "CLASS" 000067 000084
  FIELD "AENNR" 000085 000096
  FIELD "DATUV" 000097 000104
  FIELD "STATU" 000105 000105
  FIELD "STDCL" 000106 000106
END
RECORD "E2AUSPM001" 1 ""
  FIELD "MSGFN" 000064 000066
  FIELD "ATNAM" 000067 000096
  FIELD "AENNR" 000097 000108
  FIELD "DATUV" 000109 000116
  FIELD "ATWRT" 000117 000146
  FIELD "ATFLV" 000147 000168
  FIELD "ATAWE" 000169 000171
  FIELD "ATFLB" 000172 000193
  FIELD "ATAW1" 000194 000196
  FIELD "ATCOD" 000197 000197
  FIELD "ATTLV" 000198 000219
  FIELD "ATTLE" 000220 000241
  FIELD "ATPRZ" 000242 000242
  FIELD "ATINC" 000243 000264
  FIELD "ATAUT" 000265 000265
  FIELD "ATIMB" 000266 000275
  FIELD "ATZIS" 000276 000278
  FIELD "UDEF_CHAR" 000279 000308

```

---

## Importing an IDoc structure file into StreamIN

To use StreamIN to create a Message for IDoc file data, you need to import the structure file for the IDoc into the StreamIN tool. You import an IDoc structure file into StreamIN using the SAP IDoc Reader.

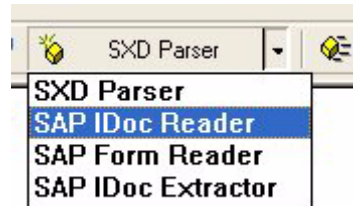
For information on generating an IDoc structure file, see [Generating an IDoc structure file](#) on page 23

### Prerequisites

- You have created a StreamIN Event configured with the Business Processes agent, see [Creating a StreamIN Event for IDoc file data](#) on page 29.
- The structure file is stored in the resource set.

### To import an IDoc structure file into the StreamIN

- 1 Open the StreamIN Event.
- 2 From the active connectors list in the Integration Tool browser, select **SAP IDoc Reader**.



- 3 Click **SAP IDoc Reader**. The Select Resource dialog box opens.
- 4 Locate the resource containing the IDoc structure file you want to import. In the File Name box, enter the name of the file and click **OK**.
 

**Note:** This file can have any extension. If the file has an extension other than \*.txt, select **All files (\*.\*)** in the Files of type list.

## Building the StreamIN Event for IDoc file data

When you have imported the IDoc structure file into StreamIN, all fields from the imported data are shown in a block structure in the Integration Tool Browser.

For information on building StreamIN Events, see the *StreamIN* documentation.

### Prerequisites

You have imported an IDoc file into a StreamIN Event, see [Importing an IDoc structure file into StreamIN](#) on page 31.

### To configure the StreamIN Event for IDoc file data

- 1 Open the StreamIN Event.
- 2 In the Integration Tool browser, open the structure of the imported data to show all blocks and fields.
- 3 From the Integration Tool browser, drag and drop the blocks or fields you want to include from the IDoc file to the appropriate location in the Message browser.



Analyze the data and determine whether to place data in a block or as fields directly under the Message (root level). Generally, you should place fields that could have multiple occurrences in the data (IDoc) in a block. You should place all fields that appear only once, for example, fields that contain an invoice date or number, at the root level directly under the Message.

- 4 Save and close the StreamIN Event.
- 5 Select to generate a description file.

- 6** In the **Select or create a resource to save the description in dialog**, create a new resource of **Description** type.
- 7** In the Enter name dialog, call the file the same name as the Event plus the `.dsc` suffix.
- 8** Double-click the new resource.
- 9** You need to specify the description file as a resource for the Event, see *Specifying the description file as an Event resource* on page 34.

## Multiple Messages for an IDoc basic type

If you have different Messages based on the same IDoc basic type (DOCUMENTTYPE), you need to create an Event for each of them.

### *Example 3 Multiple Messages for IDoc basic type*

---

ORDERS01 (DOCUMENTTYPE) is used for Sales Orders, Purchase Orders, and changes and confirmations of these documents. You would need to create the following Events to handle these Messages:

- ORDERS01\_ORDERS for order creation messages
- ORDERS01\_ORDRSP for order acknowledgement
- ORDERS01\_ORDCHG for order change messages

To achieve this scenario, you would only need to download the IDoc description file once, but you would need to incorporate the description file into each Event and save a description file (`*.dsc`) for each Event. For example:

- `orders01_orders.dsc`
  - `orders01_ordrsp.dsc`
  - `orders01_ordchg.dsc`
- 

## Comparing Messages

In the SAP IDoc Reader you can use **Difference** mode to compare the current Event (Message) with an imported Message, and display the differences.

Comparing Messages can be useful, for example, if you have built your own Invoice template Message, you could import a new Invoice SAPscript Form into the SAP Form Reader and compare the new Form with your existing Message. If you have added three new fields in the new SAPscript Form, those three fields are indicated as different (**Diff**) in the SAP IDoc Reader.

### **To set Difference mode**

- 1** In the StreamIN Integration Tool browser, click **Mode** and select **Diff**. (Normal is the default.)
- 2** Select **StreamIN > Import Message** to import a new Message. The Import Message dialog box opens.

- 3 Click **Browse**. The Open dialog box opens.
- 4 Select the dictionary you want to use to create the Message structure and click **Open**.

If there are items in the dictionary that StreamServe cannot process, or that already exist in the Message structure, they will be removed before the Message structure is created. The SAP IDoc Reader indicates objects that differ from those already in the current Event with **(Diff)** shown after the name of the object.

## Specifying the description file as an Event resource

When you close the StreamIN tool and save the Event for an IDoc file, StreamServe automatically creates a description file (\*.dsc) based on and named after the Event. StreamServer uses this description file to convert the IDoc data from the SAP system to an internal StreamServe format.

The description file is located in the same directory the IDoc structure file was imported from.

You need to specify the description file as a resource for the Event. When you export the Message, the description file will automatically be inserted as an argument in StreamServer start-up argument file.

### To specify the description file as a resource for the Event

- 1 In Design Center, right-click the StreamIN Event and select **Settings**. The Event Settings dialog box opens.
- 2 In the Description resource field, click the browse button. The Select description dialog box opens.
- 3 Browse to the description file generated for the Event.
- 4 Double-click on the description file to select the file and close the browser.
- 5 On the Event Settings dialog box, click **OK**.

# Configuring the IDoc ALE interface (StreamServe inbound)

---

This section describes how to configure the Business Processes Connect solution to use the IDoc ALE interface to receive IDoc data from your SAP system. This is called the SAP IDoc outbound - StreamServe IDoc inbound scenario.

## SAP IDoc outbound - StreamServe IDoc inbound scenario

In the SAP IDoc outbound - StreamServe IDoc inbound scenario, StreamServe receives IDoc data from the SAP system via the SAP IDoc Converter, and processes the IDoc data as a StreamIN Event.

The IDoc ALE interface uses the StreamServe SAP IDoc Converter as a link between the SAP system and StreamServer. Before configuring StreamServe, you need to configure the SAP system for ALE distribution and IDoc integration.

## Prerequisites

Before configuring StreamServe, you need to configure the SAP system for ALE distribution and IDoc integration.



Configuring the SAP system for ALE distribution and IDoc integration can be a complex task requiring SAP expertise. This section provides an overview describing how the StreamServe integration works with the SAP environment. It does not provide a complete guide describing how to set up the ALE distribution or the IDoc integration of application data in the SAP system. Only a person with high SAP expertise should perform these configurations steps.

---

## Required activities

To configure SAP and StreamServe for the SAP IDoc outbound - StreamServe IDoc inbound scenario, you need to complete the following steps:

- **Configuring SAP to send IDoc data**  
See [Configuring SAP 4.6/4.7 for the IDoc ALE interface \(SAP outbound\)](#) on page 36
- **Configuring the SAP IDoc Converter**  
See [Configuring the IDoc Converter](#) on page 57
- **Configuring StreamServe to receive IDoc data**  
See [Configuring StreamServe to receive IDoc data](#) on page 39

## Configuring SAP 4.6/4.7 for the IDoc ALE interface (SAP outbound)

### Required activities

- [Creating an RFC server destination](#) on page 36
- [Creating a port definition](#) on page 37
- [Creating a partner profile](#) on page 37

## Creating an RFC server destination

Because communication between the SAP system and the StreamServe SAP IDoc Converter uses the Transactional RFC protocol, you need to define an RFC server destination in the SAP system that recognizes the SAP IDoc Converter. The IDoc Converter runs as a registered RFC server.

### To create an RFC server destination

- 1 Logon to your SAP system as a user with administrative permissions.
- 2 In the transaction box, enter `/nsm59`. The Display and Maintain RFC Destinations window opens.
- 3 Select the TCP/IP Connections folder, then click **Create**. The RFC Destination window opens.
- 4 Specify the RFC server destination settings.

RFC server destination settings	
<b>RFC Destination</b>	A name for the RFC destination, such as <code>STRSSAPIDOCCONVERTER</code> .
<b>Connection Type</b>	Select <b>T</b> . Enter a description for the connection type, such as <code>TCP/IP Connection</code> .
<b>Description</b>	A description for the RFC destination. This destination establishes a connection to the StreamServe SAP IDoc Converter.

- 5 Click **Enter**. The RFC Destination window shows the new RFC destination.
- 6 Click **Registration**.
- 7 In the Program field, enter the program ID for this RFC destination, such as `STRSSAPIDOCCONVERTER1`. This program ID must be a unique ID for the SAP gateway you are using, and must match the ID configured in the StreamServe IDoc Converter.
- 8 Click **Enter**.
- 9 Save the RFC server destination.

## Creating a port definition

In order to use the connection between the SAP IDoc Converter and the SAP gateway, the connection must be attached to a port definition.

The port definition acts as a channel for communication between external system using EDI process. To transfer IDoc data, you need to create at least one port definition.

### To create a port definition

- 1 In the transaction box, enter `/nwe21`. The Port Definition window opens.
- 2 From the Ports folder, select the Transactional RFC folder, then click **Create**. The Ports in IDoc processing dialog box opens.
- 3 Select either **Generate port name** or **Own port name** to specify whether you want to use a system-generated name, or enter your own name. If you select **Own port name**, enter the port name.
- 4 Click **Enter**. The Creating a tRFC port window opens.
- 5 Specify the port settings.

Port settings	
<b>Description</b>	A description for the port definition, for example, StreamServe Port.
<b>Version</b>	Ensure <b>IDoc record types SAP Release 4.x</b> is selected.
<b>RFC destination</b>	The RFC destination you created in <i>Creating an RFC server destination</i> on page 36, for example, STRSSAPIDOC CONVERTER.

- 6 Save the port definition.

## Creating a partner profile

You need to create profiles of partners with whom you want to establish outbound and inbound communication through IDoc data. For example, if IDoc data needs to be sent to a ‘Customer’, you need to create a partner profile of customer type.

### Prerequisites

This section assumes the partner you want to communicate with is already configured in your SAP system. If the partner is not configured, you will need to create the partner first. See your SAP system documentation for information on creating partners.

### To create a partner profile

- 1 In the transaction box, enter `/nwe20`. The Partner Profiles window opens.

**Configuring the IDoc ALE interface (StreamServe inbound)**

- 2 Select the partner you want to use.
- 3 In the Outbound parameters table, select the type of messages that will be communicated with the specified partner, for example ORDERS.
- 4 Click the **Create Outbound Parameters** toolbar button. The Partner profiles window opens.
- 5 Specify the partner profile settings.

<b>Partner profile settings</b>	
<b>Partner function</b>	Enter 2B for the IDoc ALE interface.
<b>Message type</b>	The type of messages you will communicate with the partner, for example ORDERS.
<b>Output mode</b>	Select <b>Transfer IDoc immed.</b>
<b>Receiver port</b>	The name of the StreamServe tRFC port definition you created for the outbound options in <i>Creating a port definition</i> on page 37, for example, StreamServe Port.
<b>Basic type</b>	The IDoc basic type, for example, ORDERS01.

- 6 Save the partner profile.

## Configuring StreamServe to receive IDoc data

In the SAP IDoc outbound - StreamServe IDoc inbound scenario, StreamServe receives IDoc data from the SAP system via the SAP IDoc Converter, and processes the IDoc data as a StreamIN Event. The SAP IDoc Extractor is an add-on module to the StreamIN tool.

**Note:** This guide only contains instructions specific for configuring the Business Processes Connect solution. For general information, see the standard StreamServe documentation.

### Platform for StreamServe to receive IDoc data

In the SAP outbound - StreamServe inbound scenario, your SAP system sends IDoc data to StreamServer via an input connector using the StreamServe Service Broker. See [Configuring an input connector to receive IDoc data](#) on page 39.

### Message for StreamServe to receive IDoc data

To process IDoc file data from your SAP system, you use a Message containing a StreamIN Event, and one or more Processes. The StreamIN Event is configured to receive XML data. The Processes can be of any type.

In this scenario, there are two ways to build the StreamIN Event for the IDoc. Using the SAP IDoc Extractor, you can either import an IDoc meta file provided in the StreamServe setup, or make a connection to the SAP system to retrieve an IDoc. See [Creating a StreamIN Event to receive IDoc data](#) on page 42.

You can use any StreamServe Process to configure the layout for the Message output, such as PageOUT or XMLOUT.

## Configuring an input connector to receive IDoc data

In the SAP outbound - StreamServe inbound scenario, your SAP system sends IDoc data to StreamServer via an input connector.

You can use the following input connectors to receive IDocs:

- Directory
- Service Channel (HTTP)
- HTTP

### To configure a Service Channel (HTTP) input connector to receive IDoc data

- 1 In Design Center, add a new Platform.
- 2 Add an input connector to the Platform.
- 3 Right-click the input connector, and select **Settings**. The Input Connector Settings dialog box opens.
- 4 From the Connector list, select **Service Channel (HTTP)**.

- 5 Specify the input connector settings.

Input connector settings	
<b>Service Description</b>	Enter <code>SAPIDoc</code> . <b>Note:</b> The service description must match the name of the InData service specified for the SAP IDoc Converter. See <a href="#">Configuring the IDoc Converter</a> on page 57.
<b>Version</b>	The version of the service. There can be several versions of the same service. A client can request a specific version of a service. A client that sends a request without specifying a version, will get the highest available version of the service.

- 6 Click **OK**.
- 7 Right-click the Platform window and select **Configure Platform**.
- 8 Select **Service Broker**, and check that the settings are valid.
- 9 Click **OK**.

#### To configure a Directory input connector to receive IDoc data

- 1 In Design Center, add a new Platform.
- 2 Add an input connector to the Platform.
- 3 Right-click the input connector, and select **Settings**. The Input Connector Settings dialog box opens.
- 4 From the Connector list, select **Directory**.
- 5 Specify the input connector settings.

Connector settings	
<b>Match Criterion</b>	The path to the directory, for example: <code>scandir/*.txt</code> Use an absolute path, or a path relative to the export directory.
<b>Sort By</b>	How to sort files when reading from the directory. Two or more files with the same value (for example the same file extension) will be sorted by file name in ascending alphabetical order.

- 6 Click **OK**.

**To configure an HTTP input connector to receive IDoc data**

- 1 In Design Center, add a new Platform.
- 2 Add an input connector to the Platform.
- 3 Right-click the input connector, and select **Settings**. The Input Connector Settings dialog box opens.
- 4 From the Connector list, select **HTTP**.
- 5 Specify the input connector settings.

<b>Input connector settings</b>	
<b>HTTP Version</b>	The HTTP version to use with this connector. Auto means that the version is determined by the client's request.
<b>Address</b>	An alternative network address for this StreamServer, if you do not want to use the default network address for the workstation. For example, IP address to a specific network card.
<b>Port</b>	Port this connector listens to for HTTP requests. Two or more HTTP(S) connectors cannot share the same Port. You must use a unique port for each connector.
<b>Input threads</b>	Maximum number of concurrent connections on this connector.
<b>Idle timeout</b>	Maximum length of time (milliseconds) the server accepts to wait when it expects a request from a client.
<b>Timeout</b>	Maximum length of time (milliseconds) the server accepts to be idle before it shuts down the connection. As soon as any fragment of data is exchanged, the time-out starts over again. This means that all data does not have to be exchanged during this period.
<b>Response timeout</b>	Maximum length of time (milliseconds) a client is expected to wait for a response from the server.
<b>Authentication</b>	<p><b>None</b> – Do not use authentication.</p> <p><b>Basic</b> – Send authentication parameters as clear text. This is the only scheme supported in HTTP/1.0.</p> <p><b>Digest</b> – Send authentication parameters as a checksum over the network. Requires HTTP/1.1.</p> <p>See <a href="#">RFC 2617</a>.</p>
<b>Publish directory</b>	The root directory for stored files.

Input connector settings	
<b>Publish extension file</b>	Maps file formats to content-types.
<b>Job resource URI</b>	URI that points to an output file stored via a Job Resource output connector. For example:  /jx

6 Click **OK**.

## Creating a StreamIN Event to receive IDoc data

An Event describes the data contained in the IDoc data from your SAP system, and extracts information from the data for use in StreamServe. For IDoc data, this Event is configured as a StreamIN Event. The StreamIN Event is configured to receive XML data.

### To create a StreamIN Event for IDoc data

- 1 In Design Center, create a new Message definition.
- 2 Add a StreamIN Event to the Message.
- 3 Name the Event according to the IDoc data sent from your SAP system. See [Naming a StreamIN Event for IDoc data](#) on page 42.
- 4 Right-click the StreamIN Event and select **Settings**. The Event Settings dialog box opens.
- 5 On the Agent Settings tab, select **StrsXML** from the Input type list.
- 6 Ensure **Use Path** and **Use XSLT** are not selected. In XML input documents, you will use the name attribute of field elements as field names, not the path attribute.
- 7 Click **OK** and open the Event.

You can now import IDoc data into StreamIN to build the Event, see [Importing IDoc data into StreamIN](#) on page 43.

## Naming a StreamIN Event for IDoc data

StreamServer must be able to recognize different types of incoming documents, and match them with the appropriate Event configuration. For IDoc file data, the Event takes its name from the values of the IDOCTYPE and MESSAGETYPE objects in the data, and if you are using extended SAP basic IDoc types, also from the values of the CIMTYP (Extended Type) objects.

### Standard Events

A standard Event takes its name from values of the following objects:

- MESSAGETYPE\_IDOCTYP



- Import an IDoc using existing StreamServe meta-data files  
The Business Processes Connect solution includes meta template files for most SAP IDocs, which you can import into the StreamIN tool to create the Message.  
See [Importing IDoc data using StreamServe meta-data files](#) on page 44.
- Retrieve an IDoc directly from the SAP system  
If you can not find the IDoc you want to use among those provided, or if you want to use a customized IDoc, you should connect to the SAP system and retrieve the particular IDoc directly.  
See [Retrieving an IDoc meta-data file from SAP](#) on page 44.

## Importing IDoc data using StreamServe meta-data files

The StreamServe Business Processes installation includes meta template files for most SAP IDocs. These meta template files are located in the following directory:

```
<StreamServe installation>\Applications\SAP connect\sapbp\meta
```

**Note:** On UNIX, the meta template files are located in the `STRS_HOME/meta` directory. Make sure that all meta files have been stored in this directory.

If the SAP IDoc you want to use is included in the templates meta directory of the Business Processes installation, you can import the IDoc meta file directly into StreamIN using the SAP IDoc Extractor.

**Note:** When you import an existing IDoc meta file into StreamIN, you do not need to convert the imported IDoc data to XML format.

### To import an IDoc using a StreamServe meta-data file

- 1 In the StreamIN tool, click **SAP IDoc Extractor** in the Integration Tool browser. The Open dialog box opens.
- 2 Browse to the StreamServe directory containing the meta files, and select the IDoc meta file you want to import.
- 3 Click **Open**. The IDoc is loaded into the Integration Tool browser. You can now use the IDoc data to build the StreamIN Event.

**Note:** If you imported an existing IDoc, you do not need to transform the imported IDoc to XML format.

You can now use StreamIN to configure the Event for the IDoc. See [Configuring the StreamIN Event for IDoc data](#) on page 47.

## Retrieving an IDoc meta-data file from SAP

If you can not find the IDoc you want to use among those provided, or if you want to use a customized IDoc, you should connect to the SAP system and retrieve the IDoc you want directly.

You can retrieve an IDoc from the SAP system with or without knowing the IDoc name.

**Requirements**

If you are retrieving an IDoc directly from the SAP system, you must also create an XML formatted meta file based on the IDoc. The SAP IDoc Converter uses this meta file during runtime to transform the IDoc. See [Exporting a meta file for an IDoc](#) on page 46.

**To connect to the SAP system**

- 1 In the StreamIN tool, click **SAP IDoc Extractor** in the Integration Tool browser. The Open dialog box opens.
- 2 Click **Cancel** to close the dialog box.
- 3 Click **Connect to SAP**, and select **Logon**. The Logon dialog box opens.
- 4 Specify the SAP Logon settings.

<b>SAP Logon settings</b>	
<b>Destination</b>	Uses the destination specified in the <code>saprfc.ini</code> file to connect to the SAP system. For example, RFCEXT_R. Enter the destination in the box provided.  <b>Note:</b> You can also specify the connection type and other RFC specific parameters in the <code>saprfc.ini</code> file, which contains logon information. The SAP IDoc Converter can only use a type R destination.
<b>Full Logon</b>	Uses a full logon to connect to the SAP system. Specify the following:  <b>Hostname</b> - The IP address or host name of the SAP application server.  <b>Server Number</b> - The system number you want to access, for example 00.
<b>Client</b>	The name of the client you want to use.
<b>User Name</b>	A valid user name.
<b>Password</b>	The password for the user name. This password is not stored anywhere, but is available at the next logon.
<b>Language</b>	The abbreviation for the preferred language.

- 5 Click **OK**.

If the logon to the SAP system was successful, you are able to list IDocs.

**Note:** If you close the StreamIN tool, the connection to the SAP system will also be closed and you will have to re-connect to the SAP system when accessing another template.

#### To retrieve an IDoc when you know the IDoc name

Before you can retrieve an IDoc meta-data file from a SAP system, you must connect to the SAP system. See [To connect to the SAP system](#) on page 45.

- 1 Connect to the SAP system. See [To connect to the SAP system](#) on page 45.
- 2 Click **Connect to SAP**, and select **Retrieve IDoc**.
- 3 Specify the IDoc details.

IDoc settings	
<b>IDoc Name</b>	The basic type name of the IDoc, for example ORDERS01.
<b>Extended Type</b>	If you are using an extended SAP basic IDoc type, enter the CIMTYP extension type.
<b>SAP release</b>	The SAP release you used when sending/posting the IDoc.

- 4 Click **OK**.

The IDoc is imported into the Integration Tool browser, and you can use StreamIN to configure the Event for the IDoc. See [Configuring the StreamIN Event for IDoc data](#) on page 47.

#### To retrieve an IDoc when you do not know the IDoc name

- 1 Connect to the SAP system. See [To connect to the SAP system](#) on page 45.
- 2 Click **Connect to SAP**, and select **Display IDoc list**. The Display IDoc list shows all available IDocs, from which you can select the IDoc you want to use.

The IDoc is imported into the Integration Tool browser, and you can use StreamIN to configure the Event for the IDoc. See [Configuring the StreamIN Event for IDoc data](#) on page 47.

## Exporting a meta file for an IDoc

If you are retrieving an IDoc directly from the SAP system, you must also create an XML formatted meta file based on the IDoc, and store this file in the same directory as the StreamServe IDoc templates. The SAP IDoc Converter uses this meta file during runtime to transform the IDoc.

When you have imported the IDoc from the SAP system into the SAP IDoc Extractor, you can use the Extractor to create the required meta file.

**To create a meta file for the IDoc**

- 1 With the IDoc imported in the SAP IDoc Extractor, click **Transform IDoc to XML** in the Integration Tool browser.
- 2 Select one of the following transform options according to the type of file you want to create:

File type transform options	
<b>Meta file</b>	Transforms the IDoc meta-data file to an XML formatted meta file. The SAP IDoc Converter uses this meta file during runtime to transform the IDoc.
<b>Template file</b>	Exports the current IDoc as an XML formatted template file. You can import this template file into XMLOUT tool to configure a StreamServe Process.
<b>Meta and Template files</b>	We recommend you select this option to export both the current IDoc as both an XML formatted meta and template file.

The Save As dialog box opens.

- 3 Save the template file in the same directory as the StreamServe IDoc templates. By default:

```
<StreamServe installation>
\Applications\StreamServer\<version>\Common\data
\sapbp\template
```

- 4 Save the meta-data file in the same directory as the StreamServe IDoc meta file. By default:

```
<StreamServe installation>\Applications\SAP connect\sapbp\meta
```

## Configuring the StreamIN Event for IDoc data

When you have imported the IDoc data into StreamIN, all fields from the imported data are shown in a block structure in the Integration Tool Browser. You can use the imported data to build the StreamIN Event for the IDoc.

For detailed information on configuring StreamIN Events, see the *StreamIN* documentation.

### Prerequisites

You have created a StreamIN Event configured with the Business Processes agent, see [Creating a StreamIN Event to receive IDoc data](#) on page 42.

You have imported an IDoc file into the Event, see [Importing IDoc data into StreamIN](#) on page 43.

### To configure the StreamIN Event for IDoc data

- 1 In Design Center, open the Event.

- 2 In the Integration Tool browser, open the structure of the imported data to show all blocks and fields.
- 3 From the Integration Tool browser, drag and drop the blocks or fields you want to include from the IDoc data to the appropriate location in the Message browser.



Analyze the data and determine whether to place data in a block or as fields directly under the Message (root level). Generally, you should place fields that could have multiple occurrences in the data (IDoc) in a block. You should place all fields that appear only once, for example, fields that contain an invoice date or number, at the root level, that is directly under the Message.

---

- 4 Save and close the StreamIN Event.

# Configuring the IDoc ALE interface (StreamServe outbound)

---

This section describes how to configure the Business Processes Connect solution with your SAP system, to use the IDoc ALE interface to send IDoc data from StreamServe to a SAP system. This is called the SAP IDoc inbound - StreamServe IDoc outbound scenario.

## SAP IDoc inbound - StreamServe IDoc outbound scenario

In this scenario, StreamServe receives non-SAP data from any data source, and after processing, distributes the data in IDoc format to a SAP system. The data output is configured as an XMLOUT Process, and is sent to the SAP system via the SAP IDoc Converter that transforms the data to an IDoc format.

In this scenario, StreamServe can also receive IDoc data from a SAP system, and process and send the IDoc data to the same or different SAP system.

## Prerequisites

Before configuring StreamServe, you need to configure the SAP system for ALE distribution and IDoc integration.



Configuring the SAP system for ALE distribution and IDoc integration can be a complex task requiring a high degree of SAP expertise. This section provides an overview describing how the StreamServe integration works with the SAP environment. It does not provide a complete guide describing how to set up the ALE distribution, or the IDoc integration of application data in the SAP system. Only a person with high SAP expertise should perform these configurations steps.

---

## Required activities

To configure SAP and StreamServe for the SAP IDoc inbound - StreamServe IDoc outbound scenario, complete the following steps:

- [Configuring SAP for the IDoc ALE interface \(SAP inbound\)](#) on page 50
- [Configuring StreamServe to send IDoc data](#) on page 51

## Configuring SAP for the IDoc ALE interface (SAP inbound)

To configure your SAP system to use the IDoc ALE interface to receive IDoc data sent from StreamServe, you need to create the profile of partners with whom you have to establish communication through IDoc data.

### Creating a partner profile

You need to create profiles of partners with whom you have to establish communication through IDoc data. For example, if IDoc data needs to be received from a business partner, you need to create a partner profile of Partner type.

#### Prerequisites

These steps assume the partner you want to communicate with is already configured in your SAP system. If the partner is not configured, you will need to create the partner first. See your SAP system documentation for information on creating partners.

#### To create a partner profile

- 1 In the transaction box, enter `/nwe20`. The Partner Profiles window opens.
- 2 Select the partner you want to communicate with.
- 3 Click **Inbound parameters**. The Change View window opens.
- 4 In the Process code box, select the process code you want to use.

**Note:** You must use a unique code for a communication process in order to guarantee the integrity of the data to be transferred. This process code is used as the name given to the ways and means of processing an incoming IDoc, i.e. the process code's attributes.

If the process code is not available, contact your SAP contact person for assistance.

- 5 Save the partner profile.

## Configuring StreamServe to send IDoc data

In the SAP IDoc inbound - StreamServe IDoc outbound scenario, StreamServe receives data from any data source, and after processing, distributes the data in IDoc format to a SAP system. The data output is configured as an XMLOUT Process, and is sent to the SAP system via the SAP IDoc Converter that transforms the data to an IDoc format.

StreamServe can receive any type of data from a data source, convert the data to an IDoc format, then send the IDoc data to a SAP system.

**Note:** This guide only contains instructions specific for configuring the Business Processes Connect solution. For general information, see the standard StreamServe documentation.

### Platform for StreamServe to send IDoc data

When using StreamServe to send IDoc data to a SAP system, the output is distributed via the SAP IDoc Converter, a service developed for Business Processes to transform data to an IDoc format before being sent to a SAP system.

See [Configuring the IDoc Converter](#) on page 57.

### Message for StreamServe to send IDoc data

In the SAP IDoc inbound - StreamServe IDoc outbound scenario, you use a Message containing an Event and one or more XMLOUT Processes. The Event can be of any type.

See [Creating an XMLOUT Process to send IDoc data](#) on page 51.

## Creating an XMLOUT Process to send IDoc data

A Process describes how the final output from StreamServer will look. The Process contains the same variable data (data objects) as the Event, but in a format and with a layout of your choice. When configuring StreamServe to send SAP IDoc data to a SAP system, you should use an XMLOUT Process.

### Prerequisites

Before you can create an XMLOUT Process, you must install the XMLOUT tool. See the *XMLOUT* documentation.

### To create an XMLOUT Event to send IDoc data

- 1 In Design Center, create a new Message definition.
- 2 Add an XMLOUT Process to the Message.
- 3 Open the Process. The XMLOUT tool opens and displays the Process.

You can now configure the Process using the XMLOUT tool. See [Configuring an XMLOUT Process to send IDoc data](#) on page 52.

## Configuring an XMLOUT Process to send IDoc data

When configuring a Process to send SAP IDoc data to a SAP system, you should use an XMLOUT Process. StreamServe provides templates that you can use to configure an XMLOUT Process to send IDoc data.

### To configure an XMLOUT Process to send IDoc data

- 1 Open the XMLOUT Process you created in *Creating an XMLOUT Process to send IDoc data* on page 51. The XMLOUT tool opens showing the Options dialog box.
- 2 From the Output Format list, select **Raw**.
- 3 Click **Tools > Import XML File**.
- 4 Right click and select **Import**. Locate the directory containing the template file, by default in:
 

```
<StreamServe
installation>\Applications\StreamServer\<version>\Common\data
\sapbp\template
```
- 5 Select the IDoc you want to use and click **Open**. The template file is imported into the XMLOUT tool, with the template details shown in the Templates browser. You can now use the imported IDoc template to configure the Process based on the data contained within the Event.
 

**Note:** The first block that starts with EDI contains the control records (header variables).
- 6 Drag the block from the Event browser to the Process browser.
- 7 Drag the same block from the Templates browser to the Process browser.
- 8 Move the element you just dragged from the Templates browser to the position under the block.
 

**Note:** These records will have multiple instances and therefore must be kept in a block. When you have positioned the block, you can then add fields to the block.
- 9 Drag all fields in the Process browser to the correct element.
- 10 Repeat these steps until all blocks from the Message browser are mapped to the correct block element in the Process browser, and all fields are mapped to the correct field element.

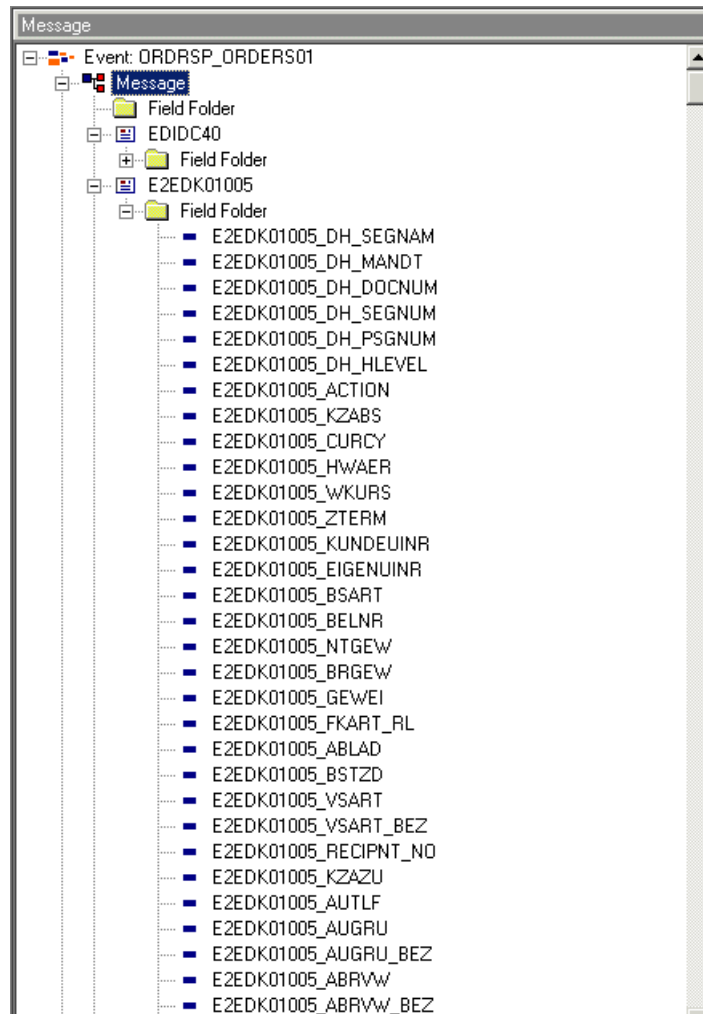
## IDoc structure and hierarchy

When configuring an IDoc to be sent to the SAP system, the order and the hierarchy of the IDoc must be correct. In order to ensure the order is correct, you need to add two fields and a script to each segment. The SAP IDoc Converter connector ensures the hierarchy level is correct by obtaining the hierarchy level information from the meta-data repository.

**Note:** If you are creating a Message that consists of an incoming IDoc and an outgoing IDoc, you do not need to add these fields and script. You only need to map the fields to the element.

*Example 1 Incoming IDoc mapped to an Outgoing IDoc*

The following example shows a StreamServe Message for an IDoc. The Message contains an StreamIN Event and an XMLOUT Process.



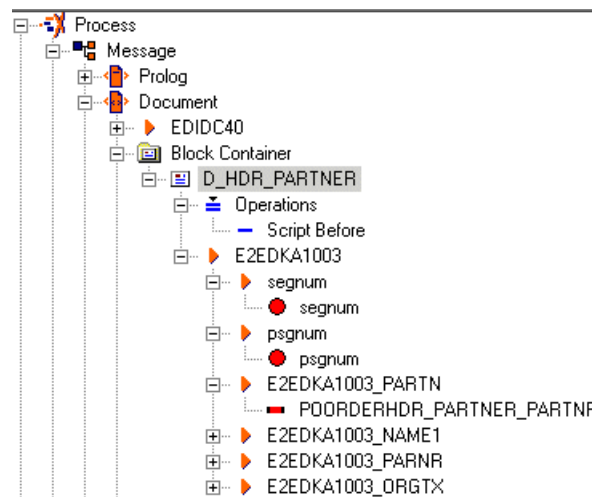
The following fields used to ensure the correct order are:

- **segnum** - specifies the number of the SAP segment.
- **psgnum** - specifies the number of the higher level SAP segment (parent segment)
- **docnum** - when sending multiple IDocs in a single outgoing file, identifies each IDoc in the file.

These two fields are included in the template file imported into the XMLOUT tool. When using these two fields, you still need to manually create the variables to contain the calculated values, and the script to calculate the values.

#### Example 2 *Incoming IDoc mapped to an Outgoing IDoc*

In this example, the `segnum` and `psgnum` variables have been added to the appropriate element. A script to calculate the values of these variables has been added to the block.



#### Step 1

- 1 Add a `$segnum` variable to the `segnum` element for each segment (block or element).
- 2 Add a `$psgnum` to the `psgnum` element for each segment (block or element).  
If you are mapping an IDoc to an IDoc, use the actual field instead of the variable.

#### Step 2

- **segnum**

If all blocks are in the correct order, you can use the following script to calculate the segment number (`segnum`).

```
$segnum = num($segnum) + 1;
if (num ($segnum) < 10)
  $segnum = "00000" + $segnum;
else
  $segnum = "0000" + $segnum;
```

The StreamServe IDoc Converter sorts segments or blocks according to respective `segnum` variable.

- **psgnum**

The `psgnum` field normally contains a zero (0) value, except when a segment has parents. When the segment has parents, the `psgnum` field indicates the number of parent segments. You can use the following scripts to set the `psgnum` value.

The following script is for blocks without a parent.

```
$psgnum = "000000";
$curParSegnum = $segnum;
```

The following script is for blocks that have a parent

```
$psgnum = $curParSegnum;
```

The IDoc structure file contains information on the hierarchy of a specific IDocType, see [Generating IDoc documentation](#) on page 26.

## Sending multiple IDocs to SAP

The SAP IDoc Converter can send an outgoing data stream from StreamServe to a SAP system that contains one or more IDocs.

In order for the SAP system to correctly identify each IDoc in the data stream, you need to associate each IDoc segment to exactly one IDoc. You do this by using a counter that fills the `EDIDC(40)_DOCNUM` field of the control record, and the `docnum` header field for the data segments.

**Note:** If you send only one IDoc at a time, you do not need to set the `docnum` value. However, if you set the `docnum` value in one segment, you have to set it in all segments.

### To specify multiple IDocs in an outgoing file

- 1 Calculate the current IDoc number within the Job. Start with the following:  

```
$docnum = "1"
```
- 2 For each IDoc in the Job, map the variable to the control record and the data segment.
- 3 Increase the `docnum` value accordingly (by 1 for each IDoc).
- 4 If the segments come in a different order, the IDoc Converter can still do the mapping accordingly. For example, when using sub-block in block structures.

For more information on the `docnum` field, see [StreamServe and IDoc format](#) on page 65.



# Configuring the IDoc Converter

---

The SAP IDoc Converter is an add-on component included in the StreamServe Business Processes installation.

The main purposes of the SAP IDoc Converter are to:

- Receive incoming IDoc data sent from the source SAP system, and convert the IDoc data into a format that StreamServer can receive - StreamServe XML format. The SAP IDoc Converter can receive an incoming file that contains one or more IDocs.
- Transform the XML data from the XMLOUT Process to the IDoc format, and send it back to the SAP system.

The SAP IDoc Converter converts data between IDoc and StreamServe XML format using IDoc meta-data stored in the following repository directory:

```
<Streamserve installation>\Applications\SAP connect\sapbp\meta
```

**Note:** On UNIX, the meta template files are located in the `STRS_HOME/meta` directory.

The meta XML files stored in the repository contain the information required to convert the IDoc data to an XML format on the inbound side, and the XML data to an IDoc format on the outbound side.

To use the SAP IDoc Converter to receive IDoc data from the SAP system, you must connect the IDoc Converter to the SAP gateway, and register the IDoc Converter with a unique program identifier. To use this connection, the connection must be attached to a port definition.

To use the SAP IDoc Converter to send IDoc data to the SAP system, you must connect an RFC client to the SAP System.

Using the Control Center, you can set up and run the SAP IDoc Converter as a service. You can set up the service to run on either a local or remote host.

When the service is created, you can select to view the properties for the client or server side of the IDoc Converter by right-clicking the IDoc Converter service.

**Note:** When you have created an IDoc Converter service, an argument file for the service is created. This file contains all runtime settings for the service. See *IDoc Converter argument file examples* on page 103

## To configure the SAP IDoc Converter as a Control Center service

- 1 Start the Control Center.
- 2 Right-click the local object or the remote host, and select **New Configuration**. The Choose Object dialog box opens showing the installed StreamServe configurations.

## Configuring the IDoc Converter

- 3 Select **SAP IDoc Converter Configuration** and click **OK**. The Service Configuration wizard opens.
- 4 Specify the Service Configuration settings.

<b>SAP IDoc Converter settings</b>	
<b>Logical name</b>	The name of the SAP IDoc Converter service. You can accept this name, or enter a new name for the service.
<b>License file</b>	StreamServer license file, which contains the SAP Business Processes license. <b>Note:</b> License file is currently not needed so this option is deactivated.
<b>Log file</b>	The file path to the log file you want to create.
<b>Log message file</b>	The file path to the log message file. For example <StreamServe installation>\Applications\SAP connect\sapbp sapidoc_logmsg.txt
<b>Log level</b>	The level of the log file (0-9).
<b>Language ID</b>	The language ID.
<b>Executable</b>	The file path to the SAP IDoc Converter executable file. For example <Streamserve installation>\Applications\SAP connect\strssapidocconverter.exe
<b>Argument file</b>	The file path to the IDoc Converter configuration file to use for this service.
<b>IDoc directions</b>	<b>Activate Server</b> – IDoc is sent from SAP to StreamServe. <b>Activate Client</b> – IDoc is sent from StreamServe to SAP.

- 5 On the SAP IDoc Converter Configuration dialog box, click **Next**. The SAP IDoc Converter Runtime Properties window opens.
- 6 Specify the SAP IDoc Converter Runtime settings.

<b>SAP IDoc Converter Runtime settings (Server)</b>	
<b>Program ID</b>	The program ID for the RFC destination in the SAP system.
<b>Gateway</b>	The IP address or host name of the computer the gateway is running on.

SAP IDoc Converter Runtime settings (Server)	
<b>Gateway service</b>	The name of the gateway, such as <code>SAPGW00</code> .
<b>Cancel running servers</b>	Cancels all RFC servers that are registered at the gateway with the specified program ID.
<b>Keep packet size</b>	<p>Select to collect multiple IDocs into one file/job before sending them to StreamServe.</p> <p>If not selected, the IDoc Converter will split the incoming IDoc stream (containing multiple IDocs) into separate files/jobs.</p>
<b>Destination Type</b>	Select the way StreamServe receives the IDoc.
<b>Destination</b>	<p>If Destination type is</p> <p><b>Directory</b> – The directory which StreamServe scans for IDocs using a Directory Scan connector. For example:  <code>c:\streamserve\saptest\bp\out</code></p> <p><b>HTTP</b> – The HTTP address of StreamServer including an HTTP access folder where the IDocs are sent. For example:  <code>http://localhost:1718/idoc</code></p> <p><b>Service Channel</b> – The HTTP address of the Service Broker where the IDocs are sent. For example:  <code>http://localhost:1818</code></p>
<b>Service Name</b>	The StreamServe IDoc Service name (only active if Destination type is Service Channel).
<b>Connection time-out (ms)</b>	If Destination type is <b>HTTP</b> or <b>Service Channel</b> , the time-out in milliseconds for the connection to the HTTP server.
<b>Communication time-out (ms)</b>	If Destination type is <b>HTTP</b> or <b>Service Channel</b> , the time-out in milliseconds for the total communication with the HTTP server.

- 7 On the SAP IDoc Converter Runtime Properties dialog box, click **Next**. The SAP IDoc Converter Configuration window opens.
- 8 Specify the settings for sending IDocs from StreamServe to SAP.

SAP IDoc Converter Configuration settings (Client)	
<b>Connection string</b>	<p>The connection parameters to connect to the SAP system, formatted as a string, for example:</p> <p><code>ASHOST=bosnt12 SYSNR=01 CLIENT=800 LANG=EN</code></p>

<b>SAP IDoc Converter Configuration settings (Client)</b>	
<b>Password</b>	The password to the SAP host.
<b>Use SNC</b>	Uses SNC (Secure Network Communication). If the receiving SAP environment uses high security, you can select this option instead of entering a user name and password.
<b>Source type</b>	Select the way StreamServe receives the XML from StreamServer.
<b>Source</b>	<p>If Source type is:</p> <p><b>Directory</b> – The directory that the IDoc Converter scans for XML files, which will be sent from StreamServe to SAP. For example:  <code>c:\streamserve\saptest\bp\in</code></p> <p><b>HTTP</b> – the HTTP access URI which identifies the incoming XML data stream. For example:  <code>/getidoc</code></p> <p><b>Service Channel</b> – The Service Broker URI where the SAP IDoc Converter service will be published to enable the service to receive XML data via the Service Broker. For example:  <code>localhost:1818</code></p>
<b>Temp directory</b>	For Source type HTTP and Service Channel, the directory where IDocs are temporarily stored before being sent to SAP.
<b>IPPort</b>	The HTTP port to the SAP IDoc Converter.
<b>Published service name</b>	For Source type Service channel, the name under which the service is published to the Service Broker.
<b>Connection time-out (ms)</b>	The time-out for the connection to the server.
<b>Communication time-out (ms)</b>	The time-out for communication with the server.

- 9 On the SAP IDoc Converter Runtime Properties dialog box, click **Next**. The Service Startup Configuration window opens.
- 10 Specify the SAP IDoc Converter Startup Configuration settings.

<b>SAP IDoc Converter Startup Configuration settings</b>	
<b>Startup type</b>	Select <b>Automatic</b> . The SAP IDoc Converter service starts automatically when the computer is restarted.

SAP IDoc Converter Startup Configuration settings	
<b>System account</b>	Runs the service under the local system account. (Default) Select <b>Allow the service to interact with desktop</b> to enable the service to send messages to your desktop.
<b>This account</b>	Assigns a logon user account to this StreamServe service. For example: <code>domain_name\user_id</code> or <code>computer\user_id</code> Enter the user ID and password for the user account.

**11 Click Finish**



**Check SAP system connection configuration**

You should verify that the connection with the SAP system for sending IDocs to StreamServe is configured correctly. In the SAP system, use the `sm59` transaction code to locate the RFC destination you created in *Configuring the IDoc Converter* on page 57, and click **Test Connection**. If an error message displays, the connection is not correctly configured.

## Viewing and editing IDoc Converter properties

When you have created the IDoc Converter service, you can select the service and edit the following properties in the Control Center:

<b>Meta file directory</b>	If you want to change the default setting, edit the value. Default value in Windows: <code>&lt;StreamServe installation&gt;\Applications\SAP connect\sapbp</code> Default value in UNIX: <code>\$STRS_HOME/meta</code>
<b>SAP Code Page</b>	If you need a specific codepage to change the behavior for the RFC codepage handling, edit the value.
<b>XML encoding</b>	If you need a specific XML encoding to change the behavior for the RFC XML encoding handling, edit the value. For example <code>UTF-8</code> .

You can select to display a list of properties for either

- the client properties of the Converter, that is the properties that specify how IDocs are sent from StreamServe to SAP.

- the server properties of the Converter, that is the properties that specify how StreamServe receives IDocs sent from SAP.

The common properties for the client and the server are always displayed.

**To view IDoc Converter client properties**

Right-click the Converter service, and select **Client properties**

**To view IDoc Converter server properties**

Right-click the Converter service, and select **Server properties**

## Running the IDoc Converter on UNIX

To start the IDoc Converter, you run the following command from the installation directory:

```
./start strssapidocconverter -a <path to IDoc argument file>
```

**64** | Running the IDoc Converter on UNIX  
**Configuring the IDoc Converter**

# StreamServe and IDoc format

---

StreamServe uses the IDoc format to retrieve and return IDoc files between StreamServer and a SAP system. IDoc files contain administrative information for technical processing, as well as the actual data.

Data in an IDoc file is divided into the following segments or ‘records’:

- Control records
- Data records
- Status records
- Header records

## Example 1 IDoc file

---

The following is an example of an IDoc file:

```
Control Record E EDI_DC40 10000000000019905446B 3012 ORDERS01
Data Record  E2EDK1005 10000000000019905400000100000001004 EUR
              E2EDK14 1000000000001990540000020000000200601
              E2EDK14 1000000000001990540000030000000200701
              E2EDK14 100000000000199054000004000000020080001
              E2EDK14 10000000000019905400000500000002012TA
              E2EDK03 100000000000199054000006000000020220010410
              E2EDK03 10000000000019905400000700000002012200104
              E2EDK03 100000000000199054000008000000020222001041
              E2EDK03 100000000000199054000009000000020252001041
              E2EDK03 100000000000199054000010000000020232001041
              E2EDK04001 10000000000019905400001100000002A2 7.000
```

**Note:** This graphic has been cropped and does not display all records.

---

## Control records

The control record is the same for all IDoc files and contains administration information, including sender, recipient and message details.

The structure of each control record is defined in the EDI\_DC/EDI\_DC40 dictionary object, which has 36 fields.

The following table lists the main fields of the dictionary object:

Field	Description
docnum	IDoc number
status	Status of the IDoc file
rcvpor	Receiver port (SAP system, EDI sub-system)
direct	Direction for IDoc transmission (outbound/inbound)
rcvprt	Partner type of receiver
rcvprn	Partner number of receiver
mestyp	Message type
sndpor	Sender port (SAP system, EDI sub-system)
sndprt	Partner type of sender
sndprn	Partner number of sender
idoctp	Basic type
rcvpfc	Partner function of receiver
maxsegnum	Number of data records

## Data records

Data records contain sections of data.

There is a relationship between control record types and data record types, where a data record with the same `docnum` document number must exist for each control record with document number `docnum`.

The structure of each data record is defined in the EDI\_DD dictionary object. The following tables list the main fields of the dictionary object.

### Fields in the EDI\_DD dictionary object for SAP 3.1

Field	Description	Variable set
tabnam	Segment name	\$idoc_dh_tabnam
mandt	Client name	\$idoc_dh_mandt
docnum	IDoc number	\$idoc_dh_docnum
segnum	Sequence number of the segment in the IDoc.	\$idoc_dh_segnum
segnam	Segment name	\$idoc_dh_segnam
psgnum	Parent segment number	\$idoc_dh_psgnum
hlevel	Hierarchy level of segment	\$idoc_dh_hlevel
sdata	Application data	\$idoc_dh_sdata

### Fields in the EDI\_DD40 dictionary object for SAP 4.x

Field	Description	Variable set
segnam	Segment name	\$idoc_dh_segnam
mandt	Client name	\$idoc_dh_mandt
docnum	IDoc number	\$idoc_dh_docnum
segnum	Segment number	\$idoc_dh_segnum
psgnum	Parent segment number	\$idoc_dh_psgnum
hlevel	Hierarchy level of SAP segment	\$idoc_dh_hlevel
sdata	Application data	\$idoc_dh_sdata

## Status records

Status records contain previous processing status details and administration information. This information is not transferred between the systems as part of the IDoc file, but as a separate data file.

The structure of each status record is defined in the EDI\_DS/EDI\_DS40 dictionary object, which has 24 fields. The following table lists the main fields of the dictionary object:

Field	Description
docnum	IDoc number
logdat	Date of status information
logtim	Time of status information
countr	IDoc status counter
credat	Date status record was created
cretim	Time status record was created
uname	User name
repid	Program name
statxt	Text for status code

## Header records

A header record acts as a trigger for StreamServe and determines which StreamServe Event should be executed. The Event name is the same as the IDOCTYPE and MESSAGETYPE data (with an underscore separator).

The header record also contains information that is automatically inserted into the following variables according to the SAP release:

### Variables available for SAP 3.1

SAP Variable	StreamServe Variable	Position	Length
\$idoc_dc_mandt	\$idoc_dc_mandt	10	3
\$idoc_dc_docnum	\$idoc_dc_docnum	13	16
\$idoc_dc_docrel	\$idoc_dc_docrel	29	4
\$idoc_dc_doctyp	\$idoc_dc_doctyp	35	8
\$idoc_dc_status	\$idoc_dc_status	33	2
\$idoc_dc_direct	\$idoc_dc_direct	43	1
\$idoc_dc_rcvpor	\$idoc_dc_rcvpor	44	10
\$idoc_dc_rcvppt	\$idoc_dc_rcvppt	54	2
\$idoc_dc_rcvprn	\$idoc_dc_rcvprn	56	10
\$idoc_dc_rcvsad	\$idoc_dc_rcvsad	66	21
\$idoc_dc_rcvlad	\$idoc_dc_rcvlad	87	70
\$idoc_dc_std	\$idoc_dc_std	157	1
\$idoc_dc_stdvrs	\$idoc_dc_stdvrs	158	6
\$idoc_dc_stdmes	\$idoc_dc_stdmes	164	6
\$idoc_dc_mescod	\$idoc_dc_mescod	170	3
\$idoc_dc_mesfcn	\$idoc_dc_mesfcn	173	3
\$idoc_dc_outmod	\$idoc_dc_outmod	176	1
\$idoc_dc_sndpor	\$idoc_dc_sndpor	178	10
\$idoc_dc_test	\$idoc_dc_test	177	1
\$idoc_dc_sndprt	\$idoc_dc_sndprt	188	2
\$idoc_dc_sndprn	\$idoc_dc_sndprn	190	10
\$idoc_dc_sndsad	\$idoc_dc_sndsad	200	21

SAP Variable	StreamServe Variable	Position	Length
\$idoc_dc_sndlad	\$idoc_dc_sndlad	221	70
\$idoc_dc_refint	\$idoc_dc_refint	291	14
\$idoc_dc_refgrp	\$idoc_dc_refgrp	305	14
\$idoc_dc_refmes	\$idoc_dc_refmes	319	14
\$idoc_dc_arckey	\$idoc_dc_arckey	333	70
\$idoc_dc_credat	\$idoc_dc_credat	403	8
\$idoc_dc_cretim	\$idoc_dc_cretim	411	6
\$idoc_dc_mestyp	\$idoc_dc_mestyp	417	6
\$idoc_dc_idotyp	\$idoc_dc_idotyp	423	8
\$idoc_dc_cimtyp	\$idoc_dc_cimtyp	431	8
\$idoc_dc_rcvpfc	\$idoc_dc_rcvpfc	439	2
\$idoc_dc_sndpfc	\$idoc_dc_sndpfc	441	2
\$idoc_dc_serial	\$idoc_dc_serial	443	20
\$idoc_dc_exprss	\$idoc_dc_exprss	463	1

## Variables available for SAP 4.x

SAP Variable	StreamServe Variable	Position	Length
\$idoc_dc_mandt	\$idoc_dc_mandt	10	3
\$idoc_dc_docnum	\$idoc_dc_docnum	13	16
\$idoc_dc_docre1	\$idoc_dc_docre1	29	4
\$idoc_dc_status	\$idoc_dc_status	33	2
\$idoc_dc_direct	\$idoc_dc_direct	35	1
\$idoc_dc_outmod	\$idoc_dc_outmod	36	1
\$idoc_dc_exprss	\$idoc_dc_exprss	37	1
\$idoc_dc_test	\$idoc_dc_test	38	1
\$idoc_dc_idotyp	\$idoc_dc_idotyp	39	30
\$idoc_dc_cimtyp	\$idoc_dc_cimtyp	69	30
\$idoc_dc_mestyp	\$idoc_dc_mestyp	99	30

SAP Variable	StreamServe Variable	Position	Length
\$idoc_dc_mescod	\$idoc_dc_mescod	129	3
\$idoc_dc_mesfcn	\$idoc_dc_mesfcn	132	3
\$idoc_dc_std	\$idoc_dc_std	135	1
\$idoc_dc_stdvrs	\$idoc_dc_stdvrs	136	6
\$idoc_dc_stdmes	\$idoc_dc_stdmes	142	6
\$idoc_dc_sndpor	\$idoc_dc_sndpor	148	10
\$idoc_dc_sndprt	\$idoc_dc_sndprt	158	2
\$idoc_dc_sndpfc	\$idoc_dc_sndpfc	160	2
\$idoc_dc_sndprn	\$idoc_dc_sndprn	162	10
\$idoc_dc_sndsad	\$idoc_dc_sndsad	172	21
\$idoc_dc_sndlad	\$idoc_dc_sndlad	193	70
\$idoc_dc_rcvpor	\$idoc_dc_rcvpor	263	10
\$idoc_dc_rcvppt	\$idoc_dc_rcvppt	273	2
\$idoc_dc_rcvpfc	\$idoc_dc_rcvpfc	275	2
\$idoc_dc_rcvprn	\$idoc_dc_rcvprn	277	10
\$idoc_dc_rcvsad	\$idoc_dc_rcvsad	287	21
\$idoc_dc_rcvld	\$idoc_dc_rcvld	308	70
\$idoc_dc_credat	\$idoc_dc_credat	378	8
\$idoc_dc_cretim	\$idoc_dc_cretim	386	6
\$idoc_dc_refint	\$idoc_dc_refint	392	14
\$idoc_dc_refgrp	\$idoc_dc_refgrp	406	14
\$idoc_dc_refmes	\$idoc_dc_refmes	420	14
\$idoc_dc_arckey	\$idoc_dc_arckey	434	70
\$idoc_dc_serial	\$idoc_dc_serial	504	20



# Configuring the BAPI interface

---

The BAPI interface comprises a number of script functions which enable you to call any BAPI (Business Application Programming Interface) function available in the SAP system from StreamServe. With the Business Processes Connect solution, the BAPI script functions are integrated into the standard StreamServe scripting language.

BAPI is a SAP standard interface technology that enables you to retrieve and update data in SAP systems using RFC (Remote Function Calls) functionality from external data sources.

The Business Processes Connect solution supports all BAPIs using the RFC interface, providing system independent access to all business objects used in the SAP system.

## The BAPI interface and StreamServe

BAPI script functions enable you to carry out synchronous BAPI calls from StreamServe to a SAP system and copy the results to script variables. Because these BAPI functions are integrated into the StreamServe scripting language, you can call the BAPI script functions from anywhere in StreamServe where you can use a script.

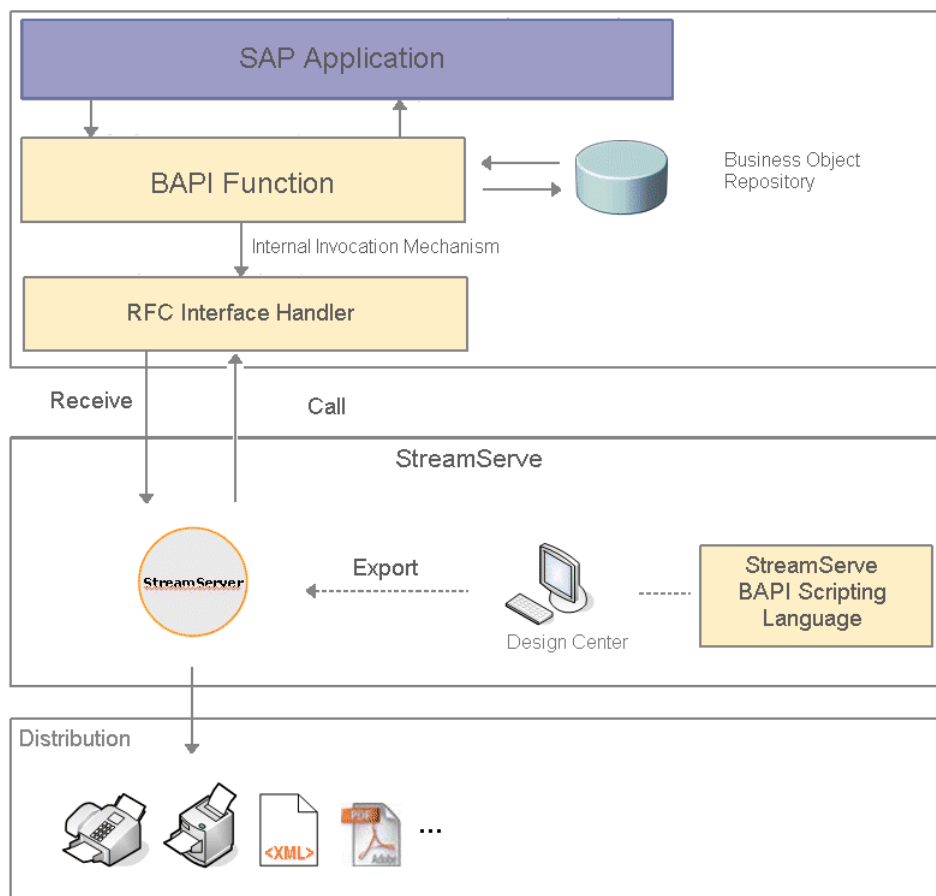
In addition to existing StreamServe script functions, BAPI script functions also enable you to call updating BAPI functions in the SAP system.

Using the BAPI interface with StreamServe does not require any configuration in the SAP system.

**Note:** The StreamServe Scripting language and SAP BAPI script functions are intended for advanced users who have some programming experience and a good working knowledge of the StreamServer and their SAP system.

The following diagram illustrates how the BAPI interface between SAP and StreamServe enables you to carry out synchronous BAPI calls to the SAP system.

*Figure 1 StreamServe and the BAPI interface*



- 1** At runtime, StreamServer executes any script functions configured for the Message. If the scripts include a BAPI script function, StreamServer sends the BAPI call to the SAP system.
- 2** In the SAP system, the RFC Interface Handler receives the BAPI call, and executes the BAPI function accordingly.
- 3** If the BAPI function contains a result value, the value is returned to StreamServer, and copied to the appropriate variables.

## Licensing StreamServe BAPI scripting

Before you can use StreamServe BAPI functions, you need to have a license for StreamServe BAPI scripting. If you did not specify StreamServe BAPI when you obtained a license for the Business Processes Connect solution, you will need to obtain a StreamServe BAPI license before you can use the BAPI script functions.

For information on obtaining a license, see the StreamServe Licensing documentation.

## Enabling BAPI scripting in StreamServe

To use BAPI script functions in StreamServe, you must enable BAPI scripting in StreamServe. You do this by entering the following argument in StreamServer startup argument file.

```
-sapbapi
```

**Note:** You must enter the `-sapbapi` argument above the lines specifying the `*.dua` and `*.dux` configuration files.

For information on the start-up argument file, see the *Design Center* documentation.

## The BAPI interface

SAP systems integrate with other systems through a set of functions called by external applications. When these functions are associated with SAP business objects and are published to the SAP Business Object Repository, they are called BAPI (Business Application Programming Interfaces) functions.

Within SAP, BAPI script functions provide a stable interface to SAP functionality. For example, `BAPI_CUSTOMER_GETLIST` and `BAPI_CUSTOMER_GETDETAIL` are BAPI functions that belong to the business object `CUSTOMER`.

In SAP, BAPI functions are flagged as RFC callable. RFC (Remote Function Call) is SAP's implementation of remote invocation mechanism. In order to call a BAPI in the SAP system, RFC Client functionality must be implemented. The BAPI script functions included in the StreamServe Business Processes installation provide RFC Client functionality between SAP and StreamServe.

## BAPI function components

A BAPI function consists of the following components:

### IMPORT parameters

Import parameters hold input values, for example the customer number. Import parameters can be:

- a scalar type, such as a string, date, time or an integer
- a structured parameter - a parameter with more than one scalar field. For example the `RETURN` parameter has fields such as `TYPE` and `MESSAGE`.

### EXPORT parameters

Export parameters hold output values, for example the success or failure of a BAPI function in the `RETURN` parameter. Export parameters can be:

- a scalar type, such as a string, date, time or an integer
- a structured parameter - a parameter with more than one scalar field.

### TABLE parameters

These are input/output parameters that hold multiple records of the same type, for example, addresses for multiple business partners. The parameter name usually indicates if the parameters are input or output.

### Exceptions

These are defined error return codes that normal RFC calls have and BAPI functions typically do not have.

*Example 1*    *BAPI function components*

---

The `BAPI_CUSTOMER_GETLIST` BAPI function contains the following components:

- **IMPORT**
    - `MAXROWS` - Maximum number of returned customer records (simple parameter).
  - **EXPORT**
    - `RETURN` - The structure that indicates the success or failure of the BAPI (structured parameter).
  - **TABLES**
    - `IDRANGE` - Serves as an input parameter containing selection criteria for the customers.
    - `ADDRESSDATA` - Serves as an input parameter containing selection criteria for the customers.
- 

## Rules for using script functions

The following rules apply when working with the BAPI interface and StreamServe BAPI script functions:

### Retrieving function IDs

For each BAPI you call, you first need to retrieve a *functionid* using `SAPCreateFunction`. This function checks if the BAPI is available in the SAP system and retrieves the actual interface.

### Using `SAPCreateFunction` first

Because all the parameter handling functions use *functionid* as a parameter, you can only call a function after using `SAPCreateFunction`.

### Setting `SAPInvokeFunction` or `SAPInvokeFunction2` parameters

Before calling a BAPI in the SAP system using `SAPInvokeFunction` or `SAPInvokeFunction2`, you need to set all input parameters using `SAPSetSimpleParameter`, `SAPSetComplexParameter` and `SAPSetTableParameter`.

### Reading output parameters

You can only read output parameters once `SAPInvokeFunction` is called and returned successfully.

**Setting IMPORT and EXPORT function parameters**

- You set IMPORT parameters using `SAPSetSimpleParameter` for the scalar input parameter, and `SAPSetComplexParameter` for the structured input parameter.
- You read EXPORT parameter values using `SAPSetSimpleParameter` for scalar parameters, and `SAPGetComplexParameter` for structured parameters.

**Setting and reading TABLE parameters**

- You set input TABLE parameter values using `SAPSetTableParameter`.
- You read output TABLE parameter values using `SAPGetTableParameter`.

**BAPI script functions and StreamServe processing phases**

**Note:** Because of the way that BAPI script functions run in StreamServe's processing phases, you should only run BAPI script functions that update data once.

BAPI script functions run in StreamServe processing phases as follows:

- `SAPConnect` establishes a connection to the SAP system in the pre-processing phase, and returns the buffered value (`$connectionid`) in the processing phase.
- `SAPDisconnect` is fully executed in the processing phase. There is no communication with the specified connection after the function is run.
- `SAPCreateFunction` establishes an interface of a BAPI function from the SAP system in the pre-processing phase, and returns the buffered value (`$functionid`) in the processing phase.
- `SAPGetComplexParameter`, `SAPGetSimpleParameter`, `SAPGetTableParameter` and `SAPGetTableRowCount` can be executed in both the pre-processing and processing phases.

**Note:** As these are local functions, the performance time is not affected.

**Note:** The results for these functions can differ between phases depending on whether a `SAPInvokeFunction` or `SAPInvokeFunction2` is executed. The correct results are only available after the invocation has taken place.

- `SAPInvokeFunction` is executed in the processing phase.
- `SAPInvokeFunction2` is executed according to the argument value:
 

0	The function is executed in both the pre-processing and processing phases.
1	The function is only executed in the pre-processing phase.

- `SAPSetComplexParameter`, `SAPSetSimpleParameter` and `SAPSetTableParameter` can be executed in both the pre-processing and processing phases.

**Note:** As these are local functions, the performance time is not affected.

## Using `SAPInvokeFunction` or `SAPInvokeFunction2`

Whether you should use `SAPInvokeFunction` or `SAPInvokeFunction2` is determined by the action you want to perform, and the processing phase in which you want to execute the function.

- `SAPInvokeFunction`  
You should use `SAPInvokeFunction` when you call an update BAPI.
- `SAPInvokeFunction2`  
When you are executing functions in the pre-processing phase only, you should use `SAPInvokeFunction2` to retrieve data.



When you are executing functions in both the pre-processing and processing phases, you should NOT use `SAPInvokeFunction2` to update data in the SAP system, as this will result in two updates being performed.

---

## Using `SAPInvokeFunction` with `CallProc` and `CallBlock`

During the processing phase, you can only use Processes and blocks which have been initialized during the pre-processing phase. Therefore, it is important when using `SAPInvokeFunction` to ensure that you run the `CallProc` and `CallBlock` script functions in the pre-processing phase, independent of the result return code of the `SAPInvokeFunction`.

## StreamServe BAPI script functions

This table lists the StreamServe BAPI script functions.

Function	Returns	Description
<code>sapcleartableparameter (\$functionid, tablename) ;</code>	0 (OK) -1 (Failed)	Clears the values of a table parameter of a BAPI function.
<code>sapconnect (connectionstring) ;</code>	String (OK) Empty (Failed)	Connects the server to a SAP system.
<code>sapcreatefunction (\$connectionid, functionname)</code>	String (OK) Empty (Failed)	Reads and creates the BAPI function interface from the SAP system.
<code>sapdisconnect (\$connectionid) ;</code>	0 (OK) -1 (Failed)	Closes the remote connection to the SAP system.
<code>sapgetcomplexparameter (\$functionid, parametername, fieldname) ;</code>	String	Returns the value for a field in a structured export parameter of the function.
<code>sapgetsimpleparameter (\$functionid, parametername) ;</code>	String	Returns the value for a scalar export parameter of the function.
<code>sapgettableparameter (\$functionid, parametername, rowindex, fieldname) ;</code>	String	Returns the value for a field in a table parameter of the function.
<code>sapgettablerowcount (\$functionid, tablename) ;</code>	>= 0 (OK) -1 (Failed)	Returns the number of rows of the specified table.
<code>sapinvokefunction (\$functionid) ;</code>	0 (OK) -1 (Failed)	Calls the function in the SAP system in the processing phase, and returns upon completion.
<code>sapinvokefunction2 (\$functionid, value) ;</code>	0 (OK) -1 (Failed)	Calls the function in the SAP system in either only the pre-processing phase, or both the pre-processing and the processing phases.
<code>saplasterror (\$functionid) ;</code>	String (OK) Empty (No Error)	Returns the last reported error from the SAP system.
<code>sapsetcomplexparameter (\$functionid, parametername, fieldname, value) ;</code>	0 (OK) -1 (Failed)	Sets the value for a field in a structured import parameter of the function.
<code>sapsetsimpleparameter (\$functionid, parametername, value) ;</code>	0 (OK) -1 (Failed)	Sets the value for a scalar import parameter of the function.

Function	Returns	Description
<code>sapsettableparameter</code> ( <i>\$functionid, parametername, rowindex, fieldname, value</i> );	0 (OK) -1 (Failed)	Sets the value for a field in a table parameter of the function.

## BAPI scripting arguments

The following arguments are used in SAP BAPI script functions.

- *connectionstring*  
The `SAPConnectString` is used to establish a connection to the SAP system, and provides all necessary information needed to establish a remote connection with an SAP system. For more information, see [SAPConnect](#) on page 81.
- *connectionid*  
The BAPI scripting returns this identifier when a connection to the SAP system is opened. This identifier is subsequently used for all calls to the SAP system. To connect to an SAP system, you need to use the *connectionstring* argument in the connect function.
- *functionid*  
The function that establishes a BAPI in StreamServer, returns this identifier. It is subsequently used in conjunction with all BAPI script functions that set or get values in the function interface, and in the actual invocation of the function in the SAP system.

## BAPI scripting return values

For all script functions, if the return value is a string, an empty string indicates either a warning or an error (except when the value returned in the RFC parameter is empty.) All functions that return a -1 value indicate an error.

## SAPConnect

The `SAPConnect` function establishes a connection to the SAP system in the pre-processing phase, and returns the buffered value (*\$connectionid*) in the processing phase. You can use this connection to do multiple function calls.

The single parameter, *connectionstring*, must contain all information needed to establish a connection with the SAP system. It uses the same syntax as `RfcOpenEx` and accepts the same values.

A connection string has two parts, one that describes the connection to the SAP system, the second part that describes the user logon parameter to this system. The system string can either contain information about a specific SAP application server or can use all necessary information for using load balancing. Before going productive you should contact your SAP administrator to find out the preferred way in your environment.

**Note:** All connections opened with `SAPConnect` must be closed using `SAPDisconnect`.

<b>Syntax</b>	<code>sapconnect (connectionstring) ;</code>	
<b>Argument</b>	<i>connectionstring</i>	A string containing all information needed to connect to the SAP system.
<b>Returns</b>	A number indicating whether the function was called successfully or not:	
	String    OK	The <i>connectionid</i> that identifies the connection to the SAP system.
	Empty    Failed	An error occurred while connecting.
<b>Example</b>	<pre>\$connectionID=sapconnect ("CLIENT=800 USER=MySAPUser PASSWORD=PASSWORD LANG=EN ASHOST=111.111.111.11 SYSNR=00");</pre> <p>You can also split the connection string into two variables and concatenate them. This can be useful if the user and password are contained in the data or retrieved from a different source.</p> <pre>User part \$Userstr="CLIENT=800 USER=MySAPUser PASSWORD=MyPassword LANGUAGE=EN";</pre> <pre>Application Server \$systemstr="ASHOST=hs0311 sysnr=53";</pre> <pre>\$connectionstring=\$systemstr + \$userstr;</pre>	

## SAPCreateFunction

The `SAPCreateFunction` establishes an interface of a BAPI function from the SAP system in the pre-processing phase, and returns the buffered value (*\$functionid*) in the processing phase.

It uses the connection identified by *connectionid* and a string with the name of the function. After this call, the SAP script functions can set the parameter values needed as input values for the function, and invoke the function in the SAP system. It returns the *functionid* that identifies the function in the other script functions.

<b>Syntax</b>	<code>sapcreatefunction (\$connectionid, functionname) ;</code>	
<b>Arguments</b>	<i>connectionid</i>	A number used to identify the connection to the SAP system returned from SAPConnect.
	<i>functionname</i>	A string containing the name of the function in the SAP system.
<b>Returns</b>	Returns a string containing the name of the valid function. If no value is returned, the function failed.	

<b>Example</b>	<pre> \$functionid = sapcreatefunction(\$connectionid, "BAPI_CUSTOMER_GETLIST"); if (\$functionid == "") {     \$error = saplasterror();     log (0, \$error); } </pre>
----------------	---

## SAPDisconnect

The `SAPDisconnect` function closes the connection identified by *connectionid* to the SAP system. The function is fully executed in the processing phase. There is no communication with the specified connection after the function is run.

**Note:** All connections opened with `SAPConnect` must be closed using `SAPDisconnect`.

<b>Syntax</b>	<code>sapdisconnect (\$connectionid) ;</code>	
<b>Argument</b>	<i>connectionid</i>	A number used to identify the connection to the SAP system returned from <code>SAPConnect</code> .
<b>Returns</b>	Returns a number indicating whether the function was called successfully or not:	
	0	OK
	-1	Failed An error occurred while executing the function.
<b>Example</b>	<code>\$result = sapdisconnect(\$connectionID);</code>	

## SAPGetComplexParameter

The `SAPGetComplexParameter` function returns the value of a field in a structured parameter. The function can be executed in both the pre-processing and processing phases.

<b>Syntax</b>	<code>sapgetcomplexparameter (\$functionid, parametername, fieldname) ;</code>	
<b>Arguments</b>	<i>functionid</i>	A number used to identify the function in the SAP system. This number is returned from the <code>SAPCreateFunction</code> function.
	<i>parametername</i>	The name of the parameter in the BAPI.
	<i>fieldname</i>	The name of the field to be set.

<b>Returns</b>	Returns the value of the field in a structured parameter. If no value is returned, either the field has no value or the field does not exist.
<b>Example</b>	<pre> \$parametername = "RETURN"; \$bapierror = sapgetcomplexparameter (\$functionid, \$parametername, "TYPE"); if ((\$bapierror = "S") or (\$bapierror = "")) {     log (1, "BAPI SUCCEEDED"); } </pre>
<b>Example (continued)</b>	<p>In this example, the value of one of the fields of the parameter RETURN is read from the function identified by <i>\$functionid</i>. The value is then tested to evaluate if there were any semantic errors in the BAPI call.</p> <p><b>Note:</b> Most SAP BAPI have a structured output parameter named RETURN. This parameter holds any value indicating the success or failure of the actual BAPI call. An S value or an empty value for the field TYPE, indicates success, while any other value indicating an error has occurred.</p>

## SAPGetSimpleParameter

The `SAPGetSimpleParameter` function returns the value of a simple parameter. The function can be executed in both the pre-processing and processing phases.

<b>Syntax</b>	<code>sapgetsimpleparameter (\$functionid, parametername) ;</code>
<b>Arguments</b>	<p><i>functionid</i>            A number used to identify the function in the SAP system returned from <code>SAPCreateFunction</code>.</p> <p><i>parametername</i>        The name of the parameter in the BAPI.</p>
<b>Returns</b>	Returns the value of the simple parameter. If no value is returned, either the field has no value or does not exist.
<b>Example</b>	<pre> \$valdocnum = sapgetsimpleparameter (\$functionid, "DOCNUMBER"); </pre> <p>In this example, a value is read from an output parameter. The value is retrieved from the parameter <code>DOCNUM</code> and written to the variable <code>\$valdocnum</code>. The SAP function in this example is <code>BAPI_DOCUMENT_CREATE</code>.</p> <p><b>Note:</b> The values retrieved from the SAP system, for example, output parameters, can only be read after <code>SAPInvokeFunction</code> has been called successfully.</p>

## SAPGetTableParameter

The `SAPGetTableParameter` function returns the value of an RFC table parameter in a specific row in a table. The function can be executed in both the pre-processing and processing phases.

<b>Syntax</b>	<code>sapgettableparameter (\$functionid, tablename, rowindex, fieldname) ;</code>	
<b>Arguments</b>	<i>functionid</i>	A number used to identify the function in the SAP system returned from <code>SAPCreateFunction</code> .
	<i>tablename</i>	The name of the table parameter in the BAPI.
	<i>rowindex</i>	Indicates the line you want to get the value for in the table.
	<i>fieldname</i>	The name of the field containing the value to get.
<b>Returns</b>	Returns a single value in an RFC table parameter. If no value is returned, either the field has no value or does not exist.	
<b>Example</b>	<pre> \$tablename = "ADDRESSDATA"; \$numofrows = sapgettablerowcount(\$functionid, \$tablename); \$idx = 0; while(num(\$idx) &lt; num(\$numofrows)){ \$idx++; \$valcustno = sapgettableparameter(\$functionid, \$tablename,num(\$idx), "CUSTOMER"); log(1, "Valcustno = "+\$valcustno); } </pre> <p>This example gets the number of row in the table parameter ADDRESSDATA, and loops through each line in the table, retrieving the value of the field CUSTOMER. This field is the customer number. The value is then set in the variable \$valcustno. This table is the table parameter returned by BAPI_CUSTOMER_GETLIST.</p> <p><b>Note:</b> Usually, you would use the <code>CallProc</code> script function to issue the variables in the Process.</p>	

## SAPGetTableRowCount

The `SAPGetTableRowCount` function returns the number of rows of an RFC table. The function can be executed in both the pre-processing and processing phases.

<b>Syntax</b>	<code>sapgettablerowcount (\$functionid, tablename) ;</code>
---------------	--

<b>Arguments</b>	<i>functionid</i>	A number used to identify the function in the SAP system returned from <code>SAPCreateFunction</code> .
	<i>tablename</i>	The table parameter name in the BAPI.
<b>Returns</b>	Returns the value of the simple parameter. If no value is returned, either the field has no value or does not exist.	
<b>Example</b>	<pre>\$tablename = "ADDRESSDATA"; \$numofrows = sapgettablerowcount(\$functionid, \$tablename);</pre> <p>This example retrieves the number of rows within the <code>\$tablename</code> table. The SAP functions uses the number is used for table parameters that the SAP function returns in order to retrieve values, and set values in variables.</p>	

## SAPInvokeFunction

The `SAPInvokeFunction` function executes the BAPI referenced by *functionid* in the SAP system. StreamServer does a synchronous call to the SAP system and waits until the function returns. The function is executed in the processing phase, therefore the results are only available in that phase.

The return code indicates if there was a communication error during the call, such as, if the wrong data was sent into the function. BAPIs typically handle invalid data by indicating the cause of the problem in the return code.



During the processing phase, you can only use Processes and blocks which have been initialized during the pre-processing phase. Therefore, it is important when using `SAPInvokeFunction` to ensure that you run the `CallProc` and `CallBlock` script functions in the pre-processing phase, independent of the result return code from the `SAPInvokeFunction`.

<b>Syntax</b>	<code>sapinvokefunction(\$functionid);</code>	
<b>Argument</b>	<i>functionid</i>	A number used to identify the function in the SAP system returned from <code>SAPCreateFunction</code> .
<b>Returns</b>	A number indicating whether the function was called successfully or not:	
	0	OK
	-1	Failed

<b>Example</b>	<pre>\$result = sapinvokefunction(\$functionid); log(1, "Result= " + \$result);</pre> <p>This example invokes the function identified with <i>functionid</i> that was created with the <code>SAPCreateFunction</code>. The <code>SAPInvokeFunction</code> calls the function module in the SAP system and returns upon completion. The function returns 0 if successful, and -1 if the function failed, indicating that either the communication layer or the function itself raised an exception. You can retrieve details of the actual exception using the <code>SAPGetLastError</code> function.</p> <p><b>Note:</b> The BAPI <code>RETURN</code> parameter indicates most semantic errors from BAPIs.</p>
----------------	--

## SAPInvokeFunction2

The `SAPInvokeFunction2` function invokes the BAPI referenced by *functionid* in the SAP system in either only the pre-processing phase, or both the pre-processing and the processing phases. This function is typically used in data retrieval. StreamServer does a synchronous call to the SAP system and waits until the function returns.

The return code indicates if there was a communication error during the call.



When you are executing functions in both the pre-processing and processing phases, you should NOT use `SAPInvokeFunction2` to update data in the SAP system, as this will result in two updates being performed.

A different problem would occur if you used the same function more than once, for example, if you use the `BAPI_CUSTOMER_GET_DETAIL` function to read the recipient address, then use the same function to read the address of the shipping partner. In this case, running the function in both phases would give you the correct result.

<b>Syntax</b>	<code>sapinvokefunction2 (\$functionid, value) ;</code>	
<b>Arguments</b>	<i>functionid</i>	A number used to identify the function in the SAP system returned from <code>SAPCreateFunction</code> .
	<i>value</i> 0	The function is invoked in both the pre-processing and processing phases.
	1	The function is only invoked in the pre-processing phase.

<b>Returns</b>	A number indicating whether the function was called successfully or not:  0            OK  -1          Failed      An error occurred while executing the function.
<b>Example</b>	<code>\$result = sapinvokefunction2(\$functionid,1);</code> The function identified by <code>\$functionid</code> is only invoked in the pre-processing phase.

## SAPLastError

The `SAPLastError` function returns the last error message from communication with the SAP system. This is typically an error that occurred when you connected to the SAP system, or after you invoked a function module in the SAP system.

This function only returns errors reported in the RFC layer of the SAP system, such as:

- communication with the SAP system could not be established
- the invoked function module does not exist
- the function module raised an exception.

**Note:** Most BAPI handle semantic calling errors by providing information in the `RETURN` parameter of the BAPI. You can *not* access this information using `SAPLastError`.

<b>Syntax</b>	<code>saplasterror(\$connectionid);</code>
<b>Argument</b>	<i>connectionid</i> A number used to identify the connection to the SAP system returned from <code>SAPConnect</code> .
<b>Returns</b>	Returns the text of the last error message from communication with the SAP system. If no value is returned, no errors have occurred since the SAP system started.
<b>Example</b>	<pre>\$result = sapconnect(connectstring); if (\$result = "-1") {     \$error = saplasterror();     log (0, \$error); }</pre>

## SAPSetComplexParameter

The `SAPSetComplexParameter` function returns the value for a field in a structured export parameter of the function. The function can be executed in both the pre-processing and processing phases.

<b>Syntax</b>	<code>sapsetcomplexparameter (\$functionid, parametername, fieldname, value) ;</code>	
<b>Arguments</b>	<i>functionid</i>	A number used to identify the function in the SAP system returned from <code>SAPCreateFunction</code> .
	<i>parametername</i>	The name of the parameter in the BAPI.
	<i>fieldname</i>	The name of the field to be set.
	<i>value</i>	The data to be passed to the function.
<b>Returns</b>	A number indicating whether the function was called successfully or not:	
	0	OK
	-1	Failed An error occurred while executing the function.
<b>Example</b>	<pre>\$structname = "IMPORTSTRUCT"; \$valflt = "0.1"; \$result = sapsetcomplexparameter(\$functionid, \$structname, "RFCFLOAT", \$valflt);</pre> <p>This example has the structure <code>IMPORTSTRUCT</code> as an import parameter. The code sets the value for the field <code>RFCFLOAT</code> to 0.1. Structured parameters are essential for records with multiple fields.</p>	

## SAPSetSimpleParameter

The `SAPSetSimpleParameter` function enables you to set the value of an input parameter of the BAPI *functionid* with the parameter *parametername*. The function can be executed in both the pre-processing and processing phases.

<b>Syntax</b>	<code>sapsetsimpleparameter (\$functionid, parametername, value) ;</code>	
<b>Arguments</b>	<i>functionid</i>	A number used to identify the function in the SAP system returned from <code>SAPCreateFunction</code> .
	<i>parametername</i>	The name of the parameter in the BAPI.
	<i>value</i>	The data to be passed to the function.

<b>Returns</b>	<p>A number indicating whether the function was called successfully or not:</p> <p>0            OK</p> <p>-1          Failed      An error occurred while executing the function.</p>
<b>Example</b>	<pre>\$functionid = sapcreatefunction(\$connectionid, "BAPI_CUSTOMER_GETLIST"); \$in_maximum = "11"; \$result = sapsetsimpleparameter(\$functionid, "MAXROWS",\$in_maximum);</pre> <p>This example sets the MAXROWS parameter for the function identified by \$functionid, (BAPI_CUSTOMER_GETLIST) MAXROWS is a scalar input parameter of the BAPI_CUSTOMER_GETLIST parameter.</p> <p><b>Note:</b> All input parameters must be set between the creation of the function interface (SAPCreateFunction), and the invoking of the actual function (SAPInvokeFunction).</p>

## SAPSetTableParameter

The SAPSetTableParameter function sets the value for a field in a table parameter of the function. The function can be executed in both the pre-processing and processing phases.

<b>Syntax</b>	<code>sapsettableparameter (\$functionid, tableparametername, rowindex, fieldname, value) ;</code>	
<b>Arguments</b>	<p><i>functionid</i></p> <p><i>tableparametername</i></p> <p><i>rowindex</i></p> <p><i>fieldname</i></p> <p><i>value</i></p>	<p>A number used to identify the function in the SAP system returned from SAPCreateFunction.</p> <p>Name of the table parameter in the BAPI.</p> <p>The line you want to set the value for in the table.</p> <p>The name of the field to be set.</p> <p>The data to pass to the function.</p>

<b>Returns</b>	<p>A number indicating whether the function was called successfully or not:</p> <p>0            OK</p> <p>-1           Failed        An error occurred while executing the function.</p>
<b>Example</b>	<pre>\$funcname = "BAPI_CUSTOMER_GETLIST"; \$functionid = sapcreatefunction(\$connectionID, \$funcname); \$tablename = "IDRANGE"; \$result = sapsettableparameter(\$functionid, \$tablename,1,"SIGN","I");</pre> <p>In this example, the BAPI_CUSTOMER_GETLIST function has a table parameter IDRANGE that serves as an input parameter for the function. Because tables can have multiple lines (records), you need to specify the row you want to set the value for in the table, using the <i>rowindex</i> parameter.</p> <p><b>Note:</b> If the row does not exist, the BAPI script function automatically creates it.</p>

## Examples of BAPI script functions

This section contains the following examples of using BAPI script functions with StreamServe:

- [Retrieving data from the SAP system](#) on page 92
- [Updating data in the SAP system](#) on page 95
- [Creating a Sales Order in StreamServe](#) on page 98

### Retrieving data from the SAP system

This example shows one way how to retrieve customer data from a SAP system for processing in StreamServe, using BAPI\_CUSTOMER\_GETLIST. In this example, the connection script is added to the Event, the 'get' script is added to the Process, and the disconnect script is run after the Process.

**Note:** Before trying to access data in the SAP system, you need to establish a connection to the SAP system. Before using any functions, you should test that the connection to the SAP system is functioning correctly.

## Script added to the Event

This script establishes a connection to the SAP system. If a valid *connectionid* is not returned, you could use the `SAPLastError` function to return the error string and log the information. You could also run an error process to notify the SAP system administrator that StreamServer could not connect with the SAP system.

```
//Connect to the SAP system
$connectionID=sapconnect("CLIENT=800 USER=MySAPUser
PASSWD=PASSWORD LANG=EN ASHOST=bostsap01 SYSNR=00");
If ($connectionID = "")
{
  //Returned Errors from the SAP system
  $errormessage = saplasterror();
  log (1, "The logon reported the following error:
"+$errormessage);
}
```

## Script added to the Process

This script retrieves customer data from the SAP system. The BAPI uses the selection criteria in the `IDRANGE` table as input for retrieving the matching set of customer records.

The first step creates the function, and checks if the function module exists in the called SAP system and retrieves the interface:

```
$funcname="BAPI_CUSTOMER_GETLIST";
$functionid=sapcreatefunction($connectionid,$funcname);
```

The next steps set all input parameters for the function call. In this example, the simple `IMPORT` parameter `MAXROWS` specifies the maximum number of returned customer records, and the selection for the customer records, which is done in the `TABLE` parameter `IDRANGE`. This table parameter checks against the customer number field in the SAP system for the specified customer record.

These steps set the values for the selection of the customers. In this example, all customers with customer numbers between (BT) 0000003330 and 0000003340 OR that start with a 1\* (CP = Compare pattern) are selected.

A similar input selection parameter exists for most BAPIs that return a list, for example `BAPI_MATERIAL_GETLIST`.

```
$result = sapsetsimpleparameter ($functionid, "MAXROWS", "100");
$tablename = "IDRANGE";
$in_option = "BT";
$in_lw_customerno = "0000003330";
$in_hg_customerno = "0000003340";
$in_option2 = "CP";
$in_lw_customerno2 = "1*";
$result = sapsettableparameter($functionid,$tablename,1,
"OPTION",$in_option);
$result = sapsettableparameter($functionid,$tablename,1,"LOW",
$in_lw_customerno);
```

```

$result = sapsettableparameter($functionid,$tablename,1,
"HIGH",$in_hg_customerno2);
$result = sapsettableparameter($functionid,$tablename,2,
"OPTION",$in_option2);
$result = sapsettableparameter($functionid,$tablename,2,"LOW",
$in_lw_customerno2);

```

Once the input values are set, you can invoke the function and check the result. This code calls the BAPI in the SAP system and checks if an error has occurred. The errors reported can be communication errors or exceptions raised by the BAPI in the SAP system. Most BAPIs report semantic errors in the RETURN parameter that you can read to check the function's success.

```

$result = sapinvokefunction2($functionid,1);
if ($result = "-1")
{
    $error = saplasterror ();
    log (1, "The Invocation of the BAPI failed: "+$error);
    // do your error handling here
}

```

The following code checks the EXPORT parameter RETURN for success, by getting the value of the TYPE field in the structured parameter RETURN. The value is then tested, with either an S (Success) or an empty value indicating that no problem was reported. This code uses a negative IF to exit the script, thereby not requiring to put the loop for retrieving the actual data into an IF statement.

```

$bapireturntype = sapgetcomplexparameter ($functionid,
"RETURN", "TYPE");
if(NOT(($bapireturn = "S") OR ($bapireturn = "")))
{
    // this code is only entered if an error occurred
    $bapimessage = sapgetcomplexparameter ($functionid,
"RETURN","MESSAGE");
    log (1, "the BAPI returns the following error : "
+$bapimessage);
    // do your error handling here
}

```

The last step retrieves all the returned data and copies it into StreamServe variables, using these variables in the Process output. The following script loops through all returned customers and sets the variables. The variable \$numofrows holds the number of returned customer records, because the code loops through each line. The table index starts with 1. The code uses the SAPGetTableParameter BAPI script function to set the variables.

```

$tablename = "ADDRESSDATA";
$numofrows = sapgettablerowcount($functionid,$tablename);
$idx = 0;
while(num($idx) < num($numofrows))
{
    $idx++;
    $valcustno = sapgettableparameter($functionid,$tablename,
num($idx),"CUSTOMER");
}

```

```
$valname = sapgettableparameter($functionid,$tablename,  
num($idx),"NAME");  
$valcountry = sapgettableparameter($functionid,$tablename,  
num($idx),"COUNTRY");  
$valstreet = sapgettableparameter($functionid,$tablename,  
num($idx),"STREET");  
$valzip = sapgettableparameter($functionid,$tablename,  
num($idx),"POSTL_COD1");  
$valcity = sapgettableparameter($functionid,$tablename,  
num($idx),"CITY");  
$valtel = sapgettableparameter($functionid,$tablename,  
num($idx),"TEL1_NUMBR");  
$valfax = sapgettableparameter($functionid,$tablename,  
num($idx),"FAX_NUMBER");  
$valadr = sapgettableparameter($functionid,$tablename,  
num($idx),"ADDRESS");  
callblock("customer");  
}
```

## Script run after the Process

The following script disconnects the connection to the SAP system:

```
//Disconnect from the SAP system  
$result=sapdisconnect($connectionID);  
log(1,"Logon Off="+$result);
```

## Updating data in the SAP system

This example shows one way how to update data in the SAP system.



You should only update data in the SAP system if you are an experienced SAP and StreamServe user, and take extreme care. StreamServe is not responsible for faulty input using the provided script functionality.

---

Before you implement one of the following scenarios, ensure you have understood the conditions of the update function in the SAP system and the business context.

This example uses the `BAPI_CUSTOMER_CHANGEFROMDATA` function to update data in the SAP system. The function has the following input parameters:

- `PI_SALESORG` - the sales organization the customer belongs to
- `PI_DISTR_CHAN` - the distribution channel the customer belongs to
- `PI_DIVISION` - the actual sales division that handles that customer
- `CUSTOMRENO` - the actual customer identifier
- `PI_ADDRESS` - the actual customer's address data that holds the data to be changed.

When such a scenario occurs in a business environment, the sales organization, division and distribution channel should be retrieved from either an LDAP directory or passed by the incoming Event.

### Script added to the Event

This script establishes a connection to the SAP system. If a valid *connectionid* is not returned, you could use the `SAPLastError` function to return the error string and log the information. You could also run an error process to notify the SAP system administrator that the StreamServer could not connect with the SAP system.

```
$connectionID=sapconnect("CLIENT=800 USER=MySAPUser
PASSWD=PASSWORD LANG=EN ASHOST=bostsap01 SYSNR=00");
If ($connectionID = "")
{
    //Returned Errors from the SAP system
    $errorMessage = saplasterror();
    log (1, "The logon reported the following error:
    "+$errorMessage);
}
```

### Script added to the Process

This script updates the customer data in the SAP system. The BAPI sets the channel information, such as sales organization, distribution channel and division, and identifies the customer by customer number. The parameter `PI_ADDRESS` holds the address data to be updated. In this script, the values are hard-coded. In a business environment, the address data, together with the customer number, should be part of the Event. The channel information could be either associated with the Event or read from an profile directory.

The first step creates the function:

```
$functionID = sapcreatefunction($connectionID,
"BAPI_CUSTOMER_CHANGEFROMDATA");
```

The following code sets the input values. These are scalar parameters set using the `SAPSetSimpleParameter` function.

```
$result = sapsetsimpleparameter($functionID, "PI_SALESORG",
"0001");
$result = sapsetsimpleparameter($functionID, "PI_DISTR_CHAN",
"01");
$result = sapsetsimpleparameter($functionID, "PI_DIVISION",
"01");
$result = sapsetsimpleparameter($functionID, "CUSTOMERNO",
"0000000020");
```

The address parameter is a structured input parameter set using the `SAPSetComplexParameter` that sets values for individual fields:

```
$result = sapsetcomplexparameter($functionID, "PI_ADDRESS",
"FIRST_NAME", "Joe");
$result = sapsetcomplexparameter($functionID, "PI_ADDRESS",
"NAME", "Klein");
```

```
$result = sapsetcomplexparameter($functionID, "PI_ADDRESS",  
"STREET","1 Stalker Lane");  
  
$result = sapsetcomplexparameter($functionID, "PI_ADDRESS",  
"CITY","Karlsruhe");  
  
$result = sapsetcomplexparameter($functionID, "PI_ADDRESS",  
"POSTL_CODE","076137");  
  
$result = sapsetcomplexparameter($functionID, "PI_ADDRESS",  
"REGION","04 ");  
  
$result = sapsetcomplexparameter($functionID, "PI_ADDRESS",  
"COUNTRY","DE ");  
  
$result = sapsetcomplexparameter($functionID, "PI_ADDRESS",  
"TELEPHONE","0721-123456");  
  
$result = sapsetcomplexparameter($functionID, "PI_ADDRESS",  
"LANGU_ISO","DE");  
  
$result = sapsetcomplexparameter($functionID, "PI_ADDRESS",  
"CURRENCY","EUR");
```

Once the values are set, the function is invoked and the result checked. The following code calls the BAPI in the SAP system and checks if an error has occurred. The errors reported can be communication errors or exceptions raised by the BAPI in the SAP system. Most BAPIs report semantic errors in parameter RETURN, which you can read to check the function's success.

```
$result = sapinvokefunction ($functionid);  
if ($result = "-1");  
{  
    $error = saplasterror ();  
    log (1, "The Invocation of the BAPI failed: "+$error);  
    // do your error handling here  
}  
  
$bapireturntype = sapgetcomplexparameter ($functionid,  
"RETURN", "TYPE");  
if(NOT(($bapireturn = "S") OR ($bapireturn = "")));  
{  
    // this code is only entered if an error occurred  
    $bapimessage = sapgetcomplexparameter ($functionid,  
"RETURN","MESSAGE");  
    log (1, "the BAPI return the following error :  
    "+$bapimessage);  
    // do your error handling here  
}
```

This code checks the EXPORT parameter RETURN for success, by getting the value of the TYPE field in the structured parameter RETURN. The value is then tested, with either an S (Success) or an empty value indicating that no problem was reported. This code uses a negative IF to exit the script and return the incomplete customer update.

The BAPI also returns the changed address data, and you can set the new variables using the following code. You can then use this data in the process to report the successful updating of the customer data.

```
$thefirstname = sapgetcomplexparameter($functionID,  
"PE_ADDRESS", "FIRST_NAME");
```

```

$thename = sapgetcomplexparameter($functionID, "PE_ADDRESS",
"NAME");
$thestreet = sapgetcomplexparameter($functionID, "PE_ADDRESS",
"STREET");
$thepostlcode = sapgetcomplexparameter($functionID,
"PE_ADDRESS", "POSTL_CODE");
$thecity = sapgetcomplexparameter($functionID, "PE_ADDRESS",
"CITY");
log(1,"first name =" + $thefirstname);
log(1,"name =" + $thename);
log(1,"street =" + $thestreet);
log(1,"postal code =" + $thepostlcode);
log(1,"city =" + $thecity);

```

### Script run after the Process

The following script disconnects the connection to the SAP system:

```

//Disconnect from the SAP system
$result=sapdisconnect($connectionID);
log(1,"Logon Off="+$result);

```

### Creating a Sales Order in StreamServe

This example shows one way how to create a Sales Order in StreamServe from data in a SAP system, using the SALESORDER\_CREATE BAPI function.

The first step creates the function:

```

$functionID = sapcreatefunction($connectionID,
"BAPI_SALESORDER_CREATEFROMDAT1");

```

The following code sets the input values. These are scalar parameters set using the SAPSetSimpleParameter function.

```

$result = 0;
If (preproc())
{
    sapsetsimpleparameter($functionID, "WITHOUT_COMMIT", "X");
    $result = sapinvokefunction2($functionID,1);
}
else
{
    sapsetsimpleparameter($functionID, "WITHOUT_COMMIT", "");
    $result = sapinvokefunction2($functionID,1);
}
//standard code continues

```

The code does the simulation in the pre-processing phase, with handling of error processes being handled in the normal way. The only difference when compared to the real phase, is that the actual update is done rather than being simulated.

You could also use the following code:

```

$result = -1;
sapsetsimpleparameter($functionID, "WITHOUT_COMMIT", "X");

```

```
$result = sapinvokefunction2($functionID,1);  
If (!preproc() AND ($result = "-1"))  
{  
    sapsetsimpleparameter($functionID, "WITHOUT_COMMIT", "");  
    $result = sapinvokefunction2($functionID,1);  
}
```

**100** | StreamServe BAPI script functions  
**Configuring the BAPI interface**

# Useful SAP transaction codes

---

This section lists SAP transaction codes which are commonly used in the SAP system to activate transactions.

**Note:** To enter a transaction code from any screen within the SAP system other than the initial screen, prefix the code with /n. For example, the /nVF03 transaction code would display the Display Billing Document screen from any screen in the SAP system.

## Configuration

SPRO	Customizing
OMFE	Processing Program/Layout Set for Purchase Order (MM)
V/30	Processing Program/Layout Set for Order Confirmation (SD)
V/34	Processing Program/Layout Set for Delivery Note (SD)
V/40	Processing Program/Layout Set for Invoice (SD)

## Spool functions

SPAD	Spool Administration
SP01	Spool Requests
SP02	Own Spool Requests

## Form processing

SE71	SAPscript
SE73	Font Maintenance
SE78	Management
SMARTFORMS	Smart Forms
SO10	Standard Texts

### Programs and reports

SE38	ABAP Editor
RSTXSCR	Import/Export SAPscript objects
RSTXSYMB	List SAP symbols
RSTXICON	List SAP icons
RSP00049	Activate Access Method Z (Spool Exit)

### Generating application output

ME22	Change Purchase Order (MM)
ME90	Print Purchase Order (MM)
VA02	Change Sales Order (Order Confirmation, SD)
VA03	Display Sales Order (Order Confirmation, SD)
VF02	Change Billing Document (Invoice, SD)
VF03	Display Billing Document (Invoice, SD)
VL02	Change Outbound Delivery (Delivery Note, SD)
VL03	Display Outbound Delivery (Delivery Note, SD)
SM69	List of external commands (for box drawing characters)
SM04	List of users currently logged on (short list)

### Data and metadata

SE11	Data Dictionary
SE16	Data Browser
WE63	IDoc Types

# IDoc Converter argument file examples

---

This section contains two examples of IDoc Converter argument files.

*Example 1 Argument file for an IDoc Service sending and receiving IDocs via a directory.*

```

<strs-configuration type="sapidocconverter">
  <section type="common">
    <propertybag id="log">
      <property id="logfile">C:\Program Files\StreamServe\4.1\Server\idoc.log</property>
      <property id="logmsgfile">C:\Program Files\StreamServe\4.1\Common\data\sapbp\sapidoc_logmsg.txt</property>
      <property id="langid">1</property>
      <property id="loglevel">9</property>
    </propertybag>
    <propertybag id="startupdata">
      <property id="useserver">1</property>
      <property id="useclient">1</property>
      <property id="metadatadirectory"></property>
    </propertybag>
    <propertybag id="lic">
      <property id="licfile"></property>
    </propertybag>
  </section>
  <section type="server">
    <propertybag id="rfcserver-connection">
      <property id="codepage"></property>
      <property id="kill">0</property>
      <property id="rfctrace">0</property>
      <property id="retrycount">3</property>
      <property id="retrywait">10</property>
    </propertybag>
    <propertybag id="channels">|
      <property id="packetsize">0</property>
      <property id="destination-channeltype">directory</property>
      <property id="destination-directory">c:\streamserve\bp\out</property>
      <property id="destination-url"></property>
      <property id="destination-indata"></property>
    </propertybag>
    <propertybag id="rfcserver-registration">
      <property id="gateway">192.168.127.24</property>
      <property id="gateway-service">sapgw02</property>
      <property id="progid">TESTIDOC CONVERTER</property>
    </propertybag>
    <propertybag id="server-http-settings">
      <property id="clientconnectiontimeout">500</property>
      <property id="clientcommunicationtimeout">500</property>
    </propertybag>
  </section>
  <section type="client">
    <propertybag id="rfcclient-connection">
      <property id="codepage"></property>
      <property id="connectionstring">CLIENT=600 USER=MTEST LANGU=EN ASHOST=192.168.127.24 SYSNR=02</property>
      <property id="passwd">strs410</property>
      <property id="usesnc">0</property>
    </propertybag>
    <propertybag id="channels">
      <property id="published_servicename"></property>
      <property id="source-scanpath">C:\streamserve\bp\in\</property>
      <property id="source-servicebroker_url"></property>
      <property id="source-channeltype">dirscan</property>
      <property id="source-httpin-URI"></property>
    </propertybag>
    <propertybag id="client-httpserver-configuration">
      <property id="ipport">1717</property>
      <property id="numberofthreads">30</property>
      <property id="connectiontimeout">500</property>
      <property id="communicationtimeout">500</property>
      <property id="tmpdir">C:\temp</property>
    </propertybag>
  </section>
</strs-configuration>

```

*Example 2*     *Argument file for an IDoc Converter service sending and receiving IDocs via HTTP*

```

<strs-configuration type="sapidocconverter">
  <section type="common">
    <propertybag id="log">
      <property id="logfile">C:\Program Files\StreamServe\4.1\Server\idoc.log</property>
      <property id="logmsgfile">C:\Program Files\StreamServe\4.1\Common\data\sapp\sapidoc_logmsg.txt</property>
      <property id="langid">1</property>
      <property id="loglevel">9</property>
    </propertybag>
    <propertybag id="startupdata">
      <property id="useserver">1</property>
      <property id="useclient">1</property>
      <property id="metadatadirectory"></property>
    </propertybag>
    <propertybag id="lic">
      <property id="licfile"></property>
    </propertybag>
  </section>
  <section type="server">
    <propertybag id="rfcserver-connection">
      <property id="codepage"></property>
      <property id="kill">0</property>
      <property id="rfctrace">0</property>
      <property id="retrycount">3</property>
      <property id="retrywait">10</property>
    </propertybag>
    <propertybag id="channels">
      <property id="packetize">0</property>
      <property id="destination-channeltype">simplehttp</property>
      <property id="destination-url">/idoc</property>
      <property id="destination-directory"></property>
      <property id="destination-indata"></property>
    </propertybag>
    <propertybag id="rfcserver-registration">
      <property id="gateway">192.168.127.24</property>
      <property id="gateway-service">sapgw02</property>
      <property id="progid">v</property>
    </propertybag>
    <propertybag id="server-http-settings">
      <property id="clientconnectiontimeout">500</property>
      <property id="clientcommunicationtimeout">500</property>
    </propertybag>
  </section>
  <section type="client">
    <propertybag id="rfcclient-connection">
      <property id="codepage"></property>
      <property id="connectionstring">CLIENT=600 USER=MTEST LANGU=EN ASHOST=192.168.127.24 SYSNR=02</property>
      <property id="passwd">strs410</property>
      <property id="usesnc">0</property>
    </propertybag>
    <propertybag id="channels">
      <property id="published_servicename"></property>
      <property id="source-sapnpatch"></property>
      <property id="source-servicebroker_url"></property>
      <property id="source-channeltype">Simplehttp</property>
      <property id="source-httpin-URI"></property>
    </propertybag>
    <propertybag id="client-httpserver-configuration">
      <property id="ipport">1717</property>
      <property id="numberofthreads">30</property>
      <property id="connectiontimeout">1000</property>
      <property id="communicationtimeout">500</property>
      <property id="tmpdir">C:\Program Files\StreamServe\4.1\Common\idoctemp</property>
    </propertybag>
  </section>
</strs-configuration>

```

