

***SeeBeyond ICAN Suite***

# **e\*Way Intelligent Adapter for SAP (BDC) User's Guide**

***Release 5.0.5 for Schema Run-time Environment (SRE)***



The information contained in this document is subject to change and is updated periodically to reflect changes to the applicable software. Although every effort has been made to ensure the accuracy of this document, SeeBeyond Technology Corporation (SeeBeyond) assumes no responsibility for any errors that may appear herein. The software described in this document is furnished under a License Agreement and may be used or copied only in accordance with the terms of such License Agreement. Printing, copying, or reproducing this document in any fashion is prohibited except in accordance with the License Agreement. The contents of this document are designated as being confidential and proprietary; are considered to be trade secrets of SeeBeyond; and may be used only in accordance with the License Agreement, as protected and enforceable by law. SeeBeyond assumes no responsibility for the use or reliability of its software on platforms that are not supported by SeeBeyond.

SeeBeyond, e\*Gate, e\*Way, and e\*Xchange are the registered trademarks of SeeBeyond Technology Corporation in the United States and/or select foreign countries. The SeeBeyond logo, SeeBeyond Integrated Composite Application Network Suite, eGate, eWay, eInsight, eVision, eXchange, eView, eIndex, eTL, ePortal, eBAM, and e\*Insight are trademarks of SeeBeyond Technology Corporation. The absence of a trademark from this list does not constitute a waiver of SeeBeyond Technology Corporation's intellectual property rights concerning that trademark. This document may contain references to other company, brand, and product names. These company, brand, and product names are used herein for identification purposes only and may be the trademarks of their respective owners.

© 2005 SeeBeyond Technology Corporation. All Rights Reserved. This work is protected as an unpublished work under the copyright laws.

**This work is confidential and proprietary information of SeeBeyond and must be maintained in strict confidence.**

Version 20050406034252.

# Contents

---

|                                     |           |
|-------------------------------------|-----------|
| <b>Preface</b>                      | <b>9</b>  |
| Intended Reader                     | 9         |
| Organization                        | 9         |
| Nomenclature                        | 10        |
| Online Use                          | 10        |
| Writing Conventions                 | 10        |
| <hr/>                               |           |
| <b>Chapter 1</b>                    |           |
| <b>Introduction</b>                 | <b>11</b> |
| SAP Interface Options               | 11        |
| Data Updating Methods               | 12        |
| The SAP BDC e*Way                   | 13        |
| Overview                            | 13        |
| e*Gate to SAP                       | 13        |
| SAP to e*Gate                       | 16        |
| Components                          | 16        |
| Supported Operating Systems         | 17        |
| <hr/>                               |           |
| <b>Chapter 2</b>                    |           |
| <b>Installation</b>                 | <b>18</b> |
| System Requirements                 | 18        |
| Environment Configuration           | 18        |
| External System Requirements        | 19        |
| External Configuration Requirements | 19        |
| Installing the e*Way                | 20        |
| Windows Systems                     | 20        |
| Installation Procedure              | 20        |
| Subdirectories and Files            | 22        |
| UNIX Systems                        | 23        |
| Installation Procedure              | 23        |

|                               |           |
|-------------------------------|-----------|
| Subdirectories and Files      | 24        |
| <b>Optional Example Files</b> | <b>25</b> |
| Installation Procedure        | 25        |
| Subdirectories and Files      | 26        |

## Chapter 3

|  |           |
|--|-----------|
| <b>System Implementation</b>           | <b>27</b> |
| <b>Overview</b>                        | <b>27</b> |
| Pre-Implementation Tasks               | 27        |
| Implementation Sequence                | 28        |
| Viewing e*Gate Components              | 28        |
| <b>Creating a Schema</b>               | <b>29</b> |
| <b>Creating Event Types</b>            | <b>30</b> |
| <b>Creating Event Type Definitions</b> | <b>30</b> |
| Using the SAP Screen Flow Recorder     | 33        |
| Using the ETD Editor's Build Tool      | 36        |
| Assigning ETDs to Event Types          | 38        |
| <b>Defining Collaborations</b>         | <b>39</b> |
| <b>Creating Intelligent Queues</b>     | <b>40</b> |
| <b>Building an SAP RFC Client</b>      | <b>41</b> |
| <b>Building an SAP RFC Server</b>      | <b>42</b> |
| <b>Sample Schemas</b>                  | <b>44</b> |
| SAP RFC Client                         | 44        |
| SAP RFC Server                         | 45        |

## Chapter 4

|  |           |
|--|-----------|
| <b>e*Way Extensions</b>                | <b>47</b> |
| <b>Overview</b>                        | <b>47</b> |
| <b>Custom ABAP Components</b>          | <b>48</b> |
| <b>Importing the Custom Components</b> | <b>50</b> |
| <b>Updating SAP R/3 Objects</b>        | <b>53</b> |
| e*Gate to SAP                          | 53        |
| SAP to e*Gate                          | 59        |

## Chapter 5

|                         |           |
|-------------------------|-----------|
| <b>Setup Procedures</b> | <b>66</b> |
| <b>Overview</b>         | <b>66</b> |

|   |           |
|---|-----------|
| <b>Setting Up the e*Way</b>                   | <b>67</b> |
| Creating the e*Way                            | 67        |
| Modifying e*Way Properties                    | 68        |
| Configuring the e*Way                         | 69        |
| Using the e*Way Editor                        | 70        |
| Section and Parameter Controls                | 71        |
| Parameter Configuration Controls              | 71        |
| Command-line Configuration                    | 72        |
| Getting Help                                  | 72        |
| Changing the User Name                        | 73        |
| Setting Startup Options or Schedules          | 73        |
| Activating or Modifying Logging Options       | 75        |
| Activating or Modifying Monitoring Thresholds | 76        |
| <b>Troubleshooting the e*Way</b>              | <b>77</b> |
| Configuration Problems                        | 77        |
| System-related Problems                       | 78        |

## Chapter 6

|   |           |
|---|-----------|
| <b>Operational Overview</b>             | <b>79</b> |
| <b>Obtaining the SAP Data Structure</b> | <b>79</b> |
| SAP Structure Export Module             | 79        |
| BDC Converter                           | 81        |
| <b>Data Mapping/Collaboration</b>       | <b>83</b> |
| e*Gate to SAP                           | 83        |
| SAP to e*Gate                           | 86        |
| <b>SAP BDC e*Way Architecture</b>       | <b>88</b> |
| Events and Collaborations               | 89        |
| Monk SAP BDC Transport Layer            | 90        |
| Monk SAP RFC Transport Layer            | 90        |
| e*Gate-to-SAP Data Exchange             | 90        |
| SAP-to-e*Gate Data Exchange             | 91        |
| <b>Basic e*Way Processes</b>            | <b>92</b> |
| Initialization Process                  | 93        |
| Connect to External Process             | 94        |
| Data Exchange Process                   | 95        |
| Disconnect from External Process        | 98        |
| Shutdown Process                        | 98        |

## Chapter 7

|                                 |            |
|---------------------------------|------------|
| <b>Configuration Parameters</b> | <b>99</b>  |
| <b>Overview</b>                 | <b>99</b>  |
| <b>General Settings</b>         | <b>100</b> |
| Journal File Name               | 100        |
| Max Resends Per Message         | 100        |
| Max Failed Messages             | 100        |
| Forward External Errors         | 101        |

|   |            |
|---|------------|
| <b>Communication Setup</b>                    | <b>102</b> |
| Start Exchange Data Schedule                  | 102        |
| Stop Exchange Data Schedule                   | 102        |
| Exchange Data Interval                        | 103        |
| Down Timeout                                  | 103        |
| Up Timeout                                    | 103        |
| Resend Timeout                                | 104        |
| Zero Wait Between Successful Exchanges        | 104        |
| <b>Monk Configuration</b>                     | <b>105</b> |
| Specifying Function or File Names             | 105        |
| Specifying Multiple Directories               | 105        |
| Load Path                                     | 105        |
| Additional Path                               | 105        |
| Auxiliary Library Directories                 | 106        |
| Monk Environment Initialization File          | 106        |
| Startup Function                              | 107        |
| Process Outgoing Message Function             | 107        |
| Exchange Data with External Function          | 108        |
| External Connection Establishment Function    | 109        |
| External Connection Verification Function     | 109        |
| External Connection Shutdown Function         | 110        |
| Positive Acknowledgment Function              | 110        |
| Negative Acknowledgment Function              | 111        |
| Shutdown Command Notification Function        | 112        |
| <b>SAP RFC Client Setup</b>                   | <b>113</b> |
| Host Name of the R/3 Target System            | 113        |
| System Number of the R/3 Target System        | 113        |
| Client  | 113        |
| User  | 113        |
| Password                                      | 113        |
| Language                                      | 114        |
| Enable RFC Trace                              | 114        |
| Log File for Failed Records                   | 114        |
| Maximum Number of Failure Before Erroring Out | 114        |
| <b>SAP RFC Server Setup</b>                   | <b>115</b> |
| Gateway Host Name                             | 115        |
| Gateway Service                               | 115        |
| Program ID                                    | 115        |
| Wait for Request Interval                     | 115        |
| Trace   | 115        |
| Log File for Failed Records                   | 116        |
| Maximum Number of Failure Before Erroring Out | 116        |
| <b>FTP Setup for File-based Transfers</b>     | <b>117</b> |
| User  | 117        |
| Password                                      | 117        |
| File Transfer Mode                            | 117        |
| Temporary File on Local System                | 117        |
| Temporary File on SAP Application Server      | 118        |
| Remote FTP Server Root Directory              | 118        |

---

## Chapter 8

|                      |            |
|----------------------|------------|
| <b>API Functions</b> | <b>119</b> |
| <b>Overview</b>      | <b>119</b> |
| BDC Transport Layer  | 120        |
| RFC Transport Layer  | 120        |

|  |            |
|--|------------|
| e*Way Kernel Layer                     | 120        |
| <b>SAP BDC Data Types</b>              | <b>121</b> |
| <b>BDC Transport Layer</b>             | <b>122</b> |
| <b>SAP BDC Template Functions</b>      | <b>122</b> |
| sapbdc-fromsap-startup                 | 122        |
| sapbdc-fromsap-connect                 | 123        |
| sapbdc-fromsap-finish                  | 123        |
| sapbdc-tosap-startup                   | 124        |
| sapbdc-tosap-connect                   | 124        |
| sapbdc-tosap-finish                    | 125        |
| sapbdc-ack                             | 125        |
| sapbdc-nack                            | 126        |
| <b>SAP BDC Utility Functions</b>       | <b>127</b> |
| sapbdc-client-connect                  | 127        |
| sapbdc-client-struct-send              | 127        |
| sapbdc-client-disconnect               | 128        |
| sapbdc-client-geterrormessage          | 128        |
| sapbdc-init                            | 129        |
| sapbdc-server-shutdown                 | 129        |
| sapbdc-server-startup                  | 130        |
| sapbdc-server-struct-fetch             | 130        |
| <b>RFC Transport Layer</b>             | <b>132</b> |
| <b>SAP RFC Client Functions</b>        | <b>132</b> |
| saprfc-client-connect                  | 132        |
| saprfc-client-callreceive              | 132        |
| saprfc-client-createtid                | 133        |
| saprfc-client-indirectcall             | 134        |
| saprfc-client-disconnect               | 134        |
| <b>SAP RFC Server Functions</b>        | <b>136</b> |
| saprfc-server-startup                  | 136        |
| saprfc-server-installfunction          | 137        |
| saprfc-server-installtransctrl         | 137        |
| saprfc-server-getinputdata             | 139        |
| saprfc-server-getcallbackfailuretid    | 140        |
| saprfc-server-getcallbackfailuretype   | 140        |
| saprfc-server-resetcallbackfailure     | 141        |
| saprfc-server-sendoutputdata           | 142        |
| saprfc-server-waitanddispatch          | 142        |
| saprfc-server-shutdown                 | 143        |
| <b>SAP Client Connection Functions</b> | <b>145</b> |
| saprfc-conn-createopt                  | 145        |
| saprfc-conn-settrace                   | 145        |
| saprfc-conn-set-clientconnpar          | 146        |
| saprfc-conn-set-clientconnmode         | 147        |
| saprfc-conn-set-clientconnopt-cpic     | 148        |
| saprfc-conn-set-clientconnopt-r3only   | 148        |
| <b>SAP Custom Structure Functions</b>  | <b>150</b> |
| saprfc-struct-create                   | 150        |
| saprfc-struct-add-entry                | 150        |
| saprfc-struct-install                  | 151        |
| <b>SAP Table List Functions</b>        | <b>153</b> |
| saprfc-tab-createlist                  | 153        |
| saprfc-tab-create                      | 154        |
| saprfc-tab-clear                       | 154        |
| saprfc-tab-appline                     | 155        |
| saprfc-tab-applines                    | 156        |
| saprfc-tab-countline                   | 156        |
| saprfc-tab-getwidth                    | 157        |
| saprfc-tab-getline                     | 157        |
| <b>SAP Parameter List Functions</b>    | <b>159</b> |

## Contents

|                                     |            |
|-------------------------------------|------------|
| saprfc-par-createlist               | 159        |
| saprfc-par-add-char                 | 160        |
| saprfc-par-add-int                  | 160        |
| saprfc-par-add                      | 161        |
| saprfc-par-add-receiving            | 162        |
| saprfc-par-get-char                 | 163        |
| saprfc-par-get-int                  | 163        |
| saprfc-par-get                      | 164        |
| <b>SAP Type-Checking Functions</b>  | <b>165</b> |
| saprfc-conn-handle?                 | 165        |
| saprfc-conn-opt?                    | 165        |
| saprfc-struct-handle?               | 166        |
| saprfc-par-list?                    | 166        |
| saprfc-tab-list?                    | 167        |
| <b>e*Way Kernel Layer</b>           | <b>168</b> |
| <b>Generic e*Way Functions</b>      | <b>168</b> |
| event-commit-to-egate               | 168        |
| event-rollback-to-egate             | 169        |
| event-send-to-egate                 | 169        |
| event-send-to-egate-ignore-shutdown | 170        |
| event-send-to-egate-no-commit       | 170        |
| get-logical-name                    | 171        |
| insert-exchange-data-event          | 171        |
| send-external-up                    | 172        |
| send-external-down                  | 172        |
| shutdown-request                    | 173        |
| start-schedule                      | 173        |
| stop-schedule                       | 174        |
| waiting-to-shutdown                 | 174        |

## Index

175



# Preface

This Preface contains information regarding the User's Guide itself.

---

## P.1 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the SeeBeyond™ e\*Gate™ Integrator system, and have a working knowledge of:

- Windows UNIX operations and administration
- Windows-style GUI operations
- SAP BDC concepts and operations
- Integrating SAP R/3 with external systems

---

## P.2 Organization

This User's Guide is organized roughly into two parts. The first part, consisting of Chapters 1-5, introduces the e\*Way and describes the procedures for installing the e\*Way and implementing a working system incorporating the e\*Way. Chapter 3 also contains descriptions of the sample schemas provided with the product. These can be used to test your system following installation and, if appropriate, as templates you can modify to produce your own custom schemas. This part should be of particular interest to a System Administrator or other user charged with the task of getting the system up and running.

The second part, consisting of Chapters 6-8, describes the architecture and internal functionality of the e\*Way. This part should be of particular interest to a Developer involved in customizing the e\*Way for a specific purpose. Information contained in this part that is necessary for the initial setup of the e\*Way is cross-referenced in the first part of the guide, at the appropriate points in the procedures.

---

## P.3 Nomenclature

Note that for purposes of brevity, the e\*Way Intelligent Adapter for SAP (BDC) is frequently referred to as the SAP BDC e\*Way, or simply the e\*Way.

---

## P.4 Online Use

This User's Guide is provided in Adobe Acrobat's Portable Document Format (PDF). As such, it can be printed out on any printer or viewed online. When viewing online, you can take advantage of the extensive hyperlinking imbedded in the document to navigate quickly throughout the Guide.

Hyperlinking is available in:

- The Table of Contents
- The Index
- Within the chapter text, indicated by **blue print**

Existence of a hyperlink *hotspot* is indicated when the hand cursor points to the text. Note that the hotspots in the Index are the *page numbers*, not the topics themselves. Returning to the spot you hyperlinked from is accomplished by right-clicking the mouse and selecting **Go To Previous View** on the resulting menu.

---

## P.5 Writing Conventions

The writing conventions listed in this section are observed throughout this document.

### Monospaced (Courier) Font

Computer code and text to be typed at the command line are set in Courier as shown below:

```
Configuration for BOB_Promotion
java -jar ValidationBuilder.jar
```

Variables within a command line, or attributes within a function signature, are set in italics as shown below:

```
stcregutl -rh host-name -un user-name -up password -sf
(sapbdc-ack string)
```

### Bold Sans-serif Font

- User Input: Click **Apply** to save, or **OK** to save and close.
- File Names and Paths: In the **Open** field, type **D:\setup\setup.exe**.
- Parameter, Function, and Command Names: The default parameter **localhost** is normally only used for testing; the Monk function **iq-put** places an Event into an IQ.

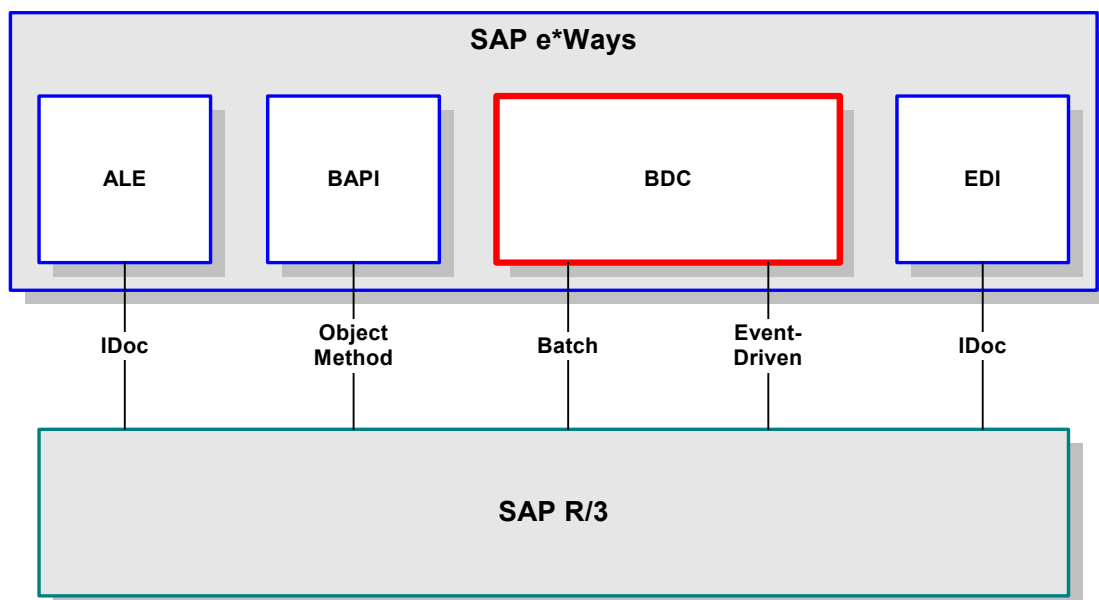
# Introduction

The e\*Way Intelligent Adapter for SAP (BDC) has been designed specifically to connect e\*Gate to SAP enterprise-management software within a network of diverse hardware and software systems. Using one or more SAP e\*Ways, e\*Gate can act as a bus, linking SAP applications and other software systems, or differently-configured SAP systems. This e\*Way allows bidirectional data exchange between e\*Gate and an SAP R/3 system via BDC (Batch Data Communications).

## 1.1 SAP Interface Options

SAP offers several interface options, including Application Link Enabling (ALE), Business Application Programming Interface (BAPI), and Electronic Data Interchange (EDI). The Batch Data Communication (BDC) interface actually is a user-emulation method that can be either batch or event-driven.

**Figure 1** SAP Interface Options



## 1.1.1 Data Updating Methods

SAP uses standard programs called *Transactions* to load data into the SAP database while following business functionality and logic. These transactions can be executed in the background using two different methods. The two methods are **Batch Data Communication** (BDC) and **Call Transaction**. Each of these standard SAP interfacing technologies emulates a user entering transactions through the SAP windows.

Being tied to the user interface forces the input data to conform to the business logic that is embedded in the interface. This provides a more robust method of data input than Intermediate Documents (IDocs), which both ALE and EDI interfacing methods employ. Since IDocs do not have to conform to the business logic, using them allows the introduction of non-conforming information into the database, which may result in corruption of the data.

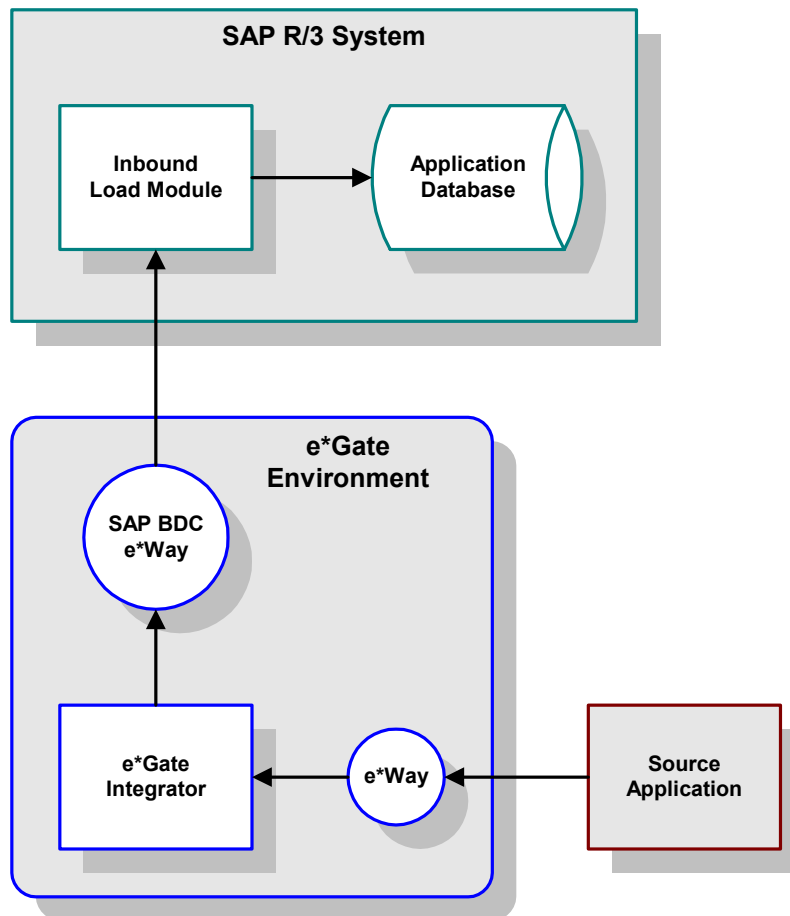
*Note:* The SAP BDC e\*Way supports both **BDC** and **Call Transaction** updating methods.

## 1.2 The SAP BDC e\*Way

### 1.2.1 Overview

#### e\*Gate to SAP

**Figure 2** e\*Gate-to-SAP Data Flow



Inbound data is extracted from some source application by the appropriate e\*Way. The data is sent to e\*Gate, which processes the data as needed to transform it into the format required by SAP, and sends it to the SAP BDC e\*Way. The e\*Way calls an RFC-enabled function module within SAP which, in turn, invokes the Inbound Load Module to load the data into the SAP database.

The data is loaded either from a data table supplied by the SAP e\*Way, or from a data file whose name is supplied as a parameter by the e\*Way. The data is placed into an internal table which, in turn, is used to load an internal BDC table. The internal BDC table is mapped either according to the standard screen flow or by customized mapping logic.

Two modes of operation are supported: **Batch** and **Event-driven**.

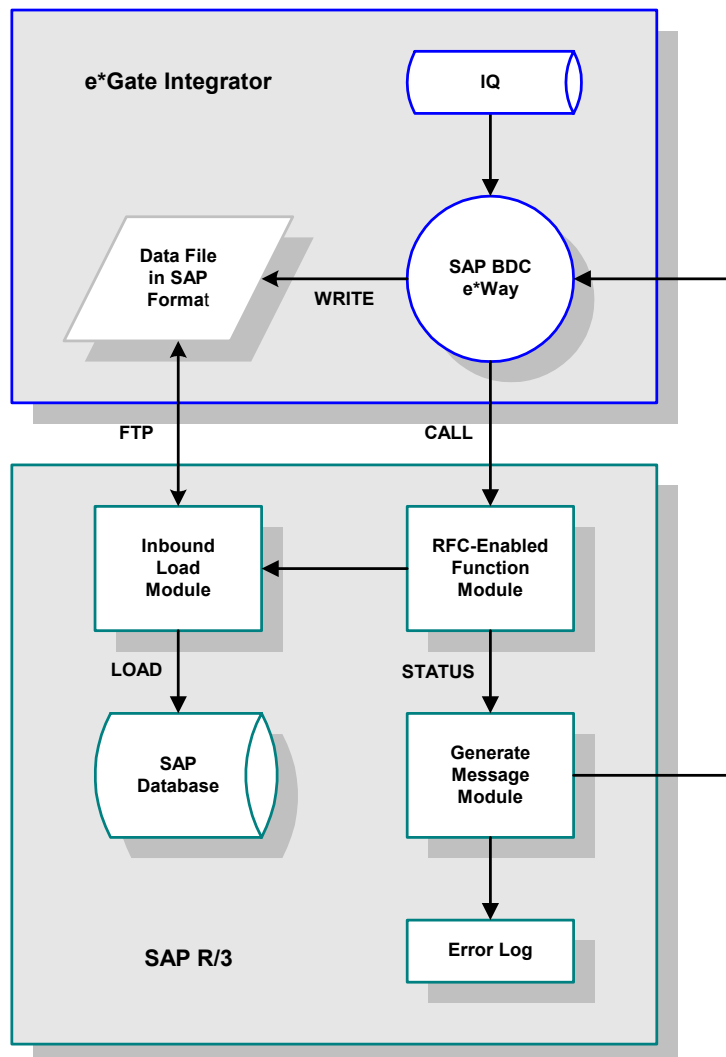
### Batch Mode

In batch mode, transactions are collected until they are triggered by a user, an application event, or a scheduling mechanism. The collection then is propagated as a whole to the SAP application, and submitted in a BDC session.

The BDC session is an asynchronous process; therefore, once a BDC session is submitted there is no return code sent from the call function to the calling program. However, BDC allows session splitting to break up large files into smaller groups of records for processing. In addition, BDC promotes multi-threading by allowing several sessions to be processed simultaneously.

Operating in batch mode, the SAP BDC e\*Way creates a data file and calls an RFC-enabled function internal to SAP. This function invokes the Inbound Batch Load Module, using the file name as a parameter. The SAP Inbound Batch Load Module then reads the data file via FTP and loads the data into the SAP database.

**Figure 3** e\*Gate-to-SAP, Batch-mode Process Flow

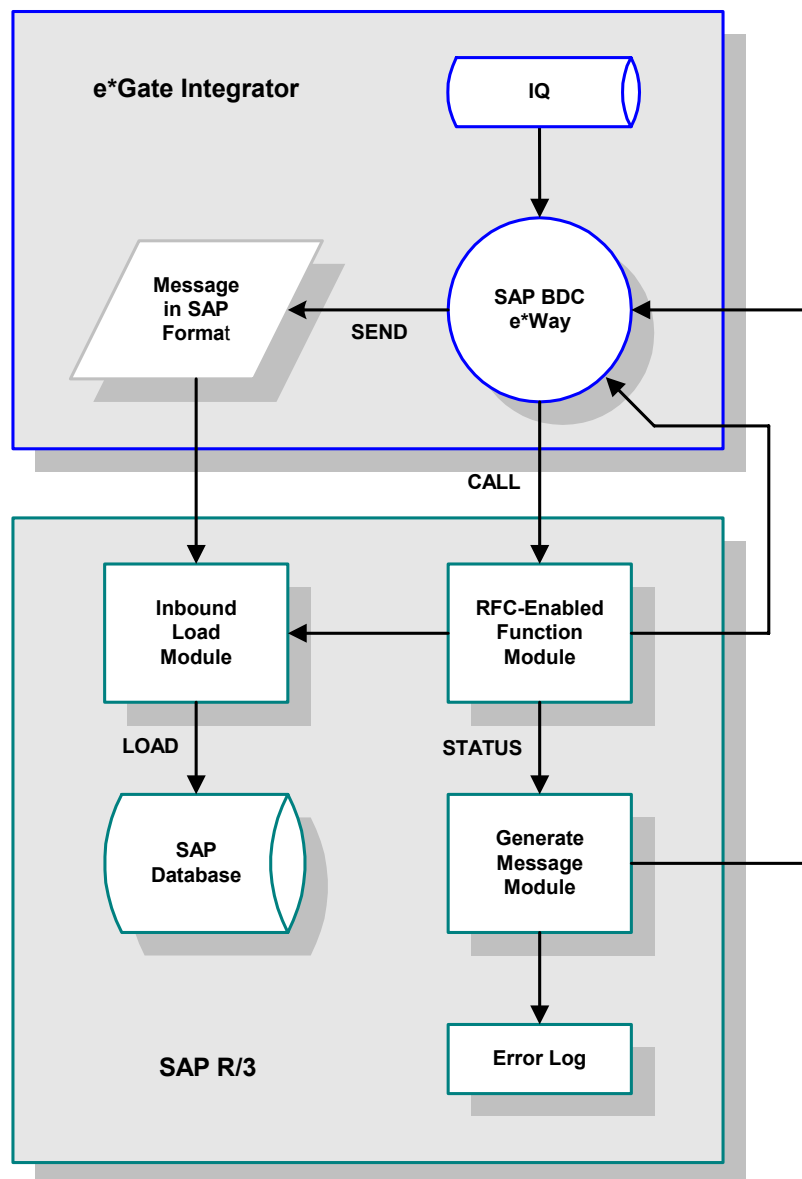


### Event-driven Mode

In event-driven mode, individual transactions are propagated to SAP as they are triggered by a user or an application event, such as the creation of a sales order or posting of a deposit. Here, the SAP BDC e\*Way creates a message and calls an RFC-enabled function internal to SAP. This function invokes the Inbound Batch Load Module which then receives the message and loads the data into the SAP database.

Data is formatted into a table much like a BDC session; however, each transaction is submitted as an individual process. Once each process has completed, a return code is sent from the Call Transaction function back to the calling program. Therefore, this method permits predicating the submission of a transaction upon the successful completion of the preceding transaction.

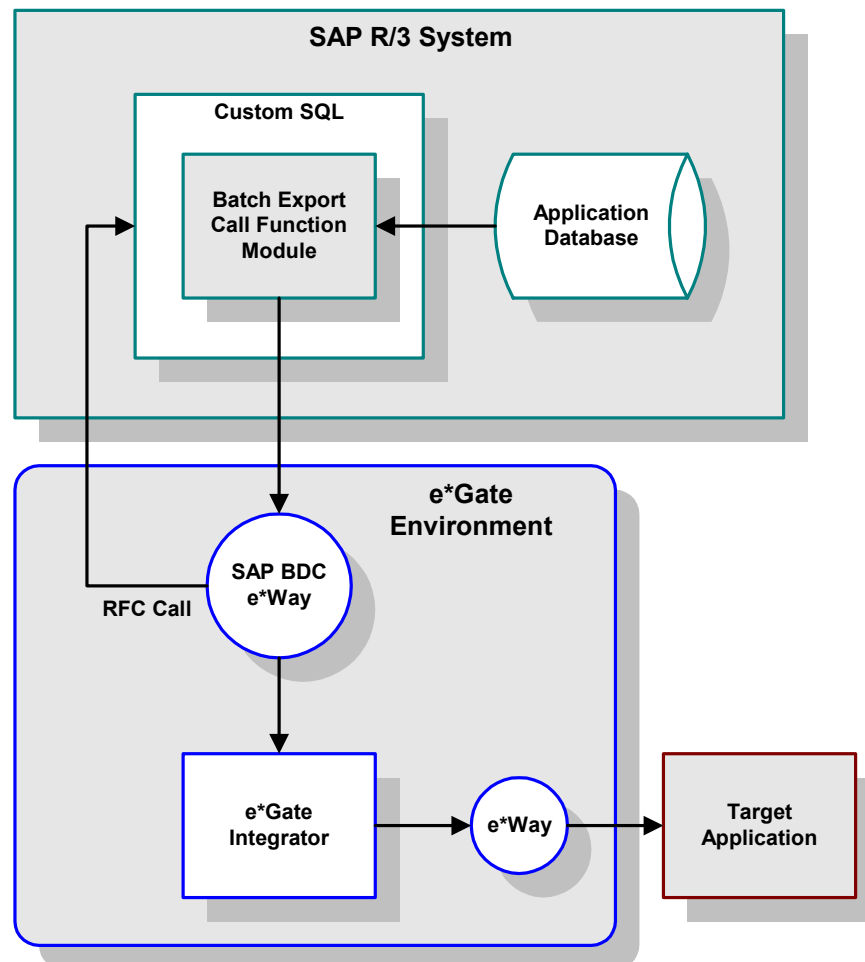
**Figure 4** e\*Gate-to-SAP, Event-driven Process Flow



## SAP to e\*Gate

The SAP-to-e\*Gate interface process begins with an SAP function module making an RFC call to initiate data transfer to the BDC e\*Way. The e\*Way receives the data from the SAP system and sends it into the e\*Gate system, which then sends the extracted data to a target system.

**Figure 5** SAP-to-e\*Gate Process Flow



### 1.2.2 Components

The SAP BDC e\*Way incorporates the following components:

- An executable file (Generic e\*Way Kernel), **stcewgenericmonk.exe**
- An accompanying dynamic load library, **stc\_monksap.dll**, which extends the executable file to form the SAP BDC e\*Way
- A default configuration file, **ewsapbdc.def**
- Monk function scripts and library files, discussed in [Chapter 8](#)



- SeeBeyond SAP BDC Converter, *stcsapbdconverter.exe*, used to build Event Type Definitions

For a list of installed files, see [Installing the e\\*Way](#) on page 20.

## 1.3 Supported Operating Systems

The e\*Way Intelligent Adapter for SAP BDC currently supports the following combinations of operating systems and releases of SAP R/3.

**Table 1** English-language Version

| Operating System                     | 4.0B | 4.5B | 4.6B | 4.6C |
|--------------------------------------|------|------|------|------|
| Windows 2000 and Windows Server 2003 | X    | X    | X    | X    |
| HP-UX 11.0 and 11i (PA-RISC)         | X    | X    | X    | X    |
| IBM AIX 5.1L                         | X    | X    | X    | X    |
| Sun Solaris 8                        | X    | X    | X    | X    |

**Table 2** Japanese-language Version

| Operating System                     | 4.0B | 4.5B | 4.6B | 4.6C |
|--------------------------------------|------|------|------|------|
| Windows 2000 and Windows Server 2003 | -    | -    | -    | X    |
| HP-UX 11.0 and 11i (PA-RISC)         | -    | -    | -    | X    |
| Sun Solaris 8                        | -    | -    | -    | X    |

**Table 3** Korean-language Version

| Operating System             | 4.0B | 4.5B | 4.6B | 4.6C |
|------------------------------|------|------|------|------|
| HP-UX 11.0 and 11i (PA-RISC) | -    | -    | -    | X    |
| IBM AIX 5.1L                 | -    | -    | -    | X    |

# Installation

This chapter describes the requirements and procedures for installing the e\*Way software. Procedures for implementing a working system, incorporating instances of the e\*Way, are described in [Chapter 3](#).

**Note:** Please read the *readme.txt* file located in the *addons\ewsapbdc* directory on the installation CD-ROM for important information regarding this installation.

---

## 2.1 System Requirements

To use the e\*Way Intelligent Adapter for SAP BDC, you need the following:

- 1 An e\*Gate Participating Host.
- 2 A TCP/IP network connection.
- 3 Sufficient free disk space to accommodate e\*Way files:
  - ♦ Approximately 1.4 MB on Windows systems
  - ♦ Approximately 5.6 MB on Solaris systems
  - ♦ Approximately 4.1 MB on HP-UX systems
  - ♦ Approximately 4.5 MB on AIX systems

**Note:** Additional disk space is required to process and queue the data that this e\*Way processes; the amount necessary varies, based on the type and size of the data being processed.

### 2.1.1 Environment Configuration

No changes are required to the Participating Host's operating environment to support this e\*Way.

---

## 2.2 External System Requirements

The e\*Way Intelligent Adapter for SAP BDC supports the following applications (see also [Supported Operating Systems](#) on page 17).

### English

- SAP R/3 release 4.0B, 4.5B, or 4.6C.

### 2.2.1 External Configuration Requirements

No changes are required to the SAP R/3 system. At your option, you can import extensions provided with the e\*Way to assist in development and system integration tasks (see [e\\*Way Extensions](#) on page 47).

## 2.3 Installing the e\*Way

### 2.3.1 Windows Systems

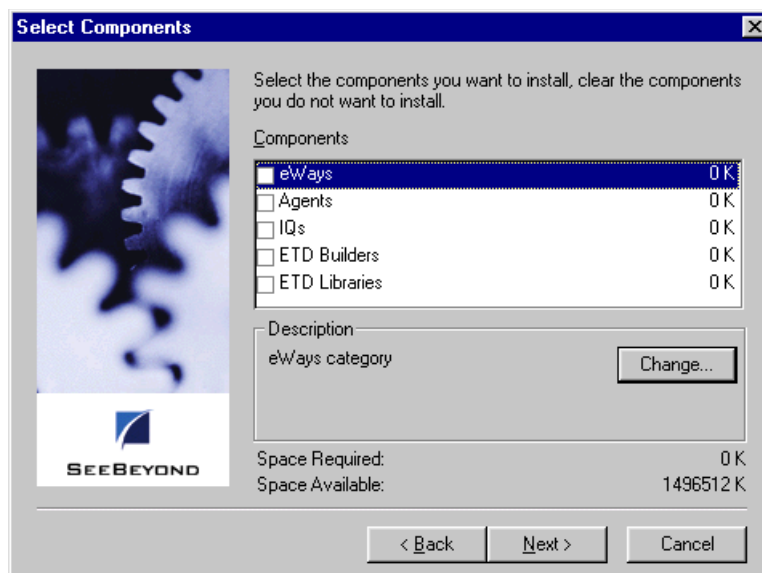
#### Installation Procedure

**Note:** *The installation utility detects and suggests the appropriate installation directory. Use this directory unless advised otherwise by SeeBeyond.*

#### To install the e\*Way on a Microsoft Windows system

- 1 Log in as an Administrator on the workstation on which you want to install the e\*Way (you must have Administrator privileges to install this e\*Way).
- 2 Exit all Windows programs and disable any anti-virus applications before running the setup program.
- 3 Insert the e\*Way installation CD-ROM into the CD-ROM drive.
- 4 Launch the setup program.
  - A If the CD-ROM drive's Autorun feature is enabled, the setup program should launch automatically. Follow the on-screen instructions until the **Choose Product** dialog box appears (see Figure 6). Check **Add-ons**, then click **Next**.

**Figure 6** Choose Product Dialog Box

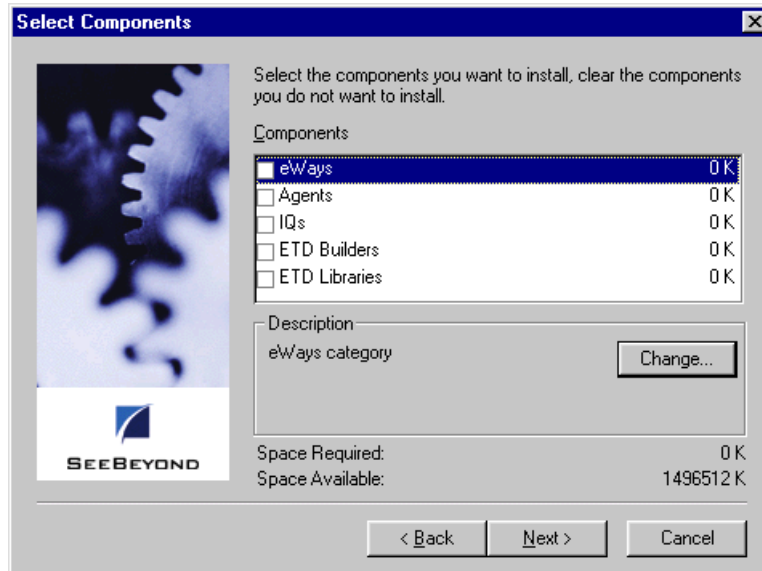


- B If the setup program does not launch automatically, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the following file on the CD-ROM drive (this bypasses the **Choose Product** dialog):

setup\addons\setup.exe

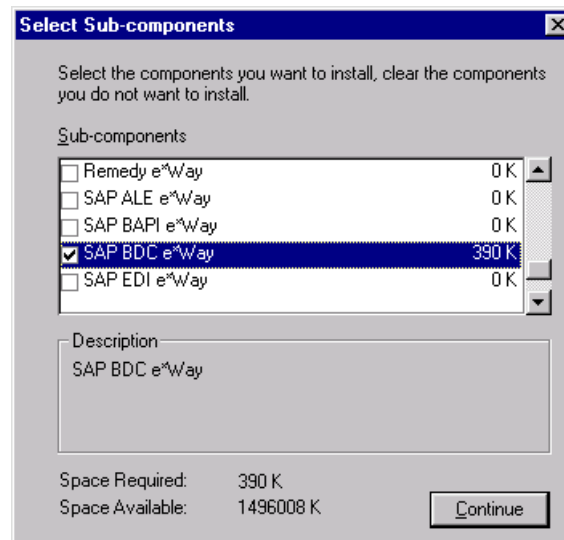
- 5 Follow the on-screen instructions until the **Select Components** dialog box appears (see Figure 7). Highlight—but do not check—**eWays** and then click **Change**.

**Figure 7** Select Components Dialog Box



- 6 When the **Select Sub-components** dialog box appears (see Figure 8), check the **SAP BDC e\*Way**.

**Figure 8** Select e\*Way Dialog



- 7 Click **Continue**, and the **Select Components** dialog box reappears.
- 8 Click **Next** and continue with the installation.

## Subdirectories and Files

By default, the InstallShield installer creates the following subdirectories and installs the following files within the \eGate\client tree on the Participating Host, and the \eGate\Server\registry\repository\default tree on the Registry Host.

**Table 4** Participating Host & Registry Host

| Subdirectories             | Files  |
|----------------------------|--|
| \bin\                      | librfc32.dll<br>stcsapbdccconverter.exe<br>stc_ewftp.ctl<br>stc_ewftp.dll<br>stc_monkfilesys.dll<br>stc_monksap.ctl<br>stc_monksap.dll   |
| \configs\stcewgenericmonk\ | ewsapbdc.def<br>sapBdc3.6To4.1Rule.txt   |
| \monk_library\             | ewsapbdc.gui<br>saprfc.gui   |
| \monk_library\sapbdc\      | sapbdc.monk<br>sapbdc-init.monk<br>sapbdc-client-connect.monk<br>sapbdc-client-disconnect.monk<br>sapbdc-client-geterrormessage.monk<br>sapbdc-client-struct-send.monk<br>sapbdc-ftp-connect.monk<br>sapbdc-server-shutdown.monk<br>sapbdc-server-startup.monk<br>sapbdc-server-struct-fetch.monk<br>saprfc-init.monk<br>saprfc-par-pad.monk |

By default, the InstallShield installer also installs the following file within the \eGate\Server\registry\repository\default tree on the Registry Host.

**Table 5** Registry Host Only

| Subdirectories | Files           |
|----------------|-----------------|
| \              | stcewsapbdc.ctl |

## 2.3.2 UNIX Systems

### Installation Procedure

**Note:** *You are not required to have root privileges to install this e\*Way. Log on under the user name that you wish to own the e\*Way files. Be sure that this user has sufficient privilege to create files in the e\*Gate directory tree.*

#### To install the e\*Way on a UNIX system

- 1 Log onto the workstation containing the CD-ROM drive and, if necessary, mount the drive.
- 2 Insert the e\*Way installation CD-ROM into the CD-ROM drive.
- 3 At the shell prompt, type  

```
cd /cdrom
```
- 4 Start the installation script by typing:  

```
setup.sh
```
- 5 A menu appears, containing several options. Select the **Install e\*Way** option, and follow any additional on-screen directions.

**Note:** *The installation utility detects and suggests the appropriate installation directory. Use this directory unless advised otherwise by SeeBeyond. Note also that **no spaces** should appear in the installation path name.*

## Subdirectories and Files

The preceding installation procedure creates the following subdirectories and installs the following files within the /eGate/client tree on the Participating Host, and the /eGate/Server/registry/repository/default tree on the Registry Host.

**Table 6** Participating Host & Registry Host

| Subdirectories             | Files  |
|----------------------------|--|
| /bin/                      | librfccm.sl (HP-UX only)<br>stc_ewftp.ctl<br>stc_monkfilesys.dll<br>stc_monksap.ctl<br>stc_monksap.dll   |
| /configs/stcewgenericmonk/ | ewsapbdc.def<br>sapBdc3.6To4.1Rule.txt   |
| /monk_library/             | ewsapbdc.gui<br>saprfc.gui   |
| /monk_library/sapbdc/      | sapbdc.monk<br>sapbdc-init.monk<br>sapbdc-client-connect.monk<br>sapbdc-client-disconnect.monk<br>sapbdc-client-geterrormessage.monk<br>sapbdc-client-struct-send.monk<br>sapbdc-ftp-connect.monk<br>sapbdc-server-shutdown.monk<br>sapbdc-server-startup.monk<br>sapbdc-server-struct-fetch.monk<br>saprfc-init.monk<br>saprfc-par-pad.monk |

The preceding installation procedure also installs the following file only within the /eGate/Server/registry/repository/default tree on the Registry Host.

**Table 7** Registry Host Only

| Subdirectories | Files           |
|----------------|-----------------|
| /              | stcewsapbdc.ctl |



## 2.4 Optional Example Files

The installation CD-ROM contains two sample schemas, **SapBdcFromSap** and **SapBdcToSap**, in the **samples\ewsapbdc** directory. To use a schema, you must load it onto your system using the following procedure. See **Sample Schemas** on page 44 for descriptions of the sample schemas.

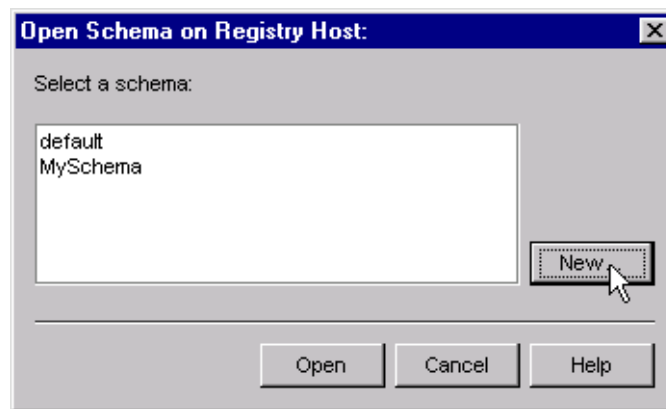
**Note:** *The SAP BDC e\*Way must be properly installed on your system before you can run the sample schema.*

### 2.4.1 Installation Procedure

To load a sample schema

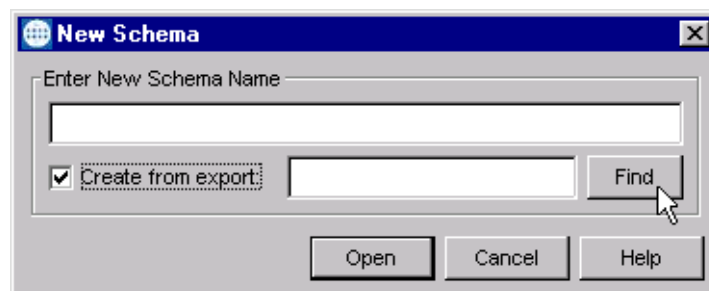
- 1 Invoke the **Open Schema** dialog box and select **New** (see Figure 9).

**Figure 9** Open Schema Dialog



- 2 Type the name you want to give to the schema (for example, **FromSap.Sample**)
- 3 Select **Create from export** and navigate to the directory containing the sample schema by clicking the **Find** button (see Figure 10).

**Figure 10** New Schema Dialog



- 4 Select the desired archive file (\*.zip) and click **Open**.

**Note:** The schema installs with the host name **localhost** and control broker name **localhost\_cb**. If you want to assign your own names, copy the file **\*.zip** to a local directory and extract the files. Using a text editor, edit the file **\*.exp**, replacing all instances of the name **localhost** with your desired name. Add the edited **.exp** file back into the **.zip** file.

## 2.4.2 Subdirectories and Files

The preceding procedure creates the following subdirectories and installs the following files within the `\eGate\Server\registry\repository\<SchemaName>` tree on the Registry Host, where `<SchemaName>` is the name you have assigned to the schema in step 2.

**Table 8** Subdirectories and Files - SapBdcFromSap

| Subdirectories                             | Files                          |
|--|--------------------------------|
| \  | SapBdcFromSap.ctl              |
| \runtime\client\data\SapBdcFromSap\input\  | fb01.dat                       |
| \runtime\client\data\SapBdcFromSap\output\ | fb01.dat                       |
| \runtime\configs\stcewfile\                | eater.cfg<br>eater.sc          |
| \runtime\configs\stcewgenericmonk\         | FromSap.cfg<br>FromSap.sc      |
| \runtime\monk_scripts\common\              | bkp.fssc<br>FromSapReceive.dsc |

**Table 9** Subdirectories and Files - SapBdcToSap

| Subdirectories                           | Files                                  |
|--|--|
| \  | SapBdcToSap.ctl                        |
| \runtime\client\data\SapBdcToSap\input\  | fb01.dat                               |
| \runtime\client\data\SapBdcToSap\output\ | fb01.dat                               |
| \runtime\configs\stcewfile\              | feeder_SAPBDC.cfg<br>feeder_SAPBDC.sc  |
| \runtime\configs\stcewgenericmonk\       | ToSapBdc.cfg<br>ToSapBdc.sc            |
| \runtime\monk_scripts\common\            | dummy.ssc<br>FB01.ssc<br>ToSapPost.tsc |

# System Implementation

In this chapter we take a more detailed look at the information presented in the Introduction, and describe the steps required for setting up a working system. Please refer to the *e\*Gate Integrator User's Guide* for additional information.

---

## 3.1 Overview

This e\*Way provides a specialized transport component for incorporation in an operational schema. The schema also contains Collaborations, linking different data or Event types, and Intelligent Queues. Typically, other e\*Way types also are used as components of the schema.

The topics discussed in this chapter include the following:

### 3.1.1 Pre-Implementation Tasks

#### Installation of SeeBeyond Software

The first task is to install the SeeBeyond software as described in [Chapter 2](#).

#### Importation of Sample Schemas

If you want to use the sample schemas supplied with the e\*Way, the schema files must be imported from the installation CD-ROM (see [Optional Example Files](#) on page 25).

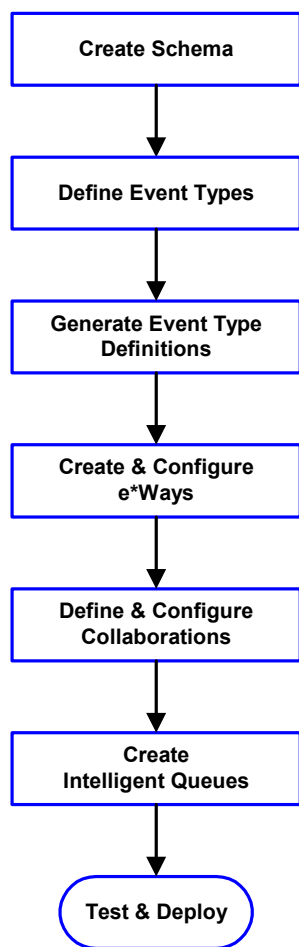
**Note:** *It is highly recommended that you make use of the sample schemas to familiarize yourself with e\*Way operation, test your system, and use as templates for your working schemas.*

#### Importation of Custom ABAP Components

If you want to use the provided custom transport extensions, your SAP R/3 environment must be modified. See [Chapter 4](#).

**Note:** *These transport extensions are optional.*

### 3.1.2 Implementation Sequence



- 1 The first step is to create a new Schema—the subsequent steps apply only to this Schema (see [Creating a Schema](#) on page 29).
- 2 The second step is to define the Event Types you are transporting and processing within the Schema (see [Creating Event Types](#) on page 30).
- 3 Third, you need to associate the Event Types created in the previous step with Event Type Definitions (ETDs) derived from the applicable Business Rules (see [Creating Event Type Definitions](#) on page 30).
- 4 The fourth step is to create and configure the required e\*Ways (see [Chapter 5](#)).
- 5 Next is to define and configure the Collaborations linking the Event Types from step 2 (see [Defining Collaborations](#) on page 39).
- 6 Now you need to create Intelligent Queues to hold published Events (see [Creating Intelligent Queues](#) on page 40).
- 7 Finally, you must test your Schema. Once you have verified that it is working correctly, you may deploy it to your production environment.

### 3.1.3 Viewing e\*Gate Components

Use the Navigator and Editor panes of the e\*Gate Schema Designer to view the various e\*Gate components. Note that you may only view components of a single schema at one time, and that all operations apply only to the current schema. All procedures in this chapter should be performed while displaying the **Components** Navigator pane. See the *e\*Gate Integrator User's Guide* for a detailed description of the features and use of the Schema Designer.

## 3.2 Creating a Schema

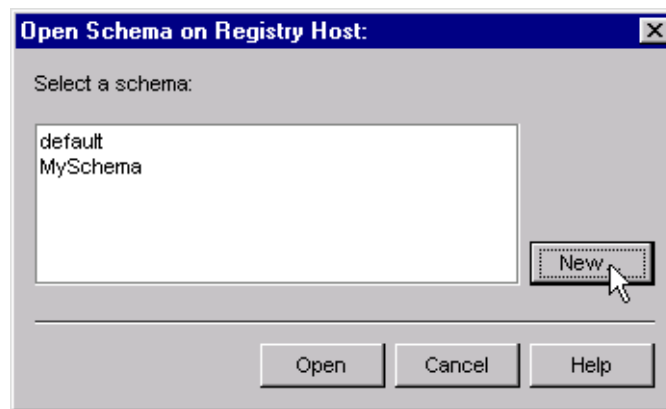
A schema is the structure that defines e\*Gate system parameters and the relationships between components within the e\*Gate system. Schemas can span multiple hosts.

Because all setup and configuration operations take place within an e\*Gate schema, a new schema must be created, or an existing one must be started before using the system. Schemas store all their configuration parameters in the e\*Gate Registry.

To select or create a schema

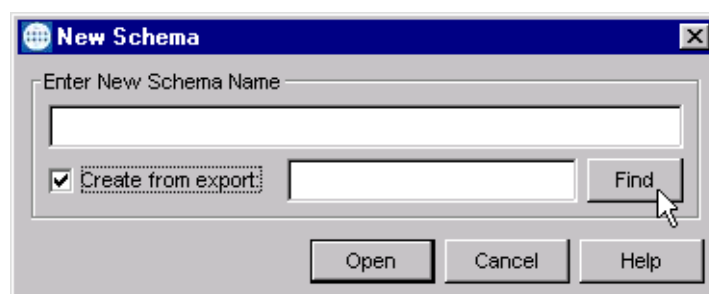
- 1 Invoke the **Open Schema** dialog box and **Open** an existing schema or click **New** to create a new schema.

**Figure 11** Open Schema Dialog



- 2 Clicking **New** invokes the **New Schema** dialog box (Figure 12).

**Figure 12** New Schema Dialog




- 3 Enter a new schema name and click **Open**.
- 4 The e\*Gate Schema Designer then opens under your new schema name.
- 5 From the **Options** menu, click on **Default Editor** and select **Monk**.
- 6 Select the **Components** tab, found at the bottom of the Navigator pane of the e\*Gate Schema Designer window.
- 7 You are now ready to begin creating the necessary components for this new schema.

---

## 3.3 Creating Event Types

Within e\*Gate, messages and/or packages of data are defined as Events. Each Event must be categorized into a specific Event Type within the schema.

### To define the Event Types

- 1 In the e\*Gate Schema Designer's Navigator pane, select the **Event Types** folder.
- 2 On the Palette, click the **New Event Type** button .
- 3 In the **New Event Type Component** box, enter the name for the input Event Type and click **Apply**. Use this method to create all required Event Types, for example:
  - ♦ **InboundEvent**
  - ♦ **ValidEvent**
  - ♦ **InvalidEvent**
- 4 After you have created the final Event Type, click **OK**.

---

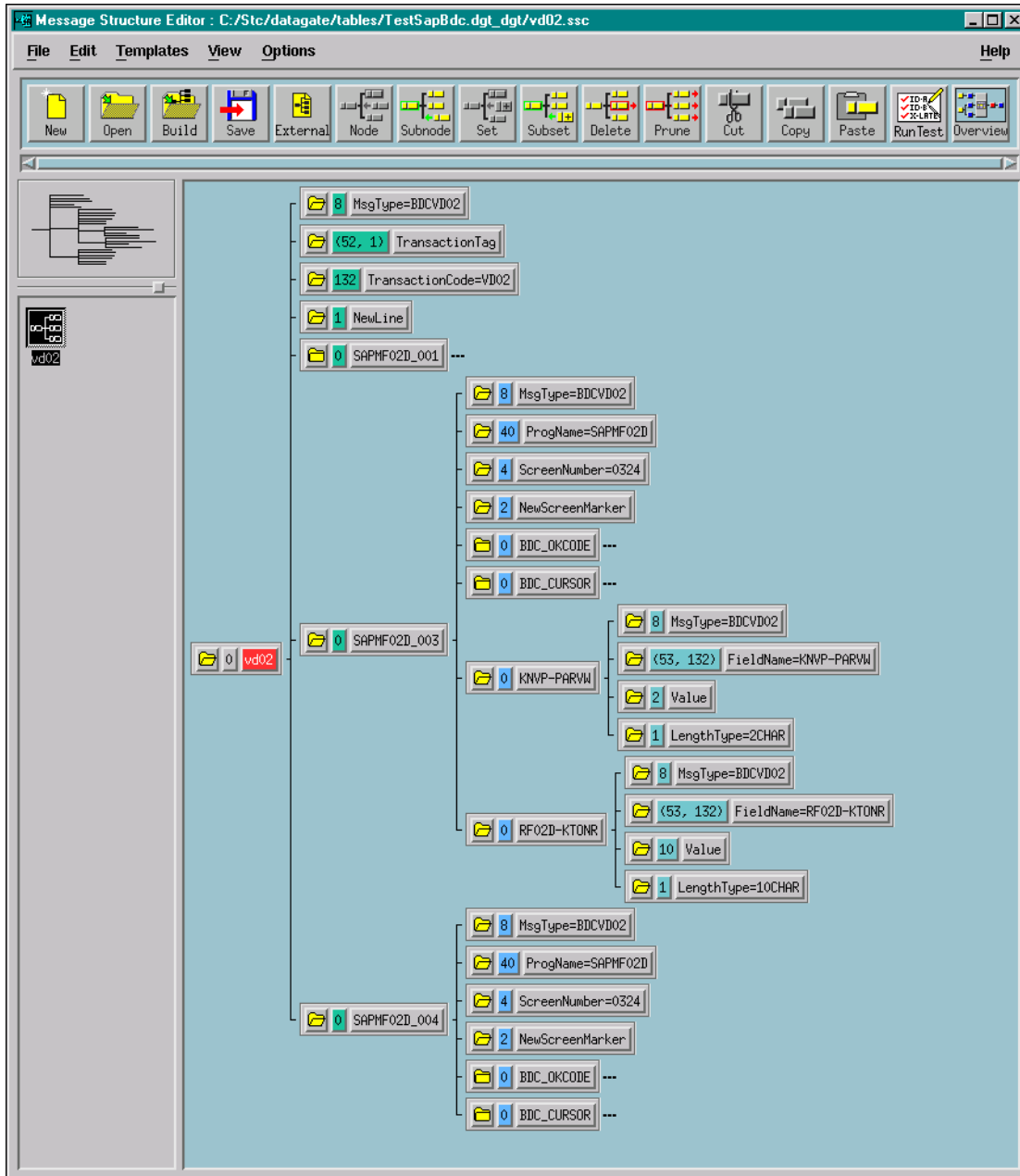
## 3.4 Creating Event Type Definitions

Before e\*Gate can process any data to or from an SAP system, you must create an Event Type Definition to package and route that data within the e\*Gate system. Once you have a file that contains a description of the SAP screen flow or the SAP custom table (see [SAP Structure Export Module](#) on page 79), you can use the ETD Editor's Build tool to create Event Type Definitions based upon the description file (see [Using the ETD Editor's Build Tool](#) on page 36). See the *e\*Gate Integrator User's Guide* for additional information about Event Type Definitions and the e\*Gate ETD Editor.

A typical BDC representation of an Event consists of multiple records, separated by "newline" characters. There are two types of SAP BDC Event Type Definitions: the Standard BDC structure for an SAP screen flow, and the Custom BDC structure for an SAP custom table. Both structures are Fixed-Length Event Type Definitions, with the first node in the structure header and the first node in the beginning of every section (record) identifying the message type. For a Standard BDC structure, the Transaction Code is also included in the structure header.

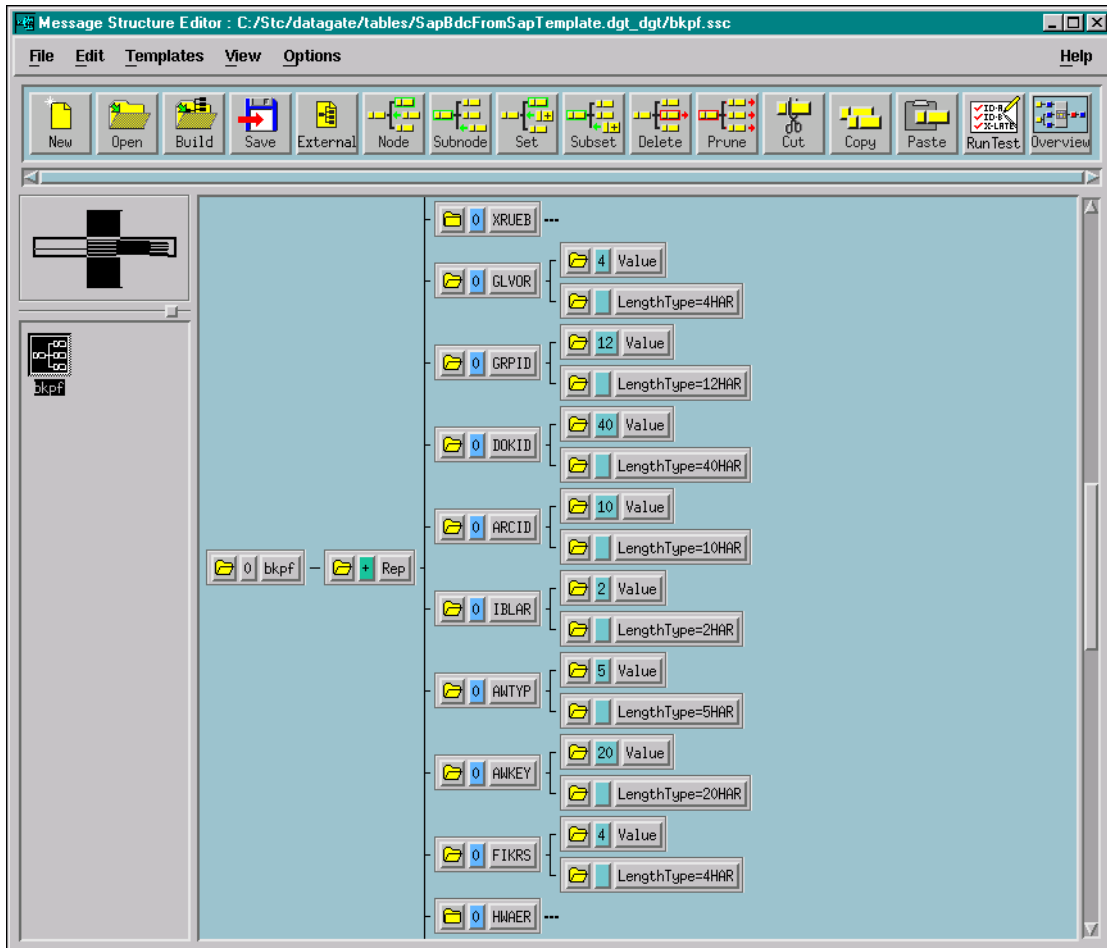
Besides the structure header, a Standard BDC structure consists of multiple sections, each corresponds to an SAP screen. Under each section are nodes that correspond to table fields referenced in the corresponding SAP screen. Each such node has a set of sub-nodes, of which only the node **Value**, which represents the value of the corresponding table field, is of concern to the e\*Way user. The following is an example Standard BDC Event Type Definition.

Figure 13 Sample Standard BDC Event Type Definition



A Custom BDC Event Type Definition simply mimics the table structure it represents. Each of the sections in such a structure represents a field in the corresponding table. Each field has a **Value** sub-node that represents the value of the table field, and an informational **TypeLength** sub-node. The entire structure is ended with a new-line character. The following is an example Custom BDC Event Type Definition.

Figure 14 Sample Custom BDC Event Type Definition



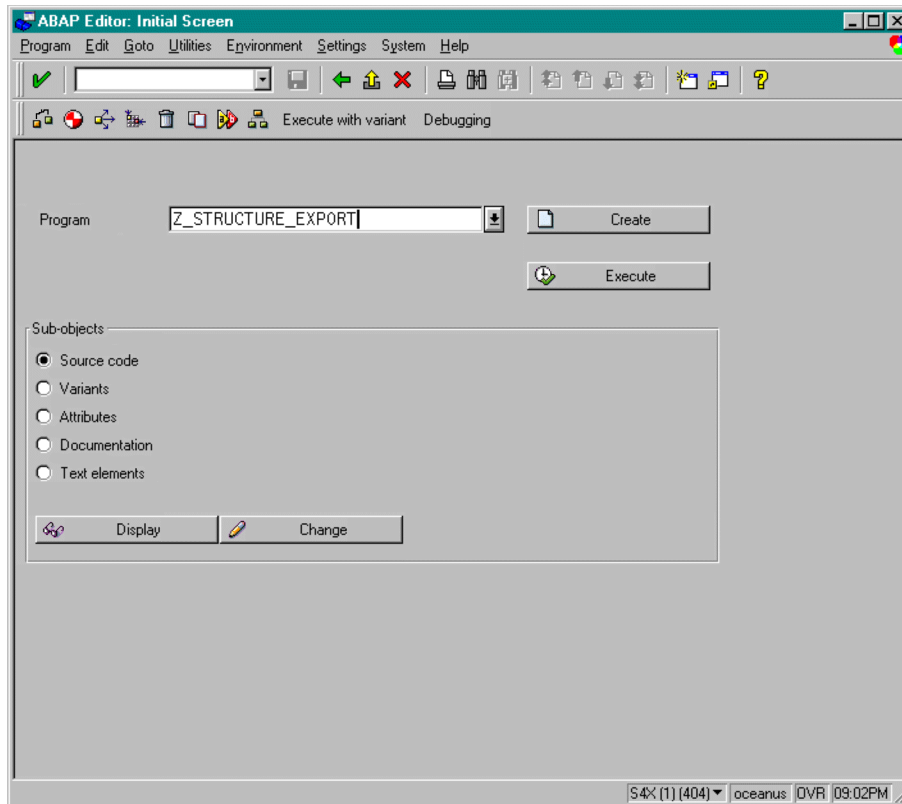


### 3.4.1 Using the SAP Screen Flow Recorder

**Note:** The SeeBeyond custom SAP ABAP components must be installed before you can use the screen recorder. See [Custom ABAP Components](#) on page 48.

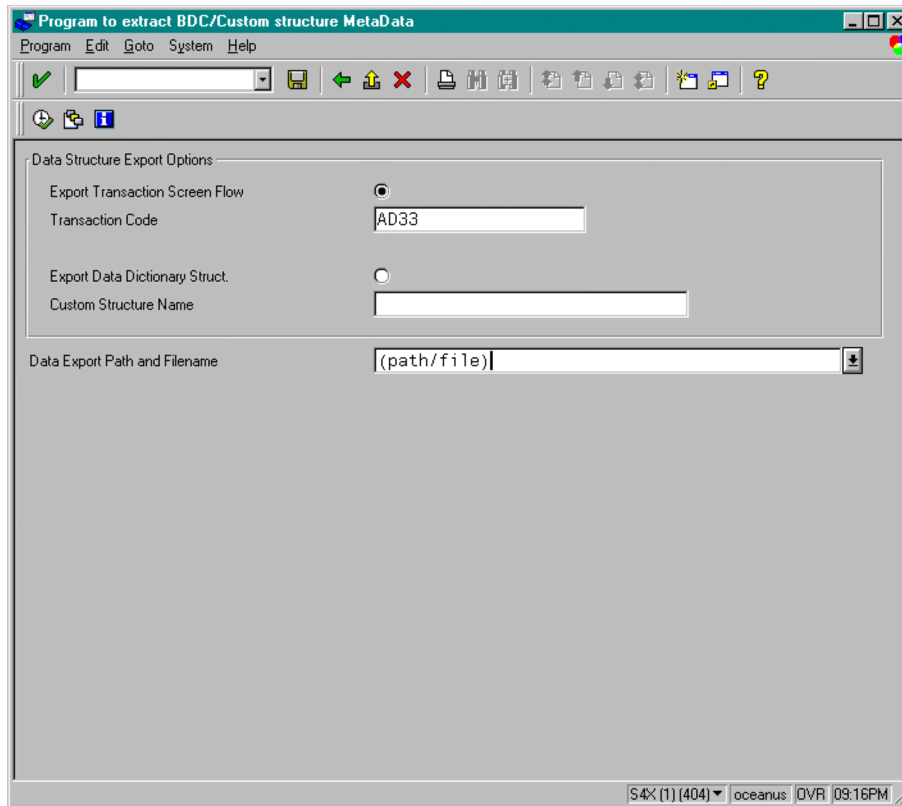
- 1 Go to transaction screen SE38.

**Figure 15** ABAP Editor Window



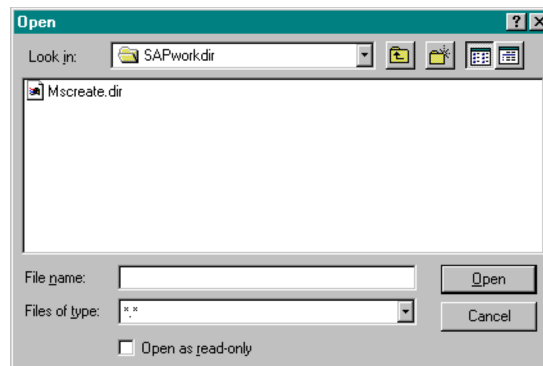
- 2 For the Program enter Z\_STRUCTURE\_EXPORT.
- 3 Click on Execute to invoke the Program to extract BDC/Custom structure MetaData window.

**Figure 16** Program to Extract ... Window





- 4 Under the **Data Structure Export Options** section, select the **Export Transaction Screen Flow** option button.
- 5 For the **Transaction Code** enter desired transaction screen to record.
- 6 Click inside the **Data Export Path and Filename** text box.
- 7 Click on the menu pull-down button to the right of the text box, which invokes a dialog box.

**Figure 17** File Selection Dialog Box



- 8 Select the desired file name and directory name for saving the metadata file.

- 9 Click on **Open**.
- 10 Click on , which invokes the starting screen for the transaction screen that you specified in step 5.
- 11 Populate all fields that are necessary for the transaction as you sequence through all transaction screens required to complete the transaction.
- 12 Once you are done, **Save** your work and **Exit** the transaction screen.
- 13 If necessary, click on  repeatedly to return to the *Program to extract BDC/Custom structure MetaData* window.
- 14 You should see a message stating that the data was successfully downloaded to the location and filename specified in step 5.


You can now use this generated metadata file as input to the SeeBeyond SAP BDC Converter to generate your ETD.

### 3.4.2 Using the ETD Editor's Build Tool

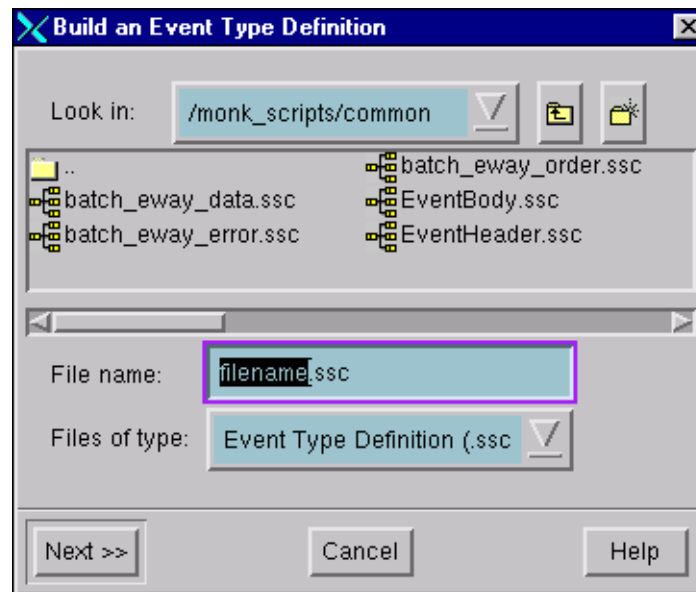
The Event Type Definition Editor's Build tool automatically creates an Event Type Definition file based upon a table structure. Use this procedure to create an Event Type Definition based upon the data your business process requires.

**Note:** Be sure to set the Default Editor to Monk, from the Options menu in the e\*Gate Schema Designer.

To create an Event Type Definition using the Build tool

- 1 Launch the ETD Editor by clicking  in the e\*Gate Schema Designer tool bar.
- 2 On the ETD Editor's tool bar, click **Build**, and the *Build an Event Type Definition* dialog box opens.

**Figure 18** Build Event Type Definition Dialog

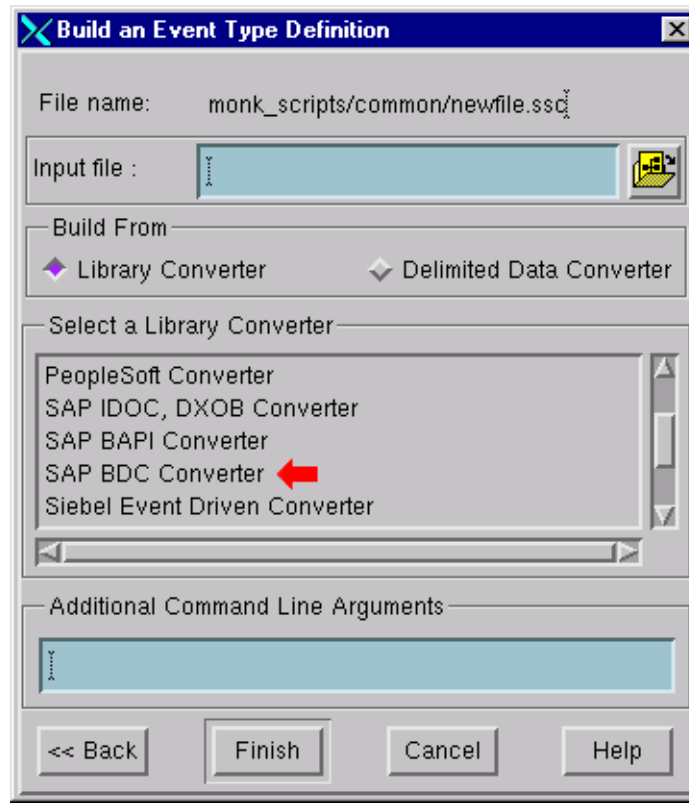


- 3 In the *File name* box, type the name of the ETD file you want to build.

**Note:** The Editor automatically supplies the .ssc extension.

- 4 Click **Next**. A new dialog box appears, as shown in Figure 19.

**Figure 19** Building the ETD




- 5 Under *Build From*, select **Library Converter**.
- 6 Under *Select a Library Converter*, select **SAP BDC Converter**.
- 7 In the *Additional Command Line Arguments* box, type any additional arguments, if desired.
- 8 Click **Finish**, and the SAP BDC Converter Wizard appears.
- 9 Follow the Wizard's instructions to finish building the ETD file.

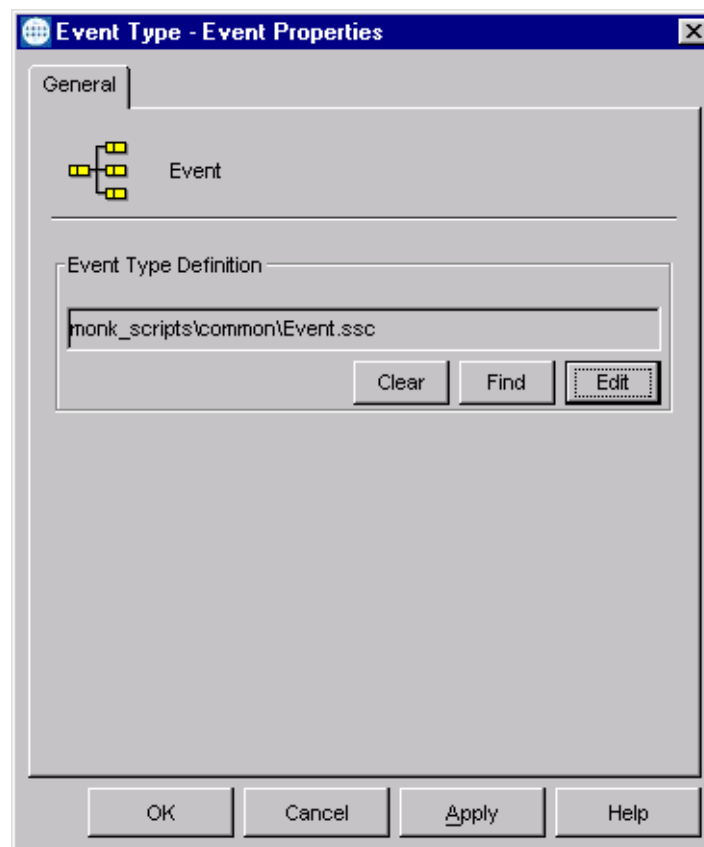
### 3.4.3 Assigning ETDs to Event Types

After you have created the e\*Gate system's ETD files, you can assign them to Event Types you have already created.

#### To assign ETDs to Event Types

- 1 In the Schema Designer window, select the **Event Types** folder in the Navigator/Components pane.
- 2 In the Editor pane, select one of the Event Types you created.
- 3 Right-click on the Event Type and select **Properties** (or click  in the toolbar). The Event Type Properties dialog box appears. See Figure 20.

**Figure 20** Event Type Properties Dialog Box



- 4 Under Event Type Definition, click **Find**, and the Event Type Definition Selection dialog box appears (it is similar to the Windows Open dialog box).
- 5 Open the `monk_scripts\common` folder, then select the desired file name (\*.ssc).
- 6 Click **Select**. The file populates the Event Type Definition field.

- 7 To save any work in the properties dialog box, click **Apply** to enter it into the system.
- 8 When finished assigning ETDs to Event Types, click **OK** to close the properties dialog box and apply all the properties.

Each Event Type is now associated with the specified Event Type Definition.

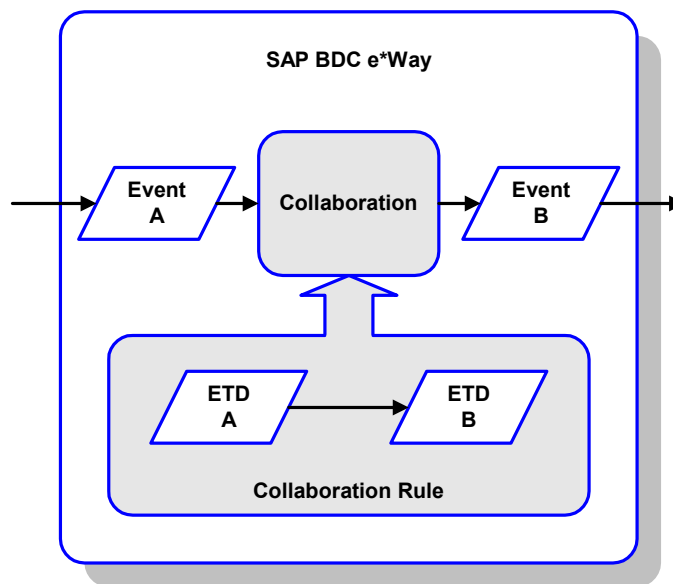
---

## 3.5 Defining Collaborations

After you have created the required Event Type Definitions, you must define a Collaboration to transform the incoming Event into the desired outgoing Event.

Collaborations are e\*Way components that receive and process Event Types, then forward the output to other e\*Gate components. Collaborations consist of the Subscriber, which “listens” for Events of a known type or from a given source, and the Publisher, which distributes the transformed Event to a specified recipient. The same Collaboration cannot be assigned to more than one e\*Gate component.

**Figure 21** Collaborations



The Collaboration is driven by a Collaboration Rule, which defines the relationship between the incoming and outgoing ETDs. You can use an existing Collaboration Rule, or use the Monk programming language to write a new Collaboration Rule script. Once you have written and successfully tested a script, you can then add it to the system’s run-time operation.

Collaborations are defined using the e\*Gate Monk Collaboration Rules Editor. See the *e\*Gate Integrator User’s Guide* for instructions on using this Editor. The file extension for Monk Collaboration Rules is **.tsc**.

---

## 3.6 Creating Intelligent Queues

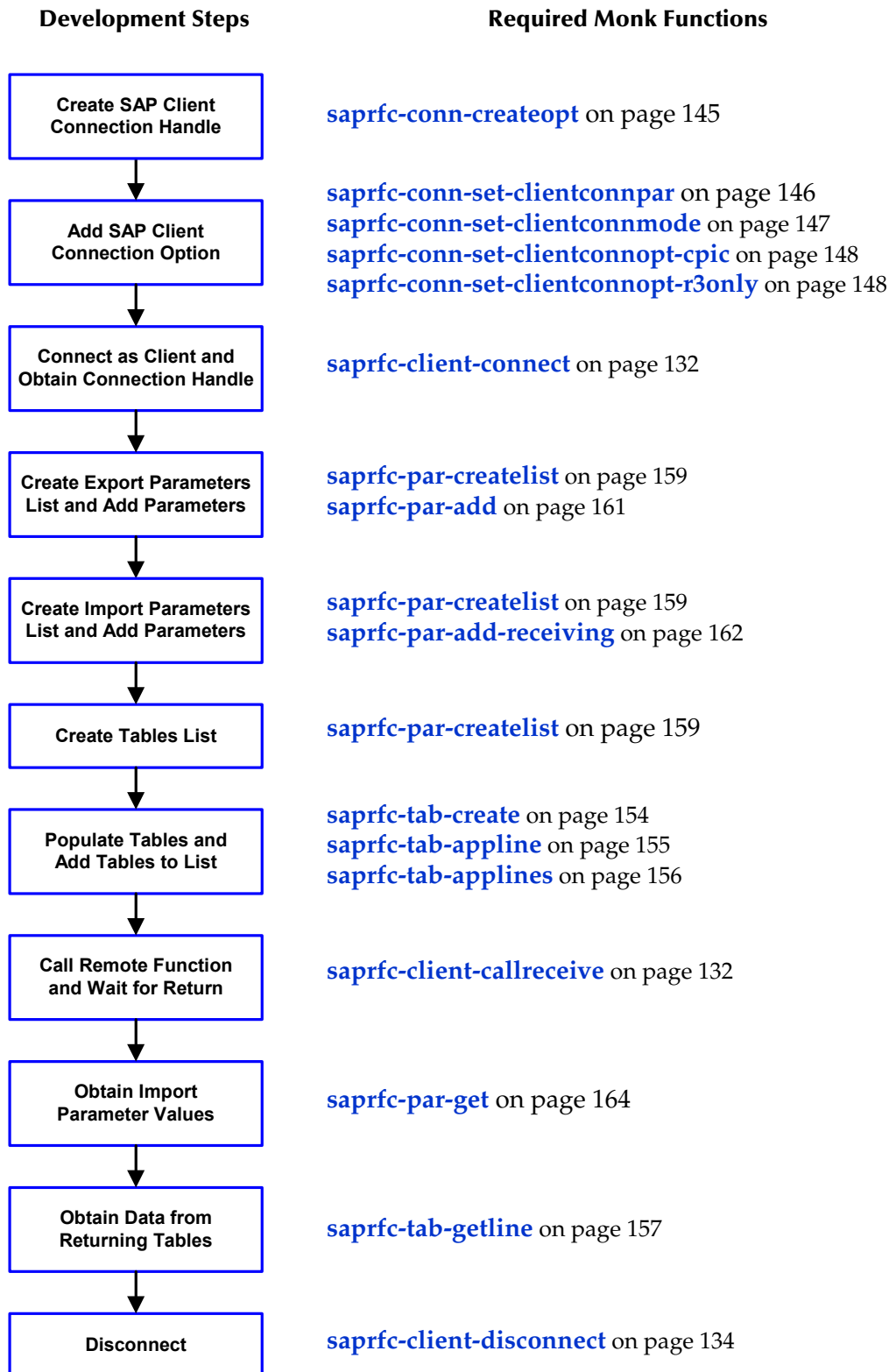
IQs are components that provide nonvolatile storage for Events within the e\*Gate system as they pass from one component to another. IQs are *intelligent* in that they are more than just a “holding tank” for Events. They actively record information about the current state of Events.

Each schema must have an IQ Manager before you can add any IQs to it. You must create at least one IQ per schema for published Events within the e\*Gate system. Note that e\*Ways that publish Events externally do not need IQs.

For more information on how to add and configure IQs and IQ Managers, see the *e\*Gate Integrator System Administration and Operations Guide*. See the *e\*Gate Integrator Intelligent Queue Services Reference Guide* and the *SeeBeyond JMS Intelligent Queue User’s Guide* for complete information on working with IQs.



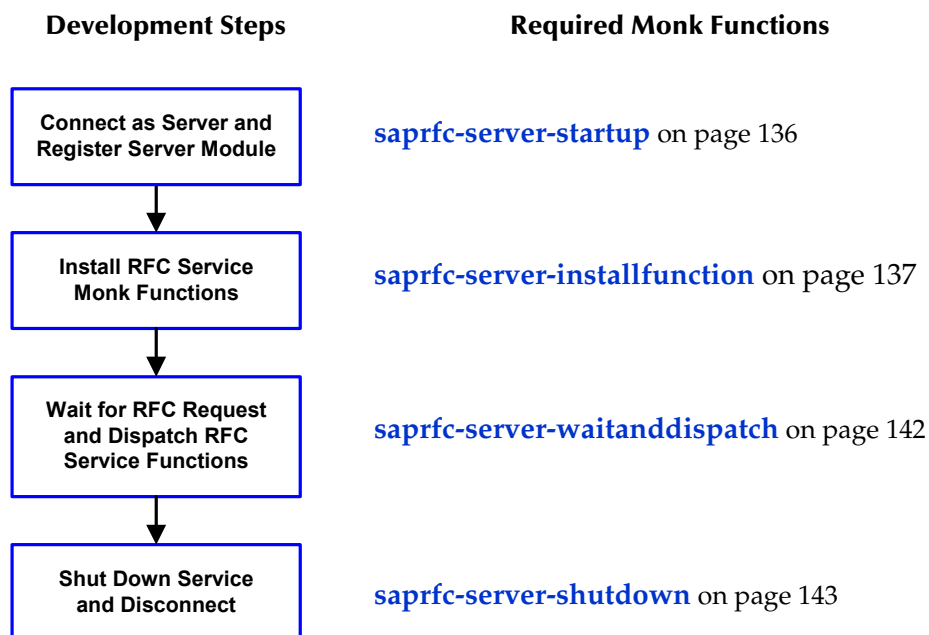
## 3.7 Building an SAP RFC Client



Building an SAP RFC Client using the SAP RFC Monk Library typically includes the preceding steps. Note that even if the remote function to be called does not require any parameters, the (empty) parameter lists are still required for the [saprfc-client-callreceive](#) call. The steps between connection and disconnection can be repeated as a group.

## 3.8 Building an SAP RFC Server

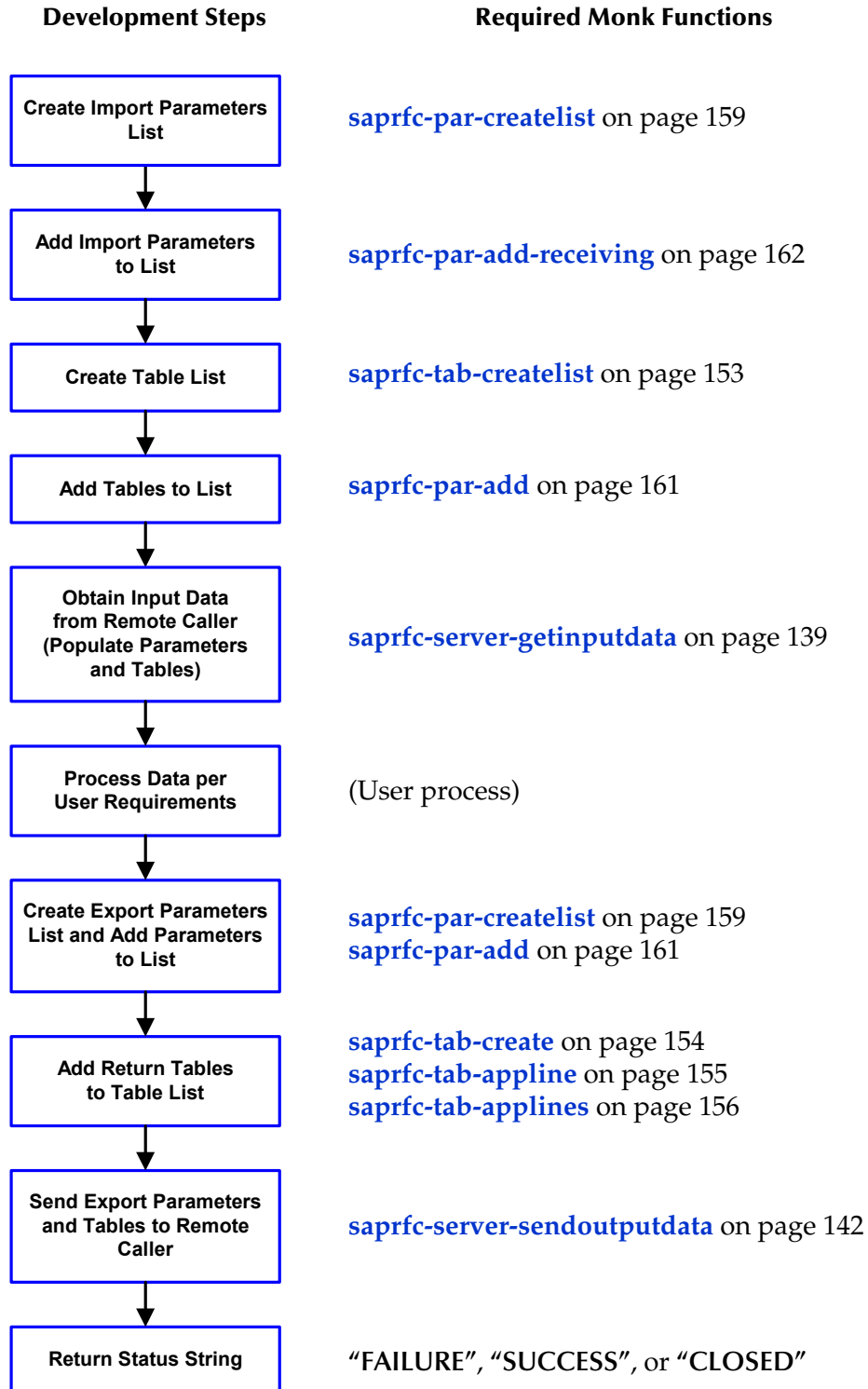
Building an SAP RFC server using the SAP RFC Monk Library typically includes the following steps.



The installed RFC service (Monk) functions are invoked to process any calls from a remote client. Such a Monk functions must be a Database (DART) Poll function (.dsc). The string returned from this function indicates the result of the service process:

- “FAILURE” - The service failed.
- “SUCCESS” - The service succeeded.
- “CLOSED” - The connection was closed by the other end.

The logic of a typical RFC service Monk function is shown in the following diagram.



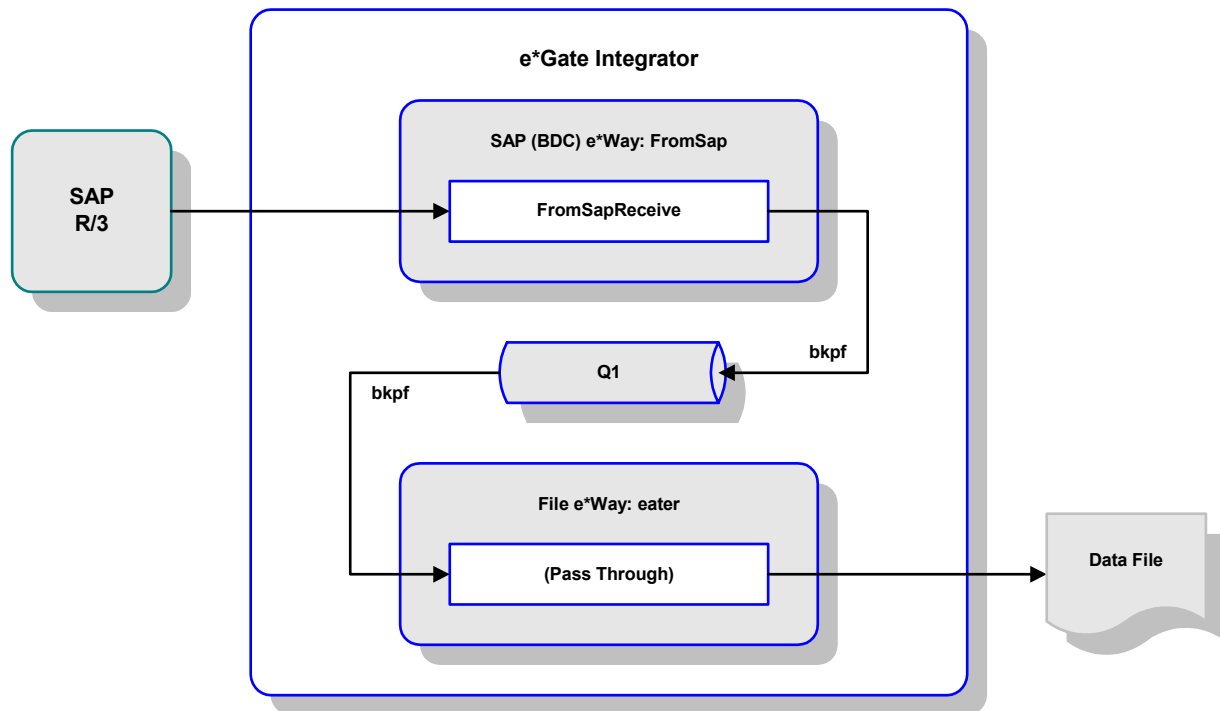
## 3.9 Sample Schemas

The installation CD-ROM contains two sample schemas, **SapBdcFromSap** and **SapBdcToSap**, in the **samples\ewsapbdc** directory. To use a schema, you must load it onto your system using the procedure given in **Optional Example Files** on page 25.

### 3.9.1 SAP RFC Client

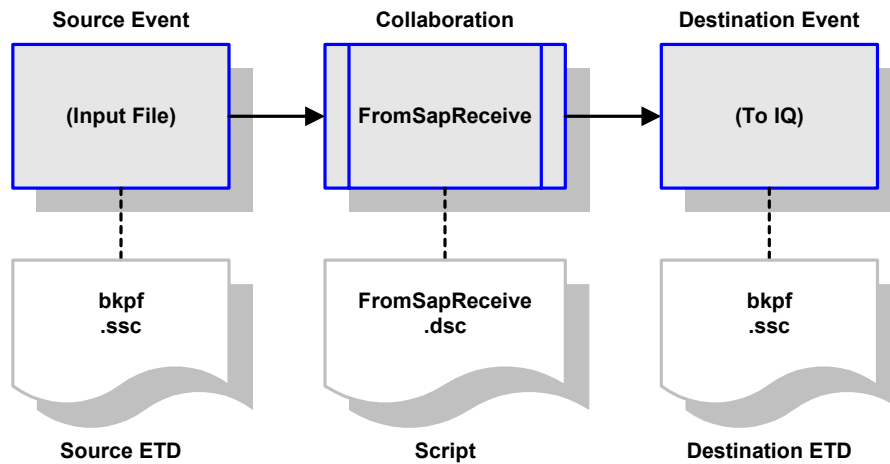
The schema **SapBdcFromSap** (see Figure 22) provides a simple example of using the e\*Way as an SAP RFC Client. A single SAP BDC e\*Way is instantiated with the logical name **FromSap**. It subscribes to data from the SAP R/3 application, runs the Collaboration **FromSapReceive**, and publishes the result to an IQ. A File e\*Way, having the logical name **eater**, subscribes to the contents of the IQ and publishes the result as a data file.

**Figure 22** SAP RFC Client Example



The Collaboration **FromSapReceive** (see Figure 23) transforms data from SAP, described by the ETD **bkpf.ssc** into data for storage in the IQ. In this example, the output data is described by the same ETD. The Collaboration is defined by the Monk script **FromSapReceive.dsc**, which provides direct database access.

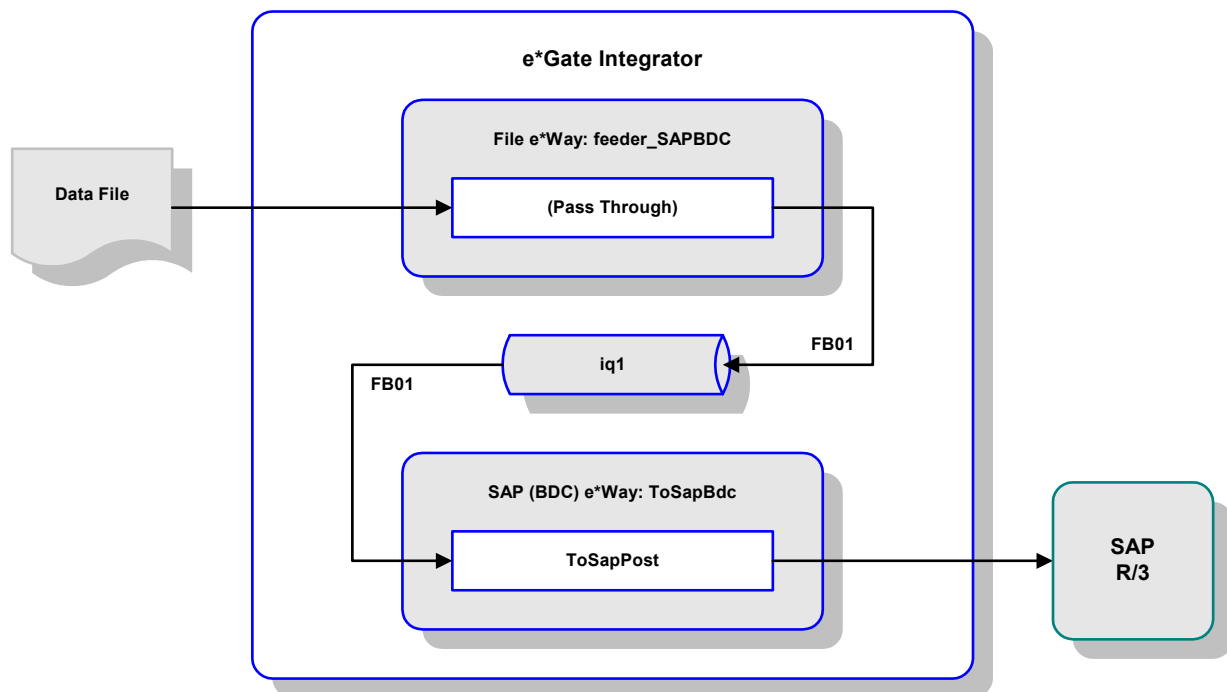
**Figure 23** FromSapReceive Collaboration



### 3.9.2 SAP RFC Server

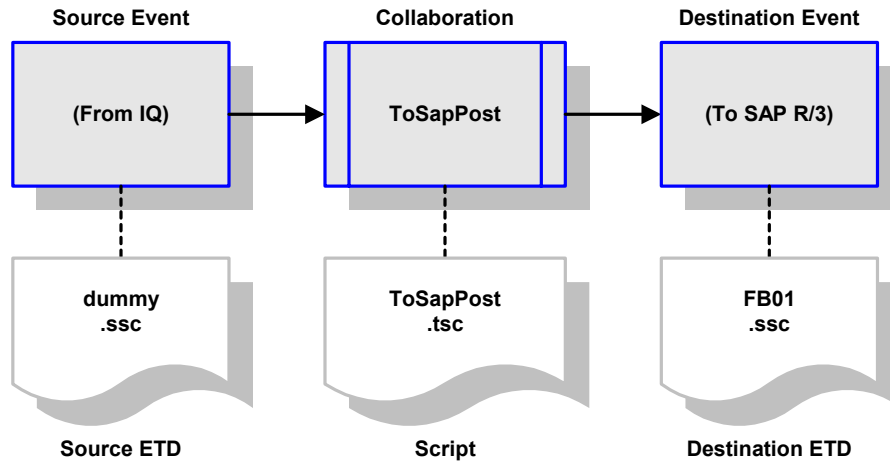
The schema `SapBdcToSap` (see Figure 24) provides a simple example of using the e\*Way as an SAP RFC Server. A File e\*Way, having the logical name `feeder_SAPBDC`, receives a data file, performs a pass-through Collaboration service, and publishes the contents of the file to the IQ `iq_1`. A single SAP BDC e\*Way is instantiated with the logical name `ToSapBdc`. It subscribes to data from the IQ, runs the Collaboration `ToSapPost` and publishes the result to the SAP R/3 application.

**Figure 24** SAP RFC Server Example



The Collaboration **ToSapPost** (see Figure 23) transforms the data from the IQ, described by the ETD **dummy.ssc**, into data for SAP R/3, described by the ETD **FB01**.

**Figure 25** ToSapPost Collaboration



# e\*Way Extensions

This chapter describes procedures for using the optional extensions provided with the e\*Way to assist in development and system integration tasks.

---

## 4.1 Overview

To use the provided extensions, custom ABAP components must be imported to your SAP application, and several SAP R/3 objects must then be updated to recognize these components. These optional extensions do not interfere with other SAP R/3 operations.

***Note:** These extensions are **not** required for e\*Way operation.*

The topics described in this chapter include the following:

[Custom ABAP Components](#) on page 48

[Importing the Custom Components](#) on page 50

[Updating SAP R/3 Objects](#) on page 53

## 4.2 Custom ABAP Components

Custom ABAP components are provided with the e\*Way to assist in development and system integration tasks. These are delivered as a set of transport files, which are located on the e\*Gate installation CD-ROM as shown in Table 10. The transport numbers contained in the file names change, so they are shown as xxxxxx. The file **BDCeWay.doc** (written in Japanese) describes the procedure for importing the files into the Japanese version of SAP R/3.

**Table 10** Transport Files

| Subdirectories                         | Files  |
|--|--|
| \utils\sapr3staging\ewsapbdc\          | Dxxxxxx.S4X<br>Kxxxxxx.S4X<br>Rxxxxxx.S4X                |
| \utils\sapr3staging\ewsapbdc\Japanese\ | Dxxxxxx.N46<br>Kxxxxxx.N46<br>Rxxxxxx.N46<br>BDCeWay.doc |

These components do not replace or alter any existing functions or files. The transport files add the following custom ABAP functions and modules, along with related tables and data, to the SAP R/3 system:

- **Z\_CUSTOM\_MAPPING**  
An ABAP module template for creating custom mapping logic (see [Data Mapping/Collaboration](#) on page 83).
- **Z\_OUTBOUND\_DGW\_INITIATE**  
A test module used to send data to the BDC e\*Way (see [SAP to e\\*Gate](#) on page 59 and [SAP-to-e\\*Gate Data Exchange](#) on page 91). The destination is determined from the table ZDGBDC.
- **Z\_R3\_BDC\_DATA\_IMPORT**  
An ABAP module template, which receives data into the e\*Way to be processed and forwarded to SAP (see [e\\*Gate to SAP](#) on page 53 and [e\\*Gate-to-SAP Data Exchange](#) on page 90).
- **Z\_STC\_DGW\_SAPBDC\_SERV**  
A pre-defined RFC server module (see [SAP-to-e\\*Gate Data Exchange](#) on page 91).
- **Z\_STRUCTURE\_EXPORT**  
An ABAP function template for standard SAP screen mapping (see [Data Mapping/Collaboration](#) on page 83).
- **ZDGBDC**  
A custom table template (see [e\\*Gate to SAP](#) on page 53).



- **ZDGLOG**  
A template for a custom table to collect both information and error messages from Inbound processing when the **LOGMSG** flag in the custom table **ZDGBDC** is activated.
- **ZDGOUT**  
A custom table template (see [SAP to e\\*Gate](#) on page 59).
- **ZOTBNDTEST2**  
Example custom ABAP module (see [SAP to e\\*Gate](#) on page 59).

## 4.3 Importing the Custom Components

The transport number (represented herein by xxxxxx) is used as input to the SAP Application Server and is also embedded in the filename of the co-files and data files.

*Note:* For the Japanese version, please read the .doc file that is located with the transport files on the installation CD-ROM.

To import the SAP ABAP components:

- 1 Copy the cofile file Kxxxxxx.S4X to the /trans/cofiles directory on the SAP Application Server.
- 2 Copy the data files Dxxxxxx.S4X and Rxxxxxx.S4X to the /trans/data directory on the SAP Application Server.
- 3 Login to the SAP Application Server and change to the /trans/bin directory.

- 4 Issue the command

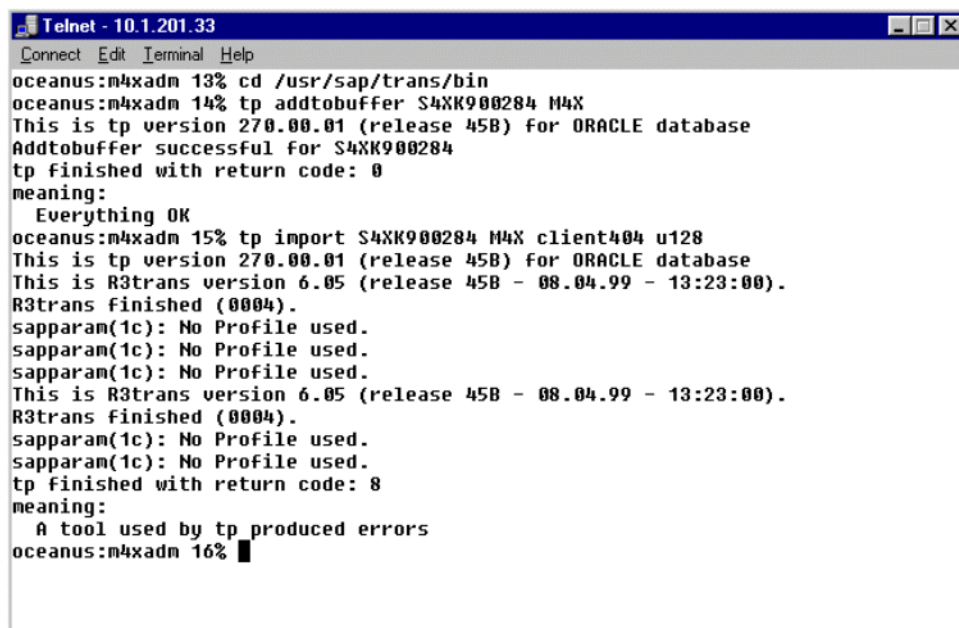
```
tp addtobuffer S4Xxxxxxxx <system>  
where <system> is the name of the target SAP system.
```

- 5 Issue the command

```
tp import S4XKxxxxxxx <system> client <client> u128  
where <client> is the client name of the target SAP system.
```

This process takes a few minutes. Ignore any **no profile used** messages that you may receive.

**Figure 26** Import Procedure Display



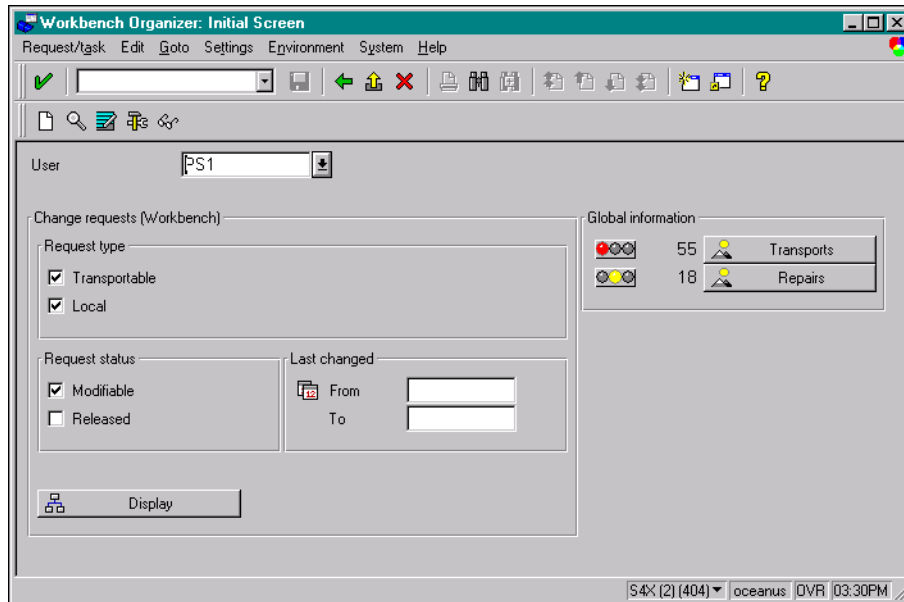
```
Telnet - 10.1.201.33  
Connect Edit Terminal Help  
oceanus:m4xadm 13% cd /usr/sap/trans/bin  
oceanus:m4xadm 14% tp addtobuffer S4XK900284 M4X  
This is tp version 270.00.01 (release 45B) for ORACLE database  
Addtobuffer successful for S4XK900284  
tp finished with return code: 0  
meaning:  
Everything OK  
oceanus:m4xadm 15% tp import S4XK900284 M4X client404 u128  
This is tp version 270.00.01 (release 45B) for ORACLE database  
This is R3trans version 6.05 (release 45B - 08.04.99 - 13:23:00).  
R3trans finished (0004).  
sapparam(1c): No Profile used.  
sapparam(1c): No Profile used.  
sapparam(1c): No Profile used.  
This is R3trans version 6.05 (release 45B - 08.04.99 - 13:23:00).  
R3trans finished (0004).  
sapparam(1c): No Profile used.  
sapparam(1c): No Profile used.  
tp finished with return code: 8  
meaning:  
A tool used by tp produced errors  
oceanus:m4xadm 16% █
```


If you encounter errors during the import process (as shown in *Figure 26*), view the error log by means of the following procedure.

**To view the error log**

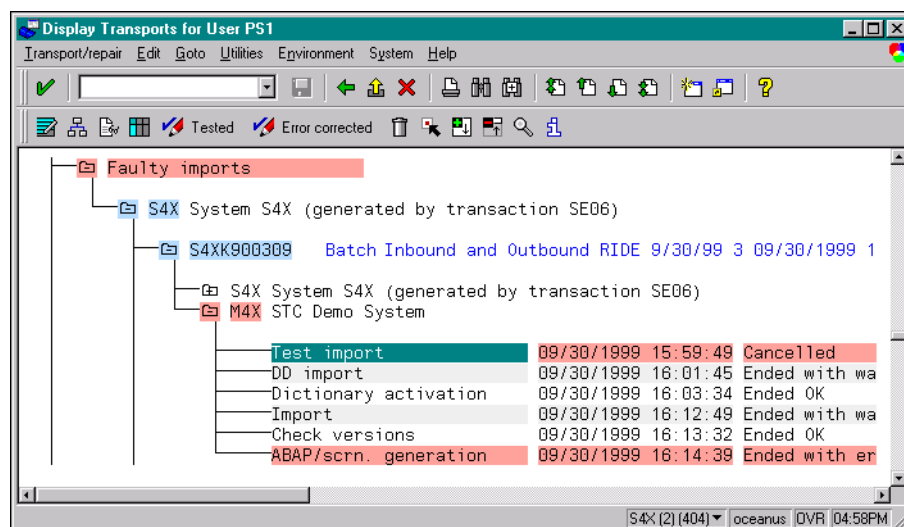
- 1 Go to transaction SE09.

**Figure 27** Workbench Organizer Window



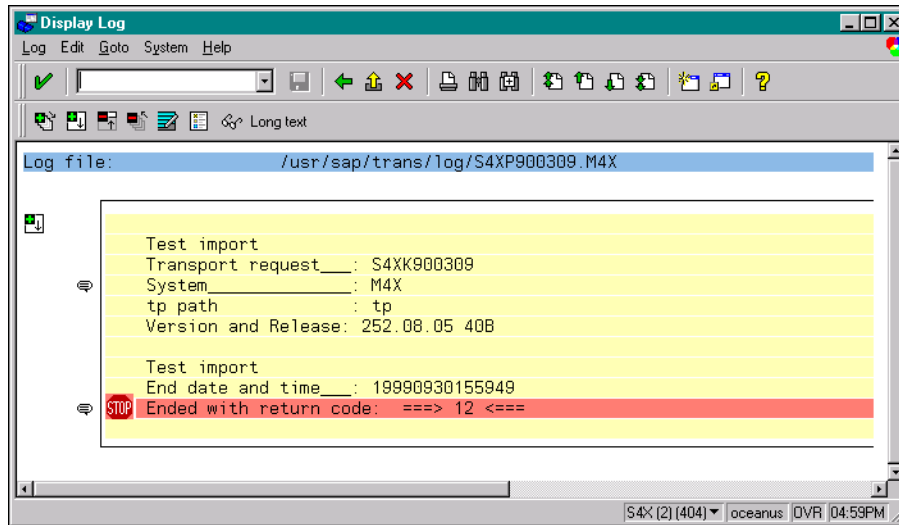
- 2 Select  Transports.
- 3 Enter the transport number, and then follow the menu path **Goto > All logs** in the *Display Transports* window.

**Figure 28** Display Transports Window



- 4 Double-click on an action field to display the individual log.

Figure 29 Display Log Window



**Note:** The figures included in this section showing the SAP GUI represent a standard SAP 4.0 installation. Your screen may appear different if you are using a different version of SAP or have modified standard version. See your SAP administrator for more information.

## 4.4 Updating SAP R/3 Objects

*Note:* All custom functions in SAP **must** begin with a “Z” prefix.

### e\*Gate to SAP

The ABAP module `Z_R3_BDC_DATA_IMPORT` is responsible for distributing data into the SAP system. This module is called via RFC by the e\*Way when the latter needs to send data to SAP. For each message it receives from the e\*Way, `Z_R3_BDC_DATA_IMPORT` does a lookup in the SAP table `ZDGBDC` to find out what to do with the message.

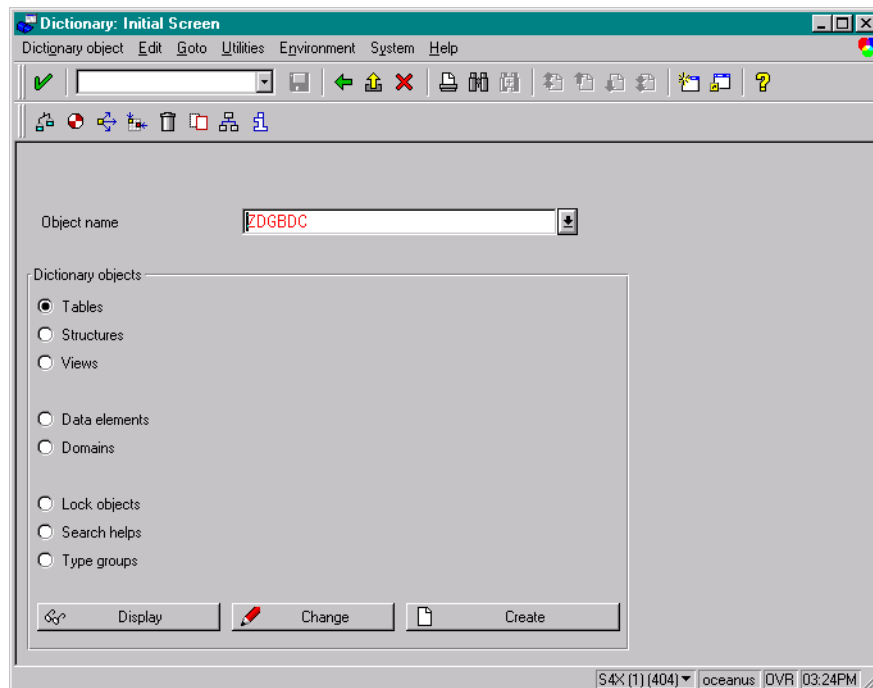
Transaction: `SE11`

After the Transport files have been imported into your SAP system, you need to perform the following two procedures:

- 1 **Insert Data into the ZDGBDC Table** on page 53
- 2 **Set up Message Counter Number Range** on page 57

### Insert Data into the ZDGBDC Table

**Figure 30** Dictionary - Initial Screen




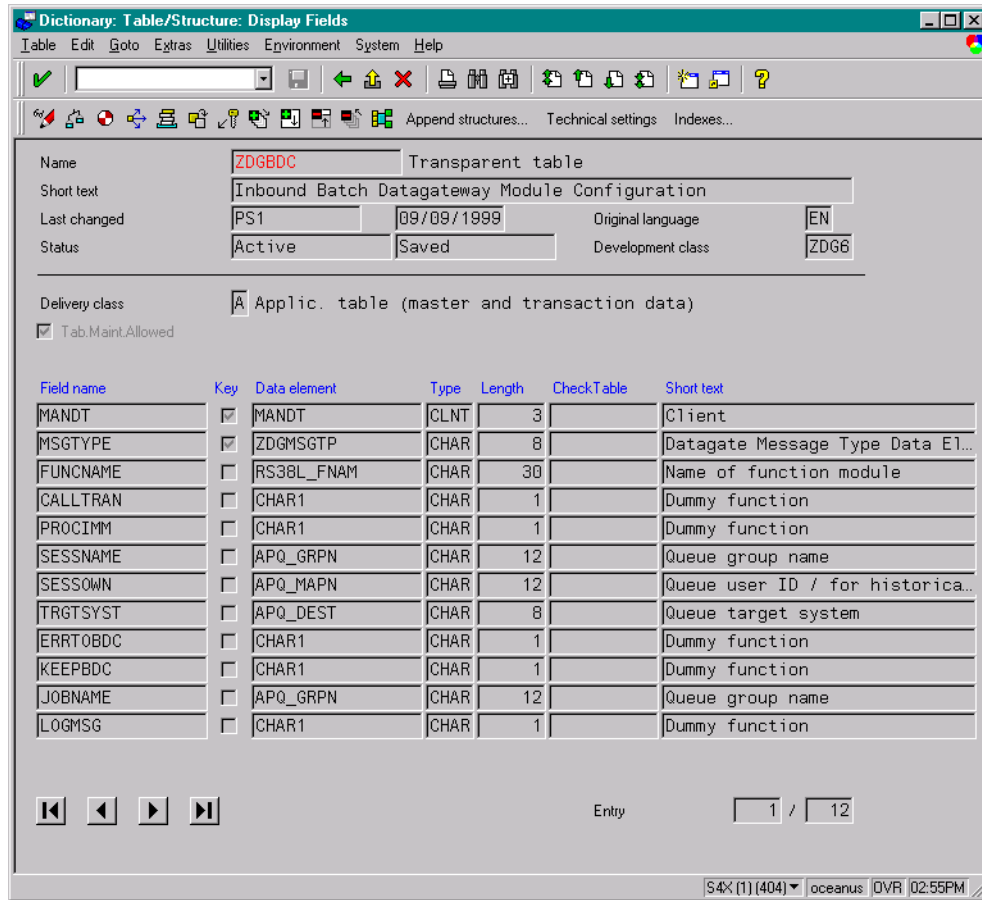
- 1 Enter `ZDGBDC` as the Object name.
- 2 Select the **Tables** option.
- 3 Select  **Display**.

Figure 31 Display Fields Window




- The *Display Fields* window shows the definition for the selected table. Follow the menu path **Utilities > Table Contents** and select  to display records from the table.

Figure 32 ZDGBDC Table

| MANDT | MSGTYPE  | FUNCNAME      | CALLTRAN | PROCIMM | SESSNAME |
|-------|----------|---------------|----------|---------|----------|
| 404   | BDCIMM   | Z_2_TRXN_TEST |          | X       | BDCIMM   |
| 404   | BDCMAN   | Z_2_TRXN_TEST |          |         | BDCMAN   |
| 404   | BDCF001  | UPDATE        |          | X       | DGBDCTST |
| 404   | BDCF002  | UPDATE        |          | X       | DGBDCTST |
| 404   | BDCF003  | UPDATE        |          | X       | DGBDCTST |
| 404   | BDCF004  | UPDATE        |          | X       | DGBDCTST |
| 404   | BDCIMM   | UPDATE        |          | X       | BDCIMM   |
| 404   | BDCMAN   | UPDATE        |          |         | BDCMAN   |
| 404   | BDCV002  | UPDATE        |          | X       | DGBDCTST |
| 404   | CTRCERR  | Z_2_TRXN_TEST | X        |         | CTRCERR  |
| 404   | CTRCNERR | Z_2_TRXN_TEST | X        |         | CTRCNERR |
| 404   | CTRERR   | UPDATE        | X        |         | CTRERR   |
| 404   | CTRNERR  | UPDATE        | X        |         | CTRNERR  |
| 404   | CUSTINST | Z_2_TRXN_TEST |          | X       | INBNDTST |
| 404   | CUSTTST  | Z_2_TRXN_TEST | X        |         | INBNDTST |
| 404   | ZFB01TST | Z_2_TRXN_TEST | X        |         | ZFB01TST |

For each message type, an entry needs to be added to the ZDGBDC table, in order for this type of message to be recognized by the module Z\_R3\_BDC\_DATA\_IMPORT. For each MSGTYPE, the FUNCNAME field tells Z\_R3\_BDC\_DATA\_IMPORT which custom function to call to process this type of message. The FUNCNAME value UPDATE represents a standard call transaction.

- 5 Entries to the ZDGBDC table are made by selecting the menu path Utilities > Create entries, from the Display Fields window, which invokes the Table Insert window.

Figure 33 Table ZDGBDC Insert Window

Table ZDGBDC Insert

MSGTYPE: BDCF001

FUNCNAME: UPDATE

CALLTRAN:

PROCIMM:

SESSNAME:

SESSOWN:

TRGTSYST:

ERRTOBDC:

KEEPBDC:

JOBNAME:

LOGMSG:

Please choose a valid function

The entries in the ZDGBDC table are as follows:


**Table 11** ZDGBDC Table Entries

| Field    | Status   | Explanation  |
|----------|----------|--|
| MSGTYPE  | Required | This is the message type which is used to identify the incoming data message. A look up during processing associates the <b>MSGTYPE</b> with any run-time configuration.   |
| FUNCNAME | Optional | This is the function module name where the screen formatting occurs.<br><ol style="list-style-type: none"> <li>1 If a <b>standard</b> SAP screen flow is used then the function name <b>UPDATE</b> must be placed in this field.</li> <li>2 If a <b>non-standard</b> screen flow is used then the function name where the custom mapping is located must be used here. Whenever, a non-standard screen flow is used, the custom module <b>Z_CUSTOM_MAPPING</b> is used as a template for creating custom mapping logic.</li> </ol>   |
| CALLTRAN | Required | This flag determines whether a Call Transaction or a BDC Session is used as the update method for loading data into SAP.<br><ol style="list-style-type: none"> <li>1 If the flag is checked, then a Call Transaction is used as the update method. During a Call Transaction, which is a synchronous process, a return code is received from SAP along with system messages to notify if the process was successful or not.</li> <li>2 If the flag is <b>not</b> checked, then a BDC Session is created as the updated method. During a BDC Session, which is an asynchronous process, a return code is not received from SAP</li> </ol> |
| PROCIMM  | Optional | This flag determines whether or not a created BDC Session is processed immediately.<br><ol style="list-style-type: none"> <li>1 If the flag is checked, then a created BDC Session is processed immediately.</li> <li>2 If the flag is <b>not</b> checked, then a created BDC Session is held in a queue for manual processing. This manual processing can be accomplished through the transaction <b>SM35</b> in SAP.</li> </ol>  |
| SESSNAME | Required | Whenever a BDC Session is created, the user must specify the session name. This unique name appears in the transaction <b>SM35</b> . If this field is not populated, then an error occurs.   |
| SESSOWN  | Required | To process a BDC Session a valid session owner (i.e. user name) must be specified. If this field is not populated, then an error occurs.   |
| TRGTSYST | Optional | This field denotes the application server where the BDC session is processed. If this field is not populated, then processing defaults to the current server the application is running on.  |
| ERRTOBDC | Optional | Since SAP does not provide error handling functionality for Call Transactions, the application is left with the responsibility for any error handling. Therefore, this option allows the user to create a BDC Session and hold it in queue for any unsuccessful Call Transactions. The user must go to transaction <b>SM35</b> where the errors can be viewed and manually corrected and reprocessed.  |



**Table 11** ZDGBDC Table Entries

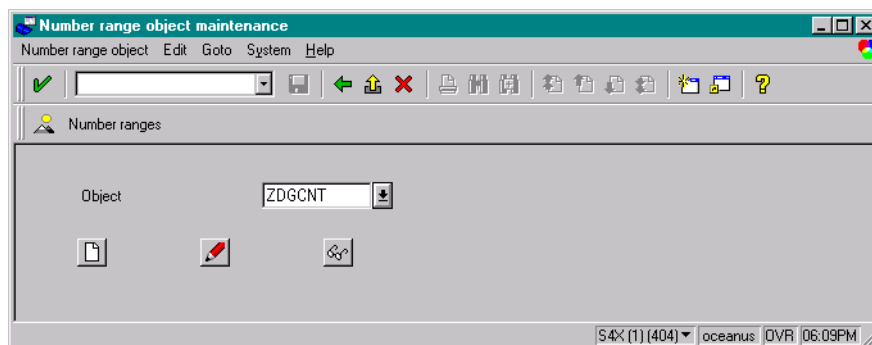
| Field   | Status   | Explanation   |
|---------|----------|---|
| KEEPBDC | Optional | This flag either keeps or discards a BDC Session after it has been successfully processed.  |
| JOBNAME | Required | This field is required by SAP whenever a BDC Session is created. If a BDC Session is chosen or a Call Transaction is used with errors to BDC, then this field must be populated |
| LOGMSG  | Optional | When activated, this flag writes all messages from Inbound processing to the custom table ZDGL0G. This includes both information and error messages.                            |

6 After typing in the desired values, select .

### Set up Message Counter Number Range

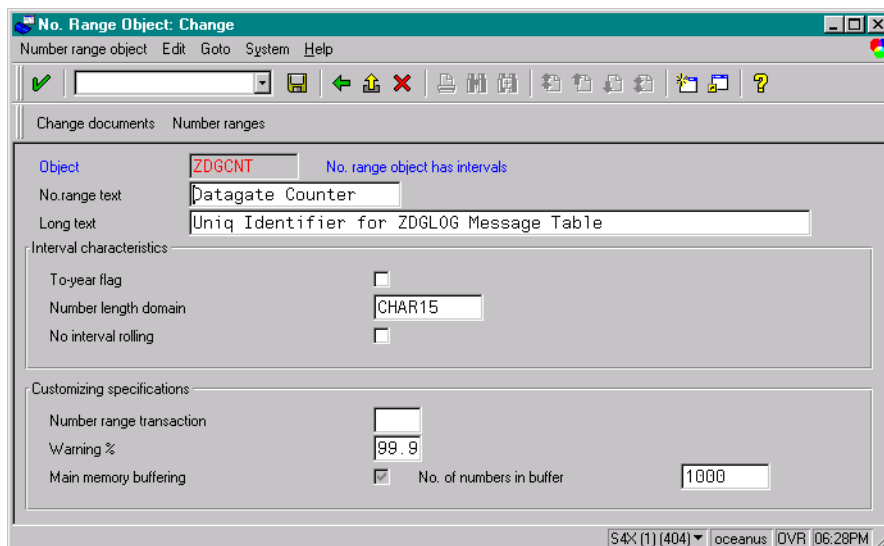
1 In SAP, go to transaction SNRO to create the number range object ZDGCNT.

**Figure 34** Create Number Range Object ZDGCNT



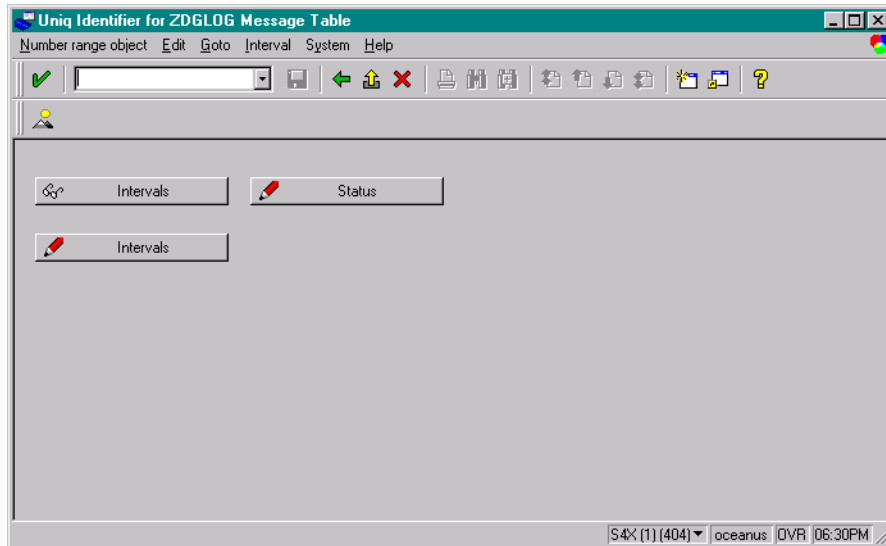
2 Press the Create push button.

**Figure 35** Enter Number Range



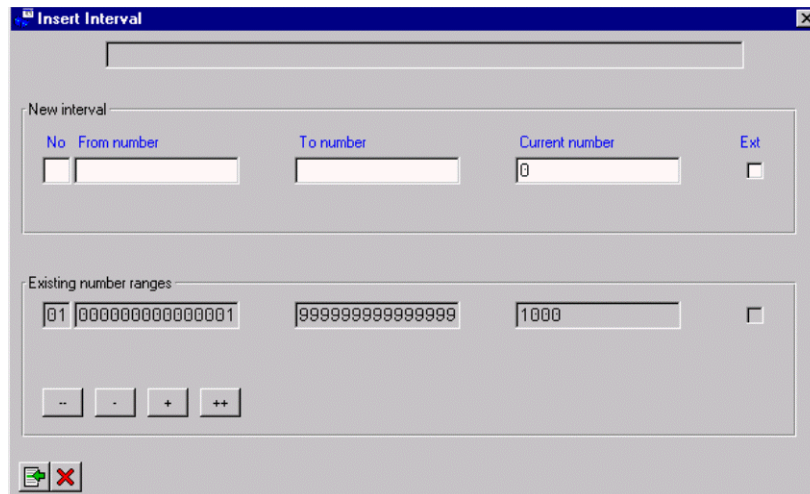
- 3 Enter the number range text and long text. The following parameters must be populated.
  - ◆ Number length domain: **CHAR15**
  - ◆ Warning %: **99.9**
  - ◆ No. of number in buffer: **1000**
- 4 Press the **Number Ranges** push button.

**Figure 36** Select Intervals Option



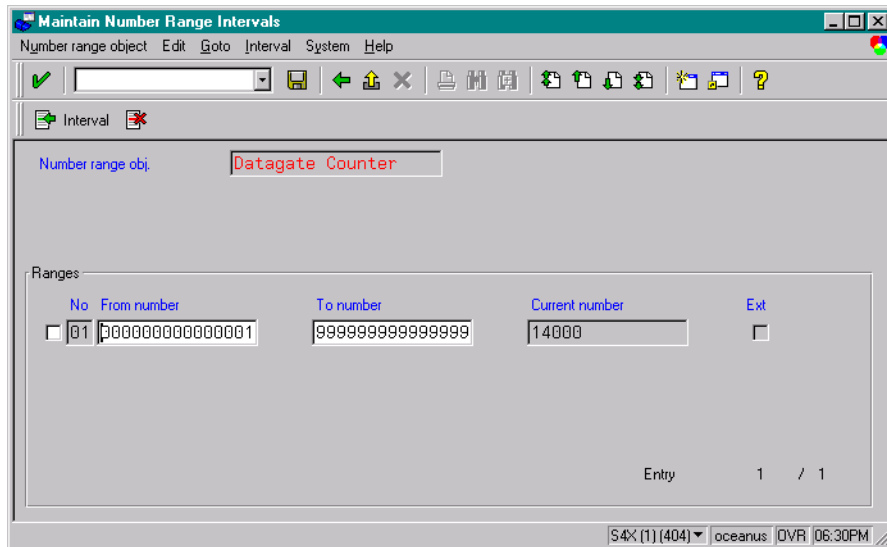
- 5 Press the **Change Intervals** push button.

**Figure 37** Enter New Interval



- 6 Enter the new interval as illustrated.

Figure 38 Save Interval

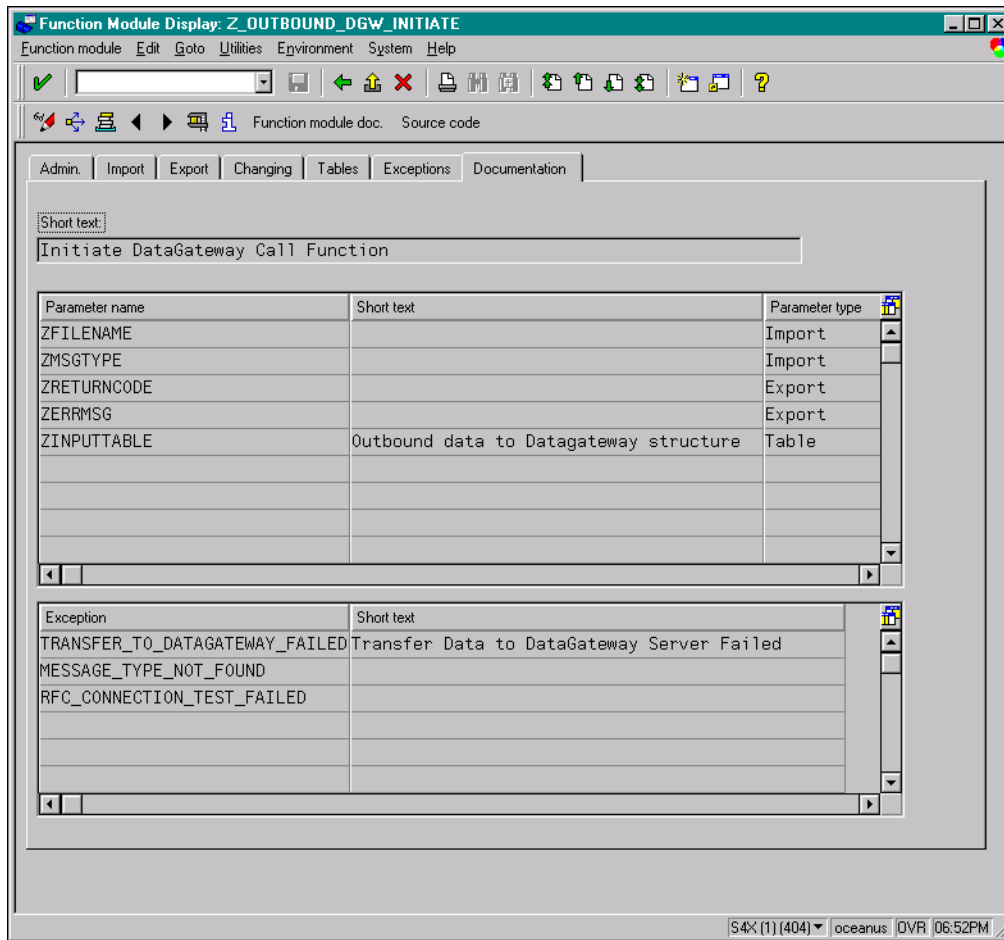


- 7 Press the **Save** push button. The new interval has been created.

## SAP to e\*Gate

In order to get data out of SAP, custom ABAP modules need to be developed to suit custom needs. However, once the data is extracted from SAP tables, the custom ABAP module can simply call `Z_OUTBOUND_DGW_INITIATE` to send data to the e\*Way. `Z_OUTBOUND_DGW_INITIATE` is delivered with the SAP BDC e\*Way. Following is the interface description of the `Z_OUTBOUND_DGW_INITIATE` function.

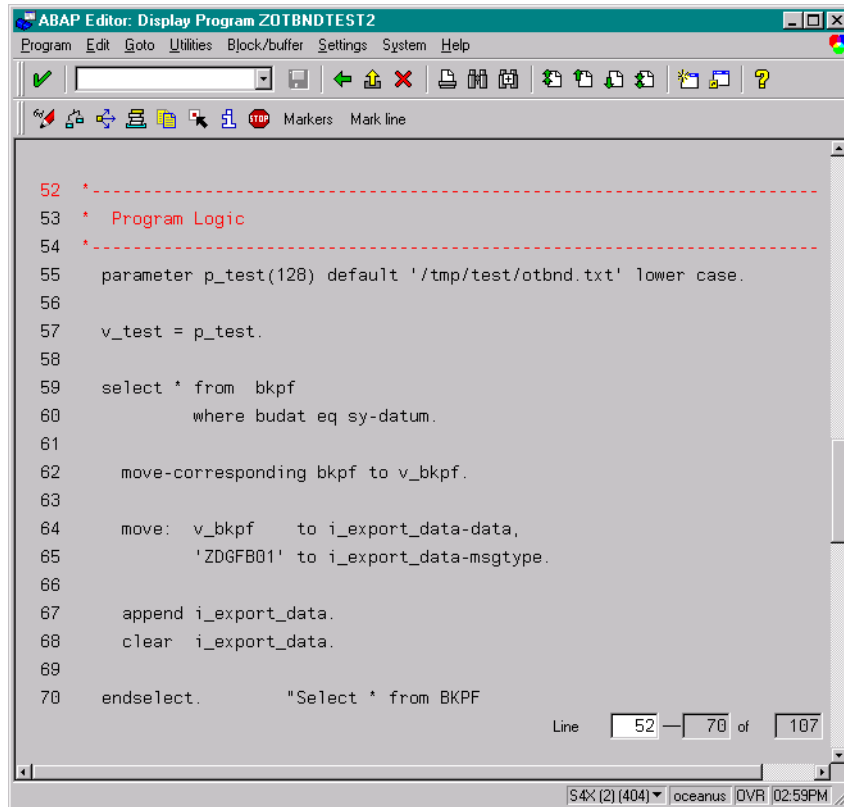
**Figure 39** Z\_OUTBOUND\_DGW\_INITIATE Description



Typically, a custom ABAP module would do the extraction of data on a set of SAP tables and then call Z\_OUTBOUND\_DGW\_INITIATE to send data to the SAP BDC e\*Way.

The following picture shows the data extraction part of an example custom ABAP module, ZOTBNDTEST2, which is delivered with the SAP BDC e\*Way.

Figure 40 ZOTBNDTEST2 (Data Extraction)



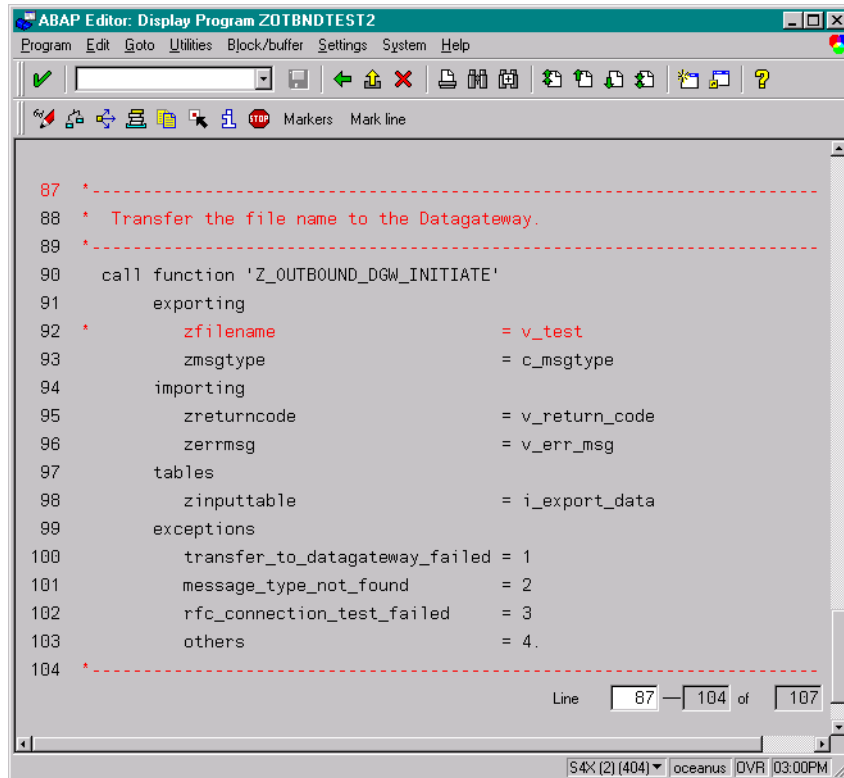
```
52  ^-----  
53  ^ Program Logic  
54  ^-----  
55  parameter p_test(128) default '/tmp/test/otbnd.txt' lower case.  
56  
57  v_test = p_test.  
58  
59  select * from bkpf  
60      where budat eq sy-datum.  
61  
62  move-corresponding bkpf to v_bkpf.  
63  
64  move: v_bkpf    to i_export_data-data,  
65      'ZDGFBO1' to i_export_data-msgtype.  
66  
67  append i_export_data.  
68  clear i_export_data.  
69  
70  endselect.      "Select * from BKPF
```

Line 52 — 70 of 107

S4X (2) (404) | oceanus | OVR | 02:59PM

In this example, data is extracted from the SAP table **BKPF** and buffered in the internal table **I\_export\_data**. In the next section of this ABAP module, the function **Z\_OUTBOUND\_DGW\_INITIATE** is called to send the data to the SAP BDC e\*Way.

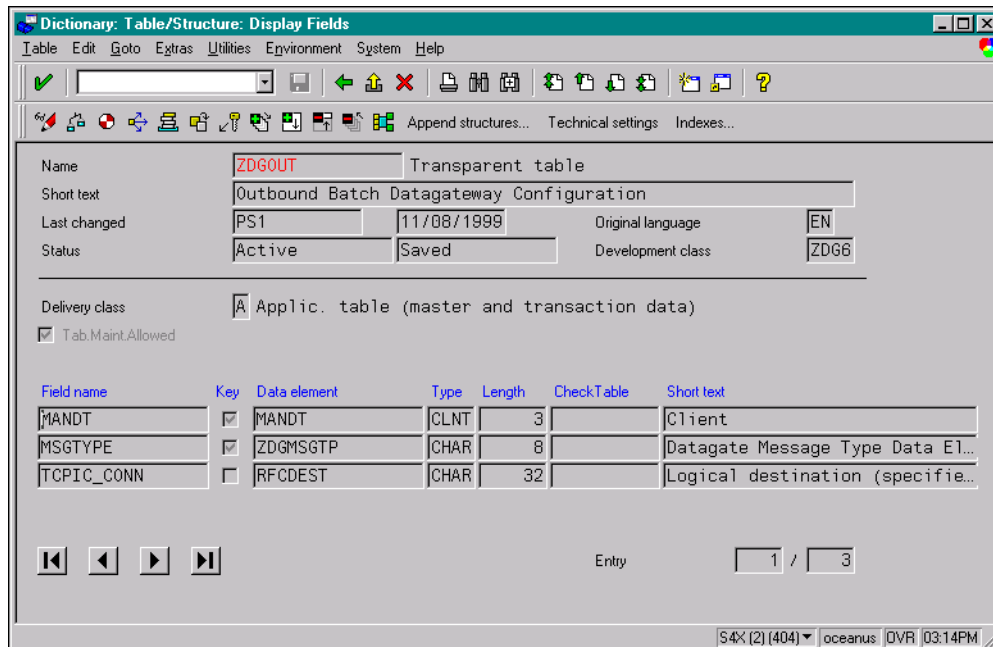
**Figure 41** ZOTBNDTEST2 (Transfer to DGW)



In this function call, the filename, message type, and the internal data table are passed to Z\_OUTBOUND\_DGW\_INITIATE. The return code from the latter is checked.

The input parameter 'message type' tells Z\_OUTBOUND\_DGW\_INITIATE where to send the data, or the RFC destination. Z\_OUTBOUND\_DGW\_INITIATE finds out the destination from an SAP table ZDGOUT, which is defined as follows.

**Figure 42** ZDGOUT definition



You can define this table through transaction SE11. To define message types, choose the menu path **Utilities > Table Content** and click the **Create** button. Then follow the SAP screen instructions.

The outbound batch configuration consists of associating the message type with the logical (registered) connection name via the table ZDGOUT. Entries to the ZDGBDC table are made by selecting the menu path **Utilities > Create Entries**, which invokes the *Table Insert* window.

The fields in this window are as follows:

**Table 12** Table Insert window fields

| Field      | Status   | Explanation   |
|------------|----------|---|
| MANDT      | Required | This field identifies the Client.   |
| MSGTYPE    | Required | This field is the Event Type which is sent as part of the call to the function Z_OUTBOUND_DGW_INITIATE.   |
| TCPIP_CONN | Required | This field is the TCP/IP connection which is set up in the transaction SM59. This connection represents the registered function name of the outbound batch e*Way. |

An example ZDGOUT table is shown below.

**Figure 43** ZDGOUT Table Example

| MANDT | MSGTYPE  | TCPIC_CONN    |
|-------|----------|---------------|
| 404   | SAPNODES | OUTBOUND_TEST |
| 404   | SAPRELN  | OUTBOUND_TEST |
| 404   | XIS_MSG  | Z_4XIS_ONLY   |
| 404   | ZOTTEST  | OUTBOUND_TEST |
| 404   | ZOUTTEST | OUTBOUND_TEST |

In this table, the message type ZOUTTEST is associated with the RFC destination (TCPIC\_CONN) OUTBOUND\_TEST. RFC destinations are defined with transaction SM59 as shown below.

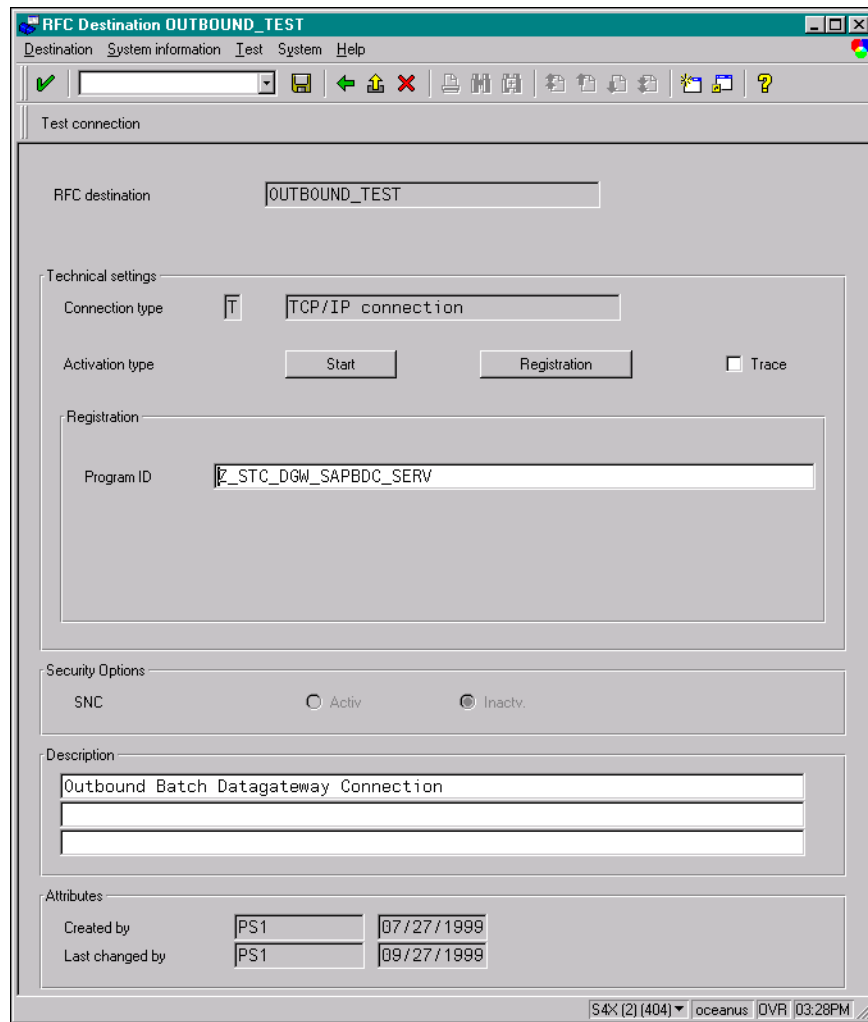
**Figure 44** RFC Destinations Tree

| Destination Name     | Description                  |
|----------------------|------------------------------|
| BSI-US-FDOC          |                              |
| BSI-US-TAX           | BSI-TAX (US) executable      |
| BSI-US-UPDATE        |                              |
| BSI40-US-TAX         |                              |
| CALLTP_HP-UX         |                              |
| CALLTP_Sun0S         | Transport tools: tp interfac |
| DATAGATE_R3_BDC_LOAD | This destination is the sour |
| DATAGATE_R3_EXPORT   | UNIX script executable.      |
| DOCUMENTATION_HELP   | Call WinHelp and WinWord fro |
| EPS_CLIENT           | Electronic Parcel Service: R |
| EU_SCRP_MF           | Graphical Screen Painter (Un |
| EU_SCRP_WN32         | Graphical Screen Painter (Wi |
| F1_HELP_SERVER       | Windows RFC server for F1 he |
| F1_HELP_SERVER_32    | Windows RFC server for F1 he |
| LOCAL_EXEC           | Starts the program 'RFCEXEC' |
| LOCAL_PRINT          |                              |
| OUTBOUND_TEST        | Outbound Batch Datagateway C |

Double clicking on the destination OUTBOUND\_TEST reveals its definition.



Figure 45 OUTBOUND\_TEST Definition



One important parameter for this destination definition is the Program ID. The Program ID identifies the e\*Way to which Z\_OUTBOUND\_DGW\_INITIATE sends messages of the associated type. At e\*Way startup, an SAP BDC e\*Way registers with the SAP Application Server using a unique Program ID. When a custom ABAP module calls Z\_OUTBOUND\_DGW\_INITIATE, a message type is passed together with the message to be sent. With the message type as the key, Z\_OUTBOUND\_DGW\_INITIATE looks up the RFC destination from the ZDGOOUT table. The message is routed to the correct SAP BDC e\*Way based on the Program ID.

# Setup Procedures

This chapter describes the procedures required to customize the SeeBeyond e\*Way Intelligent Adapter for SAP (BDC) to operate within your production system.

---

## 5.1 Overview

After creating a schema, you must instantiate and configure the SAP BDC e\*Way to operate within the schema. A wide range of setup options allow the e\*Way to conform to your system's operational characteristics and your facility's operating procedures.

The topics discussed in this chapter include the following:

### Setting Up the e\*Way

[Creating the e\\*Way](#) on page 67

[Modifying e\\*Way Properties](#) on page 68

[Configuring the e\\*Way](#) on page 69

[Changing the User Name](#) on page 73

[Setting Startup Options or Schedules](#) on page 73

[Activating or Modifying Logging Options](#) on page 75

[Activating or Modifying Monitoring Thresholds](#) on page 76

### Troubleshooting the e\*Way

[Configuration Problems](#) on page 77

[System-related Problems](#) on page 78

## 5.2 Setting Up the e\*Way

**Note:** The e\*Gate Schema Designer GUI runs only on the Windows operating system.

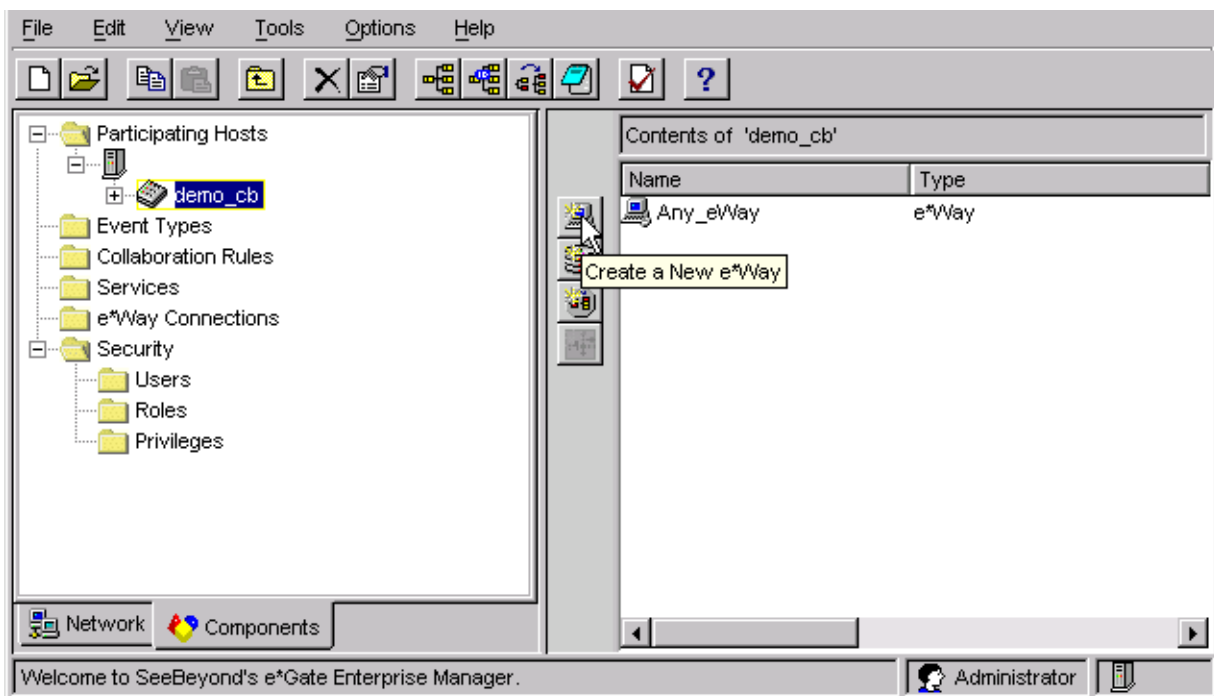
### 5.2.1 Creating the e\*Way

The first step in implementing an e\*Way is to define the e\*Way component using the e\*Gate Schema Designer.

To create an e\*Way

- 1 Open the schema in which the e\*Way is to operate.
- 2 Select the e\*Gate Schema Designer Navigator's **Components** tab.
- 3 Open the host on which you want to create the e\*Way.
- 4 Select the Control Broker you want to manage the new e\*Way.

**Figure 46** e\*Gate Schema Designer Window (Components View)



- 5 On the Palette, click Create a New e\*Way.
- 6 Enter the name of the new e\*Way, then click OK.
- 7 All further actions are performed in the e\*Gate Schema Designer Navigator's Components tab.

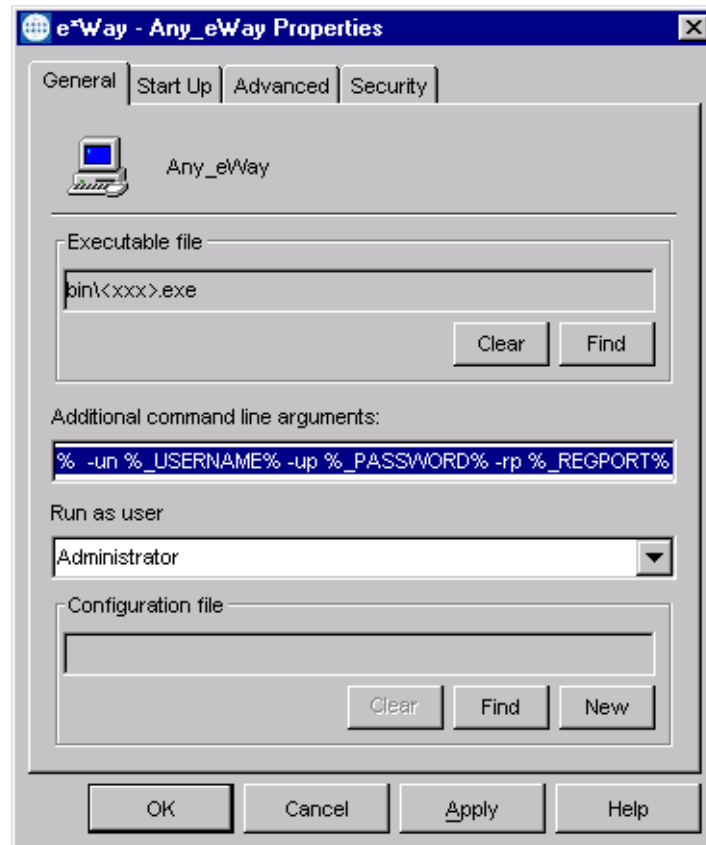
## 5.2.2 Modifying e\*Way Properties

To modify any e\*Way properties

- 1 Right-click on the desired e\*Way and select **Properties** to edit the e\*Way's properties. The properties dialog opens to the **General** tab (shown in Figure 47).

*Note:* The executable file is `stcewgenericmonk.exe`.

**Figure 47** e\*Way Properties (General Tab)



- 2 Make the desired modifications, then click **OK**.

### 5.2.3 Configuring the e\*Way

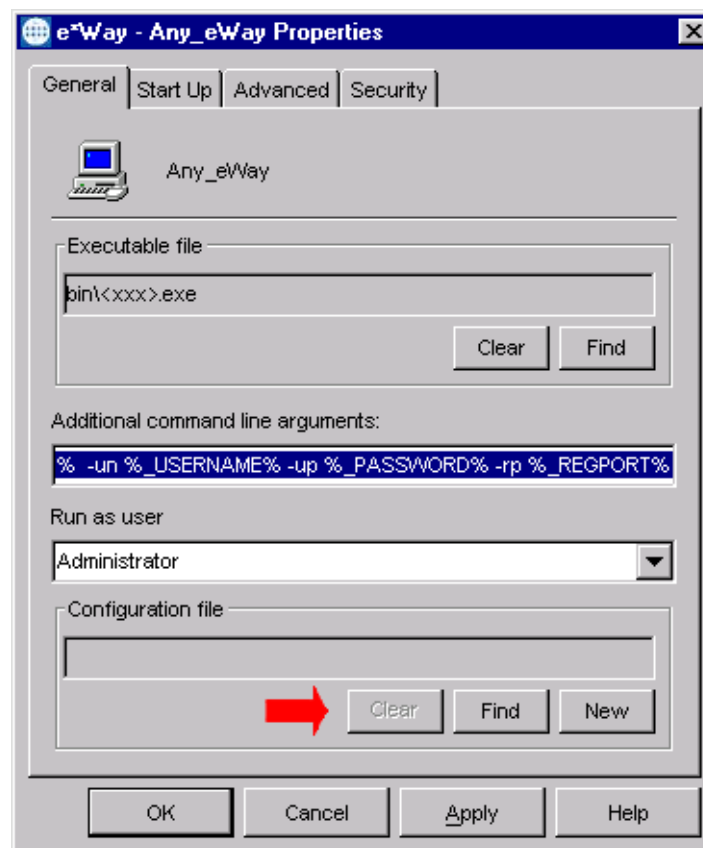
The e\*Way's default configuration parameters are stored in an ASCII text file with a .def extension. The e\*Way Editor provides a simple graphical interface for viewing and changing those parameters to create a working configuration (.cfg) file.

To change e\*Way configuration parameters

- 1 In the e\*Gate Schema Designer's Component editor, select the e\*Way you want to configure and display its properties.

**Note:** The default configuration file is `ewsapbdc.def`.

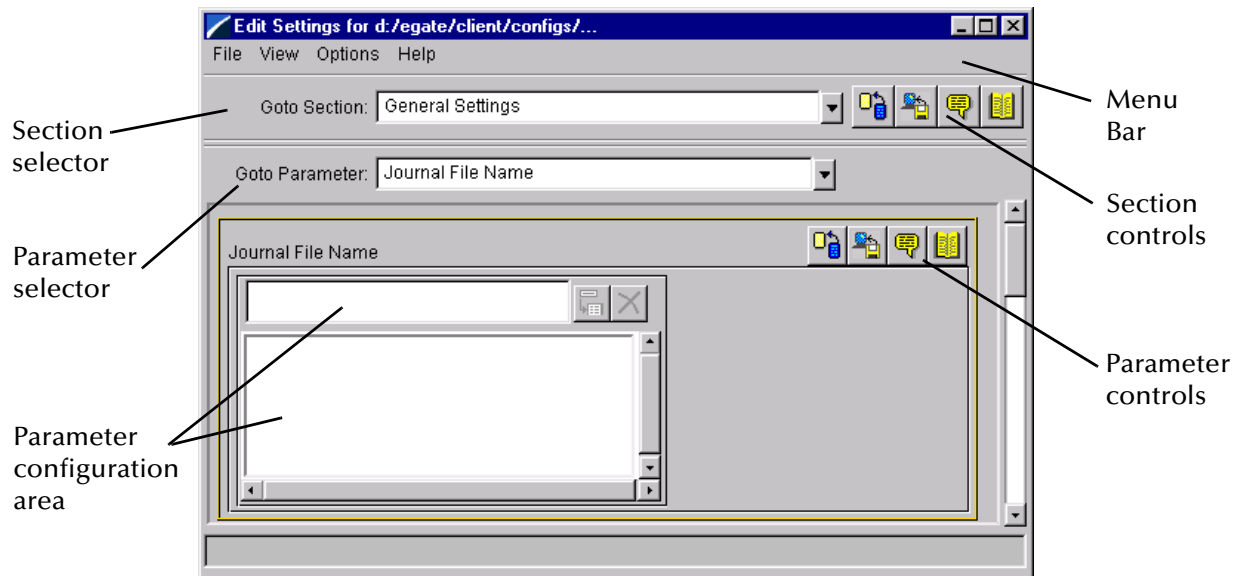
**Figure 48** e\*Way Properties - General Tab



- 2 Under **Configuration File**, click **New** to create a new file or **Find** to select an existing configuration file. If you select an existing file, an **Edit** button appears. Click this button to edit the currently selected file.
- 3 You are now in the e\*Way Configuration Editor.

## Using the e\*Way Editor

**Figure 49** The e\*Way Configuration Editor







The e\*Way Editor controls fall into one of six categories:

- The **Menu bar** allows access to basic operations (e.g., saving the configuration file, viewing a summary of all parameter settings, and launching the Help system)
- The **Section selector** at the top of the Editor window enables you to select the category of the parameters you wish to edit
- **Section controls** enable you to restore the default settings, restore the last saved settings, display tips, or enter comments for the currently selected section
- The **Parameter selector** allows you to jump to a specific parameter within the section, rather than scrolling
- **Parameter controls** enable you to restore the default settings, restore the last saved settings, display tips, or enter comments for the currently selected parameter
- **Parameter configuration controls** enable you to set the e\*Way's various operating parameters

## Section and Parameter Controls

The section and parameter controls are shown in Table 13 below.

**Table 13** Parameter and Section Controls

| Button  | Name                   | Function                |
|---|------------------------|-------------------------|
|  | <b>Restore Default</b> | Restores default values |
|  | <b>Restore Value</b>   | Restores saved values   |
|  | <b>Tips</b>            | Displays tips           |
|  | <b>User Notes</b>      | Enters user notes       |



*Note: The section controls affect all parameters in the selected section, whereas the parameter controls affect only the selected parameter.*

## Parameter Configuration Controls

Parameter configuration controls fall into one of two categories:

- Option buttons
- Selection lists, which have controls as described in Table 14

**Table 14** Selection List Controls

| Button  | Name                | Function  |
|---|---------------------|---|
|  | <b>Add to List</b>  | Adds the value in the text box to the list of available values.           |
|  | <b>Delete Items</b> | Displays a “delete items” dialog box, used to delete items from the list. |

---

## Command-line Configuration

In the **Additional Command Line Arguments** box, type any additional command line arguments that the e\*Way may require, taking care to insert them *at the end* of the existing command-line string. Be careful not to change any of the default arguments unless you have a specific need to do so.

---

## Getting Help

To launch the e\*Way Editor's Help system

From the **Help** menu, select **Help topics**.

To display tips regarding the general operation of the e\*Way

From the **File** menu, select **Tips**.

To display tips regarding the selected Configuration Section

In the **Section Control** group, click .

To display tips regarding the selected Configuration Parameter

In the **Parameter Control** group, click .

**Note:** *“Tips” are displayed and managed separately from the Help system that launches from the Toolbar's Help menu. You cannot search for Tips within the Help system, or view Help system topics by requesting Tips.*

For detailed descriptions and procedures for using the e\*Way Configuration Editor, see the *e\*Gate Integrator User's Guide*.



## 5.2.4 Changing the User Name

Like all e\*Gate executable components, e\*Ways run under an e\*Gate user name. By default, all e\*Ways run under the **Administrator** user name. You can change this if your site's security procedures so require.

### To change the user name

- 1 Display the e\*Way's properties dialog.
- 2 On the **General** tab, use the **Run as user** list to select the e\*Gate user under whose name this component is to run.

See the *e\*Gate Integrator System Administration and Operations Guide* for more information on the e\*Gate security system.

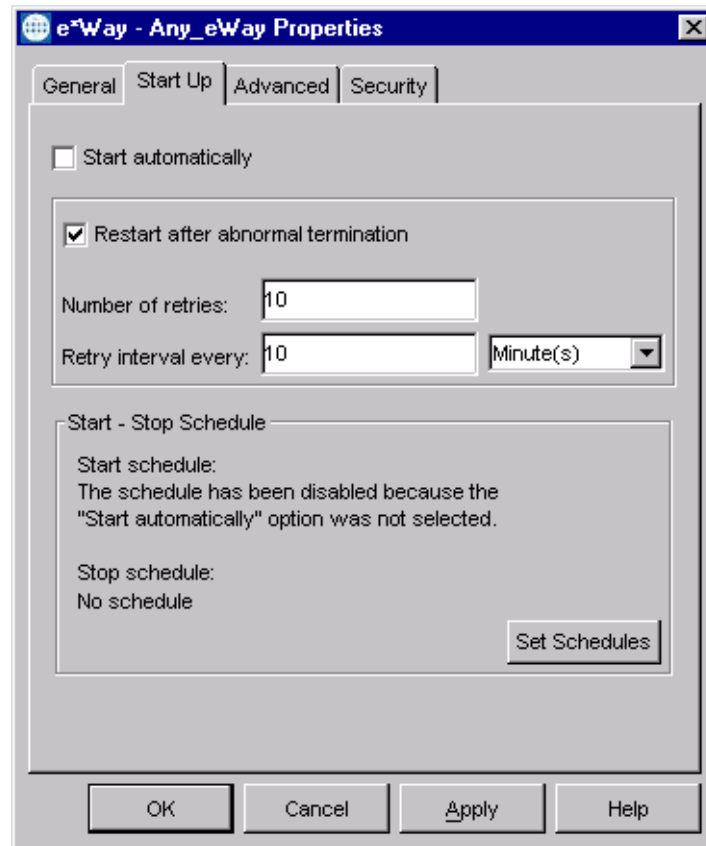
## 5.2.5 Setting Startup Options or Schedules

SeeBeyond e\*Ways can be started or stopped by any of the following methods:

- The Control Broker can start the e\*Way automatically whenever the Control Broker starts.
- The Control Broker can start the e\*Way automatically whenever it detects that the e\*Way terminated execution abnormally.
- The Control Broker can start or stop the e\*Way on a schedule that you specify.
- Users can start or stop the e\*Way manually using an interactive monitor.

You determine how the Control Broker starts or shuts down an e\*Way using options on the e\*Way properties **Start Up** tab (see Figure 50). See the *e\*Gate Integrator System Administration and Operations Guide* for more information about how interactive monitors can start or shut down components.

Figure 50 e\*Way Properties (Start-Up Tab)



#### To set the e\*Way's startup properties

- 1 Display the e\*Way's properties dialog.
- 2 Select the **Start Up** tab.
- 3 To have the e\*Way start automatically when the Control Broker starts, select the **Start automatically** check box.
- 4 To have the e\*Way start manually, clear the **Start automatically** check box.
- 5 To have the e\*Way restart automatically after an abnormal termination:
  - A Select **Restart after abnormal termination**.
  - B Set the desired number of retries and retry interval.
- 6 To prevent the e\*Way from restarting automatically after an abnormal termination, clear the **Restart after abnormal termination** check box.
- 7 Click **OK**.

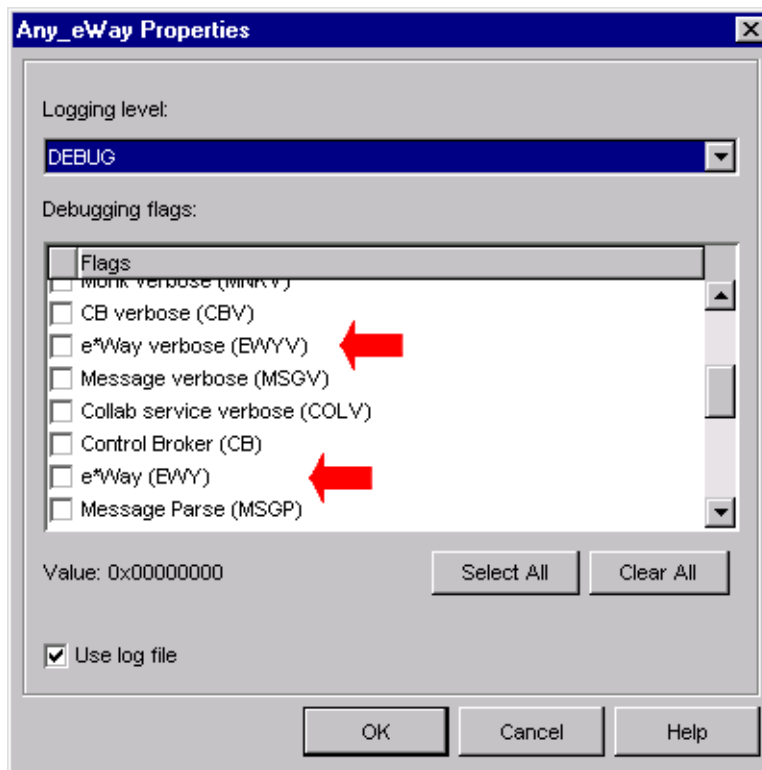
## 5.2.6 Activating or Modifying Logging Options

Logging options enable you to troubleshoot problems with the e\*Way and other e\*Gate components.

To set the e\*Way debug level and flag

- 1 Display the e\*Way's Properties dialog.
- 2 Select the **Advanced** tab.
- 3 Click **Log**. The dialog window appears (see Figure 51).

**Figure 51** e\*Way Properties (Advanced Tab - Log Option)



- 4 Select **DEBUG** for the **Logging level**.
- 5 Select either **e\*Way (EWY)** or **e\*Way Verbose (EWYV)** for the **Debugging flag**. Note that the latter has a significant negative impact on system performance.
- 6 Click **OK**.

The other options apply to other e\*Gate components and are activated in the same manner. See the *e\*Gate Integrator Alert and Log File Reference* for additional information concerning log files, logging options, logging levels, and debug flags.

## 5.2.7 Activating or Modifying Monitoring Thresholds

Monitoring thresholds enable you to monitor the throughput of the e\*Way. When the monitoring thresholds are exceeded, the e\*Way sends a Monitoring Event to the Control Broker, which routes it to the e\*Gate Schema Manager and any other configured destinations.

- 1 Display the e\*Way's properties dialog.
- 2 Select the **Advanced** tab.
- 3 Click **Thresholds**.
- 4 Select the desired threshold options and click **OK**.

See the *e\*Gate Integrator Alert and Log File Reference* for more information concerning threshold monitoring, routing specific notifications to specific recipients, or for general information about e\*Gate's monitoring and notification system.

---

## 5.3 Troubleshooting the e\*Way

In the initial stages of developing your e\*Gate Integrator system administration system, most problems with e\*Ways can be traced to configuration.

### 5.3.1 Configuration Problems

#### In the Schema Designer

- Does the e\*Way have the correct Collaborations assigned?
- Do those Collaborations use the correct Collaboration Services?
- Is the logic correct within any Collaboration Rules script employed by this e\*Way's Collaborations?
- Do those Collaborations subscribe to and publish Events appropriately?
- Are all the components that *feed* this e\*Way properly configured, and are they sending the appropriate Events correctly?
- Are all the components that this e\*Way *feeds* properly configured, and are they subscribing to the appropriate Events correctly?

#### In the e\*Way Editor

- Check that all configuration options are set appropriately.
- Check that all settings you changed are set correctly.
- Check all required changes to ensure they have not been overlooked.
- Check the defaults to ensure they are acceptable for your installation.

#### On the e\*Way's Participating Host

- Check that the Participating Host is operating properly, and that it has sufficient disk space to hold the IQ data that this e\*Way's Collaborations publish.
- Check that your *path* environment variable includes the location of the XXX dynamically-loaded libraries. The name of this variable on the different operating systems is:
  - ♦ PATH (Windows)
  - ♦ LD\_LIBRARY\_PATH (Solaris)
  - ♦ LIBPATH (AIX)
  - ♦ SHLIB\_PATH (HP-UX)

#### In the SAP Application

- Check that the application is configured correctly, is operating properly, and is sending or receiving the correct data appropriately.

## 5.3.2 System-related Problems

- Check that the connection between the external application and the e\*Way is functioning appropriately.
- Once the e\*Way is up and running properly, operational problems can be due to:
  - ♦ External influences (network or other connectivity problems).
  - ♦ Problems in the operating environment (low disk space or system errors)
  - ♦ Problems or changes in the data the e\*Way is processing.
  - ♦ Corrections required to Collaboration Rules scripts that become evident in the course of normal operations.

One of the most important tools in the troubleshooter's arsenal is the e\*Way log file. See the *e\*Gate Integrator Alert and Log File Reference Guide* for an extensive explanation of log files, debugging options, and using the e\*Gate Schema Manager system to monitor operations and performance.

# Operational Overview

Data communication with SAP is a multi-step process. In this chapter, we examine these steps in greater detail. The transaction processing is discussed within an architectural context.

Topics discussed in the chapter include:

[Obtaining the SAP Data Structure](#) on page 79

[Data Mapping/Collaboration](#) on page 83

[SAP BDC e\\*Way Architecture](#) on page 88

[Basic e\\*Way Processes](#) on page 92

---

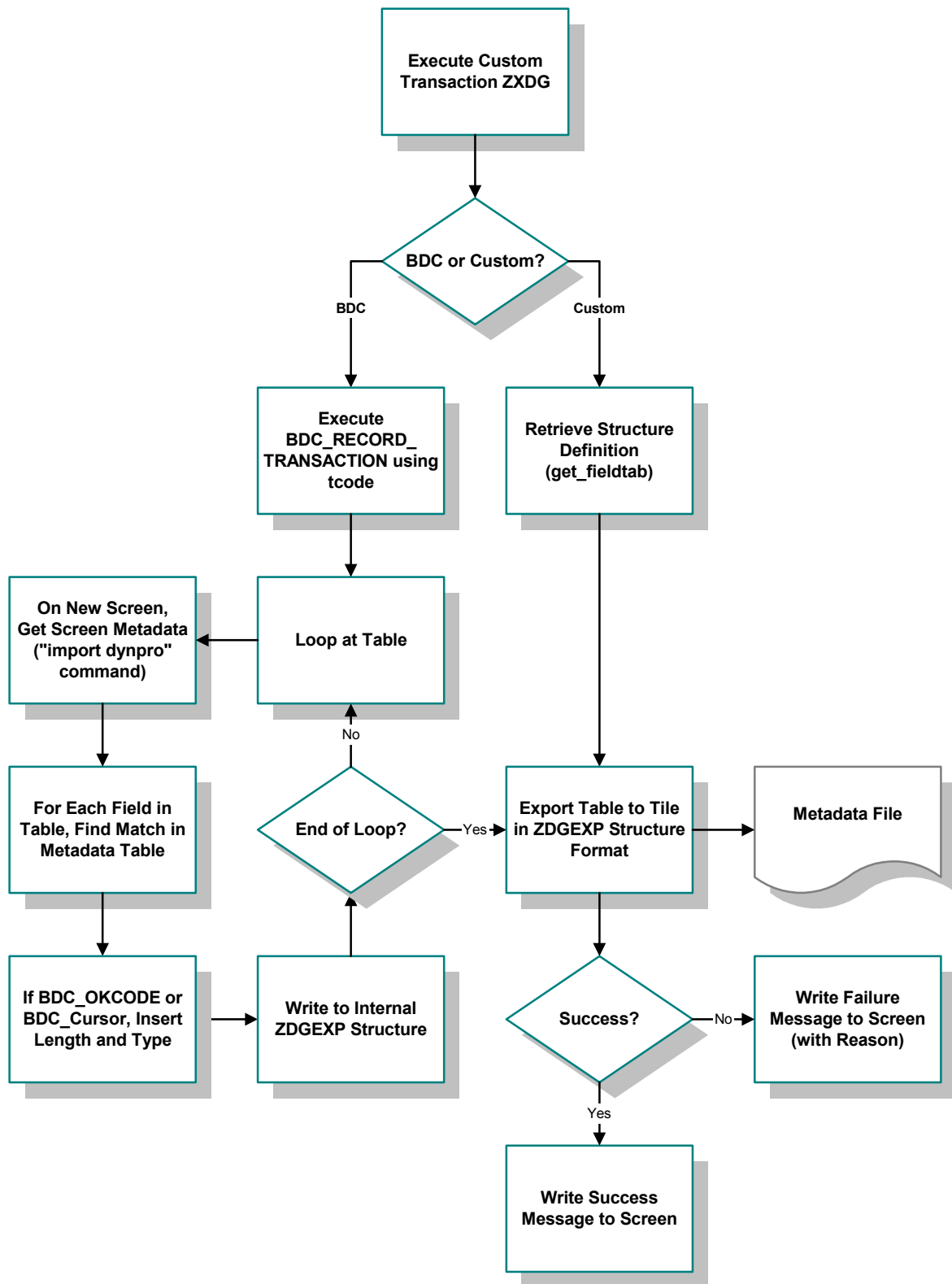
## 6.1 Obtaining the SAP Data Structure

### 6.1.1 SAP Structure Export Module

SAP expects incoming data to correspond to fields in the user interface, following a specific flow pattern. To accomplish this, the flow pattern and field structure must be exported from SAP to e\*Gate to serve as a data structure for mapping the incoming data format.

The SAP BDC interface incorporates an SAP export routine that merges this screen flow data with the screen field metadata and exports this data to a file. For situations where the screen recorder cannot be used (for example, high screen complexity), it allows the use of a custom structure for mapping.

Figure 52 SAP Structure Export Process Flow





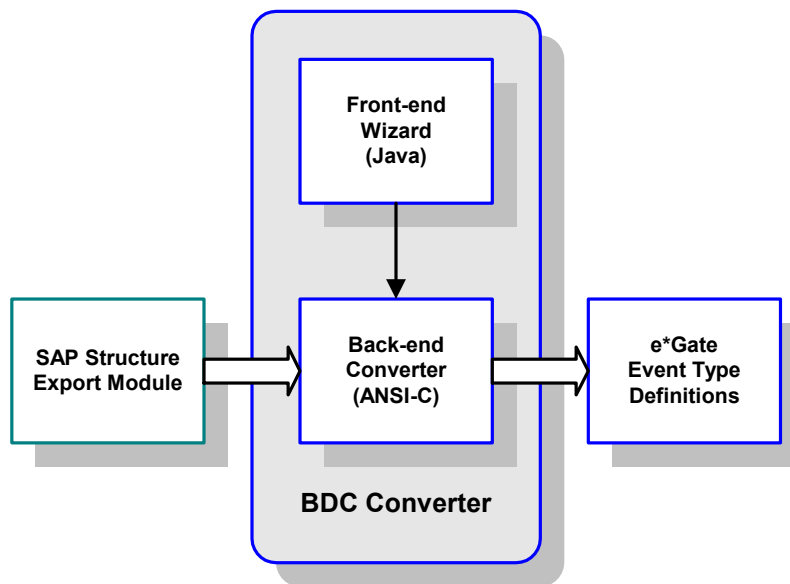
## 6.1.2 BDC Converter

The metadata file produced by the SAP Structure Export Module is used by the e\*Gate BDC Converter to produce an Event Type Definition (ETD). The Converter works within the e\*Gate Structure Builder which, in turn, is part of the ETD Editor.

The SAP BDC Converter is an e\*Way utility that enables you to import data structure information from SAP. In this case, either screen flows or custom structure definitions are exported from the SAP Structure Export Module. The SAP BDC Converter interprets the output from the SAP Structure Export Module into a graphical Event Type Definition, and presents it to the user via the e\*Gate ETD Editor.

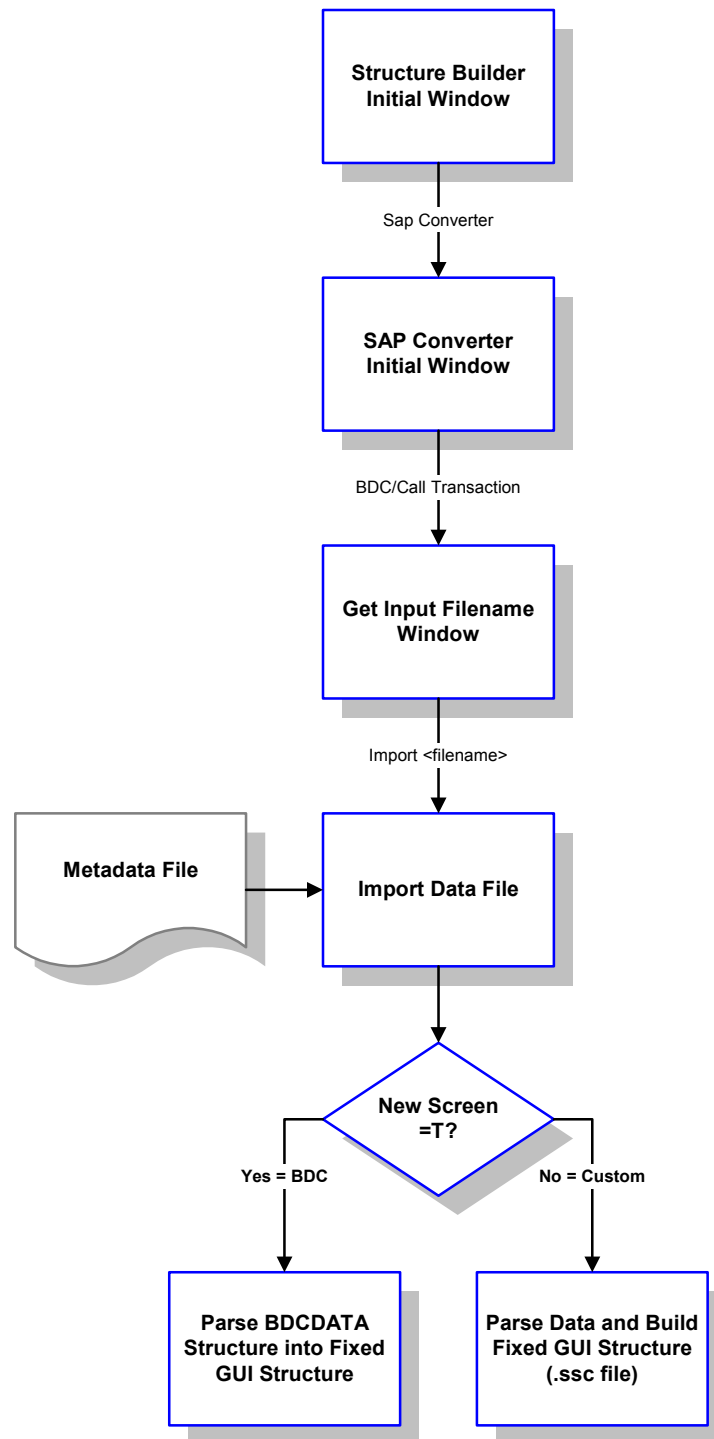
The BDC Converter Wizard GUI is implemented in Java, and calls an ANSI-C program that performs the actual conversion between the SAP data structure and e\*Gate Event Type Definitions.

**Figure 53** SAP BDC Converter



The BDC Converter process flow is shown in Figure 54.

**Figure 54** BDC Converter Process Flow



This information provides the destination data structure existing within SAP. Corresponding information from the external application provides the source data structure. Given these two sets of structure information, e\*Gate Collaboration Rules can be defined to map the data from the external system to SAP.

---

## 6.2 Data Mapping/Collaboration

### 6.2.1 e\*Gate to SAP

Both update methods (BDC and Call Transaction) input data to the SAP system by simulating users entering the data directly onto the transaction windows. SAP provides a **Screen Recorder** utility, which allows users to run through a transaction and record each window that is displayed, and write this data to a file.

- Transaction: **SE38**
- Function Name: **Z\_STRUCTURE\_EXPORT**

A SAP export routine merges the screen flow data with the screen field metadata and exports the combined data to a file. For complex screen flows it often is impossible to successfully record the screen flow using the recorder. In these situations, you must create a custom input structure in SAP, which also serves as the mapping structure in e\*Gate (see Figure 55). An example ABAP module template, **Z\_CUSTOM\_MAPPING**, is included in the transport files accompanying the e\*Way.

If the standard SAP screen mapping is used, then a BDC or Call Transaction (whichever is configured) is executed immediately. If a custom screen mapping structure is used, then a custom function module is executed to format any custom screen mapping and submit the BDC session or Call Transaction (see Figure 56).

Figure 55 Custom Mapping Logic

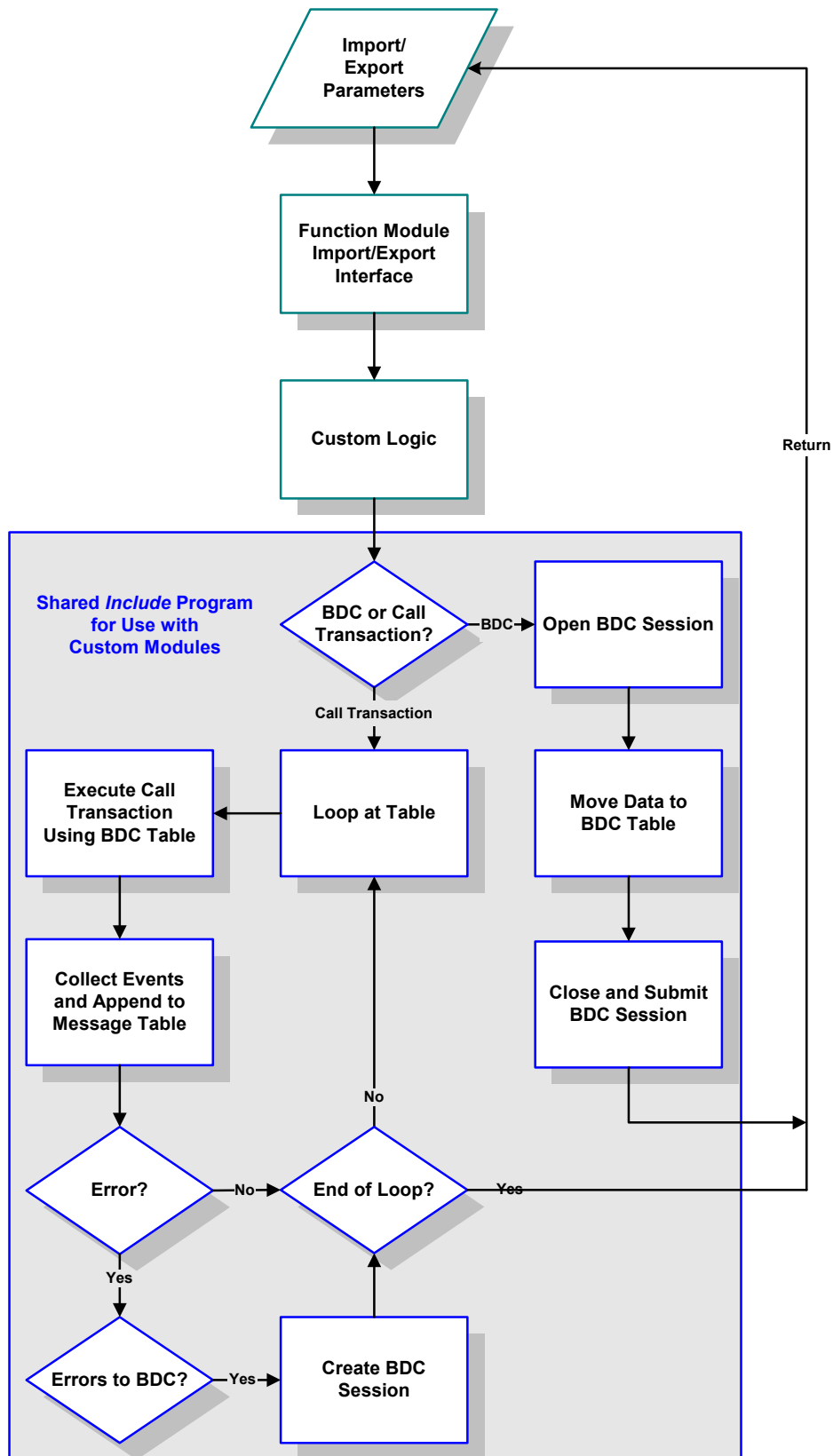
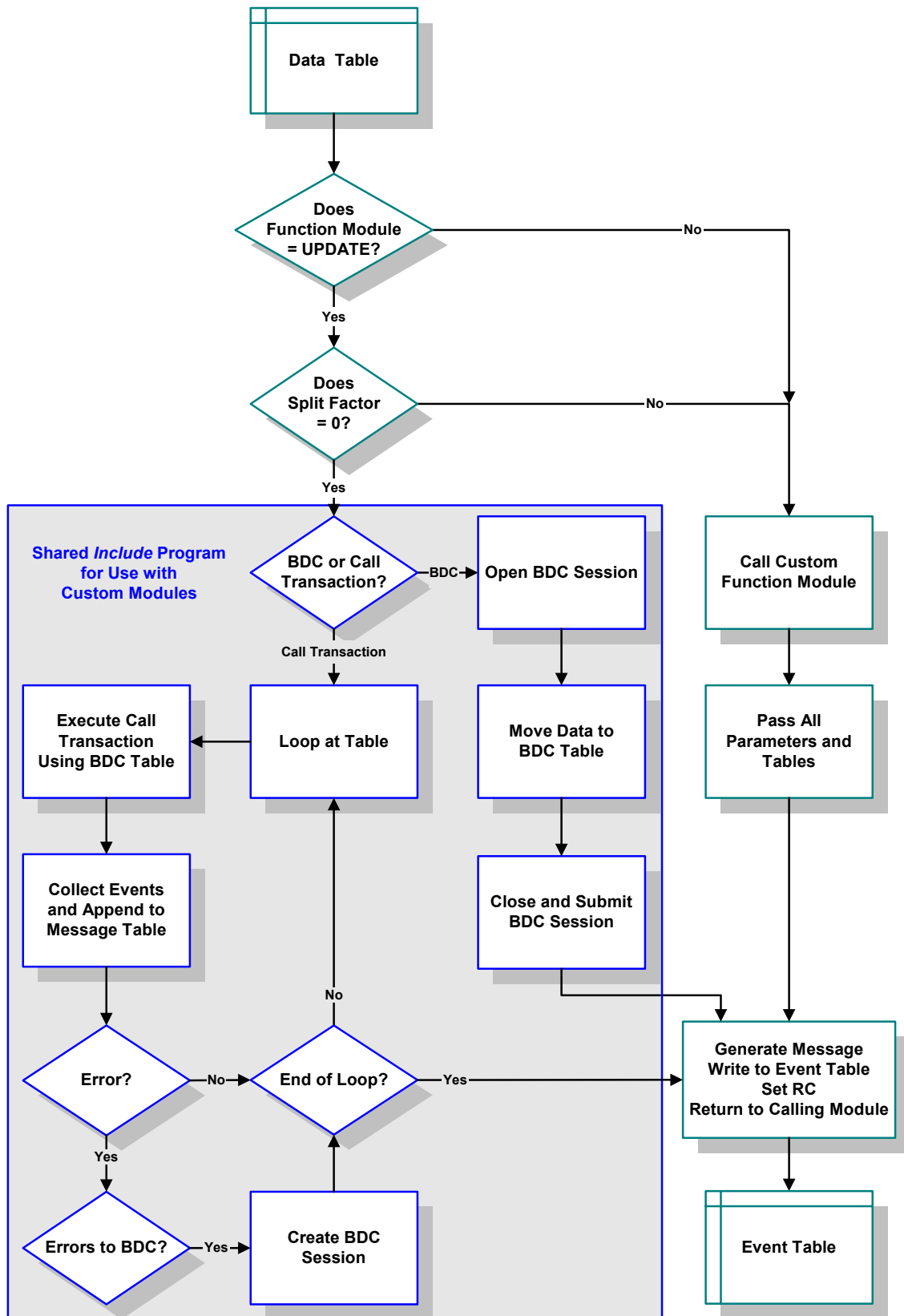


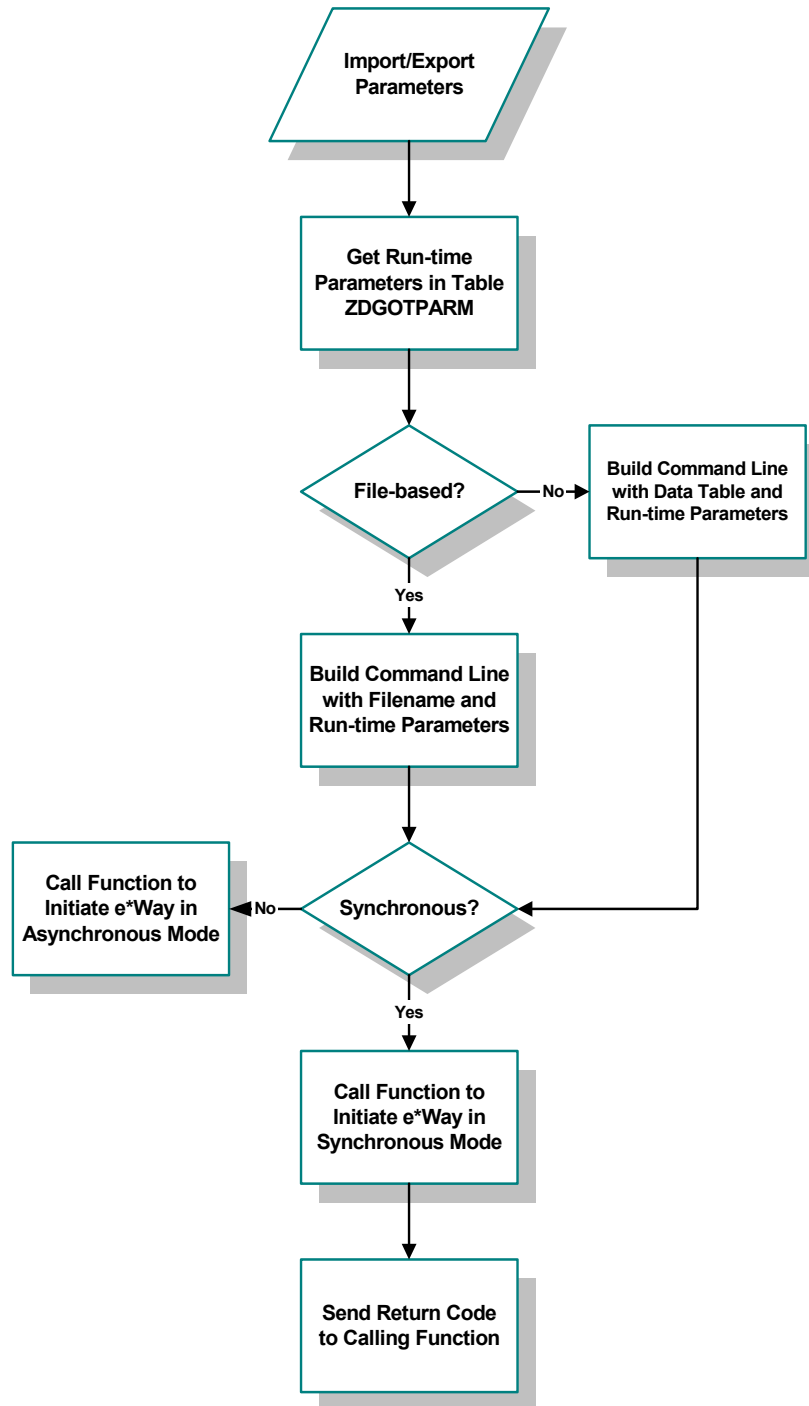
Figure 56 Event Processing



## 6.2.2 SAP to e\*Gate

The BDC e\*Way provides both synchronous and asynchronous TCP/IP connectivity for message-based scenarios, as well as asynchronous FTP functionality for file-based interfaces. The logic followed by the e\*Way is depicted in Figure 57.

**Figure 57** SAP Data Export Logic Flow



A custom SQL process extracts data from the SAP system by querying tables based on custom-developed logic. In order to accomplish this, the custom structure metadata must first be exported from SAP. After a metadata file has been created, this metadata is imported to e\*Gate via the SAP Converter.

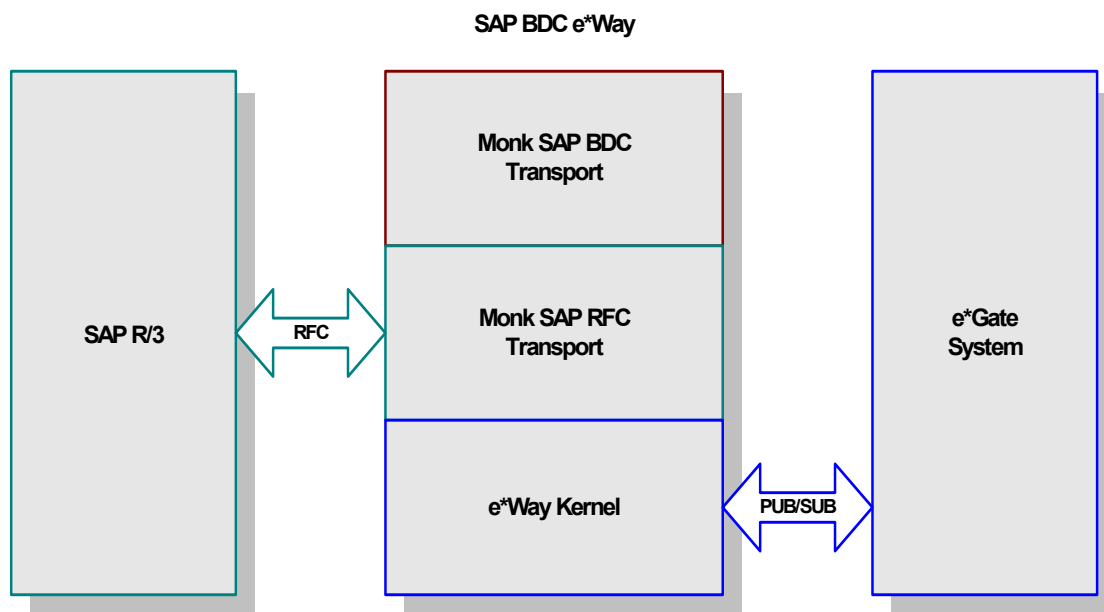
The BDC Converter creates a graphical structure definition in e\*Gate that is used to map the SAP data to the internal e\*Gate structure. A similar procedure is followed to obtain data mapping information from e\*Gate to the target application.

## 6.3 SAP BDC e\*Way Architecture

Conceptually, the e\*Way can be viewed as a multi-layered structure (see Figure 58), consisting of:

- Monk SAP BDC Transport layer, which manages BDC communication logic
- Monk SAP RFC Transport layer, which manages RFC communications with the SAP R/3 system
- e\*Way Kernel layer, which manages the processing of data and subscribing or publishing to other e\*Gate components

**Figure 58** SAP BDC e\*Way Architecture



The upper layers of the e\*Way use Monk functions to perform Business Process modeling and ETD mapping, package data as e\*Gate *Events*, send those Events to Collaborations, and manage interaction with the external system. These layers are built upon an e\*Way Kernel layer that manages the basic operations of the e\*Way, data processing, and communication with other e\*Gate components.

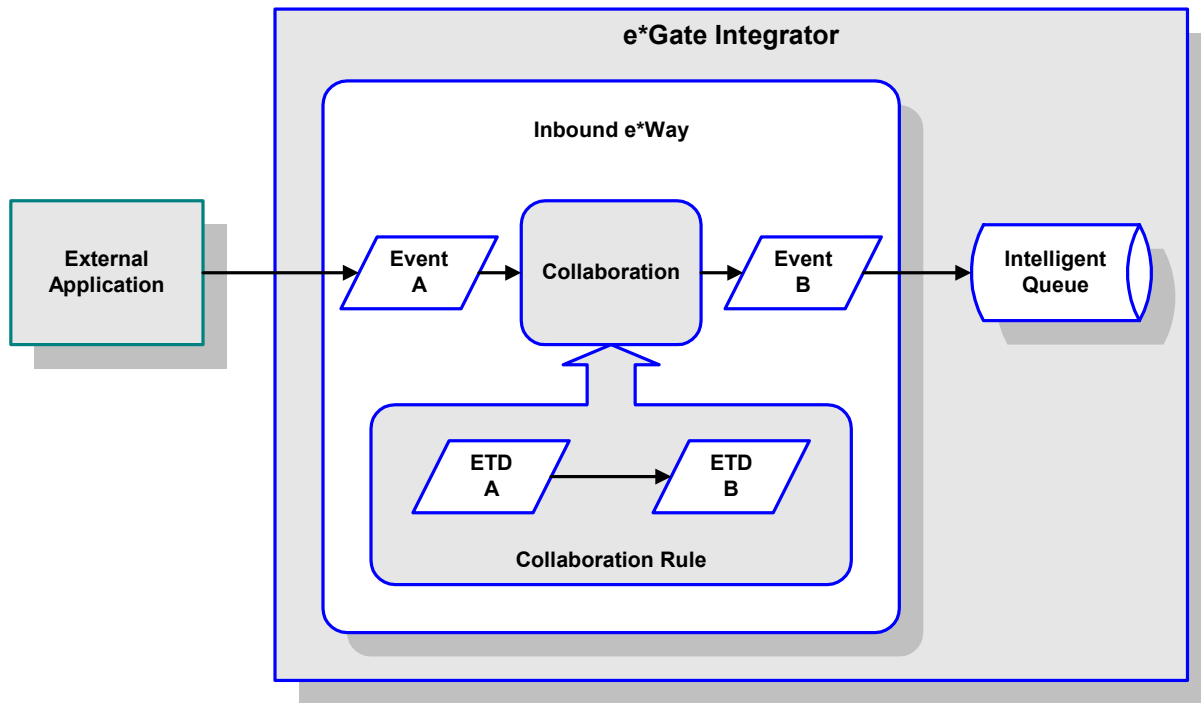
The communication layers of the e\*Way are single-threaded. Functions run serially, and only one function can be executed at a time. Processing layers are multi-threaded, with one executable thread for each Collaboration. Each thread maintains its own Monk environment; therefore, information such as variables, functions, path information, and so on cannot be shared between threads.



### 6.3.1 Events and Collaborations

Collaborations execute the business logic that enable the e\*Way to do its intended work. In turn, each Collaboration executes a Collaboration Rule, containing the actual instructions to execute the business logic. Each Collaboration that publishes its processed Events internally (within e\*Gate Integrator) requires one or more IQs to receive the Events, as shown in Figure 59. Any Collaboration that publishes its processed Events only to an external system does *not* require *any* IQs.

**Figure 59** Collaborations and IQs



Configuration options that control the Monk environment and define the Monk functions used to perform various e\*Way operations are discussed in [Chapter 7](#). You can create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as *Microsoft Word* or *Notepad*, or UNIX *vi*). The available set of e\*Way API functions is described in [Chapter 8](#). Generally, e\*Way Kernel Monk functions should be called directly only when there is a specific need not addressed by higher-level Monk functions, and should be used only by experienced developers.

For more information on defining Collaborations, defining IQs, assigning Collaborations to e\*Ways, or configuring Collaborations to publish Events, see the *e\*Gate Integrator User's Guide*.

## 6.3.2 Monk SAP BDC Transport Layer

The Monk SAP BDC Transport layer controls the BDC logic, including both the **Batch** and **Event-driven** modes of operation, and the **Batch Data Communication (BDC)** and **Call Transaction** data-update modes. It also initiates the calls to RFC-enabled function modules within both the e\*Way and the SAP application.

## 6.3.3 Monk SAP RFC Transport Layer

The Monk SAP RFC Transport layer controls the exchange of data between the SAP application and e\*Gate.

### e\*Gate-to-SAP Data Exchange

Data exchange from e\*Gate to SAP proceeds as follows:

- 1 The SAP BDC e\*Way connects to the target SAP system as an RFC client.
- 2 The e\*Way invokes the SAP ABAP module **Z\_R3\_BDC\_DATA\_IMPORT** on the remote SAP system. This module is provided with the SAP BDC e\*Way, and must be installed on the remote SAP system. See [Updating SAP R/3 Objects](#) on page 53 for more information.
- 3 The e\*Way invokes its own **Process Outgoing Message Function**, which sends data from the e\*Way to the SAP system. Depending on how the e\*Way is configured, data can be transmitted in event-driven mode or batch mode; the former is sent directly via TCP/IP, while the latter is sent via FTP.
- 4 The remote SAP system returns a status message to the e\*Way.

**Note:** *The custom ABAP function for SAP-inbound data presumes that the incoming data uses standard SAP data formats. If your data uses a custom format, you may need to write a custom ABAP function to process it.*

Depending on the situation, connecting to and disconnection from the target SAP system can be done once per data exchange (within the **Process Outgoing Message Function**), or once for each instance of the e\*Way's execution (within the **Startup Function** and **Shutdown Command Notification Function**).

## SAP-to-e\*Gate Data Exchange

Data exchange from SAP to e\*Gate proceeds as follows:

- 1 The SAP BDC e\*Way's **Startup Function** connects to the target SAP system as an RFC server and registers an RFC server module `Z_STC_DGW_SAPBDC_SERV`. An ABAP function, which must be custom-written for each installation, extracts data from the SAP system, and then calls another ABAP module, `Z_OUTBOUND_DGW_INITIATE` (delivered with the e\*Way), to send the data to the e\*Way via RFC. See [Updating SAP R/3 Objects](#) on page 53 for more information.
- 2 At the e\*Gate system, the e\*Way executes its **Exchange Data with External Function**, which checks for incoming data. When that data is received, one of the following happens:
  - ♦ If the e\*Way is operating in Event-driven mode, the e\*Way receives the SAP data in its entirety. The e\*Way processes the data and puts the data into an e\*Gate IQ.
  - ♦ If the e\*Way is operating in batch mode, the e\*Way receives only the file name of the SAP data to be processed. The e\*Way uses FTP to fetch the data file from the remote SAP system, then processes and enqueues the data stored in that file.
- 3 Once the data from the SAP system has been packaged within an Event Type Definition and placed within an IQ, it can be picked up and processed by other e\*Gate components and routed to its eventual destination.

**Note:** Connection to the source SAP system must be kept alive during the life cycle of the e\*Way.

**Note:** Obsolete SAP data types such as the following are **not** supported:

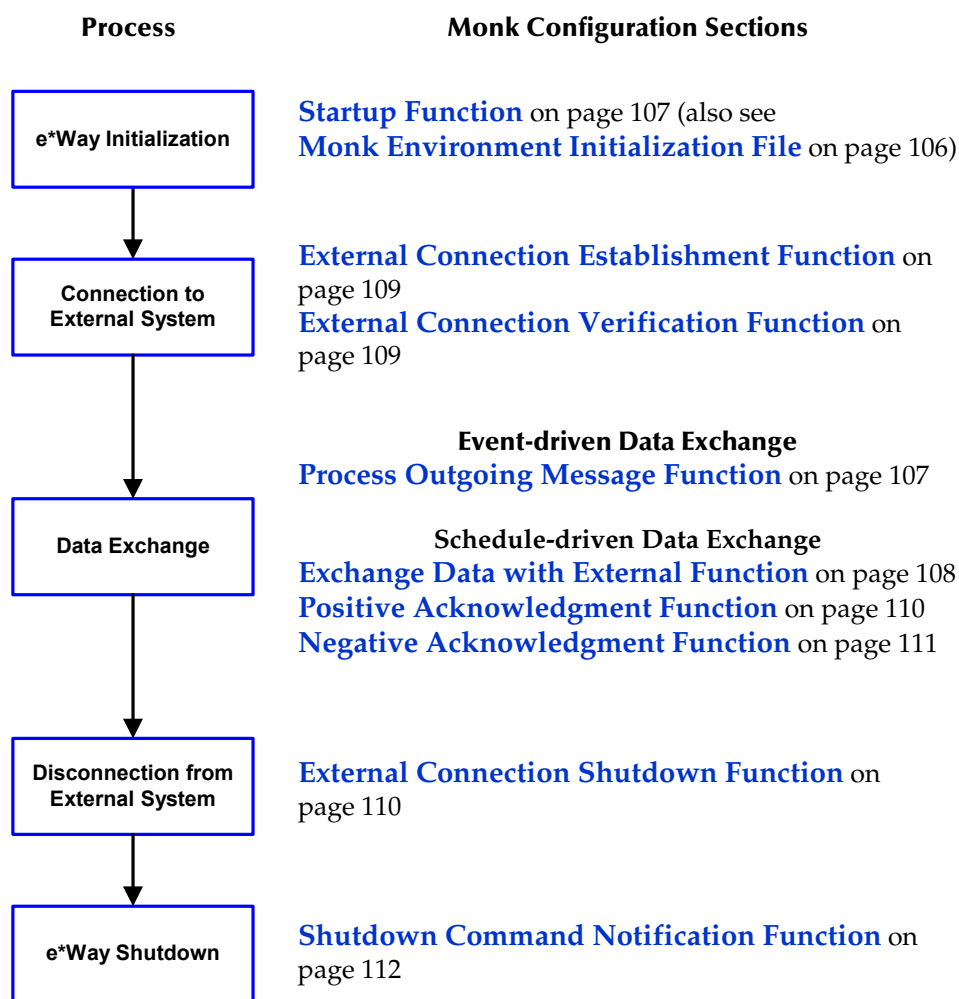
`RFCTYPE_INT1`  
`RFCTYPE_INT2`  
`RFCTYPE_DATE_1`  
`RFCTYPE_DATE_1`

## 6.4 Basic e\*Way Processes

**Note:** This section describes the basic operation of a typical e\*Way based on the Generic e\*Way Kernel. Not all functionality described in this section is used routinely by the SAP BDC e\*Way.

The most basic processes carried out by an e\*Way are listed in Figure 60. In e\*Ways based on the Generic Monk e\*Way Kernel (using `stcewgenericmonk.exe`), these processes are controlled by the listed Monk functions. Configuration of these functions is described in the referenced sections of this User's Guide.

**Figure 60** Basic e\*Way Processes

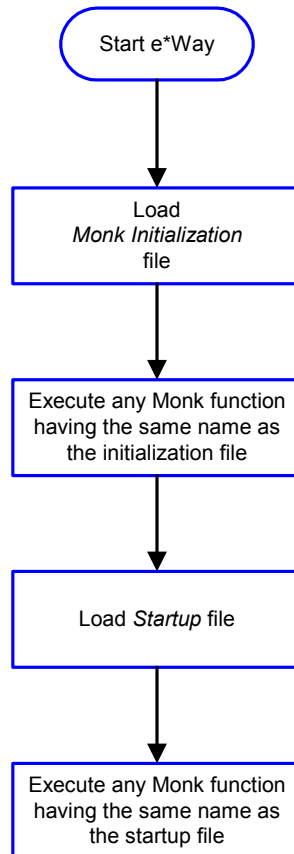


A series of diagrams on the next several pages illustrate the interaction and operation of these functions during the specified processes. Configuring the parameters associated with these functions is covered in [Chapter 7](#), while the functions themselves are described in [Chapter 8](#).

## Initialization Process

Figure 61 illustrates the e\*Way's initialization process, using the **Monk Environment Initialization File** and **Startup Function**.

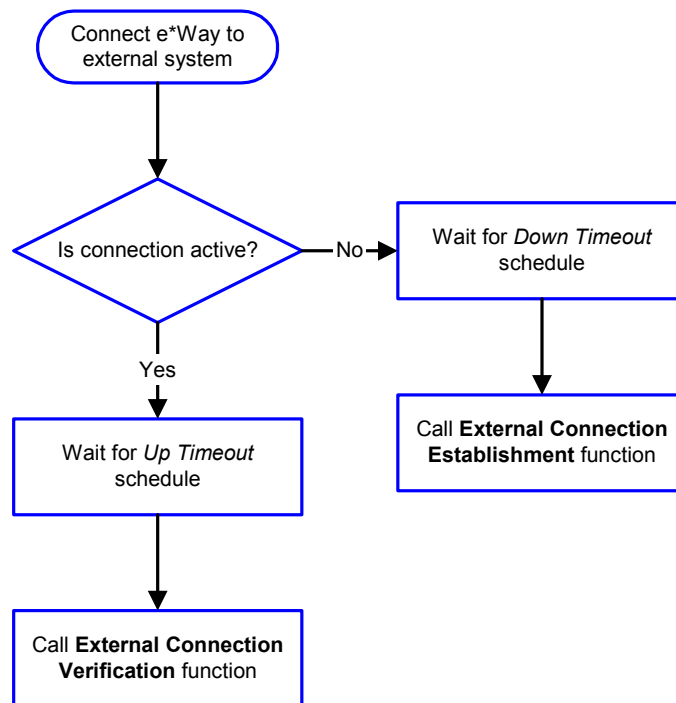
**Figure 61** Initialization Process



## Connect to External Process

Figure 62 illustrates how the e\*Way connects to the external system, using the **External Connection Establishment Function** and **External Connection Verification Function**.

**Figure 62** Connection Process



**Note:** The e\*Way selects the connection function based on an internal **up/down** flag rather than a poll to the external system. See **Figure 64 on page 96** and **Figure 63 on page 95** for examples of how different functions use this flag.

User functions can manually set this flag using Monk functions. See **send-external-up** on page 172 and **send-external-down** on page 172 for more information.

## Data Exchange Process

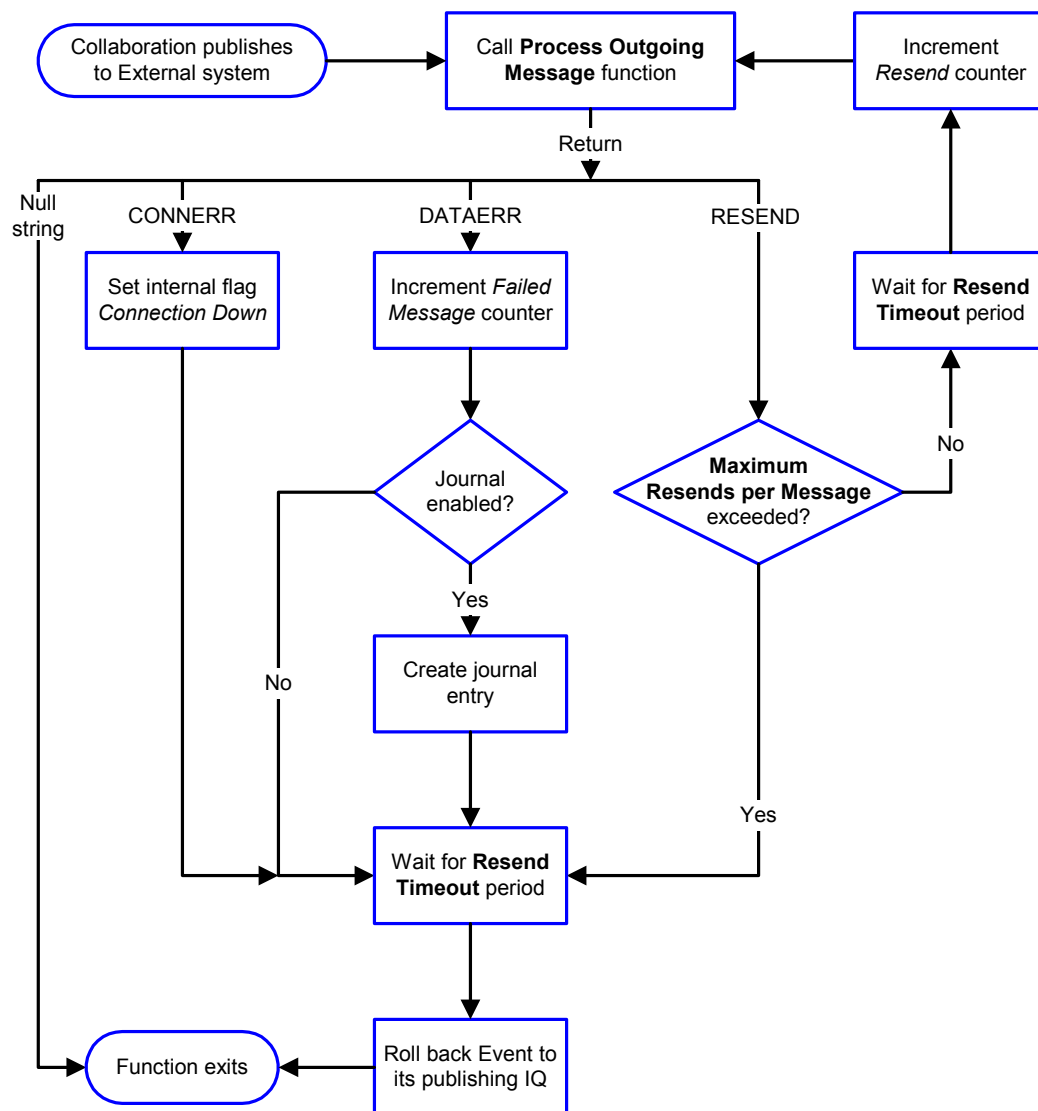
### Event-driven

Figure 63 illustrates how the e\*Way’s event-driven data exchange process works, using the **Process Outgoing Message Function**.

The e\*Way periodically checks the *Failed Message* counter against the value specified by the **Max Failed Messages** parameter. When the *Failed Message* counter exceeds the specified maximum value, the e\*Way logs an error and shuts down.

After the function exits, the e\*Way waits for the next outgoing Event.

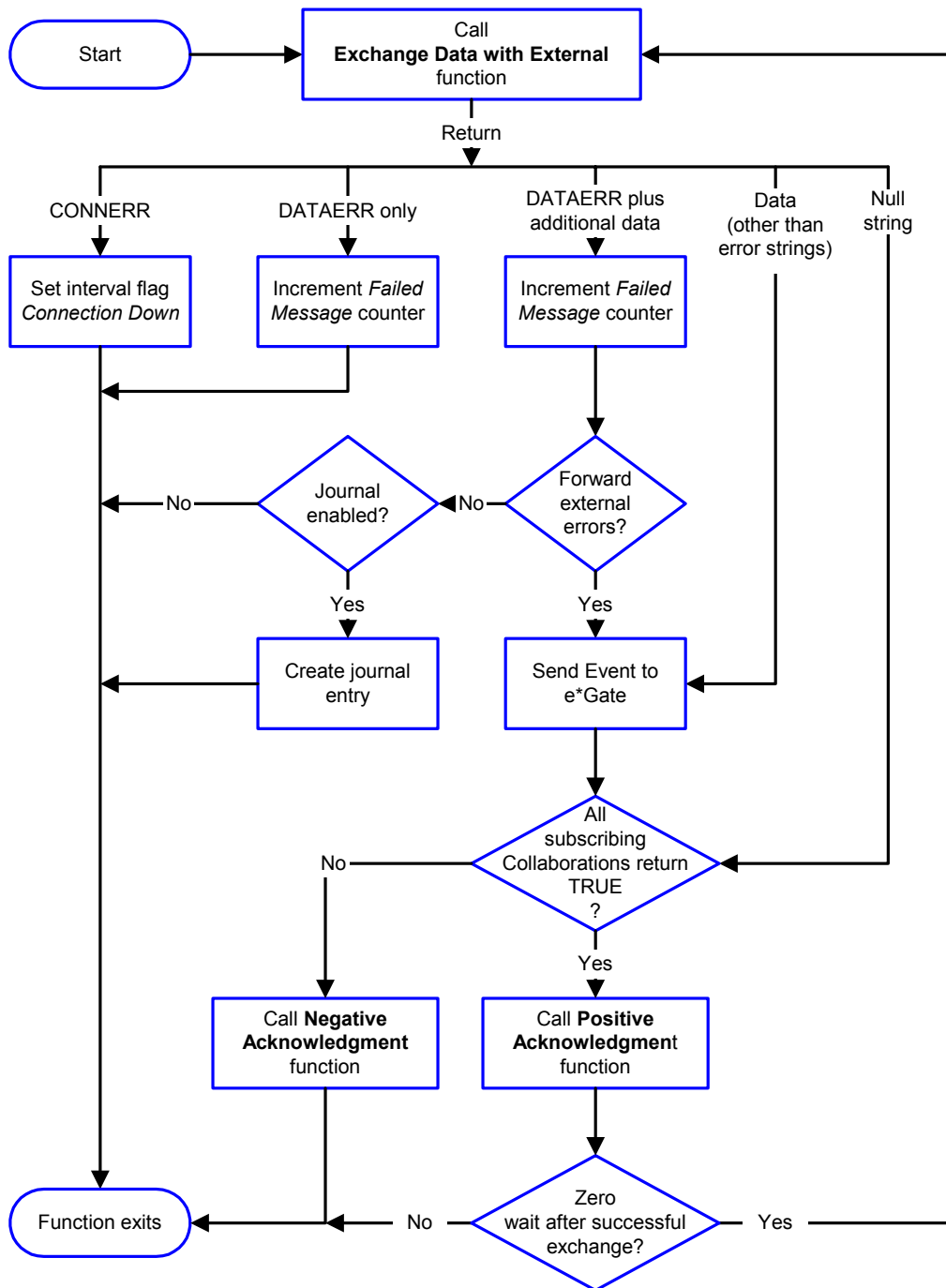
**Figure 63** Event-Driven Data Exchange Process



Schedule-driven

Figure 64 illustrates how the e\*Way’s schedule-driven data exchange process works for incoming data, using the **Exchange Data with External Function, Positive Acknowledgment Function**, and **Negative Acknowledgment Function**.

**Figure 64** Schedule-Driven Data Exchange Process





*Start* can occur in any of the following ways:

- *Start Data Exchange* time occurs
- Periodically during data-exchange schedule (after *Start Data Exchange* time, but before *Stop Data Exchange* time), as set by **Exchange Data Interval**
- The **start-schedule** Monk function is called

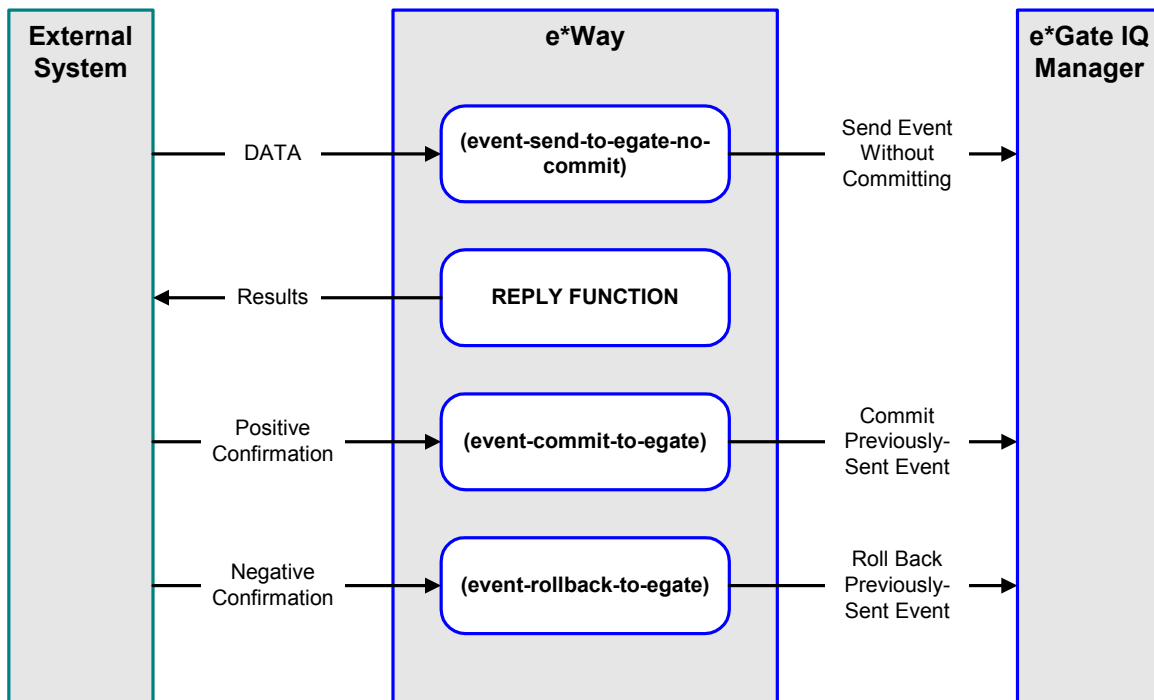
*Send Events to e\*Gate* can be implemented using any of the following Monk functions:

- **event-send-to-egate**
- **event-send-to-egate-ignore-shutdown**
- **event-send-to-egate-no-commit**

The last of these is used when confirmation of correct transmission is required from the external system. In this case, the e\*Way sends information back to the external system after receiving data. Depending upon whether the acknowledgment is positive or negative, you subsequently use one of the following functions to complete the process (see Figure 65):

- **event-commit-to-egate**
- **event-rollback-to-egate**

**Figure 65** Send Event to e\*Gate with Confirmation

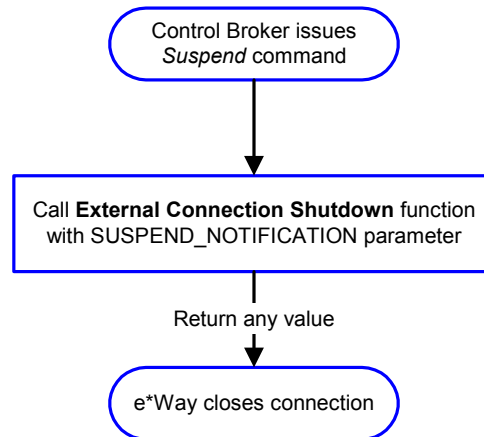


After the function exits, the e\*Way waits for the next *Start* time or command.

## Disconnect from External Process

Figure 66 illustrates how the e\*Way disconnects from the external system, using the **External Connection Shutdown Function**.

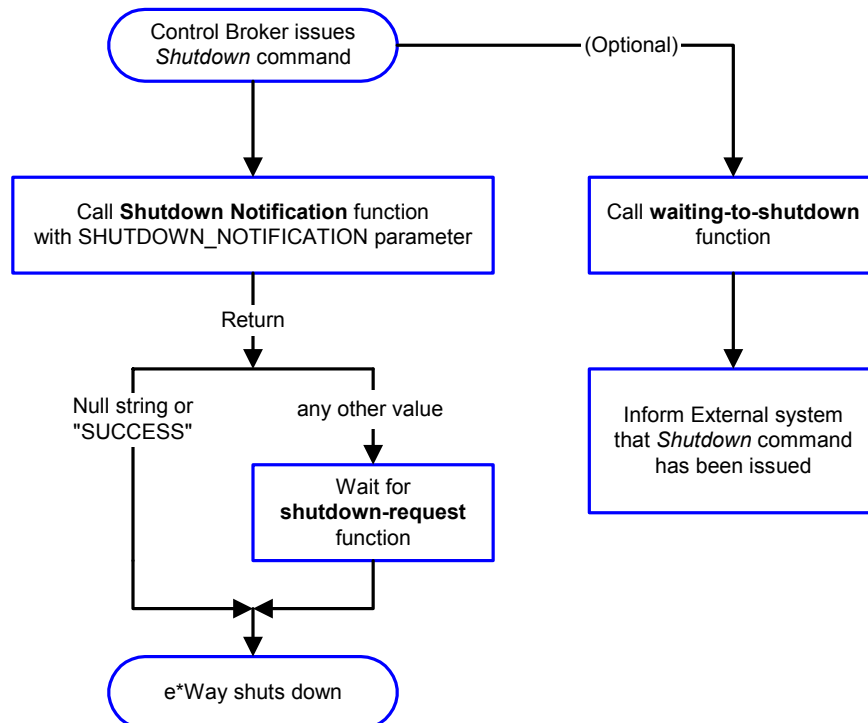
**Figure 66** Disconnect Process



## Shutdown Process

Figure 67 illustrates how the e\*Way shuts itself down, using the **Shutdown Command Notification Function**.

**Figure 67** Shutdown Process



# Configuration Parameters

This chapter describes the configuration parameters for the SAP BDC e\*Way.

---

## 7.1 Overview

The e\*Way's configuration parameters are set using the e\*Way Editor; see [Configuring the e\\*Way](#) on page 69 for procedural information. The default configuration is provided in `ewsapbdc.def`. The SAP BDC e\*Way's configuration parameters are organized into the following sections:

[General Settings](#) on page 100

[Communication Setup](#) on page 102

[Monk Configuration](#) on page 105

[SAP RFC Client Setup](#) on page 113

[SAP RFC Server Setup](#) on page 115

[FTP Setup for File-based Transfers](#) on page 117

---

## 7.2 General Settings

The General Settings control basic operational parameters.

---

### Journal File Name

#### Description

Specifies the name of the journal file.

#### Required Values

A valid filename, optionally including an absolute path (for example, **c:\temp\filename.txt**). If an absolute path is not specified, the file is stored in the **e\*Gate \SystemData\** directory. See the *e\*Gate Integrator System Administration and Operations Guide* for more information about file locations.

#### Additional Information

An Event is journaled for the following conditions:

- When the number of resends is exceeded (see **Max Resends Per Message** below)
  - When its receipt is due to an external error, but **Forward External Errors** is set to **No**
- 

### Max Resends Per Message

#### Description

Specifies the number of times the e\*Way attempts to resend a message (Event) to the external system after receiving an error. When this maximum is reached, the e\*Way waits for the number of seconds specified by the **Resend Timeout** parameter, and then rolls back the Event to its publishing IQ.

#### Required Values

An integer between 1 and 1,024. The default is 5.

---

### Max Failed Messages

#### Description

Specifies the maximum number of failed messages (Events) that the e\*Way allows. When the specified number of failed messages is reached, the e\*Way shuts down and exits.

#### Required Values

An integer between 1 and 1,024. The default is 3.

---

---

## Forward External Errors

### Description

Selects whether or not error messages received from the external system beginning with the string "DATAERR" are queued to the e\*Way's configured queue. See [Exchange Data with External Function](#) on page 108 for more information.

### Required Values

**Yes** or **No**. The default value, **No**, specifies that error messages are not to be forwarded. See [Data Exchange Process](#) on page 95 for more information about how the e\*Way uses this function.

---

## 7.3 Communication Setup

The Communication Setup parameters control the schedule by which the e\*Way obtains data from the external system.

**Note:** *The schedule you set using the e\*Way's properties in the e\*Gate Schema Designer controls when the e\*Way executable runs. The schedule that you set within the parameters discussed in this section (using the e\*Way Editor) determines when data are exchanged. Be sure you set the "exchange data" schedule to fall within the "run the executable" schedule.*

---

### Start Exchange Data Schedule

#### Description

Establishes the schedule to invoke the e\*Way's **Exchange Data with External Function**.

#### Required Values

One of the following:

- One or more specific dates/times
- A single repeating interval (such as yearly, weekly, monthly, daily, or every *n* seconds)

**Also required:** If you set a schedule using this parameter, you must also define all three of the following:

- **Exchange Data with External Function**
- **Positive Acknowledgment Function**
- **Negative Acknowledgment Function**

If you do not do so, the e\*Way terminates execution when the schedule attempts to start.

#### Additional Information

When the schedule starts, the e\*Way determines whether it is waiting to send an ACK or NAK to the external system (using the **Positive Acknowledgment Function** and **Negative Acknowledgment Function**) and whether or not the connection to the external system is active. If no ACK/NAK is pending and the connection is active, the e\*Way immediately executes the **Exchange Data with External Function**. Thereafter, the **Exchange Data with External Function** is called according to the **Exchange Data Interval** parameter until the **Stop Exchange Data Schedule** time is reached.

---

### Stop Exchange Data Schedule

#### Description

Establishes the schedule to stop data exchange.

### Required Values

One of the following:

- One or more specific dates/times
- A single repeating interval (such as yearly, weekly, monthly, daily, or every *n* seconds)

---

## Exchange Data Interval

### Description

Specifies the number of seconds the e\*Way waits between calls to the **Exchange Data with External Function** during scheduled data exchanges.

### Required Values

An integer between 0 and 86,400. The default is 120.

### Additional Information

If **Zero Wait Between Successful Exchanges** is set to **Yes** and the **Exchange Data with External Function** returns data, The **Exchange Data Interval** setting is ignored and the e\*Way invokes the **Exchange Data with External Function** immediately.

If this parameter is set to zero, then no exchange data schedule is set and the **Exchange Data with External Function** is never called.

See **Down Timeout** on page 103 and **Stop Exchange Data Schedule** on page 102 for more information about the data-exchange schedule.

---

## Down Timeout

### Description

Specifies the number of seconds that the e\*Way waits between calls to the **External Connection Establishment Function**.

### Required Values

An integer between 1 and 86,400. The default is 15.

---

## Up Timeout

### Description

Specifies the number of seconds the e\*Way waits between calls to the **External Connection Verification Function**.

### Required Values

An integer between 1 and 86,400. The default is 15.

---

## Resend Timeout

### Description

Specifies the number of seconds the e\*Way waits between attempts to resend a message (Event) to the external system, after receiving an error message from the external system.

### Required Values

An integer between 1 and 86,400. The default is 10.

---

## Zero Wait Between Successful Exchanges

### Description

Selects whether to initiate data exchange after the [Exchange Data Interval](#), or immediately after a successful previous exchange.

### Required Values

Yes or No. The default is No.

If this parameter is set to Yes, the e\*Way immediately invokes the [Exchange Data with External Function](#) if the previous exchange function returned data.

If this parameter is set to No, the e\*Way always waits the number of seconds specified by [Exchange Data Interval](#) between invocations of the [Exchange Data with External Function](#).



---

## 7.4 Monk Configuration

The parameters in this section help you set up the information required by the e\*Way to utilize Monk for communication with the external system.

### Specifying Function or File Names

Parameters that require the name of a Monk function accept either a function name (implied by the absence of a period <.>) or the name of a file (optionally including path information) containing a Monk function. If a file name is specified, the function invoked is given by the base name of the file (for example, for a file named **my-startup.monk**, the e\*Way would attempt to execute the function **my-startup**). If path information is specified, that path is appended to the **Load Path**.

If you specify a file name, be sure that the file has one of the following extensions:

- .monk
- .tsc
- .dsc

### Specifying Multiple Directories

To specify multiple directories, manually enter the directory names rather than selecting them with the **File Selection** button. Directory names must be separated with semicolons, and you can mix absolute paths with relative e\*Gate paths. For example:

```
monk_scripts\my_dir;c:\my_directory
```

The internal e\*Way function that loads this path information is called only once, when the e\*Way first starts up.

### Load Path

The Monk *load path* is the path Monk uses to locate files and data (set internally within Monk). The default load paths are determined by the **SharedExe** and **SystemData** settings in the **.egate.store** file. See the *e\*Gate Integrator System Administration and Operations Guide* for more information about this file.

---

## Additional Path

### Description

Specifies a path to be appended to the **Load Path**. A directory specified here is searched *after* searching the default load path.

### Required Values

A pathname, or a series of paths separated by semicolons. There is no default value for this parameter.

**Note:** *This parameter is optional and may be left blank.*

### Additional information

The internal e\*Way function that loads this path information is called only once, when the e\*Way first starts up.

---

## Auxiliary Library Directories

### Description

Specifies a path to auxiliary library directories. Any **.monk** files found within those directories is automatically loaded into the e\*Way's Monk environment.

### Required Values

A pathname, or a series of paths separated by semicolons. The default value is **monk\_library\ewsapbdc**.

*Note:* This parameter is optional and may be left blank.

---

## Monk Environment Initialization File

### Description

Specifies a file that contains environment initialization functions, which is loaded after the [Auxiliary Library Directories](#) are loaded.

### Required Values

A filename within the [Load Path](#), or filename plus path information (relative or absolute). If path information is specified, that path is appended to the load path. The default value is **monk\_library/ewsapbdc/sapbdc-init.monk**.

*Note:* This parameter is optional and may be left blank.

### Returns

The string **"FAILURE"** indicates that the function failed, and the e\*Way exits; any other string, including a *null string*, indicates success.

### Additional information

- Use this feature to initialize the e\*Way's Monk environment (for example, to define Monk variables that are used by the e\*Way's function scripts); it is good practice to initialize any global Monk variables that may be used by any other Monk Extension scripts
- The internal function that loads this file is called once when the e\*Way first starts up
- The e\*Way loads this file and try to invoke a function of the same base name as the file name

---

## Startup Function

### Description

Specifies a Monk function that the e\*Way loads and invokes upon startup or whenever the e\*Way's configuration is reloaded. It is called after the e\*Way loads the specified **Monk Environment Initialization File** and any files within the specified **Auxiliary Library Directories**. This function accepts no input, and must return a string.

This function should be used to initialize the external system before data exchange starts.

### Required Values

The name of a Monk function or the name of a file containing a Monk function.

*Note:* This parameter is optional and may be left blank.

### Returns

The string "FAILURE" indicates that the function failed, and the e\*Way exits; any other string (including a *null string*) indicates success.

---

## Process Outgoing Message Function

### Description

Specifies the Monk function responsible for sending outgoing messages (Events) from the e\*Way to the external system. This function is event-driven, rather than schedule-driven). The function requires a non-null string as input (i.e., the outgoing Event to be sent), and must return a string.

### Required Values

The name of a Monk function or the name of a file containing a Monk function.

*Note:* This parameter is **required**, and must **not** be left blank.

### Returns

- A *null string* ("") indicates that the Event was published successfully to the external system
- A string beginning with **RESEND** indicates that the Event should be resent
- A string beginning with **CONNERR** indicates that there is a problem with the connection to the external system, and causes a rollback of the Event
- A string beginning with **DATAERR** indicates that there is a problem with the message (Event) data itself, and causes a rollback of the Event
- A string beginning with **SHUTDOWN** indicates that the e\*Way must exit immediately

- If any string other than one of the preceding is returned, the e\*Way creates an entry in the log file indicating that an attempt has been made to access an unsupported function

#### Additional Information

- The e\*Way invokes this function when one of its Collaborations publishes an Event to an *external* destination (as specified within the e\*Gate Schema Designer).
- Once this function has been called with a *non-null string*, the e\*Way does not process another Event until the current Event has been completely processed.

**Note:** *If you wish to use **event-send-to-egate** to enqueue failed Events in a separate IQ, the e\*Way must have an inbound Collaboration (with appropriate IQs) configured to process those Events.*

---

## Exchange Data with External Function

### Description

Specifies a Monk function that initiates the transmission of data from the external system to the e\*Gate system and forwards that data as an inbound Event to one or more e\*Gate Collaborations. This function is invoked automatically by the **Down Timeout** or manually by the **start-schedule** Monk function, and is responsible for either sending data to or receiving data from the external system. If this function returns data, it is queued to e\*Gate in an inbound Collaboration. The e\*Way must have at least one Collaboration configured suitably to process the inbound Event, as well as any required IQs.

### Required Values

The name of a Monk function or the name of a file containing a Monk function.

**Note:** *This parameter is optional and may be left blank.*

### Returns

- A *null string* ("" ) indicates that the data exchange was completed successfully, but with no resultant data sent back to the e\*Gate system
- A string beginning with **CONNERR** indicates that there is a problem with the connection to the external system
- A string beginning with **DATAERR** indicates that there is a problem with the message (Event) data itself. If the error string contains data beyond the keyword, the entire string is queued to e\*Gate if an inbound Collaboration is so configured and **Forward External Errors** is set to **Yes**. Queueing, however, is performed without the subsequent sending of a **ACK** or **NAK** to the external system.
- Any other string indicates that the contents of the string are packaged as an inbound Event

### Additional Information

- Data can be queued directly to e\*Gate by using the **event-send-to-egate** Monk function or, if a two-phase approach is required, by using **event-send-to-egate-no-commit** and then **event-commit-to-egate** or **event-rollback-to-egate** to commit or rollback the enqueued events, as appropriate

*Note:* Until an Event is committed, it is not revealed to subscribers of that Event.

---

## External Connection Establishment Function

### Description

Specifies a Monk function that the e\*Way calls (repeatedly) when it has determined that the connection to the external system is down. The function accepts no input and must return a string.

This function is executed according to the interval specified within the **Down Timeout** parameter, and is called *only* according to this schedule. Once the e\*Way has determined that its connection to the external system is up, it calls the **External Connection Verification Function** (see next).

### Required Values

The name of a Monk function or the name of a file containing a Monk function.

*Note:* This parameter is **required**, and must **not** be left blank.

### Returns

- A string beginning with **SUCCESS** or **UP** indicates that the connection was established successfully
- A string beginning with **DOWN** indicates that the connection was not established successfully
- Any other string, including a *null string*, indicates that the attempt to establish the connection failed and the external state is unknown

---

## External Connection Verification Function

### Description

Specifies a Monk function that the e\*Way calls when its internal variables show that the connection to the external system is up. It is executed according to the interval specified within the **Up Timeout** parameter, and is called *only* according to this schedule.

### Required Values

The name of a Monk function or the name of a file containing a Monk function.

*Note:* This parameter is **optional** and may be left blank.

### Returns

- “SUCCESS” or “UP” indicates that the connection was established successfully
- Any other string (including the null string) indicates that the attempt to establish the connection failed

### Additional Information

If this function is not specified, the e\*Way executes the [External Connection Establishment Function](#) in its place. This latter function also is called when the e\*Way has determined that its connection to the external system is down.

---

## External Connection Shutdown Function

### Description

Specifies a Monk function that the e\*Way calls to shut down the connection to the external system. This function is invoked only when the e\*Way receives a *suspend* command from a Control Broker.

### Required Values

The name of a Monk function or the name of a file containing a Monk function.

*Note:* This parameter is optional and may be left blank.

### Input

A string indicating the purpose for shutting down the connection.

- “SUSPEND\_NOTIFICATION” - the e\*Way is being suspended or shut down
- “RELOAD\_NOTIFICATION” - the e\*Way is being reconfigured

### Returns

A string, the value of which is ignored. Any return value indicates that the *suspend* command can proceed and that the connection to the external system can be broken immediately.

*Note:* Include in this function any required “clean up” operations that must be performed as part of the shutdown procedure, but before the e\*Way exits.

---

## Positive Acknowledgment Function

### Description

This function is loaded during the initialization process and is called when all data received from the external system has been processed and enqueued successfully.

### Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is [sapbdc-ack](#).

**Note:** This parameter is *conditional* and must be supplied only if the **Exchange Data with External Function** is set to a non-zero value.

### Required Input

A string, the inbound Event to e\*Gate.

### Returns

- The string beginning with **CONNERR** indicates a problem with the connection to the external system; when the connection is re-established, the function is called again, with the same input data
- Any other string, including a *null string*, indicates that the acknowledgement has been sent to the external system successfully

### Additional Information

- After the **Exchange Data with External Function** returns a string that is transformed into an inbound Event, the Event is handed off to one or more Collaborations for further processing. The e\*Way executes this function only if the Event's processing is completed successfully by *all* the Collaborations to which it was sent; otherwise, the e\*Way executes the **Negative Acknowledgment Function**.
- This function can return data to be queued, but the e\*Way will *not* acknowledge the data with an **ACK** or **NAK**.

**Note:** If you configure the acknowledgment function to return a non-null string, you must configure a Collaboration (with appropriate IQs) to process the returned Event.

---

## Negative Acknowledgment Function

### Description

This function is loaded during the initialization process and is called when the e\*Way fails to process or enqueue data received from the external system successfully.

### Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is **sapbdc-nack**.

**Note:** This parameter is *conditional* and must be supplied only if the **Exchange Data with External Function** is set to a non-zero value.

### Required Input

A string, the inbound Event to e\*Gate.

### Returns

- The string beginning with **CONNERR** indicates a problem with the connection to the external system; when the connection is re-established, the function is called again, using the same input data

- Any other string, including a *null string*, indicates that the acknowledgement has been sent to the external system successfully

### Additional Information

- This function is called only during the processing of inbound Events. After the [Exchange Data with External Function](#) returns a string that is transformed into an inbound Event, the Event is handed off to one or more Collaborations for further processing. The e\*Way executes this function if the Event's processing is not completed successfully by *all* the Collaborations to which it was sent; otherwise, the e\*Way executes the [Positive Acknowledgment Function](#).
- This function can return data to be queued, but the e\*Way will *not* acknowledge the data with an ACK or NAK.

**Note:** *If you configure the acknowledgment function to return a non-null string, you must configure a Collaboration (with appropriate IQs) to process the returned Event.*

---

## Shutdown Command Notification Function

### Description

The e\*Way calls this Monk function automatically to notify the external system that it is about to shut down. This function also can be used to shut down the connection with the external. The function accepts a string as input and must return a string.

### Required Values

The name of a Monk function or the name of a file containing a Monk function. There is no default value for this parameter.

**Note:** *This parameter is optional and may be left blank.*

### Input

When the Control Broker issues a shutdown command to the e\*Way, the e\*Way calls this function with the string "SHUTDOWN\_NOTIFICATION" passed as a parameter.

### Returns

- A *null string* or "SUCCESS" indicates that the shutdown can occur immediately
- Any other string indicates that shutdown must be postponed; once postponed, shutdown does not proceed until the Monk function [shutdown-request](#) is executed

### Additional Information

If you postpone a shutdown using this function, be sure to use the [shutdown-request](#) function to complete the process in a timely manner.



---

## 7.5 SAP RFC Client Setup

The parameters in this section control the setup of the SAP RFC client.

---

### Host Name of the R/3 Target System

#### Description

Specifies the host name of the R/3 target system.

#### Required Values

A valid host name.

---

### System Number of the R/3 Target System

#### Description

Specifies the system number of the R/3 target system.

#### Required Values

A valid system number.

---

### Client

#### Description

Specifies the name of the client.

#### Required Values

A valid client name.

---

### User

#### Description

Specifies the name of a user on the client system.

#### Required Values

A valid user name.

---

### Password

#### Description

Specifies the password corresponding to the specified user name.

## Required Values

A valid password

---

## Language

### Description

Specifies the language of the SAP system.

### Required Values

DE (German) or EN (English). The default is EN.

---

## Enable RFC Trace

### Description

Activates or deactivates the SAP RFC Trace feature. The trace files are created in the directory `eGate\client`, and have the format `rfc<number>.trc`.

### Required Values

On or Off. The default is Off.

---

## Log File for Failed Records

### Description

Specifies the name of the file in which records that could not be processed are stored.

### Required Values

The name of a file, including absolute path information.

---

## Maximum Number of Failure Before Erroring Out

### Description

Specifies the number of failures after which the e\*Way shuts down.

### Required Values

An integer between **1** and **65,535**. The default is **1**.

---

## 7.6 SAP RFC Server Setup

The parameters in this section describe the SAP RFC Server.

---

### Gateway Host Name

#### Description

Specifies the host name of the gateway.

#### Required Values

A valid host name.

---

### Gateway Service

#### Description

Specifies the name of the gateway service.

#### Required Values

A valid service name.

---

### Program ID

#### Description

Specifies the Program ID.

#### Required Values

A valid Program ID

---

### Wait for Request Interval

#### Description

Determines the interval to wait for requests.

#### Required Values

An integer between 1 and 65,535 and an interval. The default is 5 seconds.

---

### Trace

#### Description

Determines whether or not the trace feature is activated.

### Required Values

Off or On. The default is Off.

---

## Log File for Failed Records

### Description

Specifies the name of the file in which records that could not be processed are stored.

### Required Values

The name of a file, including absolute path information.

---

## Maximum Number of Failure Before Erroring Out

### Description

Specifies the number of failures after which the e\*Way shuts down.

### Required Values

An integer between 1 and 65,535. The default is 1.

---

## 7.7 FTP Setup for File-based Transfers

The parameters in this section control the configuration settings used for data transfers that use FTP.

---

### User

#### Description

Specifies the user name on the remote FTP server under which to log in.

#### Required Values

A valid user name.

---

### Password

#### Description

Specifies the password corresponding to the user name specified in the above parameters.

#### Required Values

A valid password.

---

### File Transfer Mode

#### Description

Selects whether files are transferred in ASCII or Binary mode.

#### Required Values

ASCII or Binary. The default is Binary.

---

### Temporary File on Local System

#### Description

Specifies the name of the temporary file containing data to be sent via FTP.

#### Required Values

A valid file name, containing an absolute path.

#### Additional Information

For a RFC client in the file-based transfer mode, messages are accumulated in a local file, which is then sent via FTP to the SAP Application Server and stored as the *temporary file* before the file name is passed to the ABAP module through RFC. The ABAP module reads data from the temporary file on the SAP Application Server.

For a RFC server in the file-based transfer mode, the calling ABAP module passes to the e\*Way (through RFC) the name of a file, which actually exists on the SAP Application Server. The e\*Way fetches the named file via FTP to the local host and stores it as the *temporary file*. The e\*Way then reads data from this file.

---

## Temporary File on SAP Application Server

### Description

Specifies the name of the temporary file on the SAP Application Server in which data are sent via FTP.

**Note:** This parameter is *not* used for an RFC server.

### Required Values

A valid file name, containing an absolute path on the SAP Application Server.

---

## Remote FTP Server Root Directory

### Description:

Specifies the root directory of the remote FTP server, when on a Windows host.

### Required Values:

A valid path name (e.g., F:\Users\Ftp). Either forward or back-slashes are acceptable.

**Note:** This parameter is required *only* if your FTP sever is hosted on Windows. If the server is hosted on UNIX, leave the value blank.

### Additional Information

When the e\*Way fetches a file from the remote server via FTP, the fetch is performed relative to the FTP root directory.

# API Functions

## 8.1 Overview

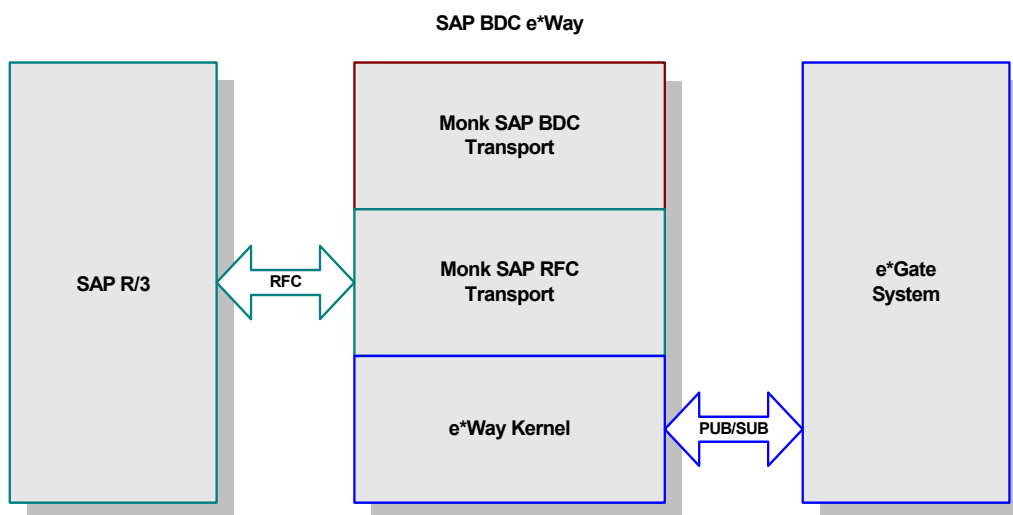
The SAP BDC e\*Way has been designed specifically to connect e\*Gate to SAP enterprise-management software within a network of diverse hardware and software systems. Using one or more SAP e\*Ways, e\*Gate can act as a bus connecting SAP applications and other software systems, or differently-configured SAP systems. This e\*Way allows data exchange between e\*Gate and an SAP system, in either SAP-inbound or SAP-outbound direction, via BDC (Batch Data Communications).

Conceptually, the e\*Way can be viewed as a three-layered structure, consisting of:

- BDC Transport Layer
- RFC Transport Layer
- e\*Way Kernel Layer

Each layer contains Monk scripts and/or functions, and makes use of lower-level Monk functions residing in the layer beneath. Developers primarily make use of the highest-level functions, which reside in the SAP BDC Transport Layer.

**Figure 68** SAP BDC e\*Way Architecture



## 8.1.1 BDC Transport Layer

The Monk functions in this group provide a high-level interface for the user to interact with the target SAP system specified in the configuration. BDC Transport-layer Monk functions are divided into the following categories:

[SAP BDC Template Functions](#) on page 122

[SAP BDC Utility Functions](#) on page 127

## 8.1.2 RFC Transport Layer

This layer provides direct RFC access to SAP R/3. Many of the SAP functions have been abstracted and encapsulated into functions available in the Monk SAP BDC Transport Layer. Generally, Monk RFC functions should be called directly only when there is a specific need not addressed by the BDC-Layer Monk functions.

RFC Transport-layer Monk functions are divided into the following categories:

[SAP RFC Client Functions](#) on page 132

[SAP RFC Server Functions](#) on page 136

[SAP Client Connection Functions](#) on page 145

[SAP Custom Structure Functions](#) on page 150

[SAP Table List Functions](#) on page 153

[SAP Parameter List Functions](#) on page 159

[SAP Type-Checking Functions](#) on page 165

## 8.1.3 e\*Way Kernel Layer

This layer implements communication between the e\*Way and e\*Gate, and also with the external system (SAP). Generally, e\*Way Kernel Monk functions should be called directly only when there is a specific need not addressed by higher-level Monk functions, and should be used only by an experienced developer.

e\*Way Kernel Monk functions are listed as

[Generic e\\*Way Functions](#) on page 168



---

## 8.2 SAP BDC Data Types

Many of the SAP BDC Monk functions reference the following data types:

### saprfc-conn-handle

**SAP Connection Handle.** A SAP Connection Handle is used to identify different connections to the SAP Application Server. Whether connected to the SAP Application Server as a RFC client or as a RFC server, a Connection Handle is required for many of the functions in the SAP RFC interface.

### saprfc-conn-opt

**SAP Client Connection Option Handle.** A client connection to the SAP Application Server takes many parameters and there are several scenarios of connecting as a client. This handle helps build the connection options. A SAP Client Connection Option Handle is supplied to the client connection function [saprfc-client-connect](#).

### saprfc-par-list

**SAP Parameter List.** In an SAP Remote Function Call (RFC), input and output data are passed through named parameter lists and named table lists (see below). When calling a remote function, the caller always specifies a list of exporting parameters, a list of importing parameters, and a list of internal tables used to pass structured data to and from the remote function. Exporting parameters are those parameters whose values are supplied to the called function by the calling function. Importing parameters are those parameters whose values are to be returned from the called function to the calling function.

### saprfc-status

Many of the Monk functions provided by the SAP BDC e\*Way return a status code, which could be any of the following values (a status code that is not listed means failure):

- **SAPRFC\_OK:** The function succeeded.
- **SAPRFC\_TIMEOUT:** The function timed out. Only applies to [saprfc-server-waitanddispatch](#).
- **SAPRFC\_CLOSED:** The SAP RFC connection has been closed by the other end. Only applies to [saprfc-server-waitanddispatch](#).
- **SAPRFC\_FAILED:** The function failed.

### saprfc-tab-list

**SAP Table List.** List of named tables are used in a Remote Function Call to pass structured data to and from the remote function.

## 8.3 BDC Transport Layer

### 8.3.1 SAP BDC Template Functions

The following functions are provided as templates to enable you to create your own basic functions (startup, shutdown, connection, and so on) with site-specific information. These functions include:

[sapbdc-fromsap-startup](#) on page 122

[sapbdc-fromsap-connect](#) on page 123

[sapbdc-fromsap-finish](#) on page 123

[sapbdc-tosap-startup](#) on page 124

[sapbdc-tosap-connect](#) on page 124

[sapbdc-tosap-finish](#) on page 125

[sapbdc-ack](#) on page 125

[sapbdc-nack](#) on page 126

---

#### sapbdc-fromsap-startup

##### Description

Provides a template for a *startup* function executed from an e\*Way that acts as a server to SAP clients.

##### Signature

(sapbdc-fromsap-startup *string*)

##### Parameters

| Name   | Type   | Description |
|--------|--------|-------------|
| string | string | A string.   |

##### Returns

A user-defined string.

##### Throws

None.

##### Location

sapbdc.monk

---

## sapbdc-fromsap-connect

### Description

Provides a template for a *connect* function from an e\*Way that acts as a server to SAP clients.

### Signature

(sapbdc-fromsap-connect *string*)

### Parameters

| Name   | Type   | Description |
|--------|--------|-------------|
| string | string | A string.   |

### Returns

A user-defined string.

### Throws

None.

### Location

sapbdc.monk

---

## sapbdc-fromsap-finish

### Description

Provides a template for a *transaction concluded* function from an e\*Way that acts as a server to SAP clients.

### Signature

(sapbdc-fromsap-finish *string*)

### Parameters

| Name   | Type   | Description |
|--------|--------|-------------|
| string | string | A string.   |

### Returns

A user-defined string.

### Throws

None.

### Location

sapbdc.monk

---

## sapbdc-tosap-startup

### Description

Provides a template for a *startup* function executed from an e\*Way that acts as a client for an SAP server.

### Signature

(sapbdc-tosap-startup *string*)

### Parameters

| Name   | Type   | Description |
|--------|--------|-------------|
| string | string | A string.   |

### Returns

A user-defined string.

### Throws

None.

### Location

sapbdc.monk

---

## sapbdc-tosap-connect

### Description

Provides a template for a *connect* function executed from an e\*Way that acts as a client for an SAP server.

### Signature

(sapbdc-tosap-connect *string*)

### Parameters

| Name   | Type   | Description |
|--------|--------|-------------|
| string | string | A string.   |

### Returns

A user-defined string.

### Throws

None.

### Location

sapbdc.monk

---

## sapbdc-tosap-finish

### Description

Provides a template for a *transaction concluded* function from an e\*Way that acts as a client for an SAP server.

### Signature

(sapbdc-tosap-finish *string*)

### Parameters

| Name   | Type   | Description |
|--------|--------|-------------|
| string | string | A string.   |

### Returns

A user-defined string.

### Throws

None.

### Location

sapbdc.monk

---

## sapbdc-ack

### Description

Provides a template for a *positive acknowledgement* function.

### Signature

(sapbdc-ack *string*)

### Parameters

| Name   | Type   | Description |
|--------|--------|-------------|
| string | string | A string.   |

### Returns

A user-defined string.

### Throws

None.

### Location

sapbdc.monk

---

## sapbdc-nack

### Description

Provides a template for a *negative acknowledgement* function.

### Signature

(sapbdc-nack *string*)

### Parameters

| Name   | Type   | Description |
|--------|--------|-------------|
| string | string | A string.   |

### Returns

A user-defined string.

### Throws

None.

### Location

sapbdc.monk

## 8.3.2 SAP BDC Utility Functions

The SAP BDC Utility functions provide a high-level interface for the user to interact with the target SAP system specified in the configuration. These functions include:

- [sapbdc-client-connect](#) on page 127
- [sapbdc-client-struct-send](#) on page 127
- [sapbdc-client-disconnect](#) on page 128
- [sapbdc-client-geterrormessage](#) on page 128
- [sapbdc-init](#) on page 129
- [sapbdc-server-shutdown](#) on page 129
- [sapbdc-server-startup](#) on page 130
- [sapbdc-server-struct-fetch](#) on page 130

---

### sapbdc-client-connect

#### Description

Establishes a client connection to the target SAP Application Server using the parameters specified in the SAP BDC e\*Way configuration. It also creates parameters lists and tables lists that are necessary for the operation of the SAP BDC e\*Way.

#### Signature

(sapbdc-client-connect)

#### Parameters

None.

#### Returns

An SAP RFC Status code (see [saprfc-status](#) on page 121).

#### Throws

None.

#### Location

sapbdc-client-connect.monk

---

### sapbdc-client-struct-send

#### Description

Sends to the target SAP system the data contained in the specified Event Type Definition or a node within an ETD. The data is sent over either in a message-based mode or in a file-based mode, depending on the configuration.

#### Signature

(sapbdc-client-struct-send *ETD\_info*)

### Parameters

| Name     | Type                 | Description                                  |
|----------|----------------------|--|
| ETD_info | ETD or ETD-node path | An ETD, or the name of a node within an ETD. |

### Returns

An SAP RFC Status code (see [saprfc-status](#) on page 121).

### Throws

None.

### Location

sapbdc-client-struct-send.monk

---

## sapbdc-client-disconnect

### Description

Closes the client connection to the target SAP Application Server specified in the SAP BDC e\*Way configuration.

### Signature

(sapbdc-client-disconnect)

### Parameters

None.

### Returns

An SAP RFC Status code (see [saprfc-status](#) on page 121).

### Throws

None.

### Location

sapbdc-client-disconnect.monk

---

## sapbdc-client-geterrormessage

### Description

Returns an error message string, which is returned by the ABAP function (for the SAP BDC e\*Way) when it fails.

### Signature

(sapbdc-client-geterrormessage)

### Parameters

None.



### Returns

An error message string.

### Throws

None.

### Location

sapbdc-client-geterrormessage.monk

---

## sapbdc-init

### Description

Sets up the SAP BDC Monk environment.

### Signature

(sapbdc-init)

### Parameters

None.

### Returns

An SAP RFC Status code (see [saprfc-status](#) on page 121).

### Throws

None.

### Location

sapbdc-init.monk

---

## sapbdc-server-shutdown

### Description

Closes the client connection to the target SAP Application Server specified in the SAP BDC e\*Way configuration.

### Signature

(sapbdc-server-shutdown)

### Parameters

None.

### Returns

An SAP RFC Status code (see [saprfc-status](#) on page 121).

### Throws

None.

**Location**

sapbdc-server-shutdown.monk

---

## sapbdc-server-startup

**Description**

Establishes a connection to the target SAP system and installs internal service modules used by the SAP BDC e\*Way. It also creates parameters lists and tables lists that are necessary for the SAP BDC e\*Way to act as an SAP RFC server.

**Signature**

(sapbdc-server-startup)

**Parameters**

None.

**Returns**

An SAP RFC Status code (see [saprfc-status](#) on page 121).

**Throws**

None.

**Location**

sapbdc-server-startup.monk

---

## sapbdc-server-struct-fetch

**Description**

Waits for RFC request from the ABAP function for the SAP BDC e\*Way until it receives a request or the waiting period specified in the e\*Way configuration has passed. If a RFC request is received, this function reads data sent over by the ABAP function and populates the specified message structure.

**Signature**

(sapbdc-client-struct-fetch *structure*)

**Parameters**

| Name      | Type      | Description          |
|-----------|-----------|----------------------|
| structure | structure | A message structure. |

**Returns**

An SAP RFC Status code (see [saprfc-status](#) on page 121).

**Throws**

None.

## Location

sapbdc-server-struct-fetch.monk

## 8.4 RFC Transport Layer

### 8.4.1 SAP RFC Client Functions

The SAP RFC Client functions manipulate connections to the SAP client. These functions include:

- [saprfc-client-connect](#) on page 132
- [saprfc-client-callreceive](#) on page 132
- [saprfc-client-createtid](#) on page 133
- [saprfc-client-indirectcall](#) on page 134
- [saprfc-client-disconnect](#) on page 134

---

#### saprfc-client-connect

##### Description

Establishes a client connection to the SAP Application Server specified in the SAP Client Connection Option, with the connection parameters also specified in the same option handle. The returned handle should be checked with [saprfc-conn-handle?](#).

##### Signature

(saprfc-client-connect *option\_handle*)

##### Parameters

| Name          | Type            | Description                              |
|---------------|-----------------|--|
| option_handle | saprfc-conn-opt | A valid SAP RFC Connection Option handle |

##### Returns

An SAP RFC Connection Option handle.

##### Throws

None.

##### Location

stc\_monksap.dll

---

#### saprfc-client-callreceive

##### Description

Calls a remote function through RFC.

### Signature

```
(saprfc-client-callreceive conn_handle tablist import_params
 export_params remote_fn)
```

### Parameters

| Name          | Type            | Description                               |
|---------------|-----------------|---|
| conn_handle   | saprfc-conn-opt | A valid SAP RFC Connection Option handle. |
| tablist       | saprfc-tab-list | A valid SAP table list.                   |
| import_params | saprfc-par-list | A valid SAP parameter list.               |
| export_params | saprfc-par-list | A valid SAP parameter list.               |
| remote_fm     | string          | The remote function to run.               |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

### Location

stc\_monksap.dll

## saprfc-client-createtid

### Description

Calls the target SAP system to create a valid SAP Transaction ID. A Transaction ID is used by the client to guarantee that each transaction is executed once and only once.

### Signature

```
(saprfc-client-createtid conn_handle)
```

### Parameters

| Name        | Type               | Description                               |
|-------------|--------------------|---|
| conn_handle | saprfc-conn-handle | A valid SAP RFC Connection Option handle. |

### Returns

If successful, a Monk string representing a valid Transaction ID created by the target SAP system. It returns an empty string ("" ) on failure. The reason of failure can be accessed via `saprfc-getlasterror`.

### Throws

None.

### Location

`stc_monksap.dll`

## saprfc-client-indirectcall

### Description

Calls a remote function through Transactional RFC.

### Signature

```
(saprfc-client-indirectcall conn_handle tablist export_params
  trans_id remote_fn)
```

### Parameters

| Name          | Type               | Description  |
|---------------|--------------------|--|
| conn_handle   | saprfc-conn-handle | A valid SAP RFC Connection Option handle.              |
| tablist       | saprfc-tab-list    | A valid SAP table list.                                |
| export_params | saprfc-par-list    | A valid SAP parameter list.                            |
| Trans_id      | String             | A valid SAP transaction ID.                            |
| remote_fn     | String             | The remote function to call through Transactional RFC. |

### Returns

If successful, the status code `SAPRFC_OK`; otherwise on failure. The reason for failure can be accessed via `saprfc-getlasterror`.

### Throws

None.

### Location

`stc_monksap.dll`

## saprfc-client-disconnect

### Description

Closes the connection to the SAP Application Server.

### Signature

(saprfc-client-disconnect *option\_handle*)

### Parameters

| Name          | Type            | Description                              |
|---------------|-----------------|--|
| option_handle | saprfc-conn-opt | A valid SAP RFC Connection Option handle |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

### Location

stc\_monksap.dll

## 8.4.2 SAP RFC Server Functions

The SAP RFC Server functions control the SAP Server. These functions include:

- [saprfc-server-startup](#) on page 136
- [saprfc-server-installfunction](#) on page 137
- [saprfc-server-installtransctrl](#) on page 137
- [saprfc-server-getinputdata](#) on page 139
- [saprfc-server-getcallbackfailuretid](#) on page 140
- [saprfc-server-getcallbackfailuretype](#) on page 140
- [saprfc-server-resetcallbackfailure](#) on page 141
- [saprfc-server-sendoutputdata](#) on page 142
- [saprfc-server-waitanddispatch](#) on page 142
- [saprfc-server-shutdown](#) on page 143

### saprfc-server-startup

#### Description

Establishes a connection to the specified SAP Application Server and registers the calling function as an RFC server with the specified program ID. The returned handle should be checked with [saprfc-conn-handle?](#).

#### Signature

*(saprfc-server-startup progID gateway\_host gateway\_service trace)*

#### Parameters

| Name            | Type    | Description   |
|-----------------|---------|---|
| progID          | string  | A program ID.   |
| gateway_host    | string  | A gateway host name.                                  |
| gateway_service | string  | A gateway service name.                               |
| trace           | integer | 1 to enable RFC tracing, or 0 to disable RFC tracing. |

#### Returns

An SAP RFC Connection Option handle.

#### Throws

None.

#### Location

stc\_monksap.dll



## saprfc-server-installfunction

### Description

Installs a callback (Monk) function as an RFC service module.

### Signature

`(saprfc-server-installfunction SAP_funciton Monk_function)`

### Parameters

| Name          | Type    | Description   |
|---------------|---------|---|
| SAP_function  | string  | The name of a function known to SAP (and thus available to any RFC caller).   |
| Monk_function | integer | The name of a Monk function to be used as the service module when this function is called via RFC. The Monk function can be either a Collaboration script (.tsc) function or a DART poll function. This function must return one of the following strings: <ul style="list-style-type: none"> <li>▪ SUCCESS</li> <li>▪ FAILURE</li> <li>▪ CLOSED (if the connection is closed by the RFC caller)</li> </ul> |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

### Location

stc\_monksap.dll

## saprfc-server-installtransctrl

### Description

Registers a set of callback routines (Monk functions) to handle transaction control.

## Signature

```
(saprfc-server-installtransctrl check_tid_monk_funcname
  commit_monk_funcname rollback_tid_monk_funcname
  confirm_tid_monk_funcname)
```

## Parameters

| Name                    | Type   | Description   |
|-------------------------|--------|---|
| check_tid_monk_funcname | String | Name of the Monk function responsible for validating SAP Transaction IDs. The Monk function is called when a local transaction is starting. It is passed a Monk string representing an SAP transaction ID, which is to be checked by this function. Since a transactional RFC call can be issued many times by the client system, this function is responsible for storing the transaction-ID in permanent storage. If the client system tries starting the same transaction a second time, this function has to return a Monk string "SKIP". It should return "STORED" if the transaction-ID is stored and the transaction can be started. It should return "FAILURE" if it cannot lock the transaction or has encountered any other internal error. |
| commit_monk_funcname    | String | Name of the Monk function responsible for the commit operation in the local database. The function is called when a local transaction ends. The function is to be used to commit the local transaction, if necessary. This function is passed a Monk string representing the SAP Transaction ID and should return a Monk string "SUCCESS" if it succeeds or "FAILURE" if it fails.  |
| rollback_monk_funcname  | String | Name of the Monk function responsible for the rollback operation in the local database. The function is called when a local transaction ends with failure. The function is to be used to roll back the local transaction, if necessary. This function is passed a Monk string representing the SAP Transaction ID and should return a Monk string "SUCCESS" if it succeeds or "FAILURE" if it fails.  |

| Name                      | Type   | Description  |
|---------------------------|--------|--|
| confirm_tid_monk_funcname | String | Name of the Monk function responsible for confirming SAP Transaction IDs. The function is called when a local transaction is completed. All informations stored about that transaction can be discarded by the server. In general, this function can be used to delete the transaction-ID from permanent storage. This function is passed a Monk string representing the SAP Transaction ID and should return a Monk string "SUCCESS" if it succeeds or "FAILURE" if it fails. |

#### Returns

The status code `SAPRFC_OK` on success and otherwise on failure.

#### Throws

None.

#### Location

`stc_monksap.dll`

## saprfc-server-getinputdata

#### Description

Used inside an RFC service module (Monk function); imports the parameters list and tables list from the caller.

#### Signature

```
(saprfc-server-getinputdata parlist tablist)
```

#### Parameters

| Name    | Type            | Description                 |
|---------|-----------------|-----------------------------|
| parlist | saprfc-par-list | A valid SAP Parameter List. |
| tablist | saprfc-tab-list | A valid SAP Table List      |

#### Returns

A status code that evaluates to one of the following:

|                             |   |
|-----------------------------|---|
| <code>SAPRFC_OK</code>      | Operations have concluded normally.                       |
| <code>SAPRFC_CLOSED</code>  | The installed service module returns the string "CLOSED". |
| <code>SAPRFC_TIMEOUT</code> | The function has timed out.                               |
| <code>SAPRFC_FAILURE</code> | The function has failed.                                  |

### Throws

None.

### Location

stc\_monksap.dll

---

## saprfc-server-getcallbackfailuretid

### Description

Returns the transaction ID on which the callback Monk function has failed. If the **Commit Monk function**, **Rollback Monk function**, or **Confirm TID Monk function** installed by [saprfc-server-installtransctrl](#) returns “FAILURE”, the return status is passed to the subsequent call to [saprfc-server-waitanddispatch](#) and the latter function returns SAPRFC\_FAILURE.

### Signature

```
(saprfc-server-getcallbackfailuretid)
```

### Parameters

None.

### Returns

The transaction ID on which the callback routine has failed.

### Throws

None.

### Location

stc\_monksap.dll

---

## saprfc-server-getcallbackfailuretype

### Description

Returns the type of callback routine that failed. If the **Commit Monk function**, the **Rollback Monk function**, or the **Confirm TID Monk function** installed by [saprfc-server-installtransctrl](#) returns “FAILURE”, the return status is passed to the subsequent call to [saprfc-server-waitanddispatch](#) and the latter function returns SAPRFC\_FAILURE.

### Signature

```
(saprfc-server-getcallbackfailuretype)
```

### Parameters

None.

### Returns

A number indicating the latest failed callback routine.

|                                  |   |
|----------------------------------|---|
| SAPRFC_CALLBACK_NOFAILURE        | No callback failure since last reset.     |
| SAPRFC_CALLBACK_CONFIRMIDFAILURE | The Confirm TID Monk function has failed. |
| SAPRFC_CALLBACK_COMMITFAILURE    | The Commit Monk function has failed.      |
| SAPRFC_CALLBACK_ROLLBACKFAILURE  | The Rollback Monk function has failed.    |

#### Throws

None.

#### Location

stc\_monksap.dll

---

## saprfc-server-resetcallbackfailure

### Description

Resets the error flag set by one of the TRFC callback routines. If the **Commit Monk** function, the **Rollback Monk** function, or the **Confirm TID Monk** function installed by [saprfc-server-installtransctrl](#) returns “FAILURE”, the return status is passed to the subsequent call to [saprfc-server-waitanddispatch](#) and the latter function returns `SAPRFC_FAILURE`.

In this case, the problem should be handled and this function must be called before [saprfc-server-waitanddispatch](#) can be used again.

### Signature

(saprfc-server-resetcallbackfailure)

### Parameters

None.

### Returns

Undefined.

### Throws

None.

### Location

stc\_monksap.dll

## saprfc-server-sendoutputdata

### Description

Used inside an RFC service module (Monk function); exports the parameters list and tables list to the caller.

### Signature

`(saprfc-server-sendoutputdata parlist tablist)`

### Parameters

| Name    | Type            | Description                 |
|---------|-----------------|-----------------------------|
| parlist | saprfc-par-list | A valid SAP Parameter List. |
| tablist | saprfc-tab-list | A valid SAP Table List      |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

### Location

stc\_monksap.dll

## saprfc-server-waitanddispatch

### Description

Waits for an RFC request until one is received or the specified timeout period has passed. If an RFC request is received, this function dispatches the requested service module (a Monk function) installed with [saprfc-server-installfunction](#).

### Signature

`(saprfc-server-waitanddispatch conn_handle wait)`

### Parameters

| Name        | Type            | Description                              |
|-------------|-----------------|--|
| conn_handle | saprfc-conn-opt | A valid SAP RFC Connection Option handle |

| Name | Type    | Description                    |
|------|---------|--------------------------------|
| wait | integer | The number of seconds to wait. |

### Returns

A status code that evaluates to one of the following:

|                        |   |
|------------------------|---|
| SAPRFC_OK              | Operations have concluded normally.                       |
| SAPRFC_CLOSED          | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT         | The function has timed out.                               |
| SAPRFC_FAILURE         | The function has failed.                                  |
| SAPRFC_CALLBACKFAILURE | One of the TRFC callback Monk functions failed.           |

### Throws

None.

### Location

stc\_monksap.dll

## saprfc-server-shutdown

### Description

Closes the connection to SAP Application Server. The connection handle should not be used any further after this function call.

### Signature

(saprfc-server-shutdown *conn\_handle*)

### Parameters

| Name        | Type            | Description                              |
|-------------|-----------------|--|
| conn_handle | saprfc-conn-opt | A valid SAP RFC Connection Option handle |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

**Throws**

None.

**Location**

`stc_monksap.dll`



### 8.4.3 SAP Client Connection Functions

The SAP Client Connection Functions manipulate the connection to the SAP client. These functions include:

- [saprfc-conn-createopt](#) on page 145
- [saprfc-conn-settrace](#) on page 145
- [saprfc-conn-set-clientconnpar](#) on page 146
- [saprfc-conn-set-clientconnmode](#) on page 147
- [saprfc-conn-set-clientconnopt-cpic](#) on page 148
- [saprfc-conn-set-clientconnopt-r3only](#) on page 148

---

#### saprfc-conn-createopt

##### Description

Creates and returns an SAP Client Connection Option Handle. The returned handle should be checked with [saprfc-conn-opt?](#).

##### Signature

(saprfc-conn-createopt)

##### Parameters

None.

##### Returns

An SAP Client Connection Option Handle.

##### Throws

None.

##### Location

**stc\_monksap.dll**

---

#### saprfc-conn-settrace

##### Description

Activates or deactivates tracing for the specified connection option.

##### Signature

(saprfc-conn-settrace *conn\_option tracing*)

##### Parameters

| Name        | Type            | Description  |
|-------------|-----------------|--|
| conn_option | saprfc-conn-opt | The SAP RFC Connection Option for which to activate or deactivate tracing. |

| Name    | Type   | Description  |
|---------|--------|--|
| tracing | string | "1" to activate tracing, or "0" to deactivate tracing. |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

### Location

**stc\_monksap.dll**

## saprfc-conn-set-clientconnpar

### Description

Sets general client-connection parameters.

### Signature

```
(saprfc-conn-set-clientconnpar conn_option client user passwd language)
```

### Parameters

| Name        | Type            | Description  |
|-------------|-----------------|--|
| conn_option | saprfc-conn-opt | The SAP RFC Connection Option to set.                  |
| client      | string          | The client string.                                     |
| user        | string          | A valid user name.                                     |
| passwd      | string          | The password corresponding to the specified user name. |
| language    | string          | A language string.                                     |

### Returns

A status code that evaluates to one of the following:

|           |                                     |
|-----------|-------------------------------------|
| SAPRFC_OK | Operations have concluded normally. |
|-----------|-------------------------------------|

|                |   |
|----------------|---|
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

#### Throws

None.

#### Location

**stc\_monksap.dll**

## saprfc-conn-set-clientconnmode

### Description

Sets the client connection mode.

### Signature

(saprfc-conn-set-clientconnmode *conn\_option mode*)

### Parameters

| Name        | Type            | Description                                     |
|-------------|-----------------|---|
| conn_option | saprfc-conn-opt | The SAP RFC Connection Option to set.           |
| mode        | string          | A mode number: "1" for CPIC and "0" for R3ONLY. |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

#### Throws

None.

#### Location

**stc\_monksap.dll**

## saprfc-conn-set-clientconnopt-cpic

### Description

Sets CPIC-specific parameters for a CPIC client connection.

### Signature

```
(saprfc-conn-set-clientconnopt-cpic conn_option gateway_host
 gateway_service)
```

### Parameters

| Name            | Type            | Description                           |
|-----------------|-----------------|---------------------------------------|
| conn_option     | saprfc-conn-opt | The SAP RFC Connection Option to set. |
| gateway_host    | string          | A gateway-host name.                  |
| gateway_service | string          | A valid gateway service.              |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

### Location

**stc\_monksap.dll**

## saprfc-conn-set-clientconnopt-r3only

### Description

Sets CPIC-specific parameters for a CPIC client connection.

### Signature

```
(saprfc-conn-set-clientconnopt-r3only conn_option host sys_number
 gateway_host gateway_service)
```

### Parameters

| Name            | Type            | Description                           |
|-----------------|-----------------|---------------------------------------|
| conn_option     | saprfc-conn-opt | The SAP RFC Connection Option to set. |
| host            | string          | A host name.                          |
| sys_number      | string          | A valid system number.                |
| gateway_host    | string          | A gateway-host name.                  |
| gateway_service | string          | A valid gateway service.              |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

### Location

**stc\_monksap.dll**

## 8.4.4 SAP Custom Structure Functions

The Custom Structure functions create and manipulate custom structures in SAP when they are required; for example, when a table field involves an non-homogeneous structure. These functions include:

[saprfc-struct-create](#) on page 150

[saprfc-struct-add-entry](#) on page 150

[saprfc-struct-install](#) on page 151

### saprfc-struct-create

#### Description

Creates an empty, custom structure that can hold up to `number_of_elements` entries.

#### Signature

`(saprfc-struct-create number_of_elements)`

#### Parameters

| Name                            | Type    | Description   |
|---------------------------------|---------|---|
| <code>number_of_elements</code> | integer | The number of elements to include in the structure. |

#### Returns

An SAP RFC Structure Handle.

#### Throws

None.

#### Location

**stc\_monksap.dll**

#### Notes

When a table field in SAP involves an non-homogeneous structure, a custom structure must be installed in SAP before RFC call/receive in order for SAP to automatically convert the data format. The `saprfc_struct_handle` returned from this function is used when adding entries to this structure.

### saprfc-struct-add-entry

#### Description

Adds an entry to the specified slot in the specified structure.

## Signature

```
(saprfc-struct-add-entry saprfc_struct_handle entry_index_number
 entry_name_string entry_type_number entry_length_number
 number_of_decimals)
```

## Parameters

| Name                 | Type    | Description  |
|----------------------|---------|--|
| saprfc_struct_handle | handle  | Handle specifying the structure, as generated by saprfc-struct-create. |
| entry_index_number   | integer | Number, beginning with 0, that specifies the element in the structure. |
| entry_name_string    | string  | Name of entry.   |
| entry_type_number    | integer | Type of entry.   |
| entry_length_number  | integer | Length of entry, in bytes.   |
| number_of_decimals   | integer | Number of decimal digits, if SAPRFC_TYPE_BCD.                          |

## Returns

Upon success, returns the string "SAPRFC\_OK", upon failure, returns "SAPRFC\_FAILURE".

## Throws

None.

## Location

**stc\_monksap.dll**

## Notes

Each entry in a custom structure created with [saprfc-struct-create](#) is defined by its name, type, length, and, if the type is SAPRFC\_TYPE\_BCD, the number of decimal digits in the BCD data.

## saprfc-struct-install

### Description

Installs a custom structure on SAP.

### Signature

```
(saprfc-struct-install saprfc_struct_handle rfc_type_number
 struct_name_string)
```

### Parameters

| Name                 | Type    | Description  |
|----------------------|---------|--|
| saprfc_struct_handle | handle  | Handle specifying the structure, as generated by saprfc-struct-create.     |
| rfc_type_number      | integer | Number representing the RFC type corresponding to the installed structure. |
| struct_name_string   | string  | New name for structure.  |

### Returns

If successful, returns a number (**rfc\_type\_number**) representing the RFC type corresponding to the installed structure; **0** indicates failure.

### Throws

None.

### Location

**stc\_monksap.dll**

### Notes

The structure is specified by the handle to the structure and the new name for it.



## 8.4.5 SAP Table List Functions

The SAP Table List functions manipulate SAP Table Lists. These functions include:

[saprfc-tab-createlist](#) on page 153

[saprfc-tab-create](#) on page 154

[saprfc-tab-clear](#) on page 154

[saprfc-tab-appline](#) on page 155

[saprfc-tab-applines](#) on page 156

[saprfc-tab-countline](#) on page 156

[saprfc-tab-getwidth](#) on page 157

[saprfc-tab-getline](#) on page 157

---

### saprfc-tab-createlist

#### Description

Creates an empty SAP table list. The returned list should be checked with [saprfc-tab-list?](#).

#### Signature

(saprfc-tab-createlist *size*)

#### Parameters

| Name | Type    | Description   |
|------|---------|---|
| size | integer | Optional: the initial size of the table list. If no size is specified, a zero-length table list is created. |

#### Returns

An SAP Table List.

#### Throws

None.

#### Location

**stc\_monksap.dll**

#### Notes

One or more named tables can be added to a table list. Table capacity grows automatically as tables are added to the list.

## saprfc-tab-create

### Description

Creates a named table and adds it to the specified table list.

### Signature

```
(saprfc-tab-create tablist tabname rowsize)
```

### Parameters

| Name    | Type            | Description                       |
|---------|-----------------|-----------------------------------|
| tablist | saprfc-tab-list | A valid SAP Table List name.      |
| tabname | string          | A valid SAP Table name.           |
| rowsize | integer         | Width in bytes of each table row. |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

### Location

**stc\_monksap.dll**

## saprfc-tab-clear

### Description

Clears the named table (erases its content) so that the table can be reused.

### Signature

```
(saprfc-tab-clear tablist tabname)
```

### Parameters

| Name    | Type            | Description                  |
|---------|-----------------|------------------------------|
| tablist | saprfc-tab-list | A valid SAP Table List name. |
| tabname | string          | A valid SAP Table name.      |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

### Location

**stc\_monksap.dll**

## saprfc-tab-appline

### Description

Appends a specified string to the specified table.

### Signature

*(saprfc-tab-appline tablist tabname string)*

### Parameters

| Name    | Type            | Description  |
|---------|-----------------|--|
| tablist | saprfc-tab-list | A valid SAP Table List name.   |
| tabname | string          | A valid SAP Table name.  |
| string  | string          | String to append to the table. The string length must not exceed the table row length. |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

### Location

**stc\_monksap.dll**

---

## saprfc-tab-applines

### Description

Appends multiple lines to the named table.

### Signature

*(saprfc-tab-applines *tablist* *tabname* *string*)*

### Parameters

| Name    | Type            | Description   |
|---------|-----------------|---|
| tablist | saprfc-tab-list | A valid SAP Table List name.  |
| tabname | string          | A valid SAP Table name.   |
| string  | string          | A string containing the data to append within multiple rows. Each row must be terminated with a newline ('\n') character. Each row in the input string must not exceed the table width. |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

### Location

**stc\_monksap.dll**

---

## saprfc-tab-countline

### Description

Counts the lines within the named table.

### Signature

*(saprfc-tab-countline *tablist* *tabname*)*

### Parameters

| Name    | Type            | Description                  |
|---------|-----------------|------------------------------|
| tablist | saprfc-tab-list | A valid SAP Table List name. |
| tabname | string          | A valid SAP Table name.      |

### Returns

If successful, the number of lines within the named table; upon error, a negative number.

### Throws

None.

### Location

**stc\_monksap.dll**

---

## saprfc-tab-getwidth

### Description

Returns the width of the named table.

### Signature

*(saprfc-tab-getwidth tablist tabname)*

### Parameters

| Name    | Type            | Description                  |
|---------|-----------------|------------------------------|
| tablist | saprfc-tab-list | A valid SAP Table List name. |
| tabname | string          | A valid SAP Table name.      |

### Returns

If successful, the width of the named table; upon error, a negative number.

### Throws

None.

### Location

**stc\_monksap.dll**

---

## saprfc-tab-getline

### Description

Returns the indexed row from the named table in the specified table list.

## Signature

(saprfc-tab-getline *tablist tabname index*)

## Parameters

| Name    | Type            | Description   |
|---------|-----------------|---|
| tablist | saprfc-tab-list | A valid SAP Table List name.  |
| tabname | string          | A valid SAP Table name.   |
| index   | integer         | A valid row-index number. The row index for a table starts from 0, and must not exceed the number of rows in the table. |

## Returns

A table row.

## Throws

None.

## Location

**stc\_monksap.dll**

## 8.4.6 SAP Parameter List Functions

The SAP Parameter List functions manipulate SAP parameter lists. These functions include:

- [saprfc-par-createlist](#) on page 159
- [saprfc-par-add-char](#) on page 160
- [saprfc-par-add-int](#) on page 160
- [saprfc-par-add](#) on page 161
- [saprfc-par-add-receiving](#) on page 162
- [saprfc-par-get-char](#) on page 163
- [saprfc-par-get-int](#) on page 163
- [saprfc-par-get](#) on page 164

---

### saprfc-par-createlist

#### Description

Creates an empty SAP parameter list. The returned list should be checked with [saprfc-par-list?](#).

#### Signature

(saprfc-par-createlist *size*)

#### Parameters

| Name | Type    | Description   |
|------|---------|---|
| size | integer | Optional: the initial size of the parameter list. If no size is specified, a zero-length parameter list is created. |

#### Returns

An SAP parameter list.

#### Throws

None.

#### Location

**stc\_monksap.dll**

#### Notes

One or more named parameters can be added to a parameter list. Capacity grows automatically as parameters are added to the list.

## saprfc-par-add-char

### Description

Adds a parameter of type **char** to the parameter list.

### Signature

*(saprfc-par-add-char list name value)*

### Parameters

| Name  | Type            | Description                       |
|-------|-----------------|-----------------------------------|
| list  | saprfc-par-list | A valid SAP parameter list.       |
| name  | string          | The name of the parameter to add. |
| value | string          | The value of the parameter.       |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

### Location

**stc\_monksap.dll**

## saprfc-par-add-int

### Description

Adds a parameter of type **integer** to the parameter list.

### Signature

*(saprfc-par-add-int list name value)*

### Parameters

| Name  | Type            | Description                       |
|-------|-----------------|-----------------------------------|
| list  | saprfc-par-list | A valid SAP parameter list.       |
| name  | string          | The name of the parameter to add. |
| value | integer         | The value of the parameter.       |



### Returns

Returns a status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

### Location

**stc\_monksap.dll**

## saprfc-par-add

### Description

Adds a parameter of the specified type.

### Signature

*(saprfc-par-add list type name value size)*

### Parameters

| Name  | Type            | Description                       |
|-------|-----------------|-----------------------------------|
| list  | saprfc-par-list | A valid SAP parameter list.       |
| type  | integer         | The type of the parameter.        |
| name  | string          | The name of the parameter to add. |
| value | string          | The value of the parameter.       |
| size  | integer         | The size of the parameter.        |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

## Location

**stc\_monksap.dll**

## Notes

If the parameter type is `SAPRFC_TYPE_NUM`, *size* should be exactly the same as what the remote function requires. In this case, if *value* contains more digits than *size*, *value* is truncated so that only the first *size* digits are used.

If *value* contains fewer digits than *size*, leading zeros are padded to *value*.

If the parameter type is `SAPRFC_TYPE_BYTE`, *size* should be exactly the same as what the remote function requires and *value* should contain exactly *size* bytes.

For other parameter types, *size* is ignored.

## saprfc-par-add-receiving

### Description

Adds a receiving parameter to the specified importing parameter list.

### Signature

(saprfc-par-add-receiving *list type name size*)

### Parameters

| Name | Type            | Description                       |
|------|-----------------|-----------------------------------|
| list | saprfc-par-list | A valid SAP parameter list.       |
| type | integer         | The type of the parameter.        |
| name | string          | The name of the parameter to add. |
| size | integer         | The size of the parameter.        |

### Returns

A status code that evaluates to one of the following:

|                |   |
|----------------|---|
| SAPRFC_OK      | Operations have concluded normally.                       |
| SAPRFC_CLOSED  | The installed service module returns the string "CLOSED". |
| SAPRFC_TIMEOUT | The function has timed out.                               |
| SAPRFC_FAILURE | The function has failed.                                  |

### Throws

None.

### Location

**stc\_monksap.dll**

## Notes

If *type* is any of the following, *size* must be at least as big as what the remote function requires:

- **SAPRFC\_TYPE\_CHAR**: Char string
- **SAPRFC\_TYPE\_NUM**: Stream of digits
- **SAPRFC\_TYPE\_BYTE**: Byte string (row data).

For other parameter types, *size* is ignored.

## saprfc-par-get-char

### Description

Returns the string value of the named parameter.

### Signature

(saprfc-par-get-char *list name*)

### Parameters

| Name | Type            | Description                       |
|------|-----------------|-----------------------------------|
| list | saprfc-par-list | A valid SAP parameter list.       |
| name | string          | The name of the parameter to add. |

### Returns

The string value of the named parameter.

### Throws

None.

### Location

**stc\_monksap.dll**

## saprfc-par-get-int

### Description

Returns the integer value of the named parameter.

### Signature

(saprfc-par-get-int *list name*)

### Parameters

| Name | Type            | Description                       |
|------|-----------------|-----------------------------------|
| list | saprfc-par-list | A valid SAP parameter list.       |
| name | string          | The name of the parameter to add. |

### Returns

An integer.

### Throws

None.

### Location

**stc\_monksap.dll**

---

## saprfc-par-get

### Description

Returns the value of the named parameter.

### Signature

*(saprfc-par-get list type name)*

### Parameters

| Name | Type            | Description                       |
|------|-----------------|-----------------------------------|
| list | saprfc-par-list | A valid SAP parameter list.       |
| type | integer         | The type of the parameter         |
| name | string          | The name of the parameter to add. |

### Returns

The value of the specified parameter as a string.

### Throws

None.

### Location

**stc\_monksap.dll**

## 8.4.7 SAP Type-Checking Functions

The SAP Type-Checking functions determine whether or not specified objects are SAP data types. These functions include:

[saprfc-conn-handle?](#) on page 165

[saprfc-conn-opt?](#) on page 165

[saprfc-struct-handle?](#) on page 166

[saprfc-par-list?](#) on page 166

[saprfc-tab-list?](#) on page 167

---

### saprfc-conn-handle?

#### Description

Determines whether or not the specified object is a valid SAP Connection Handle.

#### Signature

(saprfc-conn-handle? *monk\_object*)

#### Parameters

| Name        | Type     | Description         |
|-------------|----------|---------------------|
| monk_object | any type | The object to test. |

#### Returns

Returns a Boolean true (**#t**) if the object is a valid SAP connection handle; otherwise, returns false (**#f**).

#### Throws

None.

#### Location

**stc\_monksap.dll**

---

### saprfc-conn-opt?

#### Description

Determines whether or not the specified object is a valid SAP Client Connection Option Handle.

#### Signature

(saprfc-conn-opt? *monk\_object*)

### Parameters

| Name        | Type     | Description         |
|-------------|----------|---------------------|
| monk_object | any type | The object to test. |

### Returns

Returns a Boolean true (**#t**) if the object is a valid Client Connection Option Handle; otherwise, returns false (**#f**).

### Throws

None.

### Location

**stc\_monksap.dll**

## saprfc-struct-handle?

### Description

Determines whether or not the specified object is a valid `saprfc_struct_handle`.

### Signature

(saprfc-struct-handle? *monk\_object*)

### Parameters

| Name        | Type     | Description         |
|-------------|----------|---------------------|
| monk_object | any type | The object to test. |

### Returns

Returns a Boolean true (**#t**) if the object is a valid SAP Parameter List; otherwise, returns false (**#f**).

### Throws

None.

### Location

**stc\_monksap.dll**

## saprfc-par-list?

### Description

Determines whether or not the specified object is a valid SAP Parameter List.

### Signature

(saprfc-par-list? *monk\_object*)

### Parameters

| Name        | Type     | Description         |
|-------------|----------|---------------------|
| monk_object | any type | The object to test. |

### Returns

Returns a Boolean true (**#t**) if the object is a valid SAP Parameter List; otherwise, returns false (**#f**).

### Throws

None.

### Location

**stc\_monksap.dll**

---

## saprfc-tab-list?

### Description

Determines whether or not the specified object is a valid SAP Table List.

### Signature

(saprfc-tab-list? *monk\_object*)

### Parameters

| Name        | Type     | Description         |
|-------------|----------|---------------------|
| monk_object | any type | The object to test. |

### Returns

Returns a Boolean true (**#t**) if the object is a valid SAP Table List; otherwise, returns false (**#f**).

### Throws

None.

### Location

**stc\_monksap.dll**

## 8.5 e\*Way Kernel Layer

### 8.5.1 Generic e\*Way Functions

The functions described in this section are implemented in the e\*Way Kernel layer and control the e\*Way's most basic operations. They can be used only by the functions defined within the e\*Way's configuration file. None of these functions is available to Collaboration Rules scripts executed by the e\*Way. These functions are located in `stcewgenericmonk.exe`.

The current set of Generic e\*Way functions is:

- [event-commit-to-egate](#) on page 168
- [event-rollback-to-egate](#) on page 169
- [event-send-to-egate](#) on page 169
- [event-send-to-egate-ignore-shutdown](#) on page 170
- [event-send-to-egate-no-commit](#) on page 170
- [get-logical-name](#) on page 171
- [insert-exchange-data-event](#) on page 171
- [send-external-up](#) on page 172
- [send-external-down](#) on page 172
- [shutdown-request](#) on page 173
- [start-schedule](#) on page 173
- [stop-schedule](#) on page 174
- [waiting-to-shutdown](#) on page 174

---

### event-commit-to-egate

#### Description

Commits the Event sent previously to the e\*Gate system using [event-send-to-egate-no-commit](#).

#### Signature

(event-commit-to-egate *string*)

#### Parameters

| Name   | Type   | Description                               |
|--------|--------|---|
| string | string | The data to be sent to the e*Gate system. |



### Returns

Boolean true (#t) if the data is committed successfully; otherwise, false (#f).

### Throws

None.

---

## event-rollback-to-egate

### Description

Rolls back the Event sent previously to the e\*Gate system using [event-send-to-egate-no-commit](#), following receipt of a rollback command from the external system.

### Signature

(event-rollback-to-egate *string*)

### Parameters

| Name   | Type   | Description                                      |
|--------|--------|--|
| string | string | The data to be rolled back to the e*Gate system. |

### Returns

Boolean true (#t) if the data is rolled back successfully; otherwise, false (#f).

### Throws

None.

---

## event-send-to-egate

### Description

Sends data that the e\*Way has already received from the external system into the e\*Gate system as an Event.

### Signature

(event-send-to-egate *string*)

### Parameters

| Name   | Type   | Description                              |
|--------|--------|--|
| string | string | The data to be sent to the e*Gate system |

### Returns

A Boolean true (#t) if the data is sent successfully; otherwise, a Boolean false (#f).

### Throws

None.

### Additional information

This function can be called by any e\*Way function when it is necessary to send data to the e\*Gate system in a blocking fashion.

### See also

[event-send-to-egate-ignore-shutdown](#) on page 170

[event-send-to-egate-no-commit](#) on page 170

## event-send-to-egate-ignore-shutdown

### Description

Sends data that the e\*Way has already received from the external system into the e\*Gate system as an Event—but ignores any pending shutdown issues.

### Signature

(event-send-to-egate-ignore-shutdown *string*)

### Parameters

| Name   | Type   | Description                               |
|--------|--------|---|
| string | string | The data to be sent to the e*Gate system. |

### Returns

Boolean true (**#t**) if the data is sent successfully; otherwise, false (**#f**).

### Throws

None.

### See also

[event-send-to-egate](#) on page 169

[event-send-to-egate-no-commit](#) on page 170

## event-send-to-egate-no-commit

### Description

Sends data that the e\*Way has received from the external system to the e\*Gate system as an Event—but without Committing, pending confirmation from the external system of correct transmission of the data.

### Signature

(event-send-to-egate-no-commit *string*)

## Parameters

| Name   | Type   | Description                               |
|--------|--------|---|
| string | string | The data to be sent to the e*Gate system. |

## Returns

Boolean true (**#t**) if the data is sent successfully; otherwise, false (**#f**).

## Throws

None.

## See also

[event-commit-to-egate](#) on page 168

[event-rollback-to-egate](#) on page 169

[event-send-to-egate](#) on page 169

[event-send-to-egate-ignore-shutdown](#) on page 170

## get-logical-name

### Description

Returns the logical name of the e\*Way.

### Signature

(get-logical-name)

### Parameters

None.

### Returns

The name of the e\*Way (as defined by the e\*Gate Schema Designer).

### Throws

None.

## insert-exchange-data-event

### Description

While the [Exchange Data with External Function](#) is still active, this function can be called to initiate a repeat call to it—whether or not data was queued to e\*Gate via the function’s return mechanism following the initial call.

### Signature

(insert-exchange-data-event)

### Parameters

None.

### Returns

None.

### Throws

None.

### See also

[Exchange Data Interval](#) on page 103

[Zero Wait Between Successful Exchanges](#) on page 104

---

## send-external-up

### Description

Informs the e\*Way that the connection to the external system is up.

### Signature

(send-external-up)

### Parameters

None.

### Returns

None.

### Throws

None.

---

## send-external-down

### Description

Informs the e\*Way that the connection to the external system is down.

### Signature

(send-external-down)

### Parameters

None.

### Returns

None.

### Throws

None.

---

## shutdown-request

### Description

Completes the e\*Gate shutdown procedure that was initiated by the Control Broker but was interrupted by returning a non-null value within the **Shutdown Command Notification Function**. Once this function is called, shutdown proceeds immediately.

### Signature

(shutdown-request)

### Parameters

None.

### Returns

None.

### Throws

None.

### Additional Information

Once interrupted, the e\*Way's shutdown cannot proceed until this Monk function is called. If you do interrupt an e\*Way shutdown, we recommend that you complete the process in a timely fashion.

---

## start-schedule

### Description

Requests that the e\*Way execute the **Exchange Data with External Function** specified within the e\*Way's configuration file. Does not affect any defined schedules.

### Signature

(start-schedule)

### Parameters

None.

### Returns

None.

### Throws

None.

---

## stop-schedule

### Description

Requests that the e\*Way halt execution of the **External Connection Establishment Function** specified within the e\*Way's configuration file. Execution is stopped when the e\*Way concludes any open transaction. Does not effect any defined schedules, and does not halt the e\*Way process itself.

### Signature

(stop-schedule)

### Parameters

None.

### Returns

None.

### Throws

None.

---

## waiting-to-shutdown

### Description

Informs the external application that a shutdown command has been issued.

### Signature

(waiting-to-shutdown)

### Parameters

None.

### Returns

Boolean true (**#t**) if successful; otherwise, false (**#f**).

### Throws

None.

# Index

## A

ABAP components, importing 50  
 ABAP functions (custom)  
   Z\_OUTBOUND\_DGW\_INITIATE 48, 59, 60, 61,  
   62, 63, 65, 91  
   Z\_STRUCTURE\_EXPORT 48, 83  
 ABAP modules (custom)  
   Z\_CUSTOM\_MAPPING 48, 56  
   Z\_R3\_BDC\_DATA\_IMPORT 48, 53, 55, 90  
   Z\_STC\_DGW\_SAPBDC\_SERV 48, 91  
   ZOTBNDTEST2 49, 60  
 Additional Path parameter 105  
 APIs - see Monk functions  
 Assigning ETDs to Event Types 38  
 Autorun 20  
 Auxiliary Library Directories parameter 106

## B

Batch Mode 14  
 BDC Converter 36, 81, 87  
 BDC session 14, 83  
 Build Tool 36, 81

## C

Call Transaction 83  
 Changing the User Name 73  
 Client parameter 113  
 Collaboration 39, 77, 89  
   Rules 39, 77, 78, 89  
   Service 77  
 components, e\*Way 16  
 configuration  
   Communication Setup 102–104  
   FTP Setup for File-Based Transfers 117–118  
   General Settings 100–101  
   Monk Configuration 105–112  
   SAP RFC Client Setup 113–114  
   SAP RFC Server Setup 115–116  
 configuration parameters  
   Additional Path 105  
   Auxiliary Library Directories 106  
   Client 113

Down Timeout 103  
 Exchange Data Interval 103  
 Exchange Data With External Function 108  
 External Connection Establishment Function 109  
 External Connection Shutdown Function 110  
 External Connection Verification Function 109  
 File Transfer Mode 117  
 Forward External Errors 101  
 Gateway Host Name 115  
 Gateway Service 115  
 Host Name of the R/3 Target System 113  
 Journal File Name 100  
 Language 114  
 Log File for Failed Records 114, 116  
 Max Failed Messages 100  
 Max Resends Per Message 100  
 Maximum Number of Failure Before Erroring  
   Out 114, 116  
 Monk Environment Initialization File 106  
 Negative Acknowledgment Function 111  
 Password 113, 117  
 Positive Acknowledgement Function 110  
 Process Outgoing Message Function 107  
 Program ID 115  
 Remote FTP Server Root Directory 118  
 Resend Timeout 104  
 Shutdown Command Notification Function 112  
 Start Exchange Data Schedule 103  
 Startup Function 107  
 Stop Exchange Data Schedule 102  
 System Number of the R/3 Target System 113  
 Temporary File on Local System 117  
 Temporary File on SAP Application Server 118  
 Trace 114, 115  
 Up Timeout 103  
 User 113, 117  
 Wait for Request Interval 115  
 Zero Wait Between Successful Exchanges 104  
 configuration procedures 69  
 conventions, writing 10

## D

Data types 121  
 Down Timeout parameter 103

## E

e\*Gate-to-SAP Data Mapping 83  
 e\*Way  
   Components 16  
   configuration 69  
   creating 67  
   Installation 20

- Properties 68
- Schedules 73
- setup 66
- Startup Options 73
- troubleshooting 77
- ETD Build Tool 36, 81
- Event Type 38
- Event Type Definition (ETD) 30, 38
- event-commit-to-egate function 168
- Event-Driven Mode 15
- event-rollback-to-egate function 169
- Events 88
- event-send-to-egate function 169
- event-send-to-egate-ignore-shutdown function 170
- event-send-to-egate-no-commit function 170
- Exchange Data Interval parameter 103
- Exchange Data with External Function parameter 108
- External Connection Establishment Function parameter 109
- External Connection Shutdown Function parameter 110
- External Connection Verification Function parameter 109

## F

- File Transfer Mode parameter 117
- Forward External Errors parameter 101
- functions
  - Generic 168–174
  - SAP BDC Template Functions 122–126
  - SAP BDC Utility Functions 127–131
  - SAP Client Connection Functions 145–149
  - SAP Custom Structure Functions 150–152
  - SAP Parameter List Functions 159–164
  - SAP RFC Client Functions 132–135
  - SAP RFC Server Functions 136–144
  - SAP Table List Functions 153–158
  - SAP Type-Checking Functions 165–167
  - see also ABAP functions
  - see also Monk functions

## G

- Gateway Host Name parameter 115
- Gateway Service parameter 115
- Generic e\*Way Functions 168–174
- get-logical-name function 171

## H

- Host Name of the R/3 Target System parameter 113

## I

- importing SAP ABAP components 50
- Inbound Batch Load Module 14
- insert-exchange-data-event function 171
- Installation procedure
  - e\*Way (UNIX) 23
  - e\*Way (Windows) 20
  - sample schema 25
- InstallShield 20
- Intelligent Queue (IQ) 40, 77

## J

- Journal File Name parameter 100

## L

- Language parameter 114
- Load Path, Monk 105
- Log File for Failed Records parameter 114, 116
- logging options 75

## M

- Max Failed Messages parameter 100
- Max Resends Per Message parameter 100
- Maximum Number of Failure Before Erroring Out parameter 114, 116
- message types (custom)
  - ZOUTTEST 64
- monitoring thresholds 76
- Monk Configuration
  - Load Path 105
  - Specifying File Names 105
  - Specifying Function Names 105
  - Specifying Multiple Directories 105
- Monk Environment Initialization File parameter 106
- Monk functions
  - event-commit-to-egate 168
  - event-rollback-to-egate 169
  - event-send-to-egate 169
  - event-send-to-egate-ignore-shutdown 170
  - event-send-to-egate-no-commit 170
  - get-logical-name 171
  - insert-exchange-data-event 171
  - sapbdc-ack 125
  - sapbdc-client-connect 127
  - sapbdc-client-disconnect 128
  - sapbdc-client-geterrormessage 128
  - sapbdc-client-struct-send 127
  - sapbdc-fromsap-connect 123
  - sapbdc-fromsap-finish 123
  - sapbdc-fromsap-startup 122



sapbdc-init 129  
 sapbdc-nack 126  
 sapbdc-server-shutdown 129  
 sapbdc-server-startup 130  
 sapbdc-server-struct-fetch 130  
 sapbdc-tosap-connect 124  
 sapbdc-tosap-finish 125  
 sapbdc-tosap-startup 124  
 saprfc-client-callreceive 132  
 saprfc-client-connect 132  
 saprfc-client-createtid 133  
 saprfc-client-disconnect 134  
 saprfc-client-indirectcall 134  
 saprfc-conn-createopt 145  
 saprfc-conn-handle? 165  
 saprfc-conn-opt? 165  
 saprfc-conn-set-clientconnmode 147  
 saprfc-conn-set-clientconnopt-cpic 148  
 saprfc-conn-set-clientconnpar 146  
 saprfc-conn-set-clienconnopt-r3only 148  
 saprfc-conn-settrace 145  
 saprfc-par-add 161  
 saprfc-par-add-char 160  
 saprfc-par-add-int 160  
 saprfc-par-add-receiving 162  
 saprfc-par-createlist 159  
 saprfc-par-get 164  
 saprfc-par-get-char 163  
 saprfc-par-get-int 163  
 saprfc-par-list? 166  
 saprfc-server-getcallbackfailuretid 140  
 saprfc-server-getcallbackfailuretype 140  
 saprfc-server-getinputdata 139  
 saprfc-server-installfunction 137  
 saprfc-server-installtransctrl 137  
 saprfc-server-resetcallbackfailure 141  
 saprfc-server-sendoutputdata 142  
 saprfc-server-shutdown 143  
 saprfc-server-startup 136  
 saprfc-server-waitanddispatch 142  
 saprfc-struct-add-entry 150  
 saprfc-struct-create 150  
 saprfc-struct-handle? 166  
 saprfc-struct-install 151  
 saprfc-tab-appline 155  
 saprfc-tab-applines 156  
 saprfc-tab-clear 154  
 saprfc-tab-countline 156  
 saprfc-tab-create 154  
 saprfc-tab-createlist 153  
 saprfc-tab-getline 157  
 saprfc-tab-getwidth 157  
 saprfc-tab-list? 167  
 send-external down 172

send-external-up 172  
 shutdown-request 173  
 start-schedule 173  
 stop-schedule 174  
 waiting-to-shutdown 174

## N

Negative Acknowledgment Function parameter 111

## O

objects (custom)  
     ZDGCNT 57

## P

Parameters - see configuration parameters  
 Participating Host 77  
 Password parameter 113, 117  
 Positive Acknowledgment Function parameter 110  
 procedures  
     configuration 69  
     installation 20  
     setup 66  
 Process Outgoing Message Function parameter 107  
 Program ID parameter 115  
 Properties, e\*Way 68

## Q

Queue - see Intelligent Queue (IQ)

## R

Remote FTP Server Root Directory parameter 118  
 Resend Timeout parameter 104

## S

sample schema  
     descriptions 44  
     installation 25  
 SAP ABAP components, importing 50  
 SAP BDC Converter 36, 81  
 SAP BDC data types 121  
 SAP Converter 81  
 SAP Structure Export Module 79  
 sapbdc-ack function 125  
 sapbdc-client-connect function 127  
 sapbdc-client-disconnect function 128  
 sapbdc-client-geterrormessage function 128  
 sapbdc-client-struct-send function 127

- sapbdc-fromsap-connect function 123
  - sapbdc-fromsap-finish function 123
  - sapbdc-fromsap-startup function 122
  - sapbdc-init function 129
  - sapbdc-nack function 126
  - sapbdc-server-shutdown function 129
  - sapbdc-server-startup function 130
  - sapbdc-server-struct-fetch function 130
  - sapbdc-tosap-connect function 124
  - sapbdc-tosap-finish function 125
  - sapbdc-tosap-startup function 124
  - saprfc-client-callreceive function 132
  - saprfc-client-connect function 132
  - saprfc-client-createtid function 133
  - saprfc-client-disconnect function 134
  - saprfc-client-indirectcall function 134
  - saprfc-conn-createopt function 145
  - saprfc-conn-handle data type 121
  - saprfc-conn-handle? function 165
  - saprfc-conn-opt data type 121
  - saprfc-conn-opt? function 165
  - saprfc-conn-set-clientconnmode function 147
  - saprfc-conn-set-clientconnopt-cpic function 148
  - saprfc-conn-set-clientconnopt-r3only function 148
  - saprfc-conn-set-clientconnpar function 146
  - saprfc-conn-settrace function 145
  - saprfc-par-add function 161
  - saprfc-par-add-char function 160
  - saprfc-par-add-int function 160
  - saprfc-par-add-receiving function 162
  - saprfc-par-createlist function 159
  - saprfc-par-get function 164
  - saprfc-par-get-char function 163
  - saprfc-par-get-int function 163
  - saprfc-par-list data type 121
  - saprfc-par-list? function 166
  - saprfc-server-getcallbackfailureid function 140
  - saprfc-server-getcallbackfailuretype function 140
  - saprfc-server-getinputdata function 139
  - saprfc-server-installfunction function 137
  - saprfc-server-installtransctrl function 137
  - saprfc-server-resetcallbackfailure function 141
  - saprfc-server-sendoutputdata function 142
  - saprfc-server-shutdown function 143
  - saprfc-server-startup function 136
  - saprfc-server-waitanddispatch function 142
  - saprfc-status data type 121
  - saprfc-struct-add-entry function 150
  - saprfc-struct-create function 150
  - saprfc-struct-handle? function 166
  - saprfc-struct-install function 151
  - saprfc-tab-appline function 155
  - saprfc-tab-applines function 156
  - saprfc-tab-clear function 154
  - saprfc-tab-countline function 156
  - saprfc-tab-create function 154
  - saprfc-tab-createlist function 153
  - saprfc-tab-getline function 157
  - saprfc-tab-getwidth function 157
  - saprfc-tab-list data type 121
  - saprfc-tab-list? function 167
  - Schedules 73
  - Screen Recorder 83
  - SE09 SAP transaction 51
  - SE11 SAP transaction 53
  - SE38 SAP transaction 83
  - send-external-down function 172
  - send-external-up function 172
  - Setting Startup Options or Schedules 73
  - Shutdown Command Notification Function parameter 112
  - shutdown-request function 173
  - SM35 SAP transaction 56
  - SM59 SAP transaction 63, 64
  - SNRO SAP transaction 57
  - Start Exchange Data Schedule parameter 103
  - start-schedule function 173
  - Startup Function parameter 107
  - Startup Options 73
  - Stop Exchange Data Schedule parameter 102
  - stop-schedule function 174
  - Structure Export Module 81
  - System Number of the R/3 Target System parameter 113
  - system requirements 18–19
- ## T
- tables (custom)
    - ZDGBDC 48, 55, 63
    - ZDGLOG 49, 57
    - ZDGOUT 49, 62, 63
  - Temporary File on Local System parameter 117
  - Temporary File on SAP Application Server parameter 118
  - Trace parameter 114, 115
  - transactions (SAP) 12
    - SE09 51
    - SE11 53, 63
    - SE38 83
    - SM35 56
    - SM59 63, 64
    - SNRO 57
  - troubleshooting the e\*Way 77
- ## U
- UNIX installation procedure 23

## Index

Up Timeout parameter 103  
User name 73  
User parameter 113, 117

## W

Wait for Request Interval parameter 115  
waiting-to-shutdown function 174  
Windows installation procedure 20  
writing conventions 10

## Z

Z\_CUSTOM\_MAPPING module 48, 56  
Z\_OUTBOUND\_DGW\_INITIATE function 48, 59,  
60, 61, 62, 63, 65, 91  
Z\_R3\_BDC\_DATA\_IMPORT module 48, 53, 55, 90  
Z\_STC\_DGW\_SAPBDC\_SERV module 48, 91  
Z\_STRUCTURE\_EXPORT function 48, 83  
ZDGBDC table 48, 53, 55, 63  
ZDGCNT object 57  
ZDGLOG table 49, 57  
ZDGOUT table 49, 62, 63, 64, 65  
Zero Wait Between Successful Exchanges parameter  
104  
ZOTBNDTEST2 module 49, 60  
ZOUTTEST message type 64