

***SeeBeyond ICAN Suite***

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

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

*Monk Version*



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 This work is protected as an unpublished work under the copyright laws.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 20050502113356.

# Contents

---

|                                      |           |
|--------------------------------------|-----------|
| <b>Preface</b>                       | <b>11</b> |
| Intended Reader                      | 11        |
| Organization                         | 11        |
| Nomenclature                         | 12        |
| Online Use                           | 12        |
| Writing Conventions                  | 12        |
| Additional Documentation             | 13        |
| <hr/>                                |           |
| <b>Chapter 1</b>                     |           |
| <b>Introduction</b>                  | <b>14</b> |
| SAP Interface Options                | 14        |
| What is a BAPI?                      | 15        |
| How is BAPI Different from ALE-IDoc? | 15        |
| How are BAPIs Invoked?               | 16        |
| Security Issues                      | 16        |
| The SAP BAPI e*Way                   | 17        |
| Overview                             | 17        |
| Supported Operating Systems          | 18        |
| <hr/>                                |           |
| <b>Chapter 2</b>                     |           |
| <b>Installation</b>                  | <b>19</b> |
| System Requirements                  | 19        |
| Environment Configuration            | 19        |
| External System Requirements         | 20        |
| External Configuration Requirements  | 20        |
| Installing the e*Way                 | 21        |
| Windows Systems                      | 21        |
| Installation Procedure               | 21        |
| Subdirectories and Files             | 23        |
| UNIX Systems                         | 25        |

|                               |           |
|-------------------------------|-----------|
| Installation Procedure        | 25        |
| Subdirectories and Files      | 25        |
| <b>Optional Example Files</b> | <b>27</b> |
| Installation Procedure        | 27        |
| Subdirectories and Files      | 28        |

## Chapter 3

|  |           |
|--|-----------|
| <b>Implementation</b>                  | <b>30</b> |
| <b>Overview</b>                        | <b>30</b> |
| Pre-Implementation Tasks               | 30        |
| Implementation Sequence                | 31        |
| Viewing e*Gate Components              | 31        |
| <b>Creating a Schema</b>               | <b>32</b> |
| <b>Creating Event Types</b>            | <b>33</b> |
| <b>Creating Event Type Definitions</b> | <b>34</b> |
| The BAPI Structure Builder             | 34        |
| Assigning ETDs to Event Types          | 40        |
| <b>Defining Collaborations</b>         | <b>41</b> |
| <b>Creating Intelligent Queues</b>     | <b>42</b> |
| <b>Registering the e*Way</b>           | <b>42</b> |
| <b>Sample Schemas</b>                  | <b>45</b> |
| bapi2egate                             | 46        |
| Collaborations                         | 46        |
| Startup Function                       | 48        |
| sapbapitoew-startup                    | 48        |
| egate2bapi                             | 49        |
| Collaborations                         | 49        |
| Startup Function                       | 52        |
| dgtosapbapi-startup                    | 52        |
| TrfcFromBapi                           | 53        |
| Collaborations                         | 53        |
| Startup Function                       | 55        |
| idocfromsaptoegate-startup             | 55        |
| TrfcToBapi                             | 56        |
| Collaborations                         | 56        |

## Chapter 4

|  |           |
|--|-----------|
| <b>e*Way Extensions</b>                  | <b>58</b> |
| <b>SAP-Initiated BAPI Calls</b>          | <b>58</b> |
| <b>Importing the SAP ABAP Components</b> | <b>59</b> |
| <b>Updating SAP R/3 Objects</b>          | <b>62</b> |
| <b>Placeholder Function</b>              | <b>66</b> |

|                          |    |
|--------------------------|----|
| Data Extraction via ABAP | 68 |
|--------------------------|----|

---

## Chapter 5

|   |           |
|---|-----------|
| <b>Setup Procedures</b>                       | <b>72</b> |
| Overview                                      | 72        |
| <b>Setting Up the e*Way</b>                   | <b>73</b> |
| Creating the e*Way                            | 73        |
| Modifying e*Way Properties                    | 74        |
| Configuring the e*Way                         | 75        |
| Using the e*Way Editor                        | 76        |
| Section and Parameter Controls                | 77        |
| Parameter Configuration Controls              | 77        |
| Command-line Configuration                    | 78        |
| Getting Help                                  | 78        |
| Changing the User Name                        | 79        |
| Setting Startup Options or Schedules          | 79        |
| Activating or Modifying Logging Options       | 81        |
| Activating or Modifying Monitoring Thresholds | 82        |
| <b>Troubleshooting the e*Way</b>              | <b>83</b> |
| Configuration Problems                        | 83        |
| System-related Problems                       | 84        |

---

## Chapter 6

|  |            |
|--|------------|
| <b>Operational Overview</b>            | <b>85</b>  |
| Contents                               | 85         |
| <b>BAPI-ETD Modeling</b>               | <b>86</b>  |
| How BAPI Methods are Modeled in e*Gate | 86         |
| The BAPI Structure Builder             | 87         |
| Terminology                            | 88         |
| Data Insertion and Extraction          | 89         |
| BAPI Transport                         | 91         |
| <b>RFC Transport Process</b>           | <b>92</b>  |
| Client Mode (e*Gate to SAP)            | 92         |
| Server Mode (SAP to e*Gate)            | 94         |
| Process Overview                       | 94         |
| TCP/IP Ports                           | 95         |
| Placeholder Function                   | 96         |
| Program ID Registration                | 98         |
| Monk Handler Installation              | 98         |
| Polling SAP for Function Calls         | 99         |
| Extracting Data                        | 100        |
| Processing Data and Sending Results    | 101        |
| Data Extraction via ABAP               | 102        |
| <b>tRFC Transport Process</b>          | <b>106</b> |
| Client Mode (e*Gate to SAP)            | 107        |

|                                    |            |
|------------------------------------|------------|
| EID Not Found                      | 107        |
| EID Found but TID Not Used         | 107        |
| EID Found and TID Reserved         | 107        |
| <b>Server Mode (SAP to e*Gate)</b> | <b>112</b> |
| Process Overview                   | 112        |
| Monk Handler Installation          | 115        |
| Checking SAP for Function Calls    | 115        |
| Extracting and Processing Data     | 116        |
| <b>e*Way Architecture</b>          | <b>119</b> |
| Events and Collaborations          | 120        |
| ETD Model Layer                    | 121        |
| SAP BAPI Transport Layer           | 122        |
| SAP RFC Transport Layer            | 122        |
| e*Way Kernel Layer                 | 123        |
| <b>Basic e*Way Processes</b>       | <b>124</b> |
| Initialization Process             | 125        |
| Connect to External Process        | 126        |
| Data Exchange Process              | 127        |
| Disconnect from External Process   | 130        |
| Shutdown Process                   | 130        |

---

## Chapter 7

|  |            |
|--|------------|
| <b>Configuration Parameters</b>            | <b>131</b> |
| <b>Overview</b>                            | <b>131</b> |
| <b>General Settings</b>                    | <b>132</b> |
| Journal File Name                          | 132        |
| Max Resends Per Message                    | 132        |
| Max Failed Messages                        | 132        |
| Forward External Errors                    | 133        |
| <b>Communication Setup</b>                 | <b>134</b> |
| Exchange Data Interval                     | 134        |
| Zero Wait Between Successful Exchanges     | 134        |
| Start Exchange Data Schedule               | 135        |
| Stop Exchange Data Schedule                | 136        |
| Down Timeout                               | 136        |
| Up Timeout                                 | 136        |
| Resend Timeout                             | 136        |
| <b>Monk Configuration</b>                  | <b>137</b> |
| Specifying Function or File Names          | 137        |
| Specifying Multiple Directories            | 137        |
| Load Path                                  | 137        |
| Additional Path                            | 137        |
| Auxiliary Library Directories              | 138        |
| Monk Environment Initialization File       | 138        |
| Startup Function                           | 139        |
| Process Outgoing Message Function          | 139        |
| Exchange Data with External Function       | 140        |
| External Connection Establishment Function | 141        |
| External Connection Verification Function  | 142        |
| External Connection Shutdown Function      | 142        |
| Positive Acknowledgment Function           | 143        |

|  |            |
|--|------------|
| Negative Acknowledgment Function               | 144        |
| Shutdown Command Notification Function         | 144        |
| <b>SAP RFC Client Setup</b>                    | <b>146</b> |
| Use Load Balancing                             | 146        |
| Host Name of the R/3 Target System             | 146        |
| Optional Router String                         | 146        |
| System Number of the R/3 Target System         | 147        |
| Client   | 147        |
| User   | 148        |
| Password                                       | 148        |
| Language                                       | 148        |
| Enable RFC Trace                               | 148        |
| Launch ABAP4 Debug Window                      | 148        |
| Optional Gateway Host Name                     | 149        |
| Optional Gateway Service                       | 149        |
| Target System ID for Load Balancing            | 150        |
| Application Server Group for Load Balancing    | 150        |
| Log File for Failed Records                    | 150        |
| Maximum Number of Failures Before Erroring Out | 151        |
| Enforce Transactional RFC                      | 151        |
| Transaction ID Verification Database           | 151        |
| <b>SAP RFC Server Setup</b>                    | <b>153</b> |
| Gateway Host Name                              | 153        |
| Optional Router String                         | 153        |
| Gateway Service                                | 154        |
| Program ID                                     | 154        |
| Wait for Request Interval                      | 154        |
| Wait for eGate Interval                        | 155        |
| Maximum Response Wait                          | 155        |
| Trace  | 155        |
| Log File for Failed Records                    | 155        |
| Maximum Number of Failures Before Erroring Out | 155        |
| Enforce Transactional RFC                      | 156        |
| Transaction ID Verification Database           | 156        |
| OnCheckTID Monk Function                       | 157        |
| OnCommit Monk Function                         | 157        |
| OnRollback Monk Function                       | 157        |
| OnConfirmTid Monk Function                     | 158        |
| <b>SAP tRFC RDBMS Setup</b>                    | <b>159</b> |
| Enable RDBMS TID Management                    | 159        |
| Database Type                                  | 160        |
| Database Name                                  | 160        |
| User Name                                      | 160        |
| Encrypted Password                             | 160        |
| Client TID Table Name                          | 160        |
| Server TID Table Name                          | 161        |

---

## Chapter 8

|                                   |            |
|-----------------------------------|------------|
| <b>API Functions</b>              | <b>162</b> |
| <b>Overview</b>                   | <b>162</b> |
| Data Types and Function Templates | 163        |
| BAPI Transport Layer              | 163        |
| RFC Transport Layer               | 163        |
| e*Way Kernel Layer                | 163        |
| <b>BAPI e*Way Data Types</b>      | <b>164</b> |
| saprfc-conn-handle                | 164        |

|                                     |            |
|-------------------------------------|------------|
| saprfc-conn-opt                     | 164        |
| saprfc-status                       | 164        |
| saprfc-par-list                     | 165        |
| saprfc-tab-list                     | 165        |
| <b>BAPI e*Way Utility Functions</b> | <b>166</b> |
| sapbapi-get-laststatus              | 166        |
| sapbapi-init                        | 166        |
| saprfc-getlasterror                 | 167        |
| saprfc-init                         | 167        |
| <b>Standard Function Templates</b>  | <b>169</b> |
| <b>Standard RFC Operation</b>       | <b>169</b> |
| ewtosapbapi-startup                 | 169        |
| ewtosapbapi-reconnect               | 170        |
| ewtosapbapi-exit                    | 170        |
| sapbapitoew-polling                 | 171        |
| sapbapitoew-exit                    | 171        |
| sapbapitofrew-polling               | 172        |
| sapbapitofrew-exit                  | 172        |
| sapbapi-ack                         | 172        |
| sapbapi-nak                         | 173        |
| sapbapi-nack                        | 173        |
| sapbapi-return-empty-string         | 174        |
| sapbapi-shutdown                    | 174        |
| sapbapi-verify-connect              | 174        |
| <b>Transactional RFC Operation</b>  | <b>176</b> |
| ewtosaptrfc-startup                 | 176        |
| ewtosaptrfc-reconnect               | 177        |
| ewtosaptrfc-exit                    | 177        |
| saptrfctoew-polling                 | 177        |
| saptrfctoew-exit                    | 178        |
| saptrfctofrew-polling               | 178        |
| saptrfctofrew-exit                  | 179        |
| saptrfc-ack                         | 179        |
| saptrfc-nak                         | 179        |
| saptrfc-nack                        | 180        |
| saptrfc-init                        | 180        |
| saptrfc-return-empty-string         | 180        |
| saptrfc-shutdown                    | 181        |
| saptrfc-verify-connect              | 181        |
| <b>BAPI Transport Layer</b>         | <b>182</b> |
| <b>SAP BAPI Functions</b>           | <b>182</b> |
| sapbapi-client-connect              | 182        |
| sapbapi-client-disconnect           | 183        |
| sapbapi-client-openex               | 183        |
| sapbapi-server-pollrequest-dispatch | 184        |
| sapbapi-server-register             | 184        |
| sapbapi-server-unregister           | 185        |
| sapbapi-struct-call                 | 185        |
| sapbapi-struct-fetch                | 186        |
| sapbapi-struct-installfunction      | 186        |
| sapbapi-struct-raise                | 187        |
| sapbapi-struct-resetall             | 188        |
| sapbapi-struct-resetexception       | 188        |
| sapbapi-struct-resetexport          | 189        |
| sapbapi-struct-resetimport          | 189        |
| sapbapi-struct-resettables          | 190        |
| sapbapi-struct-send                 | 190        |
| <b>RFC Transport Layer</b>          | <b>192</b> |
| <b>SAP RFC Client Functions</b>     | <b>192</b> |
| saprfc-client-callreceive           | 192        |



|  |            |
|--|------------|
| saprfc-client-connect                        | 193        |
| saprfc-client-createtid                      | 193        |
| saprfc-client-disconnect                     | 194        |
| saprfc-client-indirectcall                   | 194        |
| saprfc-client-openex                         | 195        |
| saprfc-conn-abort                            | 195        |
| saprfc-conn-createopt                        | 196        |
| saprfc-conn-handle?                          | 196        |
| saprfc-conn-opt?                             | 197        |
| saprfc-conn-set-clientconnmode               | 197        |
| saprfc-conn-set-clientconnopt-cpic           | 198        |
| saprfc-conn-set-clientconnopt-r3only         | 199        |
| saprfc-conn-set-clientconnpar                | 199        |
| saprfc-conn-settrace                         | 200        |
| <b>SAP RFC Server Functions</b>              | <b>201</b> |
| saprfc-server-getcallbackfailuretid          | 201        |
| saprfc-server-getcallbackfailuretype         | 202        |
| saprfc-server-getinputdata                   | 202        |
| saprfc-server-installfunction                | 203        |
| saprfc-server-installtransctrl               | 204        |
| saprfc-server-resetcallbackfailure           | 205        |
| saprfc-server-sendoutputdata                 | 206        |
| saprfc-server-shutdown                       | 206        |
| saprfc-server-startup                        | 207        |
| saprfc-server-waitanddispatch                | 208        |
| <b>SAP tRFC Server Functions</b>             | <b>209</b> |
| saptrfc-commit-tid                           | 209        |
| saptrfc-delete-tid                           | 210        |
| saptrfc-get-current-event-id                 | 210        |
| saptrfc-get-tid                              | 211        |
| saptrfc-on-check-tid                         | 211        |
| saptrfc-on-commit                            | 212        |
| saptrfc-on-commit-tid                        | 213        |
| saptrfc-on-confirm-tid                       | 213        |
| saptrfc-on-rollback                          | 214        |
| saptrfc-on-rollback-tid                      | 215        |
| saptrfc-receive-idoc4                        | 215        |
| saptrfc-send-idoc4                           | 216        |
| saptrfc-struct-call                          | 216        |
| <b>SAP TID Database Management Functions</b> | <b>218</b> |
| saptrfc-tid-db-bind                          | 218        |
| saptrfc-tid-db-delete                        | 219        |
| saptrfc-tid-db-insert                        | 219        |
| saptrfc-tid-db-on-check                      | 220        |
| saptrfc-tid-db-on-commit                     | 220        |
| saptrfc-tid-db-on-confirm                    | 221        |
| saptrfc-tid-db-on-rollback                   | 221        |
| saptrfc-tid-db-reconnect                     | 222        |
| saptrfc-tid-db-reserve                       | 222        |
| saptrfc-tid-db-select                        | 223        |
| saptrfc-tid-db-update                        | 223        |
| saptrfc-tid-file-delete                      | 224        |
| saptrfc-tid-file-on-check                    | 224        |
| saptrfc-tid-file-on-commit                   | 225        |
| saptrfc-tid-file-on-confirm                  | 225        |
| saptrfc-tid-file-on-rollback                 | 226        |
| saptrfc-tid-file-reserve                     | 226        |
| <b>SAP Custom Structure Functions</b>        | <b>228</b> |
| saprfc-struct-add-entry                      | 228        |
| saprfc-struct-create                         | 229        |
| saprfc-struct-handle?                        | 229        |
| saprfc-struct-install                        | 230        |
| <b>SAP Parameter List Functions</b>          | <b>231</b> |

|                                     |            |
|-------------------------------------|------------|
| saprfc-par-add                      | 231        |
| saprfc-par-add-char                 | 232        |
| saprfc-par-add-float                | 233        |
| saprfc-par-add-int                  | 234        |
| saprfc-par-add-receiving            | 234        |
| saprfc-par-bcd->char                | 235        |
| saprfc-par-char->bcd                | 236        |
| saprfc-par-createlist               | 236        |
| saprfc-par-get                      | 237        |
| saprfc-par-get-char                 | 237        |
| saprfc-par-get-float                | 238        |
| saprfc-par-get-int                  | 238        |
| saprfc-par-list?                    | 239        |
| saprfc-par-pad                      | 239        |
| <b>SAP Table List Functions</b>     | <b>241</b> |
| saprfc-tab-appline                  | 241        |
| saprfc-tab-applines                 | 242        |
| saprfc-tab-clear                    | 242        |
| saprfc-tab-countline                | 243        |
| saprfc-tab-create                   | 244        |
| saprfc-tab-createlist               | 244        |
| saprfc-tab-getline                  | 245        |
| saprfc-tab-getwidth                 | 245        |
| saprfc-tab-list?                    | 246        |
| <b>e*Way Kernel Layer</b>           | <b>247</b> |
| <b>Generic e*Way Functions</b>      | <b>247</b> |
| event-commit-to-egate               | 247        |
| event-rollback-to-egate             | 248        |
| event-send-to-egate                 | 248        |
| event-send-to-egate-ignore-shutdown | 249        |
| event-send-to-egate-no-commit       | 249        |
| get-logical-name                    | 250        |
| insert-exchange-data-event          | 250        |
| send-external-up                    | 251        |
| send-external-down                  | 251        |
| shutdown-request                    | 252        |
| start-schedule                      | 252        |
| stop-schedule                       | 253        |
| waiting-to-shutdown                 | 253        |
| <b>Index</b>                        | <b>254</b> |

# 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:

- Operation and administration of the appropriate operating systems (see [Supported Operating Systems](#) on page 18)
- Windows-style GUI operations
- SAP BAPI concepts and operations
- Integrating SAP R/3 with external systems

---

## P.2 Organization

This User's Guide is organized 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 (BAPI) is frequently referred to as the SAP BAPI 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
```

### 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.

---

## P.6 Additional Documentation

Many of the procedures included in this User's Guide are described in greater detail in the *e\*Gate Integrator User's Guide*. For more information on the Java Collaboration Service, see the *e\*Gate Integrator Collaboration Services Reference Guide*.

Comprehensive information on SAP R/3 can be found at the SAP Help Portal:

<http://help.sap.com>

Once you have selected the appropriate SAP R/3 version and language, you are presented with the SAP Library, which contains an index and a search facility (**Find**). Under *Business Framework Architecture*, you can locate information such as:

- BAPI User Guide
- BAPI Programming Guide
- BAPI Enhancements and Modifications

If you have a SAP customer or partner SAPNet user ID, you also can access the following links for additional information on:

- The SAP Java Connector:  
<https://www013.sap-ag.de/connectors>
- Working with and developing BAPI solutions:  
<https://www013.sap-ag.de/BAPI>

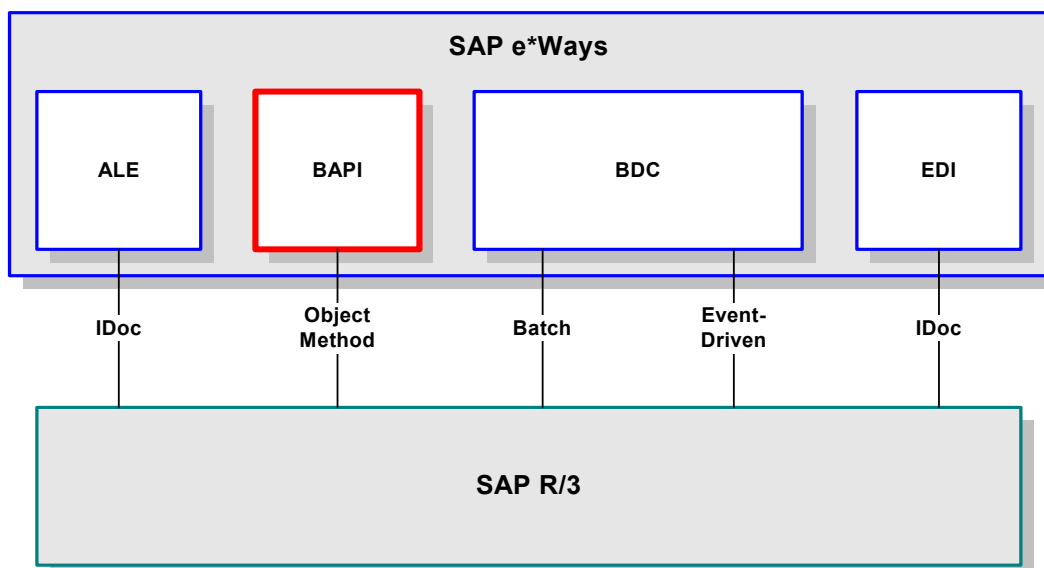
# Introduction

The SeeBeyond e\*Way Intelligent Adapter for SAP (BAPI) 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 system via SAP's Business Application Programming Interface (BAPI).

## 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.

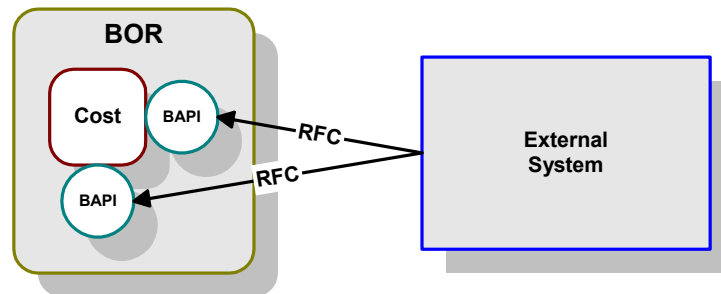
**Figure 1** SAP Interfacing Options



### 1.1.1 What is a BAPI?

**BAPI** is an acronym for **B**usiness **A**pplication **P**rogramming **I**nterface and is SAP's way of providing precise access to processes and data residing in their system. More specifically, BAPIs are **methods** of SAP Business Objects stored in the Business Object Repository (BOR) and are implemented by function modules programmed in ABAP/4. Also, these functions are Remote Function Call (RFC) enabled, and thus can be called by an external process such as the SAP BAPI e\*Way.

**Figure 2** BAPI Example



### 1.1.2 How is BAPI Different from ALE-IDoc?

SAP's ALE-IDoc mechanism also provides access to processes and data residing in the SAP system, but with one major difference: that access is asynchronous. In other words, data retrieved from the SAP system is not guaranteed to be the most current; there could have been several seconds lapse between when data was captured into an IDoc and when it was sent out via ALE to the external system. Also, if data in an IDoc is sent into SAP, it may not be posted immediately into the database tables. Another difference is data communicated through IDocs tend to be overly comprehensive and lengthy.

In contrast, SAP's BAPIs provide synchronous access to SAP. That is, a request by an external process for data to be retrieved from or posted into the SAP system is executed immediately and control is returned to the caller only when the transaction has completed (or failed). Moreover, the data exchanged is brief as compared with IDocs.

| Topic         | ALE-IDoc                          | BAPI                |
|---------------|-----------------------------------|---------------------|
| Communication | Asynchronous                      | Synchronous         |
| Data from SAP | Sending may lag capture into IDoc | Immediate sending   |
| Data to SAP   | Posting may lag receipt of data   | Immediate posting   |
| Data content  | Lengthy and overly comprehensive  | Brief and selective |

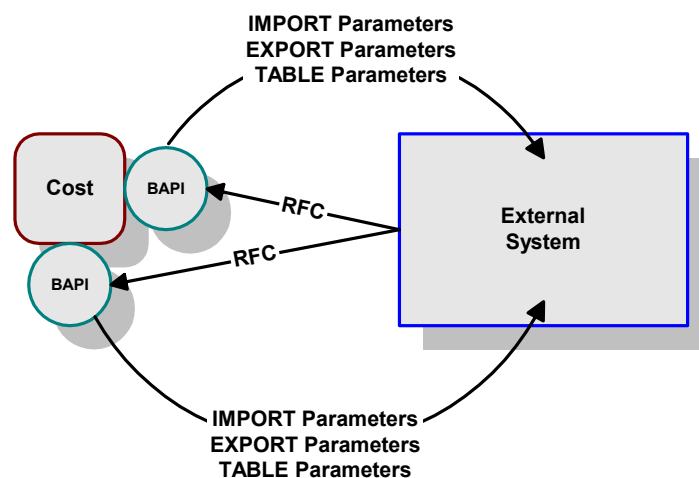
### 1.1.3 How are BAPIs Invoked?

Before it can be invoked, a BAPI (or any RFC-enabled function module on SAP for that matter) requires the following:

- **IMPORT** parameters - data to be provided to the BAPI
- **EXPORT** parameters - data to be returned by the BAPI
- **TABLE** parameters - data that may be provided to and/or returned by the BAPI

The detailed metadata for these parameters such as descriptions of their value types, mandatory or optional nature, can be perused under SAP transaction **SE37** and can prove rather tedious to implement.

**Figure 3** Required Information



The functionality of the SAP BAPI e\*Way simplifies the whole process of determining the requisite **IMPORT**, **EXPORT** and **TABLE** parameters—marshalling all the necessary data using the correct type and format, calling the ABAP/4 function module that represents the BAPI, and then extracting and parsing data from the **EXPORT** and/or **TABLE** parameters.

### 1.1.4 Security Issues

To provide an acceptable and secure B2B environment, SAP recommends that the e\*Gate user be of a special user class to limit interaction with the system. This class is either:

- **System** (in recent SAP R/3 versions)
- **CPI-C** (in earlier SAP R/3 versions)

These user classes are limited to external applications, since neither can log on to a SAPGUI session.



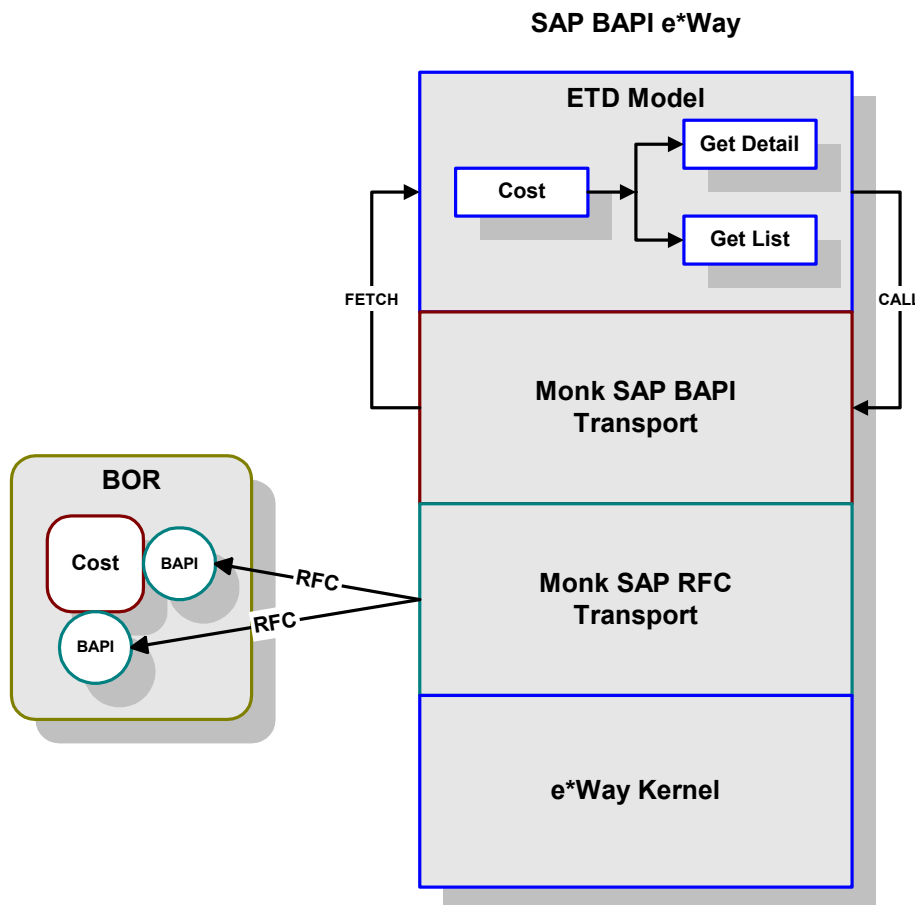
## 1.2 The SAP BAPI e\*Way

### 1.2.1 Overview

The SAP BAPI e\*Way replaces the External System of Figure 2 with a multi-layered structure as shown in Figure 4, which consists of:

- Event Type Definition (ETD) Model layer, which translates SAP Business Objects into Event Type Definitions
- Monk SAP BAPI Transport layer, which manages communication to and from BAPIs
- 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 4** SAP BAPI e\*Way



This structure is explained in detail in [Chapter 6](#).

## 1.3 Supported Operating Systems

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

**Table 1** English-language Version

| Operating System                                | SAP R/3 & BASIS |      |      |      |     |     |
|---|-----------------|------|------|------|-----|-----|
|   | 4.0B            | 4.5B | 4.6B | 4.6C | 4.7 | 6.2 |
| Windows 2000 and Windows Server 2003            | X               | X    | X    | X    | X   | X   |
| HP Tru64 5.1A                                   | X               | X    | X    | X    | X   | X   |
| HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23) | X               | X    | X    | X    | X   | X   |
| IBM AIX 5.1L                                    | X               | X    | X    | X    | X   | X   |
| Sun Solaris 8                                   | X               | X    | X    | X    | X   | X   |

**Table 2** Japanese-language Version

| Operating System                                | SAP R/3 & BASIS |      |      |      |     |     |
|---|-----------------|------|------|------|-----|-----|
|   | 4.0B            | 4.5B | 4.6B | 4.6C | 4.7 | 6.2 |
| Windows 2000 and Windows Server 2003            | -               | -    | -    | X    | X   | X   |
| HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23) | -               | -    | -    | X    | X   | X   |
| Sun Solaris 8                                   | -               | -    | -    | X    | X   | X   |

**Table 3** Korean-language Version

| Operating System                                | SAP R/3 & BASIS |      |      |      |     |     |
|---|-----------------|------|------|------|-----|-----|
|   | 4.0B            | 4.5B | 4.6B | 4.6C | 4.7 | 6.2 |
| HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23) | -               | -    | -    | X    | X   | X   |
| IBM AIX 5.1L                                    | -               | -    | -    | X    | X   | X   |
| Sun Solaris 8                                   | -               | -    | -    | X    | X   | 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\ewsapbapi` 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 BAPI, 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 (not including sample achemas):
  - ♦ Approximately 1.3 MB on Windows systems
  - ♦ Approximately 1.5 MB on Solaris systems
  - ♦ Approximately 1.5 MB on HP-UX systems
  - ♦ Approximately 1.3 MB on AIX systems
  - ♦ Approximately 1.5 MB on Compaq Tru64 systems

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 BAPI supports the following applications (see also [Supported Operating Systems](#) on page 18).

### English

- SAP R/3 release 4.0B, 4.5B, 4.6B, 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 58).

## 2.3 Installing the e\*Way

### 2.3.1 Windows Systems

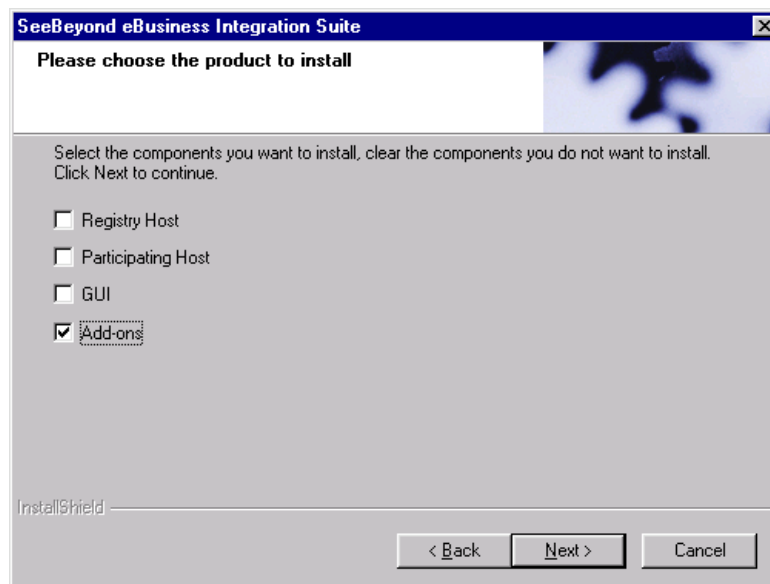
#### Installation Procedure

**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 5). Check **Add-ons**, then click **Next**.

**Figure 5** Choose Product Dialog Box

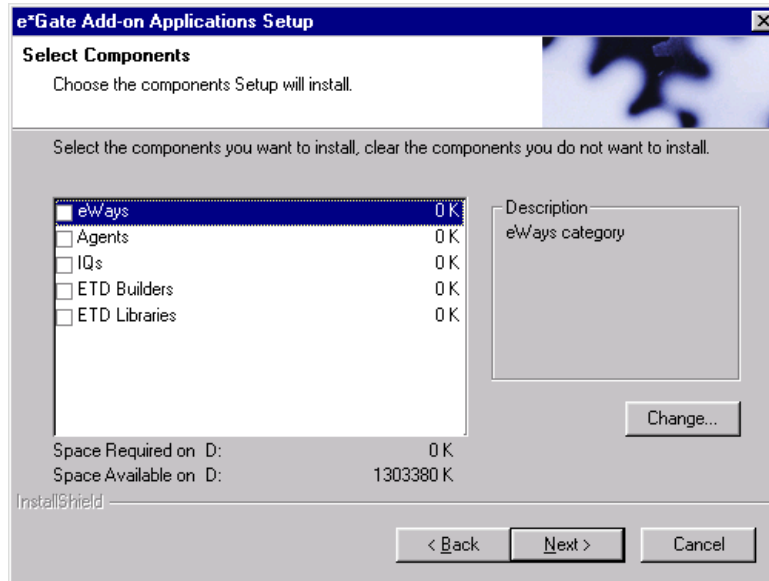


- B If the setup program does not launch automatically, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the following file on the CD-ROM drive (this bypasses the **Choose Product** dialog):

setup\addons\setup.exe

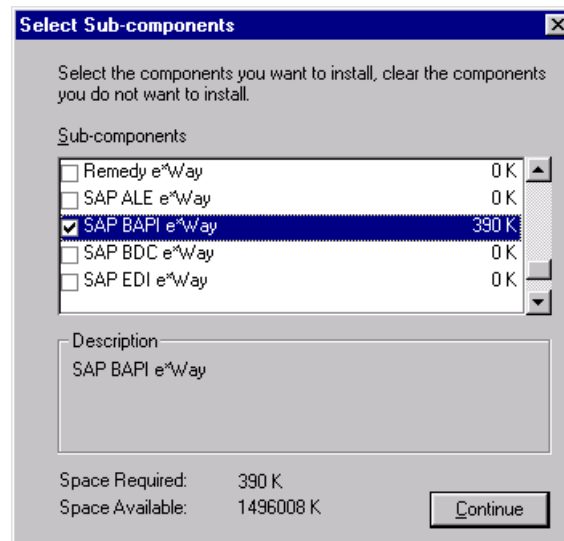
- 5 Follow the on-screen instructions until the **Select Components** dialog box appears (see Figure 6). Highlight—but do not check—**eWays** and then click **Change**.

**Figure 6** Select Components Dialog Box



- 6 When the **Select Sub-components** dialog box appears (see Figure 7), check the **SAP BAPI e\*Way**.

**Figure 7** 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

**Note:** *Installing the e\*Way Intelligent Adapter for SAP BAPI installs both Java and Monk versions. Only the files used by the Monk version are listed in this section.*

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

**Table 4** Registry Host Only

| Subdirectory | File             |
|--------------|------------------|
| \            | stcewsapbapi.ctl |

By default, the InstallShield installer also 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 5** Participating Host & Registry Host

| Subdirectories             | Files  |
|----------------------------|--|
| \bin\                      | librfc32.dll<br>stcbapiconvert.exe<br>stc_monksap.ctl<br>stc_monksap.dll |
| \configs\stcewgenericmonk\ | ewsapbapi.def<br>sapBapi3.6To4.1Rule.txt                                 |
| \monk_library\             | ewsapbapi.gui<br>saprfc.gui  |

**Table 5** Participating Host & Registry Host

| Subdirectories           | Files  |
|--------------------------|--|
| \monk_library\ewsapbapi\ | sapbapi.monk<br>sapbapi-client-connect.monk<br>sapbapi-client-disconnect.monk<br>sapbapi-client-openex.monk<br>sapbapi-get-laststatus.monk<br>sapbapi-init.monk<br>sapbapi-server-pollrequest-dispatch.monk<br>sapbapi-server-register.monk<br>sapbapi-server-unregister.monk<br>sapbapi-struct-call.monk<br>sapbapi-struct-fetch.monk<br>sapbapi-struct-installfunction.monk<br>sapbapi-struct-raise.monk<br>sapbapi-struct-resetall.monk<br>sapbapi-struct-resetexception.monk<br>sapbapi-struct-resetexport.monk<br>sapbapi-struct-resetimport.monk<br>sapbapi-struct-resettables.monk<br>sapbapi-struct-send.monk<br>saprfc-init.monk<br>saprfc-par-pad.monk<br>saptrfc-commit-tid.monk<br>saptrfc-delete-tid.monk<br>saptrfc-get-current-event-id.monk<br>saptrfc-get-tid.monk<br>saptrfc-on-check-tid.monk<br>saptrfc-on-commit.monk<br>saptrfc-on-confirm-tid.monk<br>saptrfc-on-rollback.monk<br>saptrfc-receive-idoc4.monk<br>saptrfc-send-idoc4.monk<br>saptrfc-stdver-eway-funcs.monk<br>saptrfc-struct-call.monk<br>saptrfc-tid-db-bind.monk<br>saptrfc-tid-db-delete.monk<br>saptrfc-tid-db-insert.monk<br>saptrfc-tid-db-on-check.monk<br>saptrfc-tid-db-on-commit.monk<br>saptrfc-tid-db-on-confirm.monk<br>saptrfc-tid-db-on-rollback.monk<br>saptrfc-tid-db-reconnect.monk<br>saptrfc-tid-db-reserve.monk<br>saptrfc-tid-db-select.monk<br>saptrfc-tid-db-update.monk |
| \stcgui\ctls\            | guisapbapi.ctl   |



## 2.3.2 UNIX Systems

### Installation Procedure

**Note:** You are not required to have root privileges to install this e\*Way..

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

**Note:** Installing the e\*Way Intelligent Adapter for SAP BAPI installs both Java and Monk versions. Only the files used by the Monk version are listed in this section.

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

**Table 6** Registry Host Only

| Subdirectory | File             |
|--------------|------------------|
| /            | stcewsapbapi.ctl |

The preceding installation procedure also 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 7** Participating Host & Registry Host

| Subdirectories | Files   |
|----------------|---|
| /bin/          | lbrfccm.sl (HP-UX only)<br>stc_monksap.ctl<br>stc_monksap.dll |

**Table 7** Participating Host & Registry Host

| Subdirectories             | Files  |
|----------------------------|--|
| /configs/stcewgenericmonk/ | ewsapbapi.def<br>sapBapi3.6To4.1Rule.txt   |
| /monk_library/             | ewsapbapi.gui<br>saprfc.gui  |
| /monk_library/ewsapbapi/   | sapbapi.monk<br>sapbapi-client-connect.monk<br>sapbapi-client-disconnect.monk<br>sapbapi-client-openex.monk<br>sapbapi-get-laststatus.monk<br>sapbapi-init.monk<br>sapbapi-server-pollrequest-dispatch.monk<br>sapbapi-server-register.monk<br>sapbapi-server-unregister.monk<br>sapbapi-struct-call.monk<br>sapbapi-struct-fetch.monk<br>sapbapi-struct-installfunction.monk<br>sapbapi-struct-raise.monk<br>sapbapi-struct-resetall.monk<br>sapbapi-struct-resetexception.monk<br>sapbapi-struct-resetexport.monk<br>sapbapi-struct-resetimport.monk<br>sapbapi-struct-resettables.monk<br>sapbapi-struct-send.monk<br>saprfc-init.monk<br>saprfc-par-pad.monk<br>saptrfc-commit-tid.monk<br>saptrfc-delete-tid.monk<br>saptrfc-get-current-event-id.monk<br>saptrfc-get-tid.monk<br>saptrfc-on-check-tid.monk<br>saptrfc-on-commit.monk<br>saptrfc-on-confirm-tid.monk<br>saptrfc-on-rollback.monk<br>saptrfc-receive-idoc4.monk<br>saptrfc-send-idoc4.monk<br>saptrfc-stdver-eway-funcs.monk<br>saptrfc-struct-call.monk<br>saptrfc-tid-db-bind.monk<br>saptrfc-tid-db-delete.monk<br>saptrfc-tid-db-insert.monk<br>saptrfc-tid-db-on-check.monk<br>saptrfc-tid-db-on-commit.monk<br>saptrfc-tid-db-on-confirm.monk<br>saptrfc-tid-db-on-rollback.monk<br>saptrfc-tid-db-reconnect.monk<br>saptrfc-tid-db-reserve.monk<br>saptrfc-tid-db-select.monk<br>saptrfc-tid-db-update.monk |
| \\stcgui\ctls\             | guisapbapi.ctl   |

## 2.4 Optional Example Files

The installation CD-ROM contains sample schemas in the `samples\ewsapbapi` directory.

- `bapi2egate` (RFC Server example, Poll and Extract)
- `egate2bapi` (RFC Client example)
- `TrfcFromBapi` (tRFC Server example)
- `TrfcToBapi` (tRFC Client example)

To use these schemas, you must load them onto your system using the following procedure. See [Sample Schemas](#) on page 45 for descriptions of the sample schemas and instructions regarding its use.

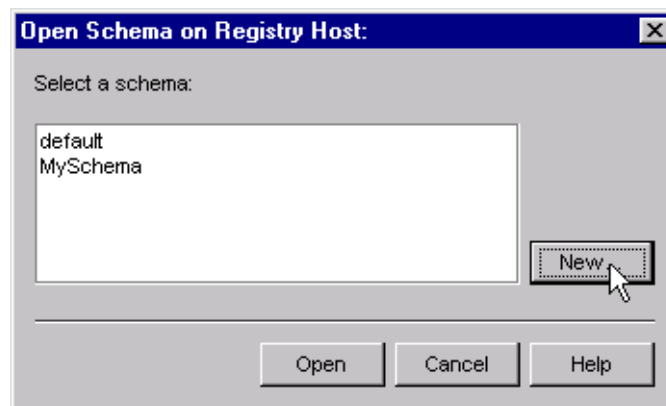
*Note:* The SAP BAPI e\*Way must be properly installed on your system before you can run the sample schema.

### 2.4.1 Installation Procedure

To load a sample schema

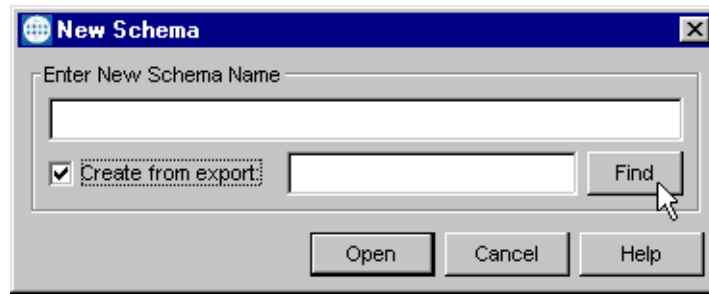
- 1 Invoke the **Open Schema** dialog box and select **New** (see Figure 8).

**Figure 8** Open Schema Dialog



- 2 Type the name you want to give to the schema (for example, `bapi2egateSample`)
- 3 Select **Create from export** and navigate to the directory containing the sample schema by clicking the **Find** button (see Figure 9).

**Figure 9** New Schema Dialog



- 4 Select the desired **.zip** file 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** Sample Schema - bapi2egate

| Subdirectories                              | Files  |
|---|--|
| \runtime\                                   | bapi2egate.ctl   |
| \runtime\configs\stcewgenericmonk\          | ewCostCenterServer.cfg<br>ewCostCenterServer.sc  |
| \runtime\data\input\                        | readmeinput.txt  |
| \sandbox\Administrator\monk_scripts\common\ | bapitoewCostCenter-handler.dsc<br>bapitoewCostCenter-reconnect.dsc<br>CostCenter.ssc<br>etResult.ssc |

**Table 9** Sample Schema - egate2bapi

| Subdirectories                              | Files  |
|---|--|
| \runtime\                                   | egate2bapi.ctl   |
| \runtime\configs\stcewfile\                 | ewEater.cfg<br>ewEater.sc<br>ewFeeder.cfg<br>ewFeeder.sc |
| \runtime\configs\stcewgenericmonk\          | ewBAPI.cfg<br>ewBAPI.sc                                  |
| \runtime\data\input\                        | CostCenter.bapi  |
| \runtime\data\output\                       | output0.dat  |
| \runtime\monk_scripts\common\               | dgwto sapbapi-startup.monk<br>ewto sapbapi-getlist.tsc   |
| \sandbox\Administrator\monk_scripts\common\ | crBAPI.tsc<br>etCostCenter.ssc<br>etInbound.ssc          |

**Table 10** Sample Schema - TrfcFromBapi

| Subdirectories                     | Files  |
|------------------------------------|--|
| \runtime\                          | TrfcFromBapi.ctl   |
| \runtime\client\data\test\output\  | output0.dat  |
| \runtime\configs\stcewfile\        | EaterBAPI.cfg<br>EaterBAPI.sc  |
| \runtime\configs\stcewgenericmonk\ | sapbapifromsap.cfg<br>sapbapifromsap.sc  |
| \runtime\monk_scripts\common\      | idoc_inbound_asyn.ssc<br>idocfromsaptoegate-reconnect.dsc<br>idocfromsaptoegate-startup.monk<br>result.ssc |

**Table 11** Sample Schema - TrfcToBapi

| Subdirectories                     | Files                            |
|------------------------------------|----------------------------------|
| \runtime\                          | TrfcToBapi.ctl                   |
| \runtime\client\data\test\input\   | idoc_order.dat                   |
| \runtime\configs\stcewfile\        | feeder.cfg<br>feeder.sc          |
| \runtime\configs\stcewgenericmonk\ | ewbapi.cfg<br>ewbapi.sc          |
| \runtime\monk_scripts\common\      | idoc4.ssc<br>send-idoc-tosap.tsc |

# 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.

Several sample schemas, included in the software package, are described at the end of this chapter. These can be used to test your system following installation and, if appropriate, as a template that you can modify to produce your own schema.

### 3.1.1 Pre-Implementation Tasks

#### Installation of SeeBeyond Software

The first task, of course, 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 27).

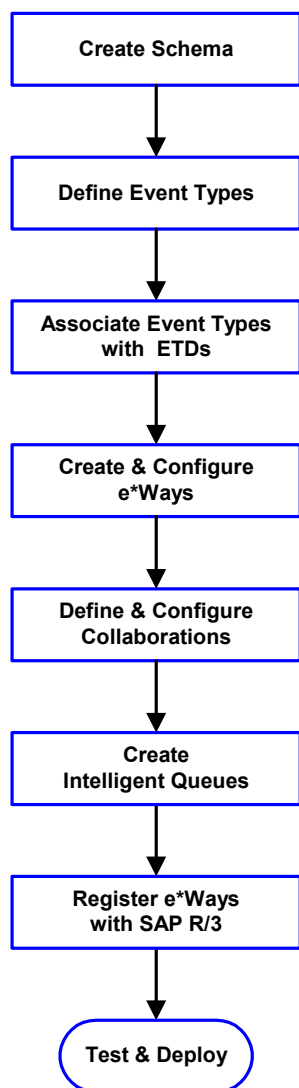
**Note:** *It is highly recommended that you make use of the sample schemas to familiarize yourself with e\*Way operation, test your system, and use as templates for your working schemas.*

#### Importation of Custom ABAP Components

If you want to use the provided mechanism for initiating BAPI calls from SAP to e\*Gate, your SAP R/3 environment must be modified. See [Chapter 4](#).

**Note:** *This extension is optional, since most ABAP calls are initiated from within e\*Gate.*

### 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 32).
- 2 The second step is to define the Event Types you are transporting and processing within the Schema (see [Creating Event Types](#) on page 33).
- 3 Next 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 34).
- 4 The fourth step is to create and configure the required e\*Ways (see [Chapter 5](#)).
- 5 Next you need to define and configure the Collaborations between Event Types (see [Defining Collaborations](#) on page 41).
- 6 Now you need to create Intelligent Queues to hold published Events (see [Creating Intelligent Queues](#) on page 42).
- 7 Before attempting to execute the Schema, you must register the e\*Ways created in step 4 with your SAP R/3 system (see [Registering the e\\*Way](#) on page 42).
- 8 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

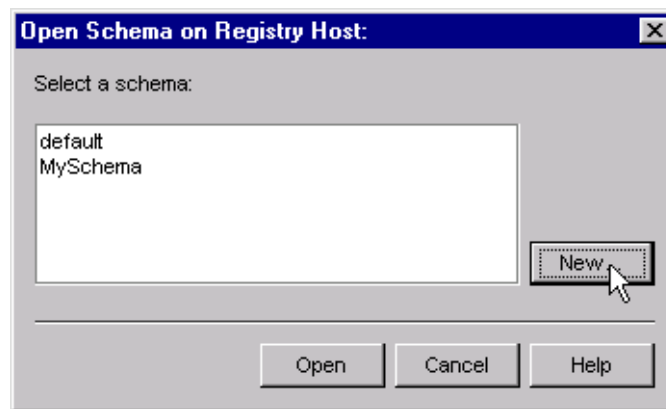
A schema is the structure that defines e\*Gate system parameters and the relationships between components within the e\*Gate system. Schemas can span multiple hosts.

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

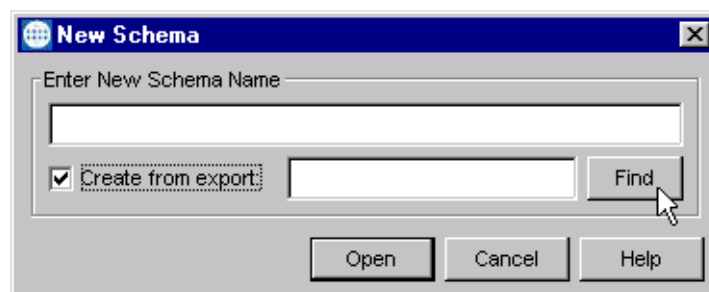
- 1 Invoke the **Open Schema** dialog box and **Open** an existing schema or click **New** to create a new schema.

**Figure 10** Open Schema Dialog



- 2 Clicking **New** invokes the **New Schema** dialog box (Figure 11).

**Figure 11** 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

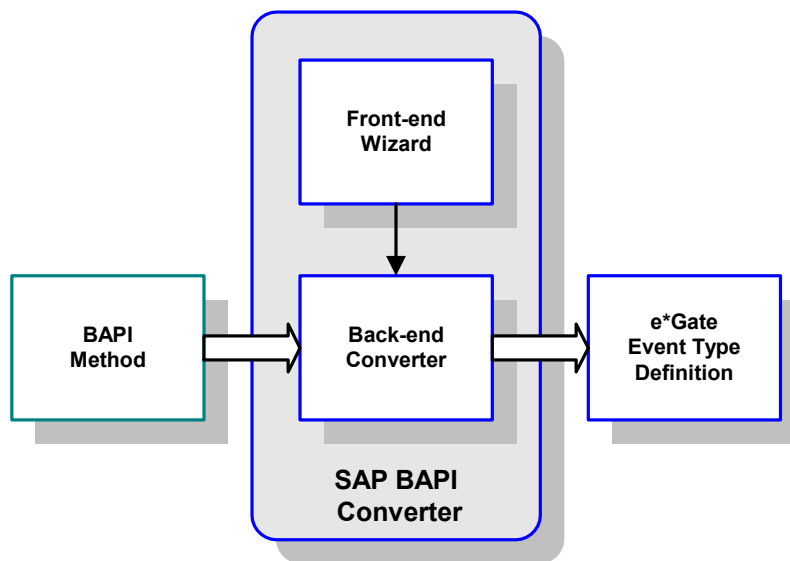
Before e\*Gate can process any data to or from an SAP system, you must create an Event Type Definition to package and route that data within the e\*Gate system. See the *e\*Gate Integrator User's Guide* for additional information about Event Type Definitions and the e\*Gate ETD Editor.

**Note:** *The BAPI ETD serves as a front end to accessing APIs that communicate with SAP, either to call and receive data from SAP ABAP/4 functions or be called by and subsequently return data to ABAP/4 functions. Hence, the ETD is **not** parsable.*

### 3.4.1 The BAPI Structure Builder


The ETD Editor contains the BAPI Structure Builder, which enables you to automatically build an ETD representing a BAPI defined in SAP, using metadata taken dynamically from SAP. Since each BAPI call is unique, the structure builder needs to create the structures corresponding to all required BAPI.

**Figure 12** BAPI Structure Builder/Converter



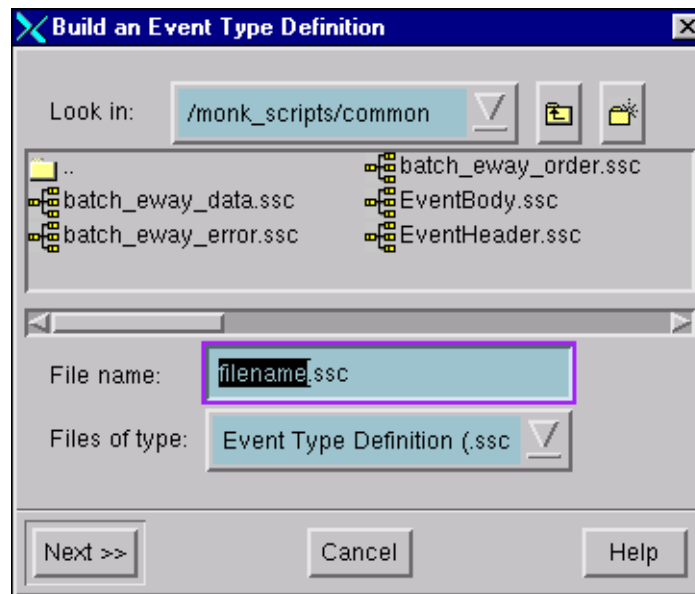
**Note:** Be sure to set the Default Editor to **Monk**, from the **Options** menu in the e\*Gate Schema Designer.

### 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**.

The *Build an Event Type Definition* dialog box opens.

**Figure 13** Build Event Type Definition Dialog

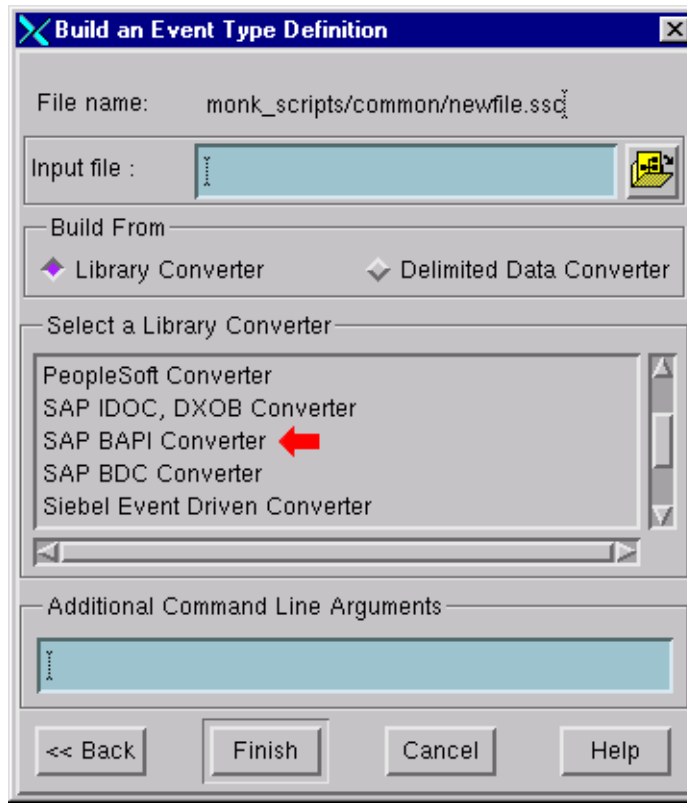


- 3 In the *File name* box, type the name of the ETD file you want to build.

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

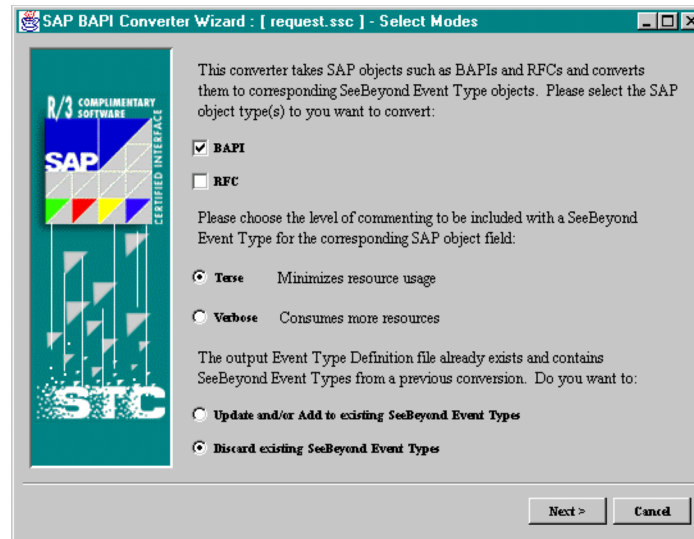
- 4 Click **Next**. A new dialog box appears, as shown in Figure 14.

**Figure 14** Building the ETD



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

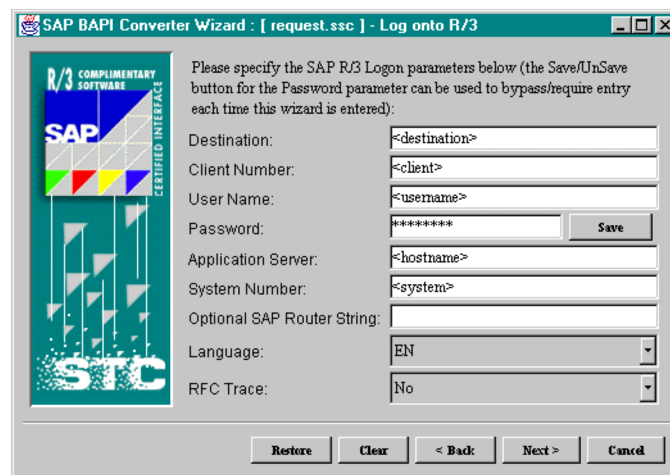
**Figure 15** SAP BAPI Converter Wizard (1)



10 First, the Wizard asks for various modal settings:

- ◆ Whether or not a BAPI and/or an RFC is to be converted. As mentioned before, a RFC-enabled function on SAP is essentially the same as a BAPI method—it is simply not associated with any BOR object.
- ◆ Level of commenting such as **Terse** or **Verbose**. For **Terse** level, only comments regarding value type and length of parameter fields are provided. In addition to those of **Terse**, the **Verbose** level adds the short text description of the parameter fields.
- ◆ When building an ETD file, the Structure Editor requires the name of an output file and if that file already exists and contains existing ETDs, you have the option to update and/or add new converted BAPI ETDs to the existing ones, or to discard them completely and start afresh.

**Figure 16** SAP BAPI Converter Wizard (2)



11 Second, the Wizard asks for SAP Logon information.

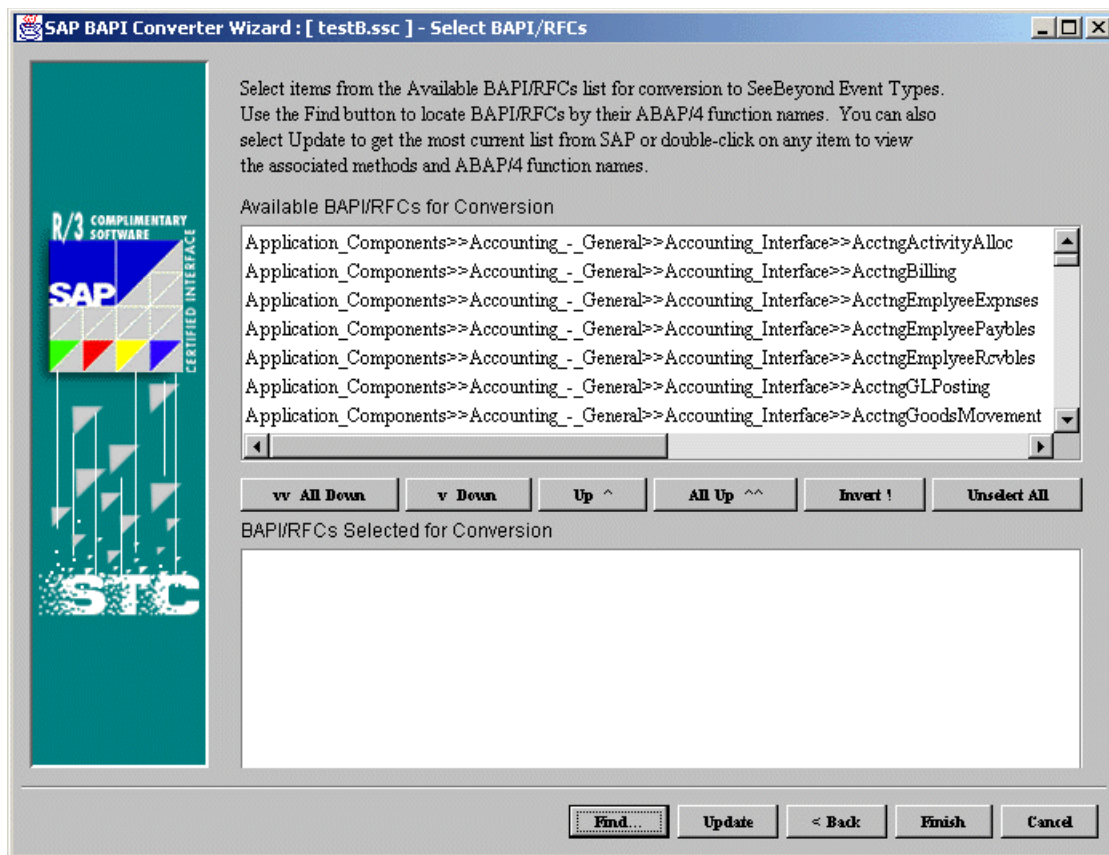
Information that is supplied here is cached per user to facilitate consecutive reuses of the converter. Some notes:

- ◆ Sensitive data such as the password are encrypted before being cached. If password entry is required every time, simply select the **UnSave** button.
- ◆ The **Restore** button restores all logon parameters to the cached values if any.
- ◆ The **Clear** button clears all logon parameter entries.

When all the required parameters are entered and you click **Next**, the wizard checks if there's a cached version of the available BAPI and/or RFC lists. The lists are cached on the local drive according to a combined key of Language type, Application Server, System Number and Destination. If the lists do not exist, the wizard interacts with the SAP system to generate the lists. This operation can take several minutes to complete.

**Note:** The user must be an SAP interactive “*dialog*” user type (*not* a CPIC user type, which is non-interactive).

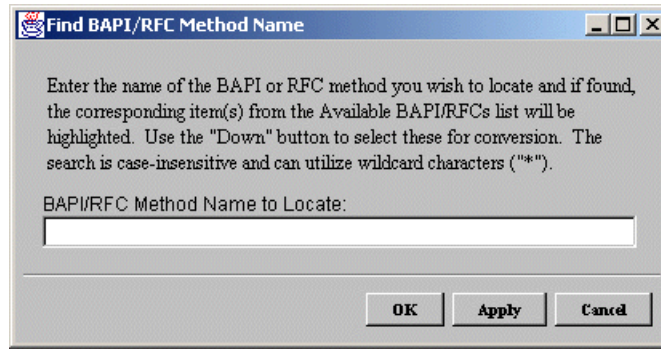
**Figure 17** SAP BAPI Converter Wizard (3)



12 Third, the Wizard asks you to select BAPIs/RFCs for conversion.

- ◆ Select BAPI and/or RFC objects from the top list to deposit into the bottom list for conversion. For BAPIs, the list items represent flattened path names of BAPI objects as presented in the SAP GUI under transaction **BAPI**.
- ◆ Selecting **Find** invokes the **Find BAPI/RFC Method Name** dialog box (Figure 18).

**Figure 18** Find Method Name Dialog Box



- ◆ Type in the name or search pattern for the ABAP/4 function name to locate within the **Available BAPI/RFCs for Conversion** list.

*Note:* Only wildcard characters (\*) can be used—not a regular expression.


- ◆ You may either hit **OK**, in which case the dialog box disappears after the search, or **Apply** which allows the dialog box to remain up for more searches. In either case, at the conclusion of a successful search a pop-up message appears to announce the number of matches found.
- ◆ On the Converter Wizard:
  - ◆ Select the **Down** button in order to reserve the items desired for conversion.
  - ◆ Select the **Back** or **Finish** button to close the dialog box.

Once an object is selected for conversion, all the BAPI methods of the object are exposed in the ensuing ETD. Click **Refresh** to refresh the cached list of BAPIs and/or RFCs after more have been added to the system.

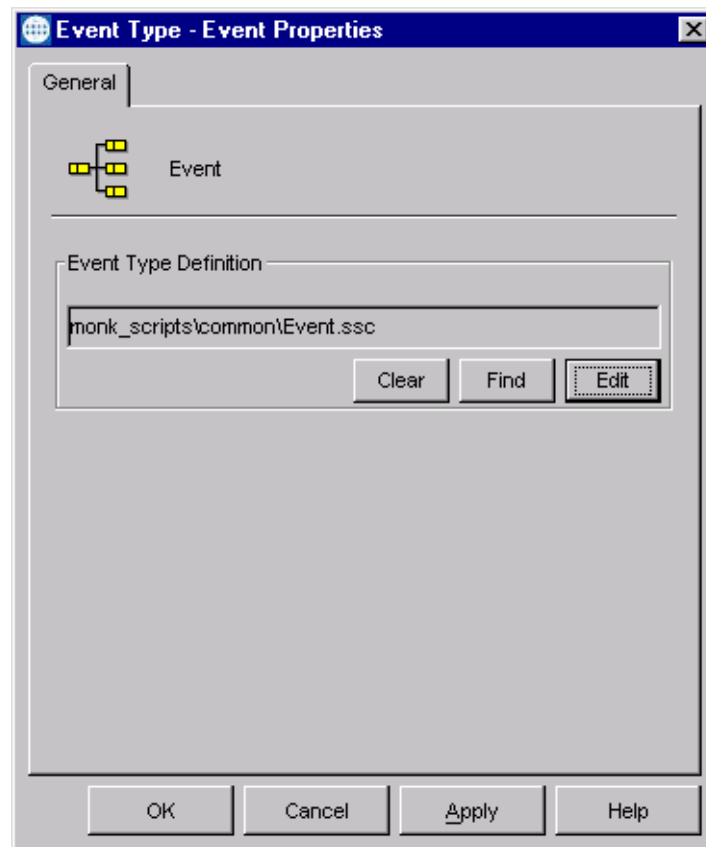
### 3.4.2 Assigning ETDs to Event Types

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 19.

**Figure 19** 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.

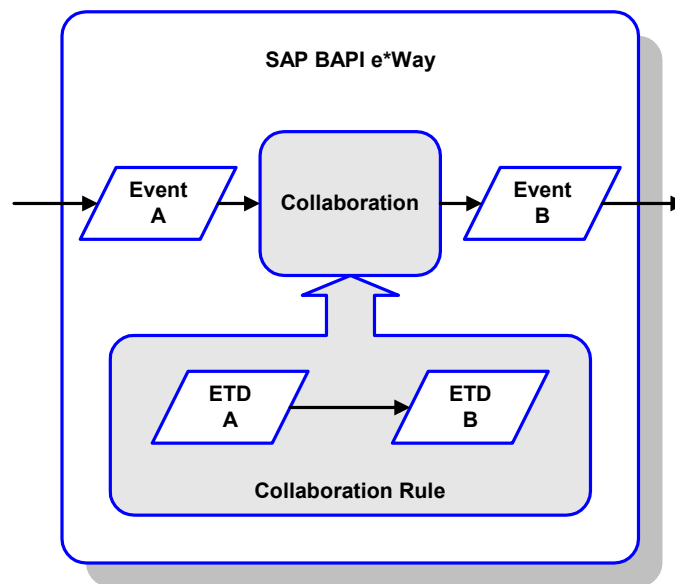
Each Event Type is now associated with the specified Event Type Definition.

## 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 20** Collaborations



The Collaboration is driven by a Collaboration Rule script, which defines the relationship between the incoming and outgoing ETDs. You can use an existing Collaboration Rule script, 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

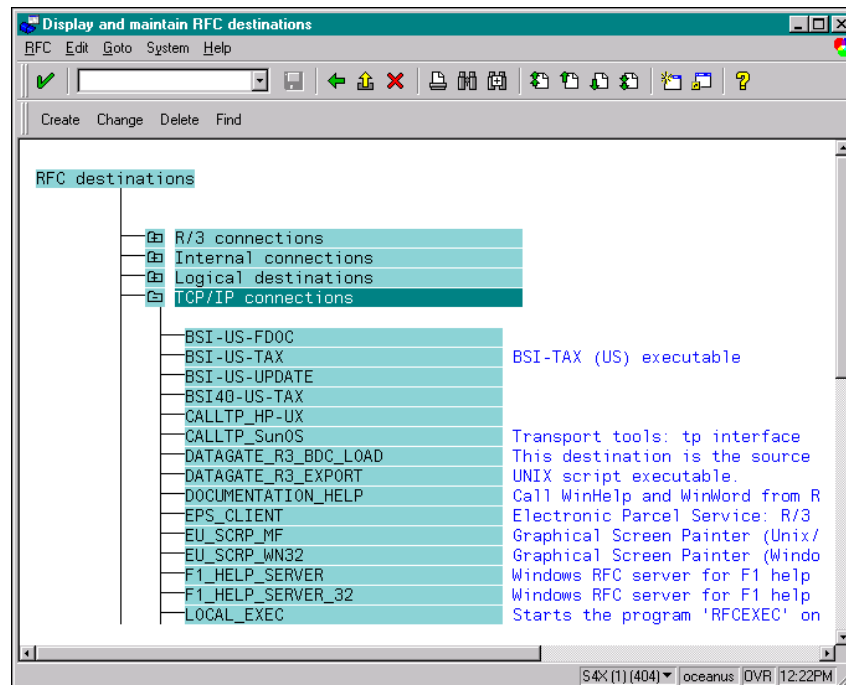
The final step is to create and associate an IQ for the SAP BAPI e\*Way. IQs manage the exchange of information between components within the e\*Gate system, providing non-volatile storage for data as it passes from one component to another. IQs use IQ Services to transport data. IQ Services provide the mechanism for moving Events between IQs, handling the low-level implementation of data exchange (such as system calls to initialize or reorganize a database). See the *e\*Gate Integrator User's Guide* for complete information on queuing options and procedures for creating IQs.

## 3.7 Registering the e\*Way

Transaction: SM59

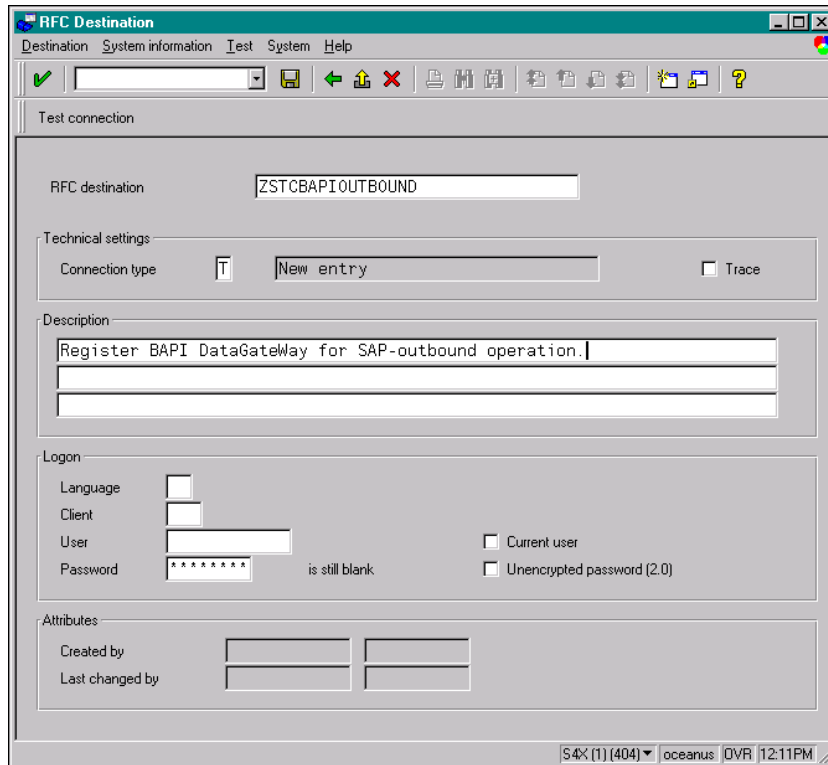
Before the e\*Way can install functions on the SAP system, it must first register its Program ID. This Program ID is associated with an SAP RFC Destination which must be configured.


**Figure 21** RFC Destination Display Window



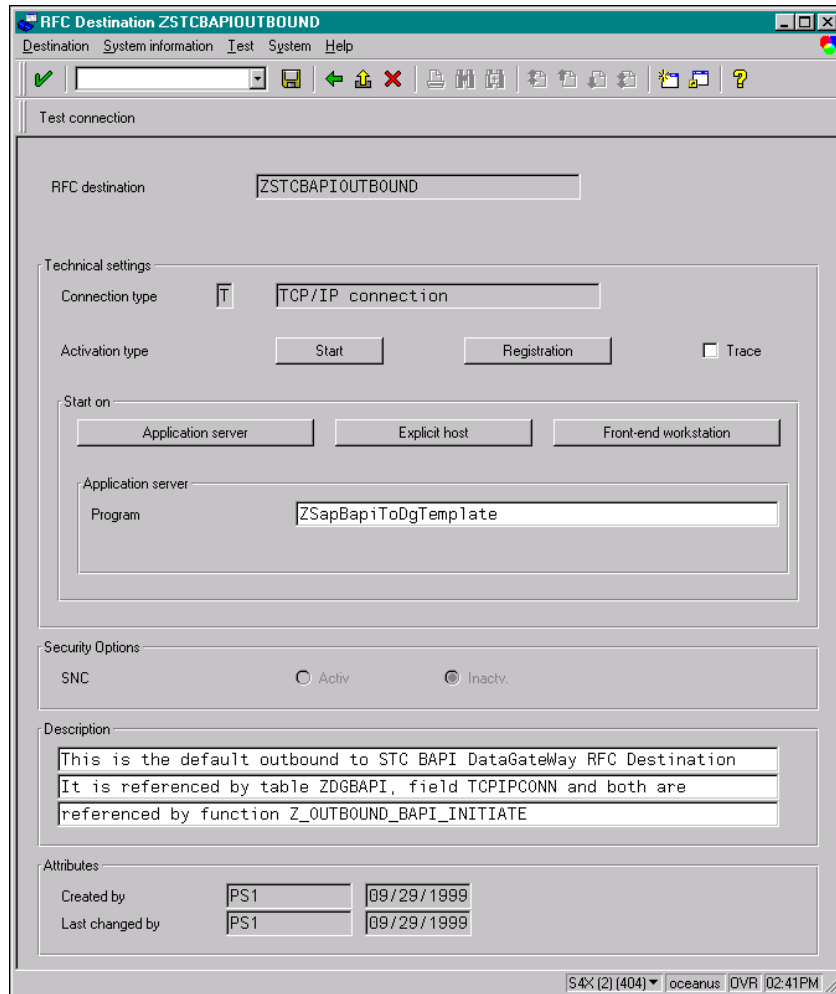
- 1 Select Create, which displays the entry window.


**Figure 22** RFC Destination Entry Window



- 2 Type the desired **RFC destination** into the text box.
- 3 Select Connection Type: **T**
- 4 Select **Test Connection**, then return  on the screen that appears (ignore any warning messages). The *RFC Destination* window for your entry now appears.

**Figure 23** RFC Destination: ZSTCBAPIOUTBOUND



- 5 Type in the name of the e\*Way Program that is currently running.
- 6 Select Activation Type: **Registration**
- 7 Save  and return to the *SAP System* main 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.

---

## 3.8 Sample Schemas

Sample implementations are available in the **samples\ewsapbapi** directory of the e\*Gate CD-ROM.

- **bapi2egate** - RFC Server example
- **egate2bapi** - RFC Client example
- **TrfcFromBapi** - tRFC Server example
- **TrfcToBapi** - tRFC Client example

These samples can be used to test your system following installation and, if appropriate, as templates that you can modify to produce your own schema. Several of the sample schema employ customized Monk functions, which are described in the respective sections.

### To use a sample schema

- 1 Install the sample schema as described in **Optional Example Files** on page 27.
- 2 From the control panel, start the newly registered Control Broker as follows:
  - A Double-click **Services**.
  - B Locate **e\*Gate Control Broker <name of schema>** and select it.
  - C Click **Start**.

### 3.8.1 bapi2egate

This schema illustrates the RFC Server process described in [Server Mode \(SAP to e\\*Gate\)](#) on page 94. The sample schema **bapi2egate** sets up a single instance of the SAP BAPI e\*Way and also of the File e\*Way, having the logical names shown in the following table. It also sets up an Intelligent Queue, with the logical name **BAPI\_EGate\_IQ**.

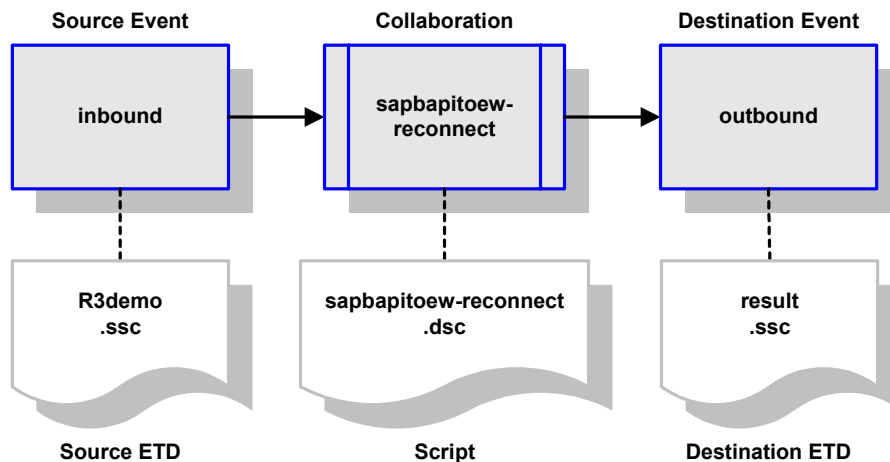
| e*Way Type     | Module LN       | Pub/Sub LN    |
|----------------|-----------------|---------------|
| SAP BAPI e*Way | BAPI_EGate_EWay | BAPI_EGate_SP |
| File e*Way     | Eater           | EaterSP       |

The e\*Way polls SAP R/3 for function calls and extracts any data for processing and routing. Processed data is sent to the IQ for publication to other e\*Gate components and to the File e\*Way, which sends any results to SAP R/3.

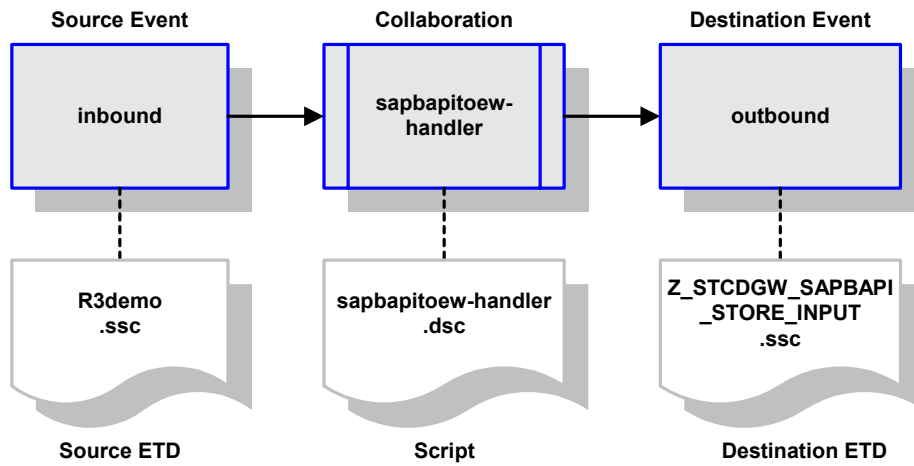
### Collaborations

The sample schema contains two Monk function/Collaborations used by the e\*Way **BAPI\_EGate\_SP** to interact with the SAP R/3 system. The first, which polls SAP for function calls (see [Polling SAP for Function Calls](#) on page 99), is shown in Figure 24. The second, a Monk function handler for fetching and processing data from SAP (see [Extracting Data](#) on page 100 and [Processing Data and Sending Results](#) on page 101), is shown in Figure 25.

**Figure 24** Polling Monk Function

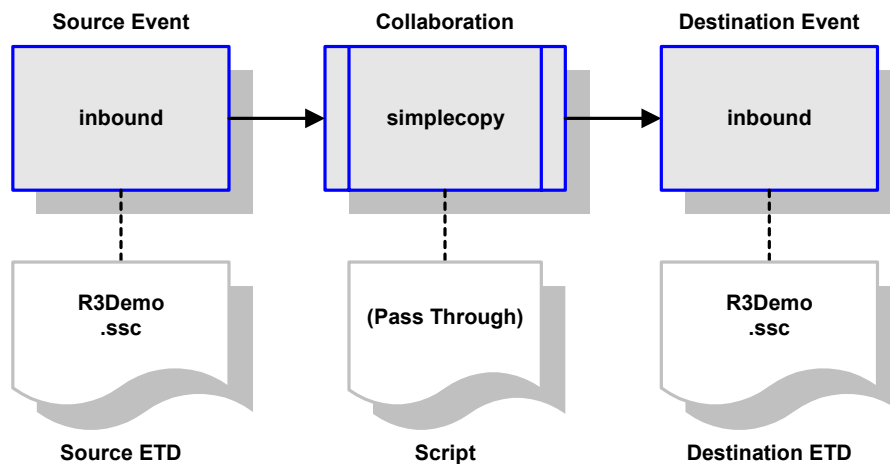


**Figure 25** Monk Function Handler

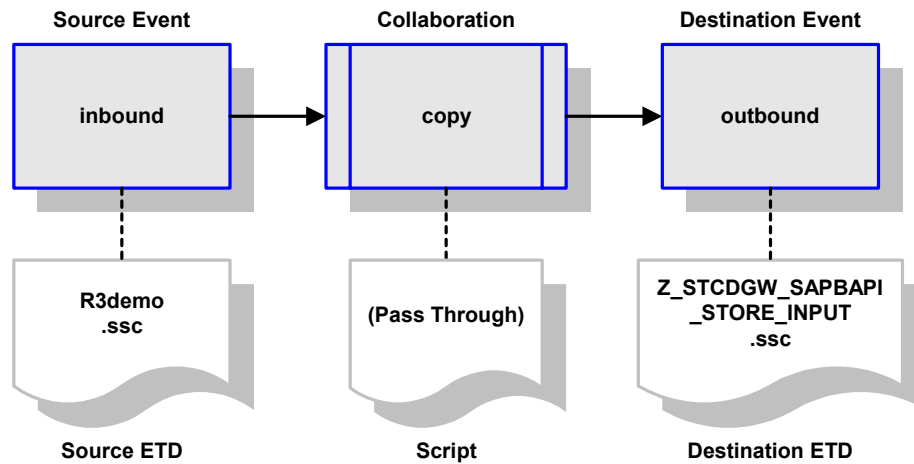


The e\*Gate components interact with each other by means of two internal pass-through Collaboration services, having the logical names **simplecopy** and **copy**. The definitions for these Collaboration services are shown in Figure 26 and Figure 27, respectively.

**Figure 26** simplecopy Collaboration Service



**Figure 27** copy Collaboration Service



## Startup Function

For this sample schema, the standard startup Monk function, [ewtosapbapi-startup](#), has been modified and renamed as follows.

---

### sapbapitoew-startup

#### Description

This Monk startup function is used with the [bapi2egate](#) Sample Schema. It defines the `sapbapi` global variables and loads the Collaboration `sapbapitoew-reconnect`.

#### Signature

(sapbapitoew-startup)

#### Parameters

None.

#### Returns

None.

#### Throws

None.

#### Location

sapbapitoew-startup.monk



### 3.8.2 egate2bapi

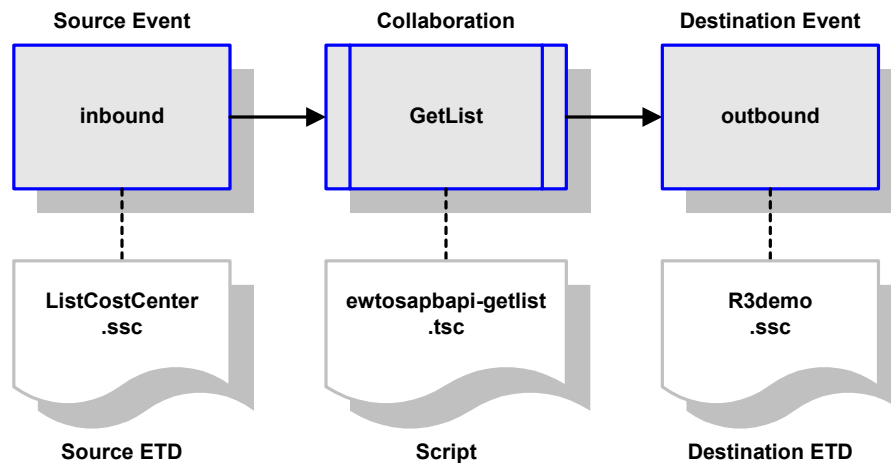
This schema illustrates the RFC Client process described in **Client Mode (e\*Gate to SAP)** on page 92. The sample schema **egate2bapi** sets up two instances of the SAP BAPI e\*Way and two instances of the File e\*Way, having the logical names shown in the following table. It also sets up two Intelligent Queues, with the logical names **iq\_bapi1** and **iq\_bapi2**.

| e*Way Type     | Module LN | Pub/Sub LN        |
|----------------|-----------|-------------------|
| SAP BAPI e*Way | SAP_BAPI  | SAP_BAPI_INCOMING |
|                |           | SAP_BAPI_SP       |
| File e*Way     | eater     | eater_bapi_sp     |
|                | feeder    | feeder_bapi_sp    |

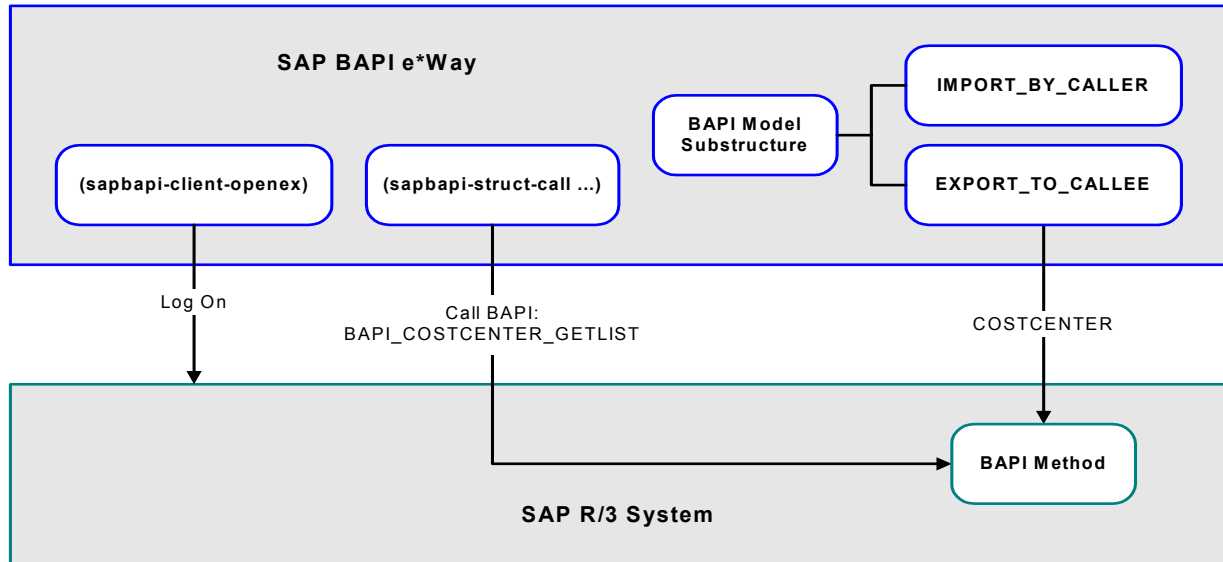
### Collaborations

The sample schema contains one primary Collaboration, shown in Figure 28, which illustrates a simple SAP client application. This Collaboration, **GetList**, calls the BAPI method **BAPI\_COSTCENTER\_GETLIST** to fetch the list of cost centers from SAP [**EXPORT\_TO\_CALLEE (COSTCENTER)**], as shown in Figure 29. It also obtains a timestamp [**EXPORT\_TO\_CALLEE (DATE)**].

**Figure 28** GetList Collaboration Definition

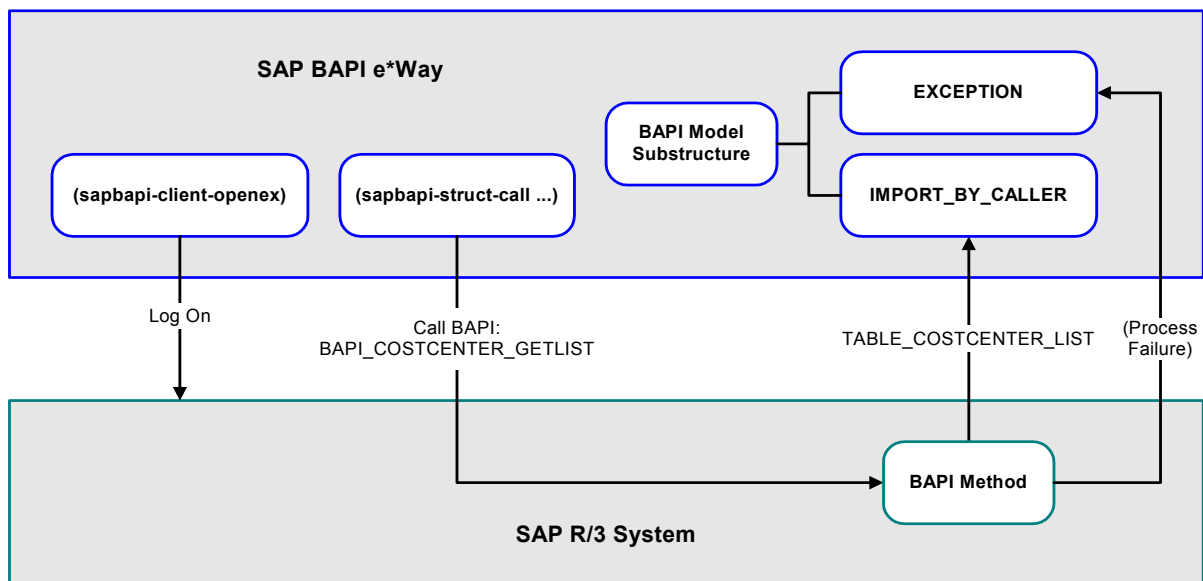


**Figure 29** EGate2BAPI Fetch



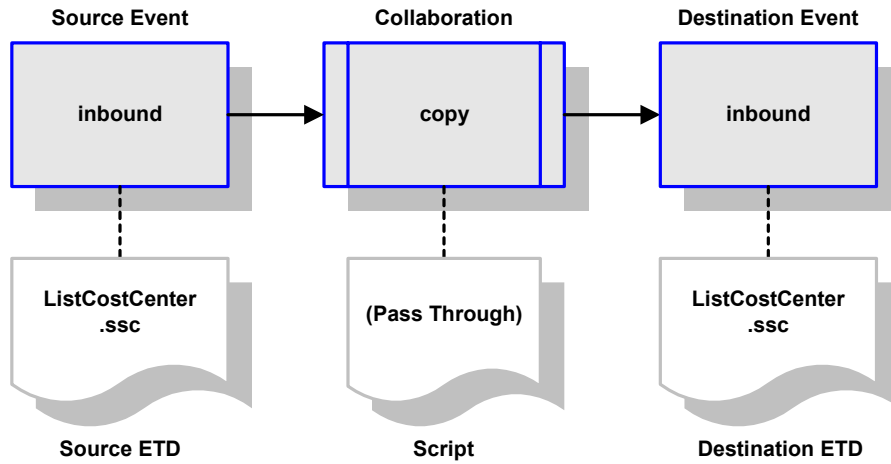
If this operation is successful, information (TABLE\_COSTCENTER\_LIST) is returned to e\*Gate using event-send-to-egate (see Figure 30). Failure returns an exception.

**Figure 30** EGate2BAPI Results

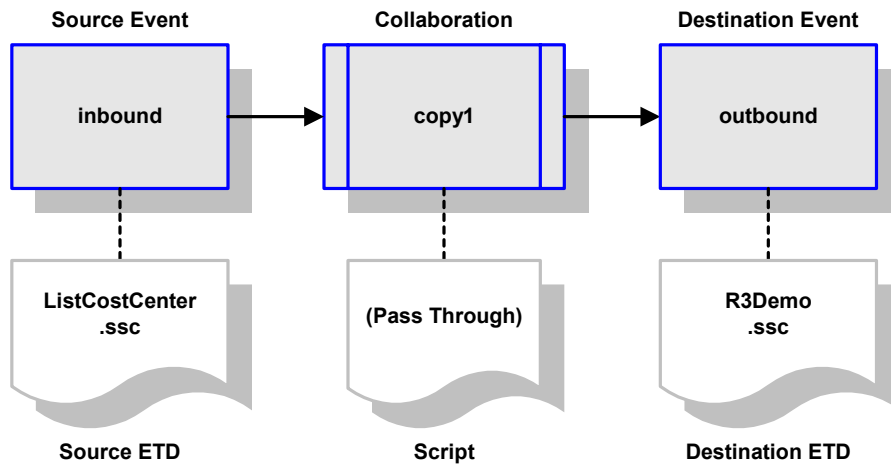


The e\*Gate components interact with each other by means of three internal pass-through Collaboration services, having the logical names **copy**, **copy1**, and **copy2**. The definitions for these Collaboration services are shown in Figure 31, Figure 32, and Figure 33, respectively.

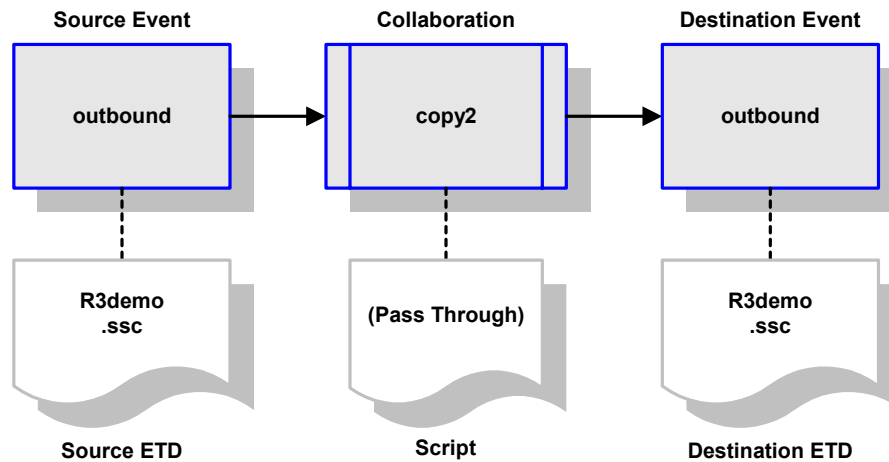
**Figure 31** copy Collaboration Service



**Figure 32** copy1 Collaboration Service



**Figure 33** copy2 Collaboration Service



## Startup Function

For this sample schema, the standard startup Monk function, `ewtosapbapi-startup`, has been modified and renamed as follows.

---

## dgwtosapbapi-startup

### Description

This Monk startup function is used with the `egate2bapi` Sample Schema. It follows the general form of `ewtosapbapi-startup`, but loads `sapbapi-init`, `saprfc-init`, and `stc_monkutils.dll`, after which it invokes `sapbapi-init`.

### Signature

`(dgwtosapbapi-startup)`

### Parameters

None.

### Returns

The string "UP" upon success, "DOWN" upon failure.

### Throws

None.

### Location

`dgwtosapbapi-startup.monk`

### 3.8.3 TrfcFromBapi

This schema illustrates the tRFC Server process described in [Server Mode \(SAP to e\\*Gate\)](#) on page 112. The sample schema **TrfcFromBapi** sets up two instances of the SAP BAPI e\*Way and two of the File e\*Way, having the logical names shown in the following table. It also sets up an Intelligent Queue, with the logical name **iq1**.

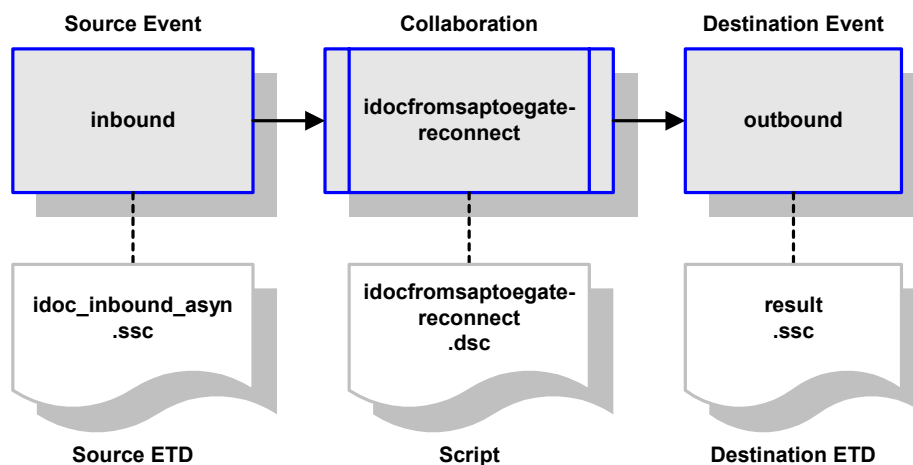
| e*Way Type     | Module LN       | Pub/Sub LN    |
|----------------|-----------------|---------------|
| SAP BAPI e*Way | BAPI_EGate_EWay | BAPI_EGATE_SP |
| File e*Way     | Eater           | EaterSP       |

The e\*Way polls SAP R/3 for function calls and extracts any data for processing and routing. Processed data is sent to the IQ for publication to other e\*Gate components and to the File e\*Way, which sends any results to SAP R/3.

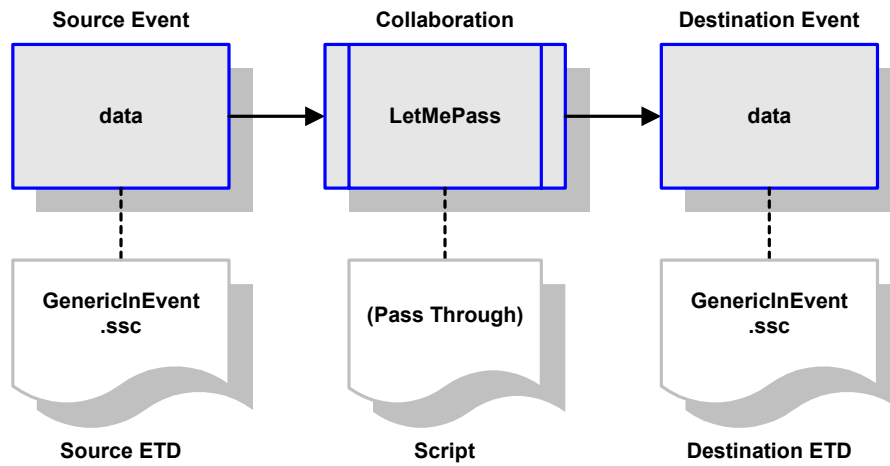
### Collaborations

The sample schema contains a Monk function/Collaboration which polls SAP for function calls, is shown in Figure 34. Another Collaboration, **LetMePass**, is used for receiving the previously-installed Monk functions from SAP (see Figure 35). This is a pass-through Collaboration service that allows the function to be received without being processed as data by the e\*Way.

**Figure 34** Polling Monk Function

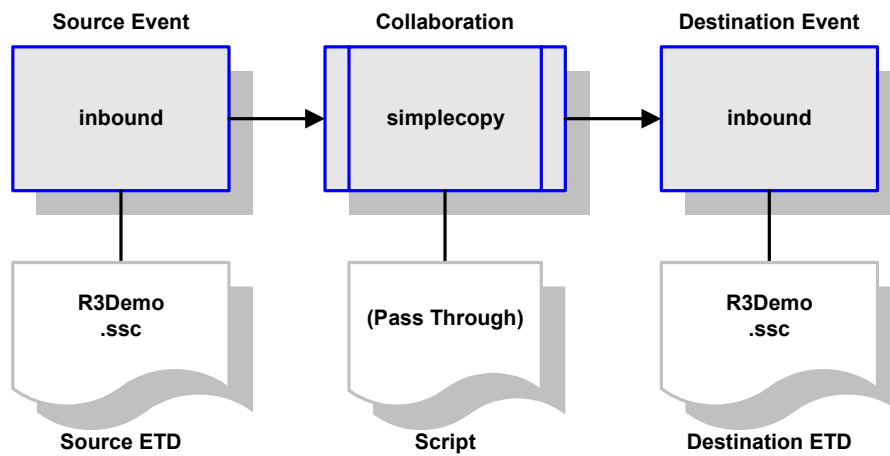


**Figure 35** LetMePass Collaboration Service

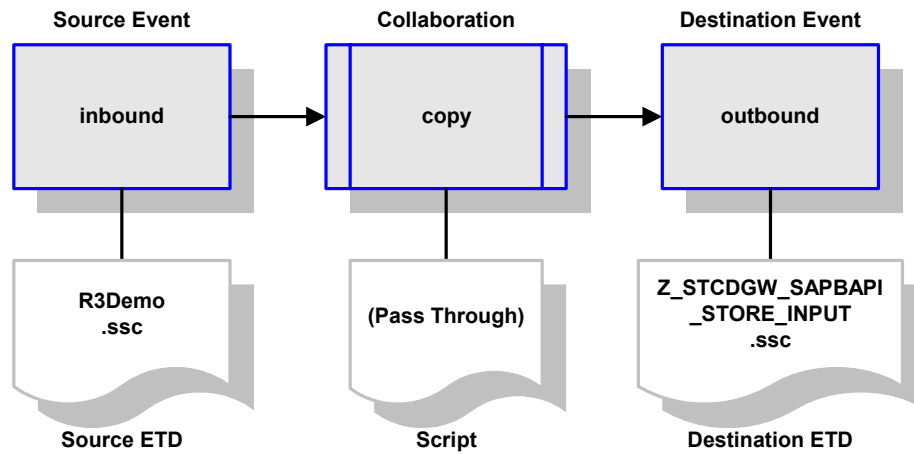


The e\*Gate components interact with each other by means of two internal pass-through Collaboration services, having the logical names **simplecopy** and **copy**. The definitions for these Collaboration services are shown in Figure 36 and Figure 37, respectively.

**Figure 36** simplecopy Collaboration Service



**Figure 37** copy Collaboration Service



## Startup Function

For this sample schema, the standard startup Monk function, `ewtosapbapi-startup`, has been modified and renamed as follows.

---

### idocfromsaptoegate-startup

#### Description

This Monk startup function is used with the `TrfcFromBapi` Sample Schema. It loads the Collaboration `idocfromsaptoegate-reconnect`.

#### Signature

`(idocfromsaptoegate-startup)`

#### Parameters

None.

#### Returns

The string "UP" upon success; anything else indicates failure.

#### Throws

None.

#### Location

`idocfromsaptoegate-startup.monk`

### 3.8.4 TrfcToBapi

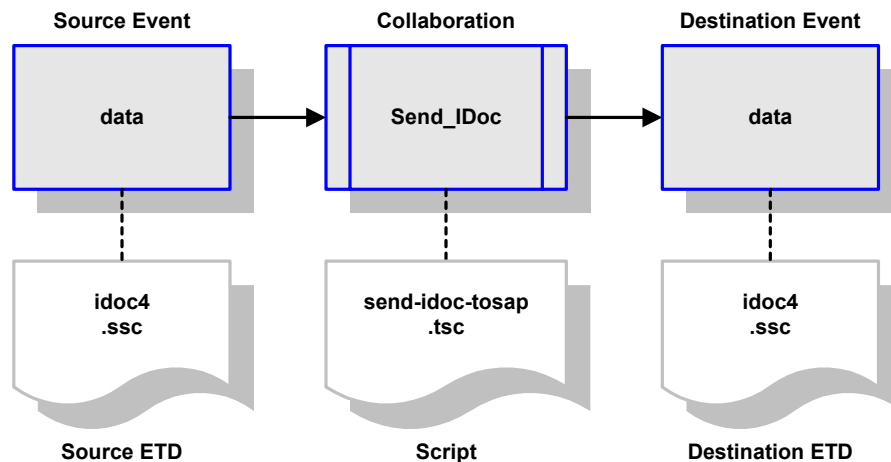
This schema illustrates the tRFC Client process described in **Client Mode (e\*Gate to SAP)** on page 107. The sample schema **TrfcToBapi** sets up two instances of the SAP BAPI e\*Way and also of the File e\*Way, having the logical names shown in the following table. It also sets up two Intelligent Queues, with the logical names **iq1** and **iq2**.

| e*Way Type     | Module LN | Pub/Sub LN    |
|----------------|-----------|---------------|
| SAP BAPI e*Way | (none)    | idoc_from_sap |
|                | ewbapi    | query         |
| File e*Way     | eater     | eat           |
|                | feeder    | feed          |

### Collaborations

The sample schema contains a Collaboration for sending an IDoc to the SAP R/3 system using tRFC, which is shown in Figure 38. This Collaboration specifies the data mapping between source and destination, and calls the appropriate TID Management Monk functions.

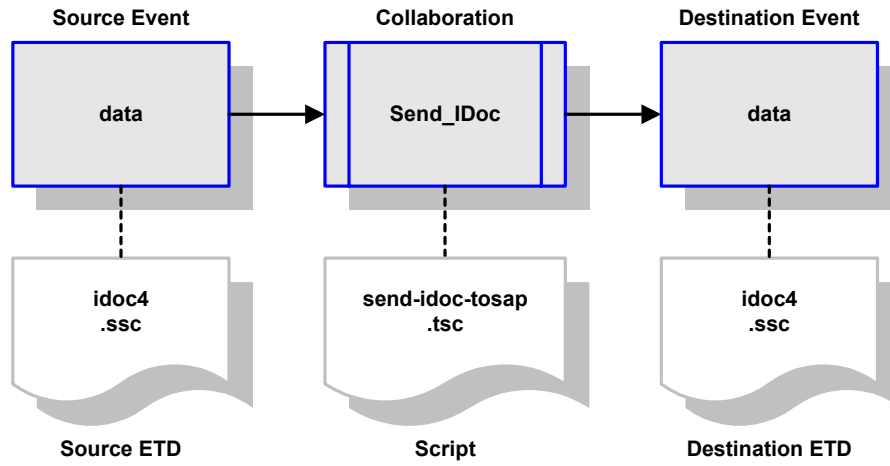
**Figure 38** Send\_IDoc Collaboration



The e\*Gate components interact with each other by means of a single internal pass-through Collaboration service, having the logical name **LetMePass**. This Collaboration is also used for extracting data from SAP. The definition for this Collaboration service is shown in Figure 39.



**Figure 39** LetMePass Collaboration Service



# 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 SAP-Initiated BAPI Calls

BAPI calls are usually made from within the e\*Gate environment; however, should you need to initiate BAPI calls from SAP R/3, a mechanism for doing so is provided with the e\*Way. The provided code allows you to send the BAPI name and key/value pairs from SAP to the e\*Way, which then can make the BAPI call to SAP R/3.

To use this mechanism, 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 are non-intrusive, and 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:

**Importing the SAP ABAP Components** on page 59

**Updating SAP R/3 Objects** on page 62

**Placeholder Function** on page 66

**Data Extraction via ABAP** on page 68

*Note:* The screen captures shown in this chapter correspond to SAP Frontend Server release 4.5B, and R/3 release 4.0B. They are shown simply to illustrate the general nature of the procedures, and contain only example values. They are not meant to replace the SAP documentation supplied with your system. The procedures for releases 4.0, 4.5 and 4.6 vary in detail, but are similar in a logical sense. Please refer to your SAP documentation to determine the exact procedures and corresponding appearance of the GUI for your installation.

## 4.2 Importing the SAP ABAP Components

The custom ABAP components provided with the e\*Way are delivered as a set of transport files, which are located on the e\*Gate installation CD-ROM in the directory:

```
\utils\sapr3staging\ewsapbapi\
```

The transport files add the following custom ABAP functions and modules to the SAP R/3 system:

- **Z\_BAPI\_OUTBOUND**

An example custom ABAP module (see [Data Extraction via ABAP](#) on page 68).

- **Z\_OUTBOUND\_BAPI\_INITIATE**

A test function used to send data to the SAP BAPI e\*Way (see [Data Extraction via ABAP](#) on page 68). The destination is determined from the table ZDGBAPI.

- **Z\_STCDGW\_SAPBAPI\_STORE\_INPUT**

A pre-defined placeholder function, which receives data into the e\*Way to be stored and/or forwarded to another BAPI (see [Placeholder Function](#) on page 66).

- **ZDGBAPI**

A pre-defined table template (see [Updating SAP R/3 Objects](#) on page 62).

- **ZSTCBAPIOUTBOUND**

A pre-defined destination (see [Data Extraction via ABAP](#) on page 68).

These components enable operation of the e\*Way and assist in building Event structures. They do not replace or alter any existing functions or files. The transport number (represented here by xxxxxx, since it changes) is used as input to the SAP Application Server and also is embedded in the filename of the cofiles and data files.

### 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 file **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 S4XKxxxxxxx <system>
```

where **<system>** is the *system 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 to complete. Ignore any **No Profile Used** messages that you may receive.

**Figure 40** Import Procedure Display

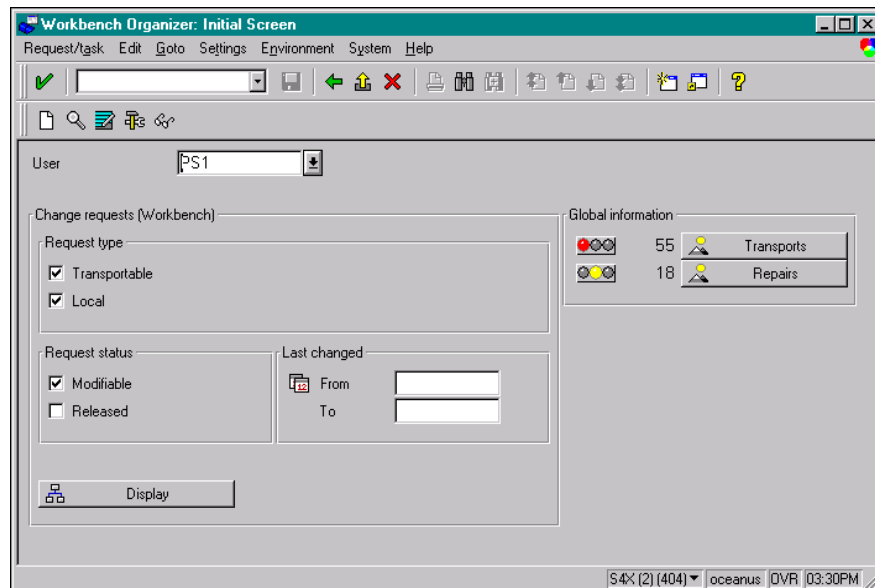
```

Telnet - 10.1.201.33
Connect Edit Terminal Help
oceanus:m4xadm 13% cd /usr/sap/trans/bin
oceanus:m4xadm 14% tp addtobuffer S4XKxxxxxx M4X
This is tp version 270.00.01 (release 45B) for ORACLE database
Addtobuffer successful for S4XKxxxxxx
tp finished with return code: 0
meaning:
  Everything OK
oceanus:m4xadm 15% tp import S4XKxxxxxx 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% █
    
```

**Note:** If you encounter errors during the import process (as shown in [Figure 40](#) on [page 60](#)), view the error log by following the procedure described below.

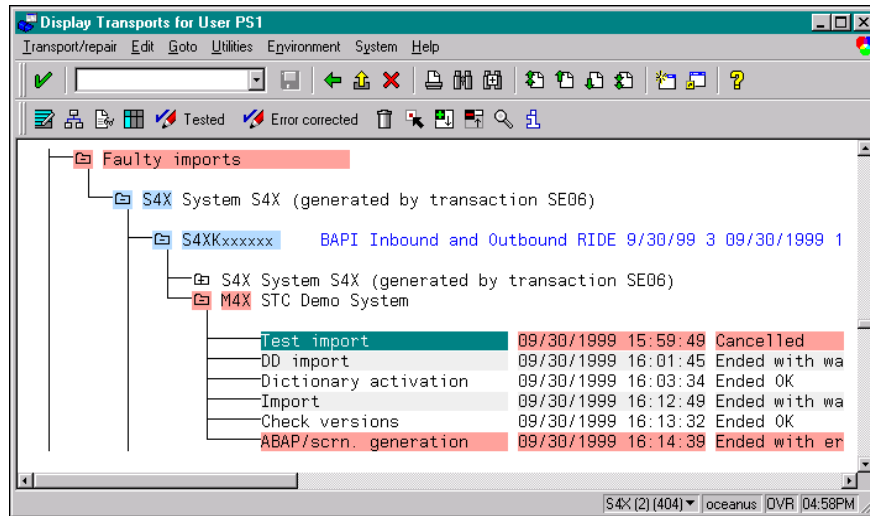
- 1 Go to transaction SE09.

**Figure 41** Workbench Organizer Window



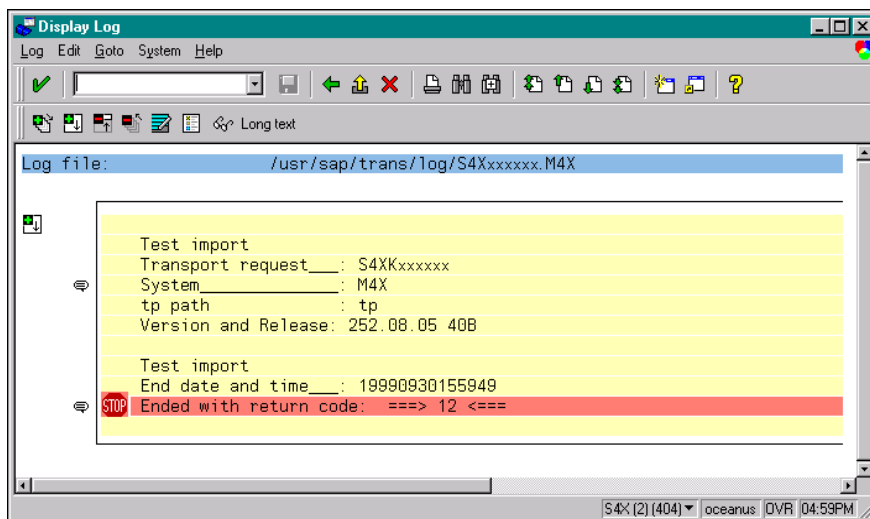
- 2 Select  Transports.
- 3 Enter the transport number, and then follow the menu path **Goto > All logs** in the *Display Transports* window.

Figure 42 Display Transports Window



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

Figure 43 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.

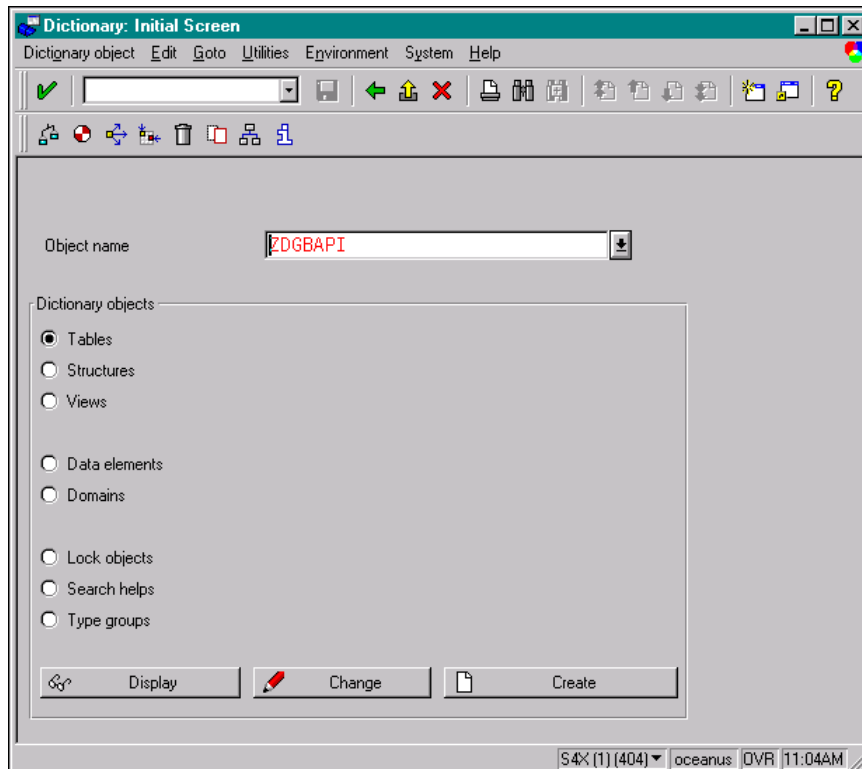
## 4.3 Updating SAP R/3 Objects

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

Transaction: SE11

After the Transport files have been imported into your SAP system, the following data needs to be inserted into the ZDGBAPI table.

**Figure 44** Dictionary - Initial Screen




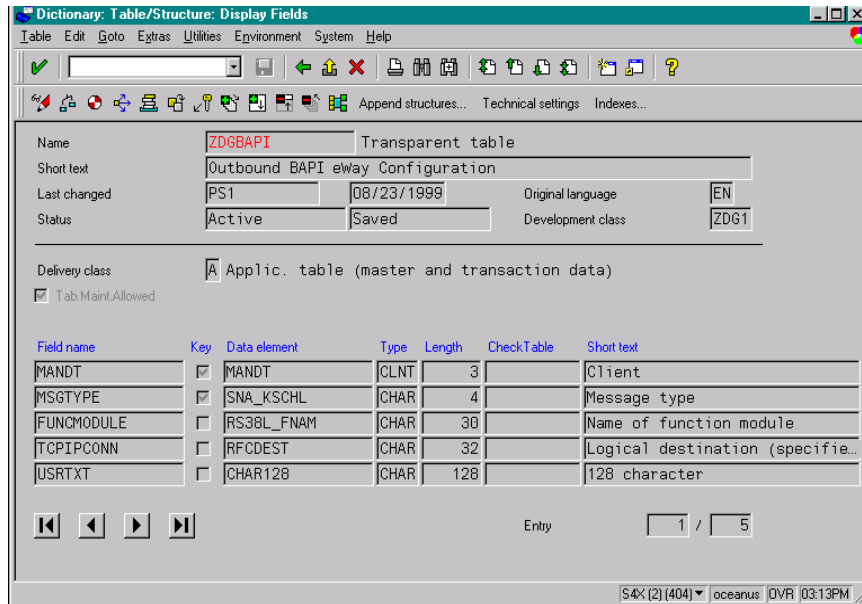
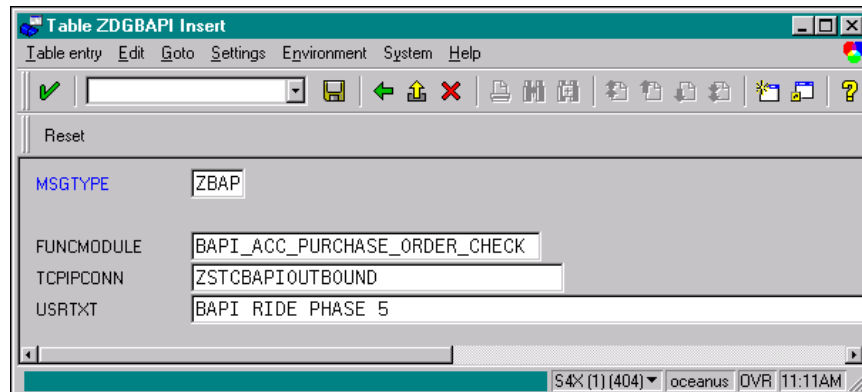
- 1 Enter **ZDGBAPI** as the **Object name**.
- 2 Select the **Tables** option.
- 3 Select  **Display**.

Figure 45 Display Fields Window



- 4 In the *Display Fields* window, select the menu path **Utilities > Create entries**.

Figure 46 Table Insert Window




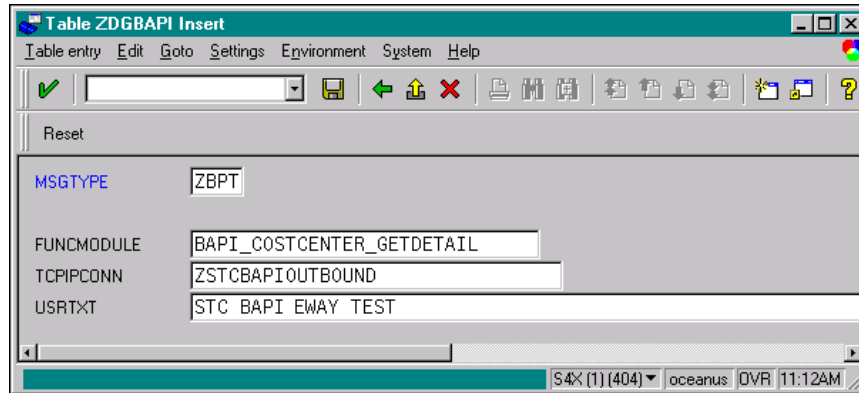
- 5 Type the following into the fields:
  - ◆ MSGTYPE: ZBAP
  - ◆ FUNCMODULE: BAPI\_ACC\_PURCHASE\_ORDER\_CHECK
  - ◆ TCPIPCONN: ZSTCBAPIOUTBOUND
  - ◆ USRTXT: BAPI RIDE PHASE 5
- 6 Save  the entries.
- 7 Now create another entry:

Figure 47 Table Insert Window




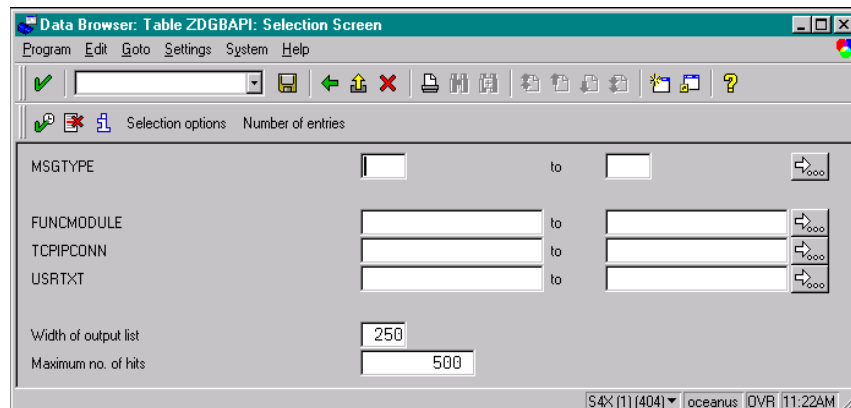
- 8 Type the following into the fields:
  - ◆ MSGTYPE: ZBPT
  - ◆ FUNCMODULE: BAPI\_COSTCENTER\_GETDETAIL
  - ◆ TCPIPCONN: ZBAPIOUTBOUND
  - ◆ USRTXT: BAPI e\*Way TEST
- 9 Once again, save  the entry.
- 10 Return to the *Dictionary-Initial Screen* and follow the menu path **Utilities > Table Contents**.

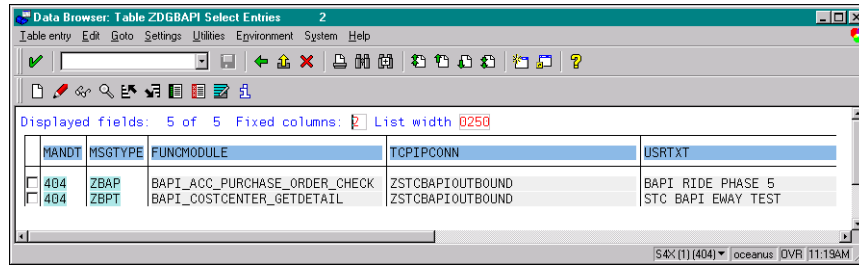
Figure 48 Data Browser Selection Window



- 11 In the *Data Browser Selection* window, select .



**Figure 49** Data Browser Select Entries Window



- 12 The entries you made should be displayed in the *Data Browser Select Entries* window. If not, repeat steps 7 or 8, as appropriate.
- 13 Return to the *SAP System* main window.

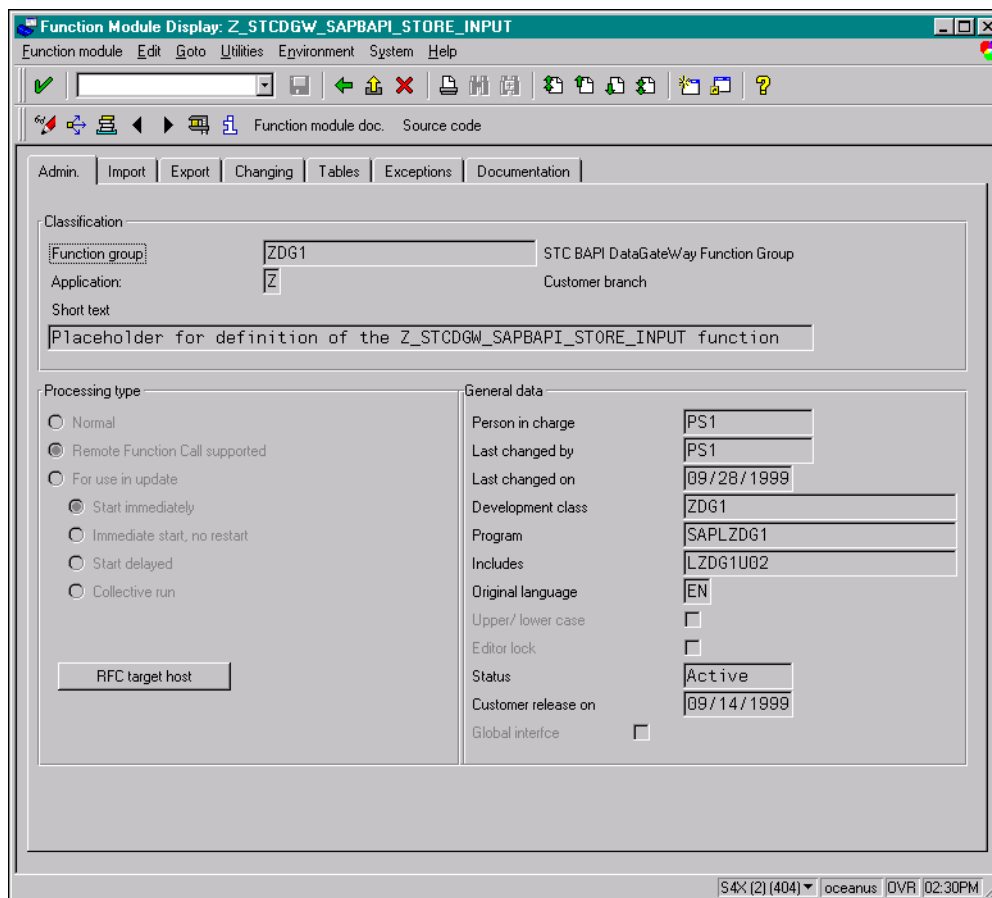
## 4.4 Placeholder Function

The pre-defined placeholder function/template `Z_STCDGW_SAPBAPI_STORE_INPUT`, receives data into the e\*Way to be stored and/or forwarded to another BAPI. This template is placed on SAP to provide the interface definition, and can be converted into a working ETD by the BAPI Converter Wizard (see [The BAPI Structure Builder](#) on page 34).

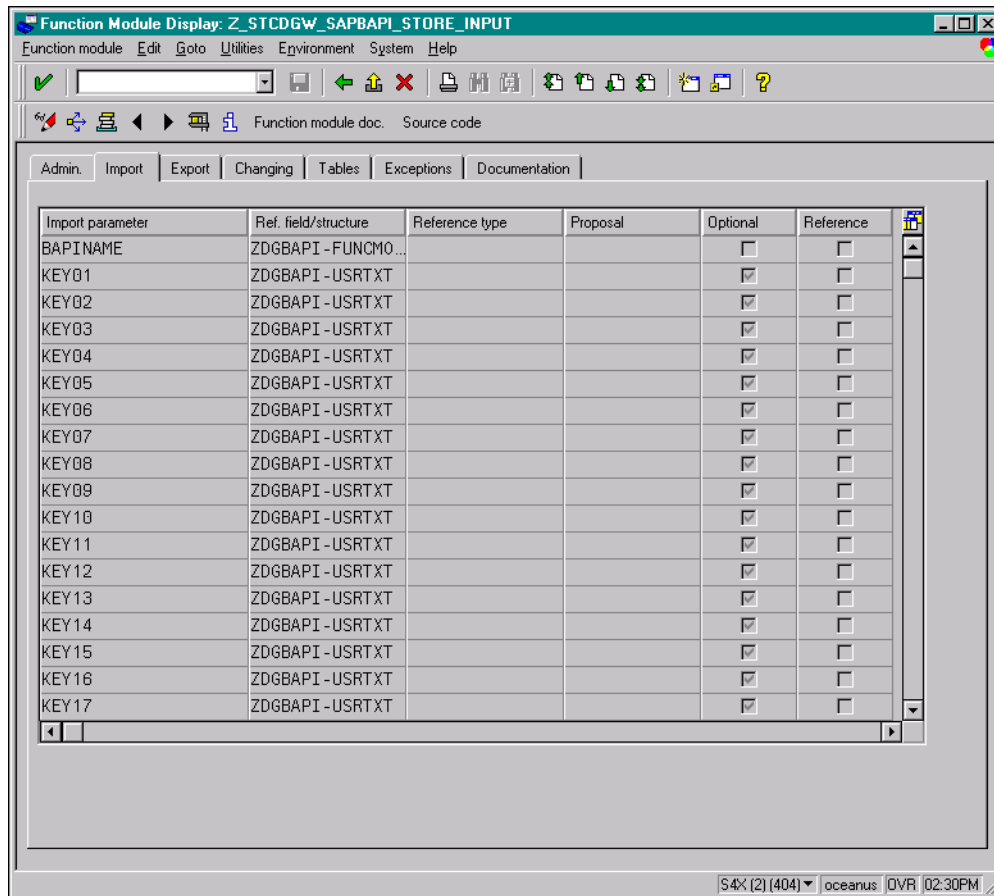
**Note:** The function must be **Activated** and **Released** before the BAPI Converter Wizard can interrogate its definition.

This function can be examined using transaction **SE37**:

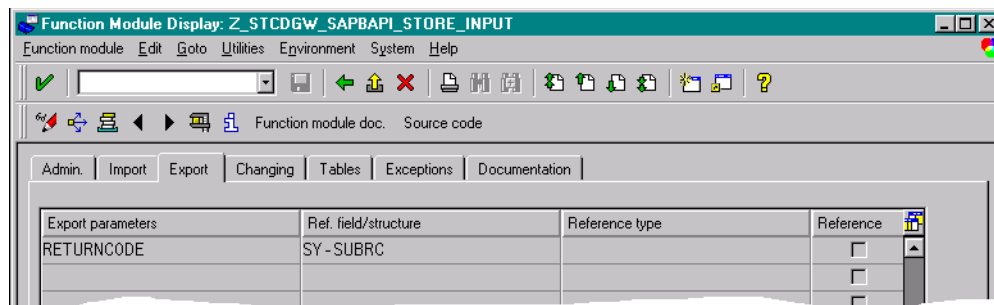
**Figure 50** `Z_STCDGW_SAPBAPI_STORE_INPUT` (Admin Tab)



**Figure 51** Z\_STCDGW\_SAPBAPI\_STORE\_INPUT (Import Tab)



**Figure 52** Z\_STCDGW\_SAPBAPI\_STORE\_INPUT (Export Tab)



## 4.5 Data Extraction via ABAP

Figure 53 shows the data extraction part of an example custom ABAP module, Z\_BAPI\_OUTBOUND.

**Figure 53** ABAP Module Z\_BAPI\_OUTBOUND

```

1 report z_bapi_outbound .
2
3 data: return like sy-subrc,
4       errmsg like bdcmsgcoll.
5
6 call function 'FUNCTION_EXISTS'
7   exporting
8     funcname = 'STC_DGW_SAPBDC_CONN_TEST'
9   *   funcname = 'Z_OUTBOUND_BAPI_INITIATE'
10  exceptions
11    function_not_exist = 1.
12
13 write:/ sy-subrc.
14
15 if sy-subrc = 1.
16   call function 'Z_OUTBOUND_BAPI_INITIATE'
17   exporting
18     zmsgtype = 'ZBPT'
19     zkey01  = 'GETDETAIL'
20     zkey02  = '00012054'
21     zkey03  = '19990923'
22     zkey04  = '19991222'
23   importing
24     zreturn  = return
25     zerrmsg  = errmsg
26   exceptions
27     transfer_to_eway_failed = 1
28     message_type_not_found = 2
29     rfc_connection_test_failed = 3
30     others = 4.
31
32   write:/ sy-subrc.
33 endif.

```

Line 1 — 33 of 33

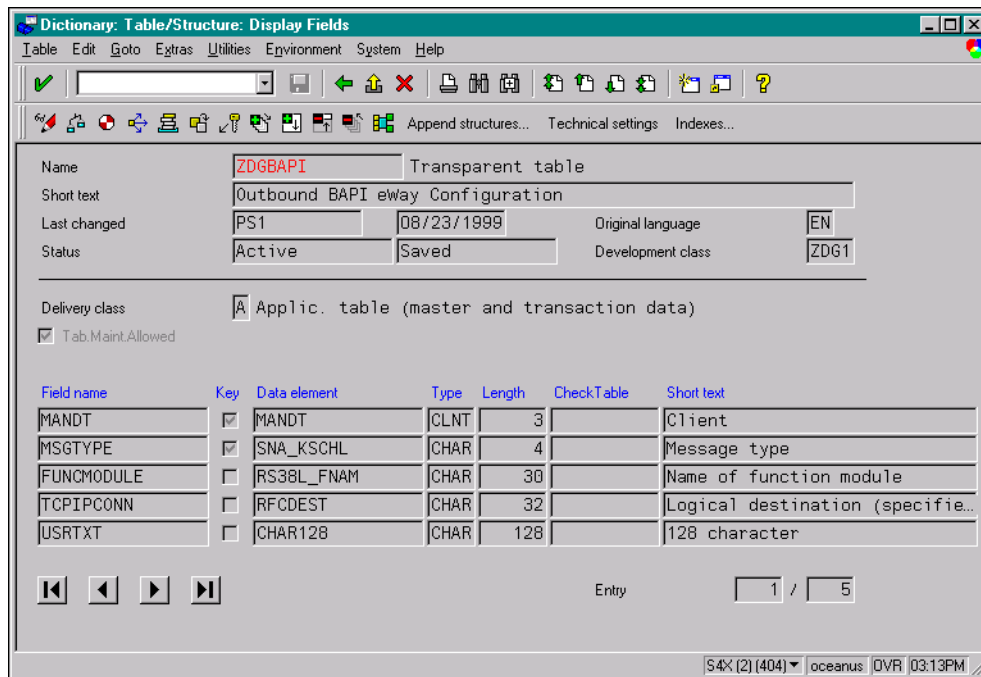
S4X [2] (404) oceanus OVR 03:28PM

In this example, a test function Z\_OUTBOUND\_BAPI\_INITIATE is called to send the data to the SAP BAPI e\*Way. In this function call, the filename, message type, and the internal data table are passed to Z\_OUTBOUND\_BAPI\_INITIATE. The return code from the latter is checked.

The input parameter **message type** tells Z\_OUTBOUND\_BAPI\_INITIATE where to send the data, or the RFC destination.

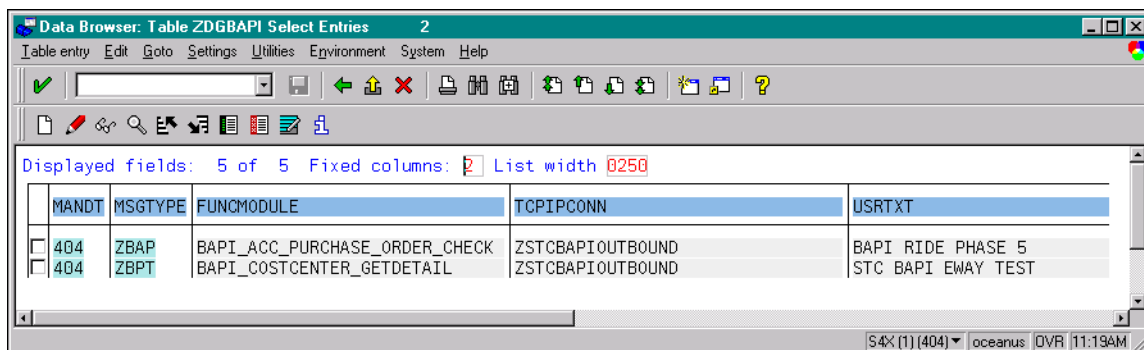
The function Z\_OUTBOUND\_BAPI\_INITIATE determines the destination from an SAP table ZDGBAPI, which is defined as follows.

Figure 54 ZDGBAPI Definition



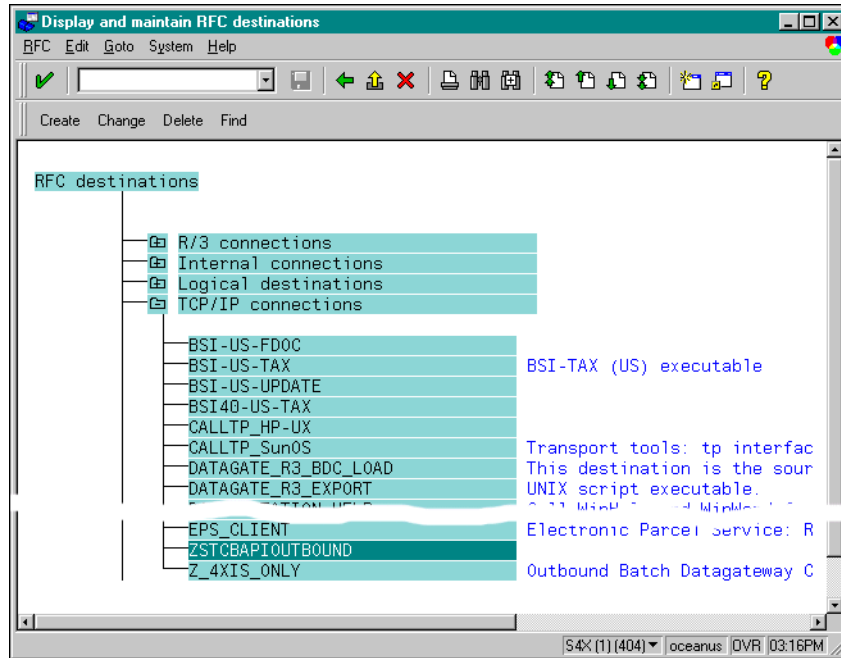
This table can be defined by the user through transaction SE11 (see [Updating SAP R/3 Objects](#) on page 62). An example ZDGBAPI table is shown below.

Figure 55 ZDGBAPI Table Example



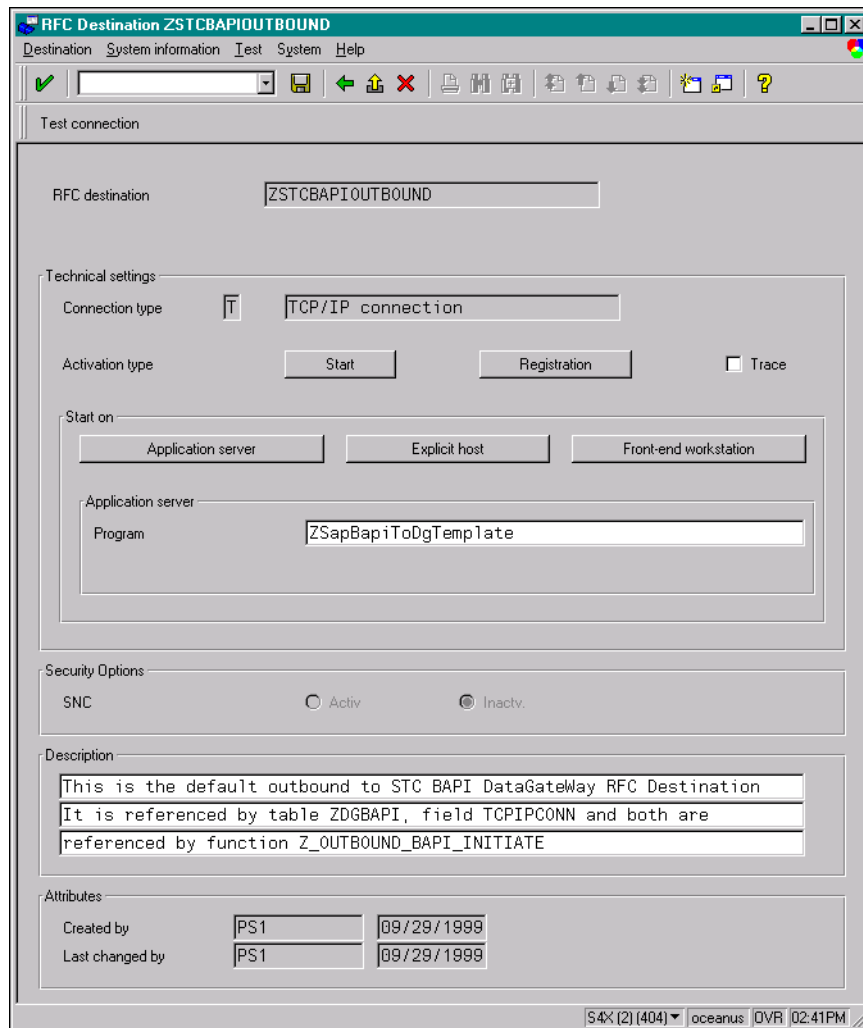
In this table, the message type ZOUTTEST is associated with the RFC destination (TCPIPCONN) OUTBOUND\_TEST. RFC destinations are defined with transaction SM59 as shown below (see [Registering the e\\*Way](#) on page 42).

Figure 56 RFC Destinations Tree



Double clicking on the destination ZSTCBAPIOUTBOUND reveals the definition for it.

Figure 57 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\_BAPI\_INITIATE sends messages of the associated type. At e\*Way startup, an SAP BAPI e\*Way registers with the SAP Application Server using a unique Program ID.

When a custom ABAP module calls Z\_OUTBOUND\_BAPI\_INITIATE, a message type is passed together with the message to be sent. With the message type as the key, Z\_OUTBOUND\_BAPI\_INITIATE looks up the RFC destination from the ZDGBAPI table. The message is routed to the correct SAP BAPI e\*Way based on the Program ID.

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

- RFCTYPE\_INT1
- RFCTYPE\_INT2
- RFCTYPE\_DATE\_1
- RFCTYPE\_DATE\_1

# Setup Procedures

This chapter describes the setup procedures for the SeeBeyond e\*Way Intelligent Adapter for SAP (BAPI).

---

## 5.1 Overview

After creating a schema, you must instantiate and configure the SAP BAPI 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 73

[Modifying e\\*Way Properties](#) on page 74

[Configuring the e\\*Way](#) on page 75

[Changing the User Name](#) on page 79

[Setting Startup Options or Schedules](#) on page 79

[Activating or Modifying Logging Options](#) on page 81

[Activating or Modifying Monitoring Thresholds](#) on page 82

### Troubleshooting the e\*Way

[Configuration Problems](#) on page 83

[System-related Problems](#) on page 84



## 5.2 Setting Up the e\*Way

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

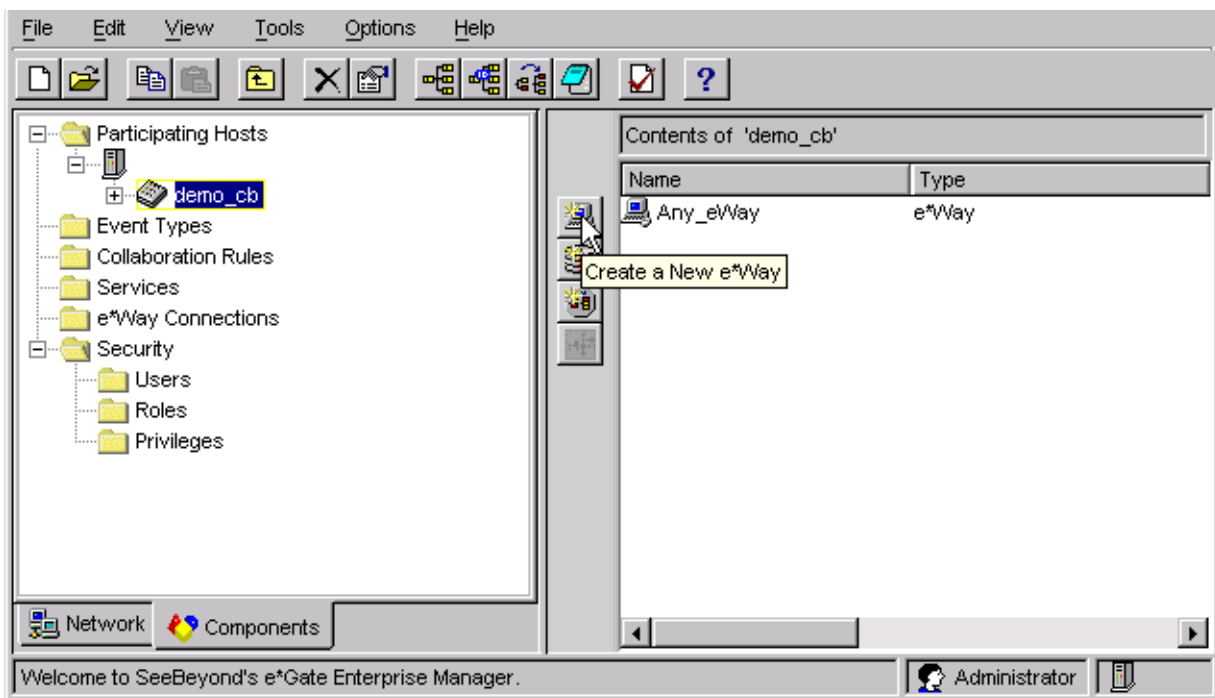
### 5.2.1 Creating the e\*Way

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 58** 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.

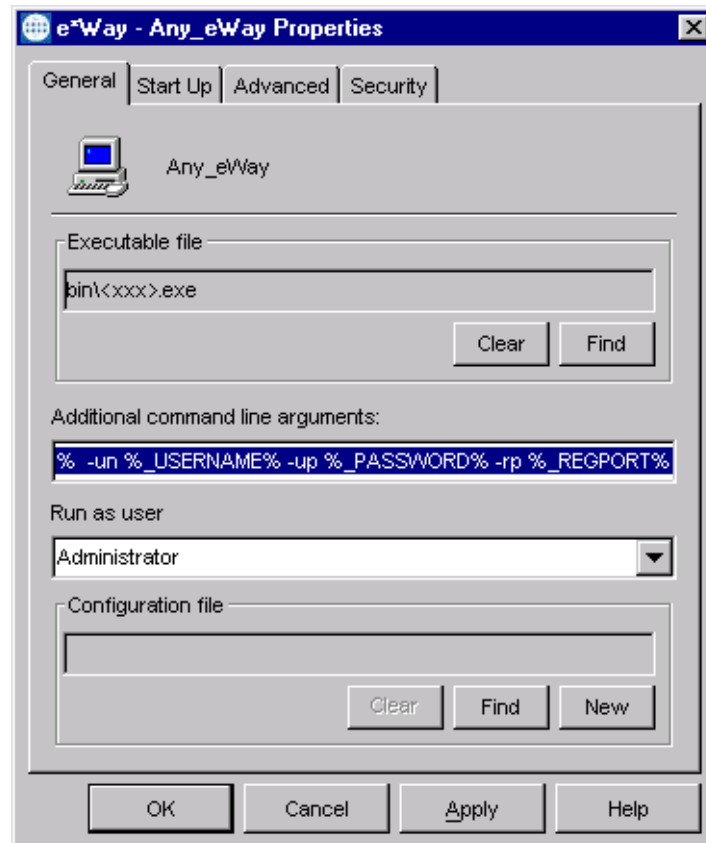
## 5.2.2 Modifying e\*Way Properties

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 59).

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

**Figure 59** e\*Way Properties (General Tab)



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

### 5.2.3 Configuring the e\*Way

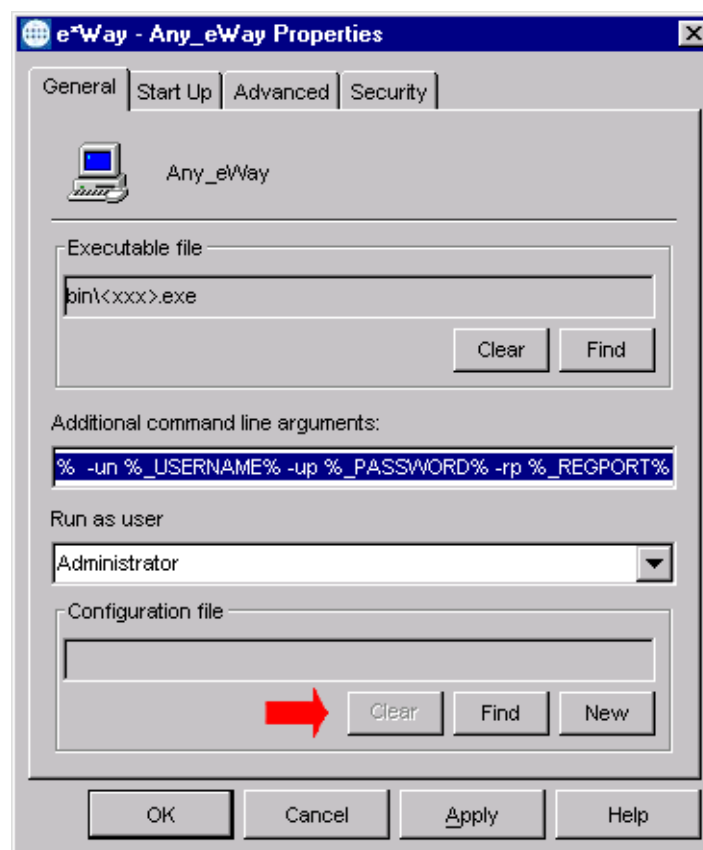
The e\*Way's default configuration parameters are stored in an ASCII text file with a .def extension. The e\*Way Editor provides a simple graphical interface for viewing and changing those parameters to create a working configuration (.cfg) file.

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 **ewsapbapi.def**.

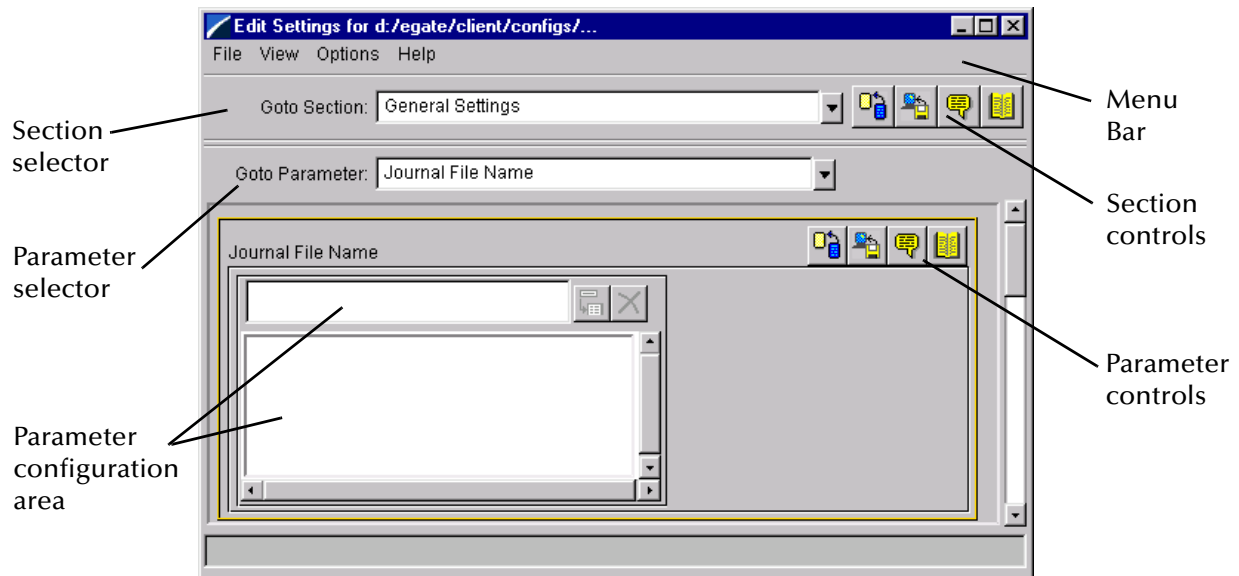
**Figure 60** e\*Way Properties - General Tab



- 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 the button to edit the currently selected file.
- 3 You are now in the e\*Way Configuration Editor.

## Using the e\*Way Editor

Figure 61 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 12 below.

**Table 12** 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 13

**Table 13** 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 runs.

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

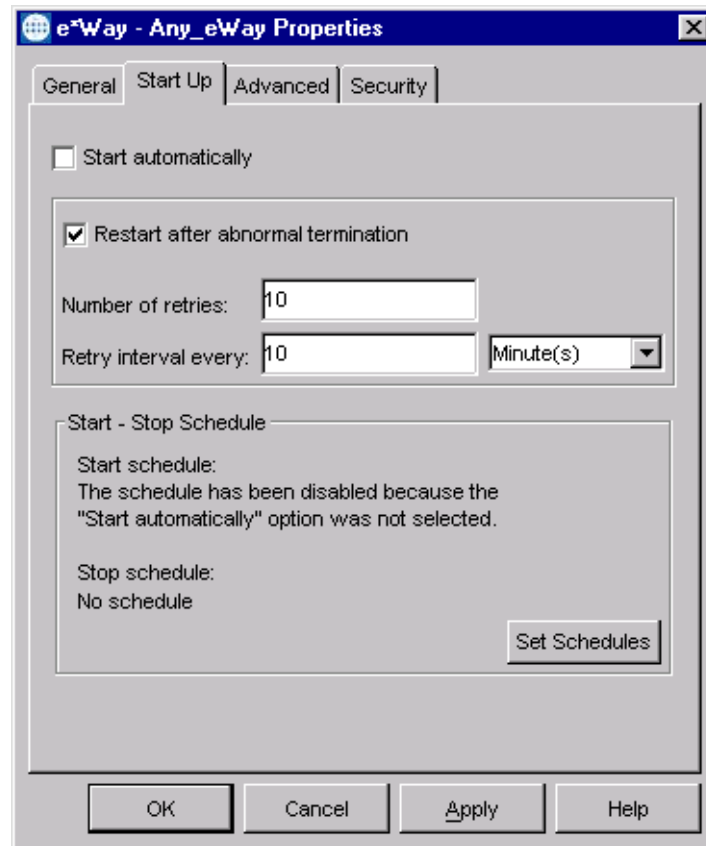
## 5.2.5 Setting Startup Options or Schedules

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 62). See the *e\*Gate Integrator System Administration and Operations Guide* for more information about how interactive monitors can start or shut down components.

**Figure 62** 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**.



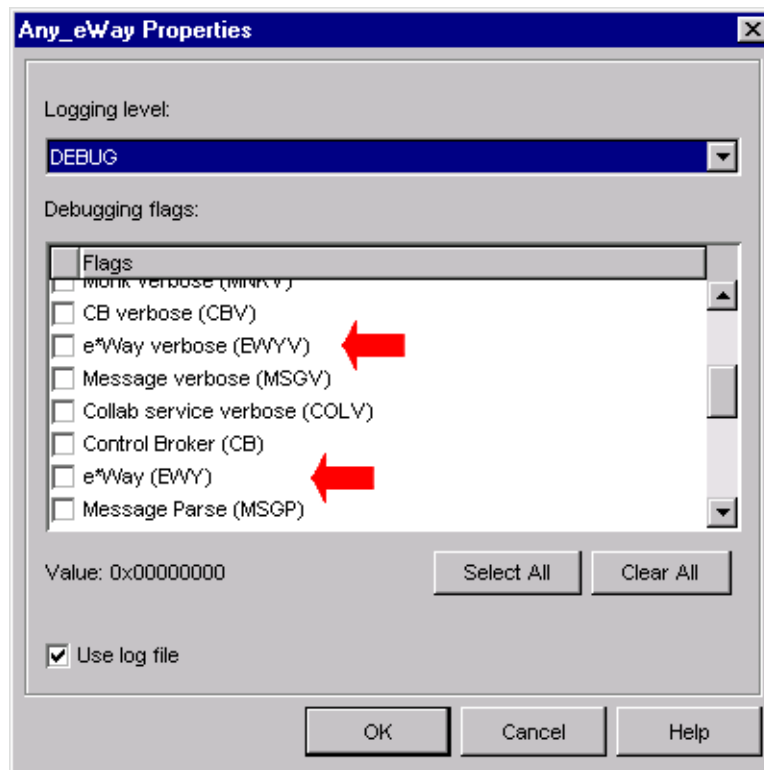
## 5.2.6 Activating or Modifying Logging Options

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**, and the dialog window appears (see Figure 63).

**Figure 63** 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 the *path* environment variable includes the location of the SAP BAPI dynamically-loaded libraries. The name of this variable on the different operating systems is:
  - ♦ PATH (Windows)
  - ♦ LD\_LIBRARY\_PATH (Solaris/Compaq)
  - ♦ 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

This chapter describes the interaction that takes place between the SeeBeyond e\*Way Intelligent Adapter for SAP (BAPI) and the SAP R/3 application, and summarizes the internal architecture and processes of the e\*Way itself.

---

## 6.1 Contents

The major topics discussed in this chapter include the following:

**BAPI-ETD Modeling** on page 86 describes how BAPI methods are modeled in e\*Gate using Monk objects and how data is inserted into and extracted from the BAPI methods. It also introduces the BAPI Structure Builder for obtaining metadata from SAP R/3 and terminology used in subsequent discussions.

**RFC Transport Process** on page 92 describes the standard RFC transport process for both Client and Server modes of operation.

**tRFC Transport Process** on page 106 describes the Transactional RFC (tRFC) transport process for both Client and Server modes of operation.

**e\*Way Architecture** on page 119 describes the conceptual architecture of the SAP BAPI e\*Way.

**Basic e\*Way Processes** on page 124 describes the basic e\*Way processes that take place within the e\*Way kernel.

## 6.2 BAPI-ETD Modeling

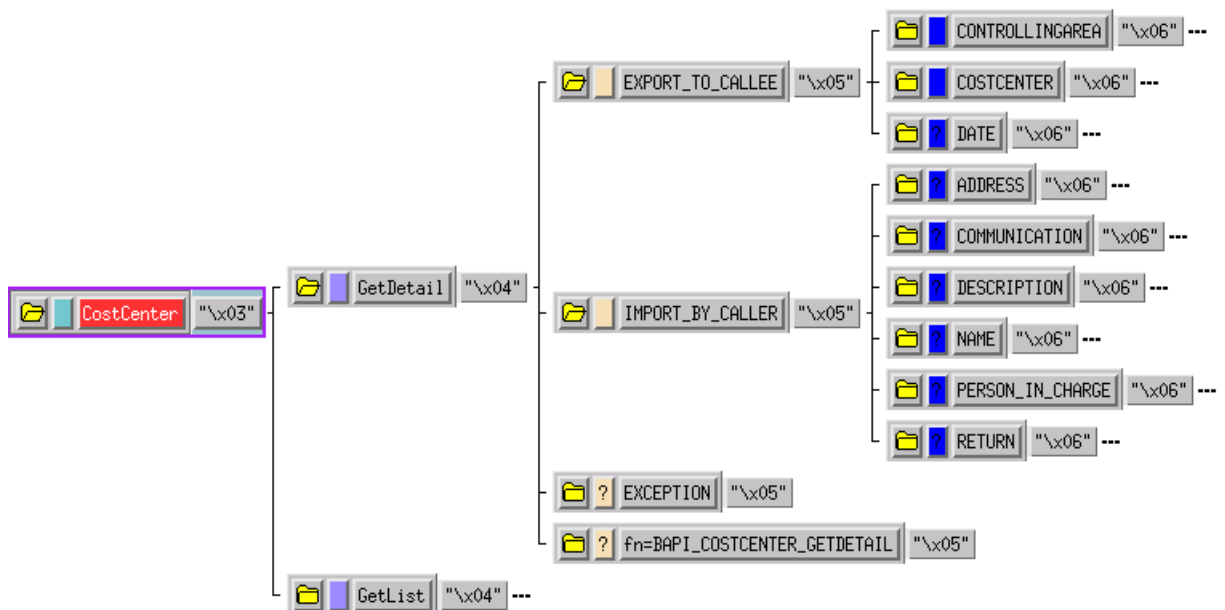
### 6.2.1 How BAPI Methods are Modeled in e\*Gate

SAP Business Objects typically have multiple BAPI methods associated with them, so the ability to map to multiple substructures is essential. Also, the ability to combine these substructures into a single (super)structure maintains the critical relationships between them.

An e\*Gate ETD also can be composed of multiple substructures, where each substructure represents an independent, but related, entity. Data can be inserted into, or extracted from, each of these message substructures individually.

In the BAPI e\*Way, a one-to-one relationship is set up between the desired BAPI methods and the ETD structure. Figure 18 below illustrates such a representation, showing the salient parts of an ETD tree.

**Figure 64** BAPI Method Substructure Example



To simplify the figure, the entire tree is not shown. On the far left is the node representing the BOR object **Chastisement**; it contains two sub-nodes representing the exposed BAPI methods **GetDetail** and **GetList**.

For the **GetDetail** method, there are sub-nodes representing the BAPI method's **IMPORT** parameters, **EXPORT** parameters, **EXCEPTION**, and the **ABAP/4 Function Name** corresponding to this BAPI. The **TABLE** parameters are not required in this BAPI and thus are absent. Both **IMPORT** and **EXPORT** parameters have their own sub-nodes representing their individual components.

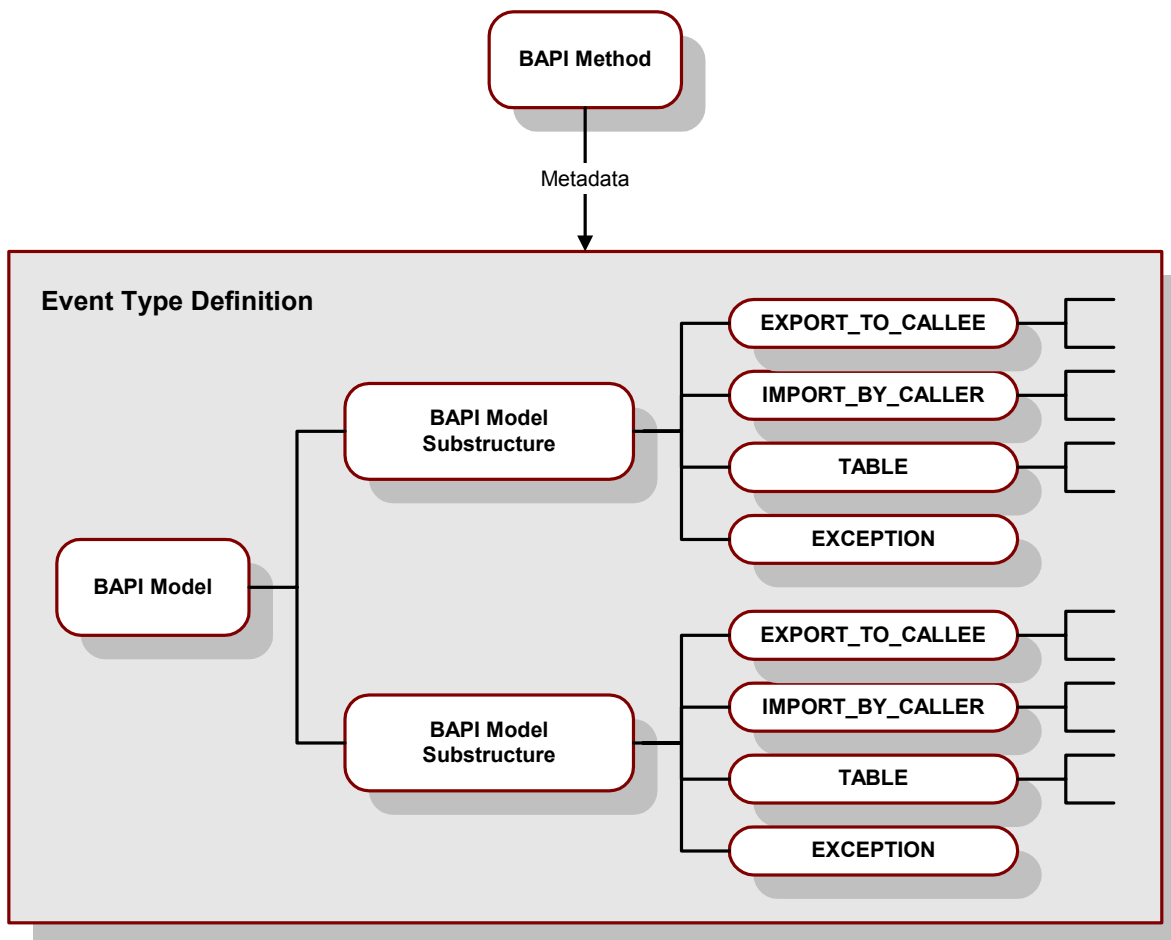
*Note: If you develop custom BAPIs, follow the instructions in the BAPI Programmers Reference to ensure your BAPI has been properly created and released to be visible in the BAPI Browser.*

## 6.2.2 The BAPI Structure Builder

The e\*Way's BAPI-ETD Model layer contains the BAPI Structure Builder, which enables you to automatically build an ETD representing a BAPI method defined in SAP, using metadata taken dynamically from SAP. Since each BAPI call is unique, the structure builder needs to create the structures corresponding to all required BAPI methods.

The Structure Builder's Converter Wizard converts the metadata for SAP BAPI methods (and other RFC-enabled function modules) into one ETD, where data can be easily dragged and dropped into the **IMPORT**, **EXPORT** and **TABLE** parameter nodes of specific BAPI-method substructures. For information on using the Structure Builder, see [The BAPI Structure Builder](#) on page 34.

**Figure 65** BAPI to ETD Conversion



### 6.2.3 Terminology

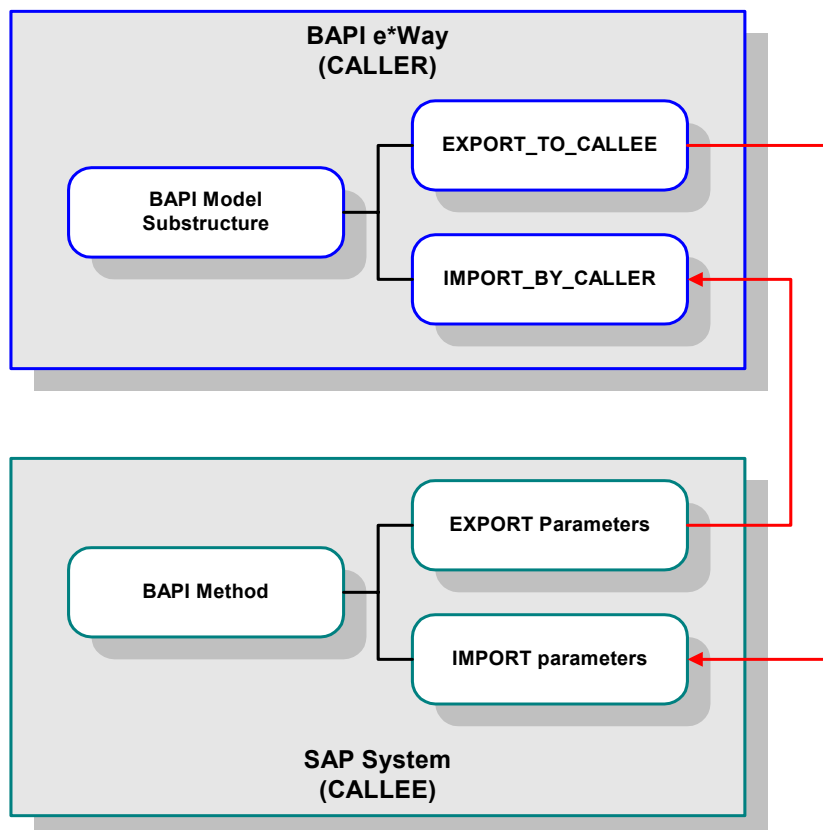
SAP's convention for **IMPORT** and **EXPORT** parameters, as described under the SAP Function Builder (SE37), is to use both terms in a *BAPI-centric* way, not relative to the system where the BAPI method resides (be it SAP or some external system like e\*Gate). That is, **IMPORT** parameters contain data that are imported into the method for processing, whereas **EXPORT** parameters pertain to results from the processing and are exported from the method.

Depending upon which system is calling the BAPI method, and which is hosting it, the meaning of **IMPORT** and **EXPORT** can be confusing if ETD nodes are simply labeled as such, since the **IMPORT** of one is the **EXPORT** of the other. With the BAPI e\*Way—for want of better terminology—**CALLEE** is used to represent the recipient of a message sent by a **CALLER**.

Additionally, rather than having one set of ETDs for SAP-to-e\*Way and another set for e\*Way-to-SAP, the BAPI e\*Way-produced ETDs are *caller-centric*; that is, *the nomenclature of the nodes is relative to the entity initiating the call*.

For example, if a BAPI e\*Way (the **caller**) calls a BAPI method that is hosted on SAP (the **callee**), data must be placed in the **EXPORT\_TO\_CALLEE** node of the ETD and is exported to the **callee's** (SAP's) **IMPORT** parameters. After the BAPI method has completed processing, the results are placed in the **callee's** **EXPORT** parameters, and imported by the **caller** (e\*Way) to the **IMPORT\_BY\_CALLER** node.

**Figure 66** Caller-Callee Example

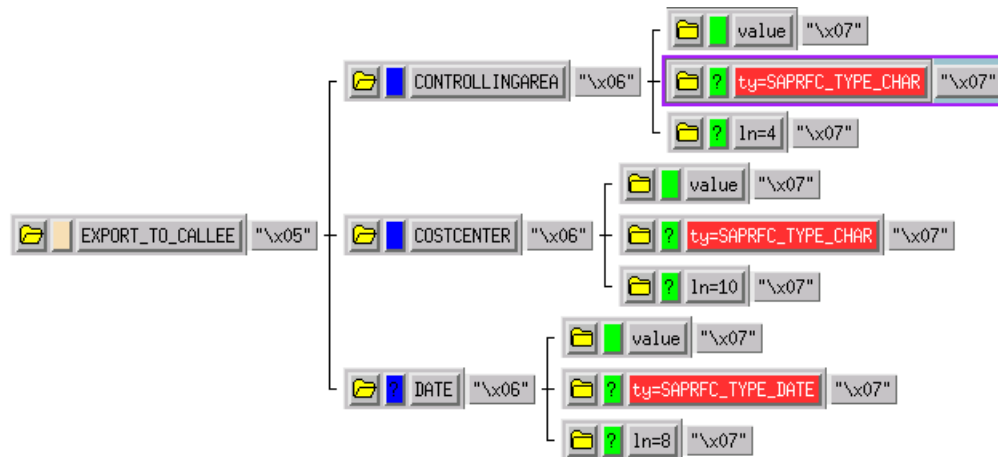




## 6.2.4 Data Insertion and Extraction

SAP uses many data types when exchanging data via RFC, and these appear as part of the commenting for a parameter field, as shown in Figure 67.

**Figure 67** Parameter Node Type Example



| Parameter Node Type | Description                                  |
|---------------------|--|
| SAPRFC_TYPE_CHAR    | Single byte character                        |
| SAPRFC_TYPE_DATE    | Date with the format CCYYMMDD                |
| SAPRFC_TYPE_TIME    | Time with the format HHMMSS                  |
| SAPRFC_TYPE_BYTE    | Binary (8-bit per byte) information          |
| SAPRFC_TYPE_NUM     | Integer number expressed in ASCII digit form |
| SAPRFC_TYPE_REAL    | Floating point number                        |
| SAPRFC_TYPE_INT     | Integer number                               |

When inserting data to, or extracting data from, parameter nodes of the above types, with the exception of `SAPRFC_TYPE_BYTE`, the ASCII string form of the data is used (e.g., 1.25 for `SAPRFC_TYPE_REAL`).

Binary information such as that used in the `SAPRFC_TYPE_BYTE` type cannot be given as-is because characters in it can be confused with the delimiters used by the BAPI ETDs. Instead, use the Monk function `string->hexdump` to encode binary data before inserting into a parameter node and inversely, use the Monk function `hexdump->string` on the data extracted from such a node.

Also, the mandatory versus optional nature of a parameter is indicated by the Repeat indicator of an ETD node. For example, in the above figure, both `CONTROLLINGAREA`

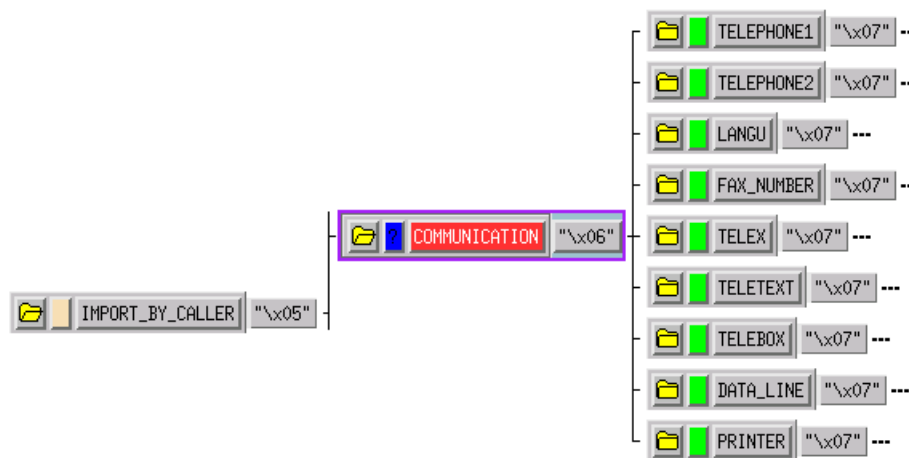
and **COSTCENTER** are mandatory (must contain some data) whereas **DATE** is optional (may have or not have data).

The Repeat indicators used are:

| Repeat Indicator | Meaning                                    |
|------------------|--|
| (blank)          | Mandatory—must contain data                |
| +                | Mandatory and can repeat one or more times |
| ?                | Optional—may or may not have data          |
| *                | Optional and can repeat one or more times  |

When building Collaboration Rules, data can be dragged directly into or from the parameter nodes; it is not necessary to reference the **value** node (as shown in Figure 67). However, if an opened parameter node does not show a **value** sub-node, it implies that the parameter is a composite and information should be placed into each of its components (see Figure 68). Note that composite nodes are opened by default.

**Figure 68** Composite Node Example

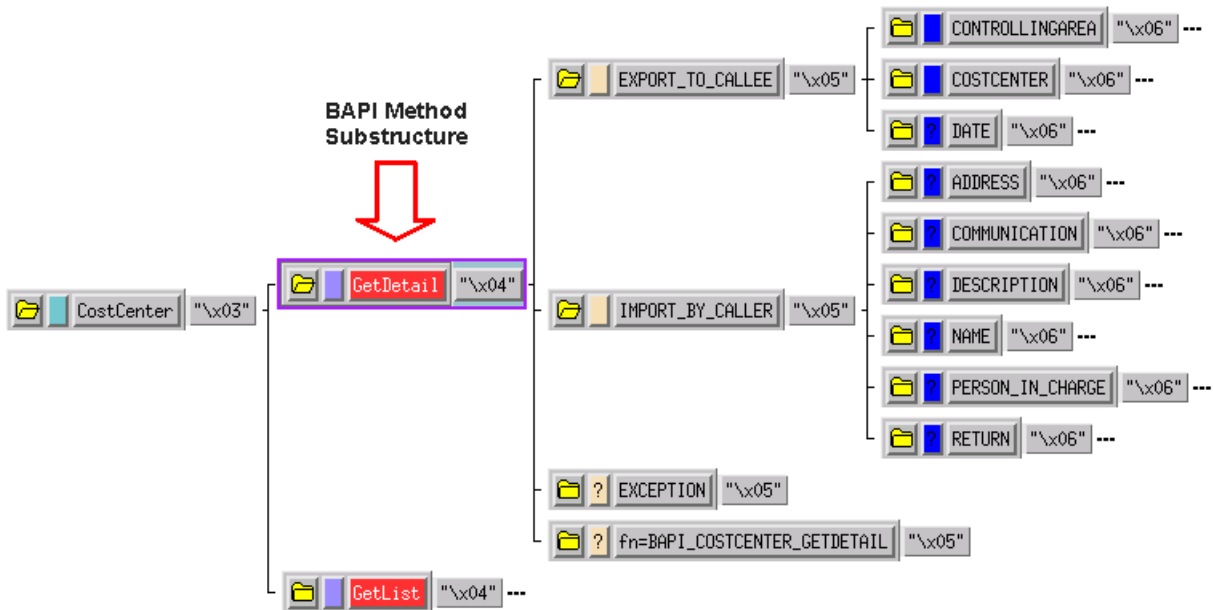


**Note:** When data field conversion is required as part of the Collaboration Rule (in the RFC layer), you must perform it in the rule manually. You must remember to pad data to full field length with leading zeros, and format date and time fields correctly.

## 6.2.5 BAPI Transport

The e\*Way's BAPI Transport layer consists of specialized Monk functions, whose names all start with the prefix **sapbapi**, that primarily take a BAPI or RFC-method message substructure as an argument (see [Program ID Registration](#) on page 98). The latter is defined as the ETD node that is the **immediate parent** of the nodes associated with the BAPI or RFC function name to be used, as shown in Figure 69.

**Figure 69** Transport Layer Monk Function Example



BAPI Transport has two operating modes, depending on whether the BAPI e\*Way is configured for Client-mode (e\*Gate to SAP) or Server-mode (SAP to e\*Gate) operation. Note that directions here refer to the *primary data flow*, since other data may be returned in the opposite direction.

The BAPI Transport layer lies on top of an RFC Transport layer, which also has two operation modes: standard RFC and Transactional RFC (tRFC). The latter is more reliable, but significantly more complex—as shown in the following sections.

For additional information, see [SAP BAPI Transport Layer](#) on page 122.

## 6.3 RFC Transport Process

### 6.3.1 Client Mode (e\*Gate to SAP)

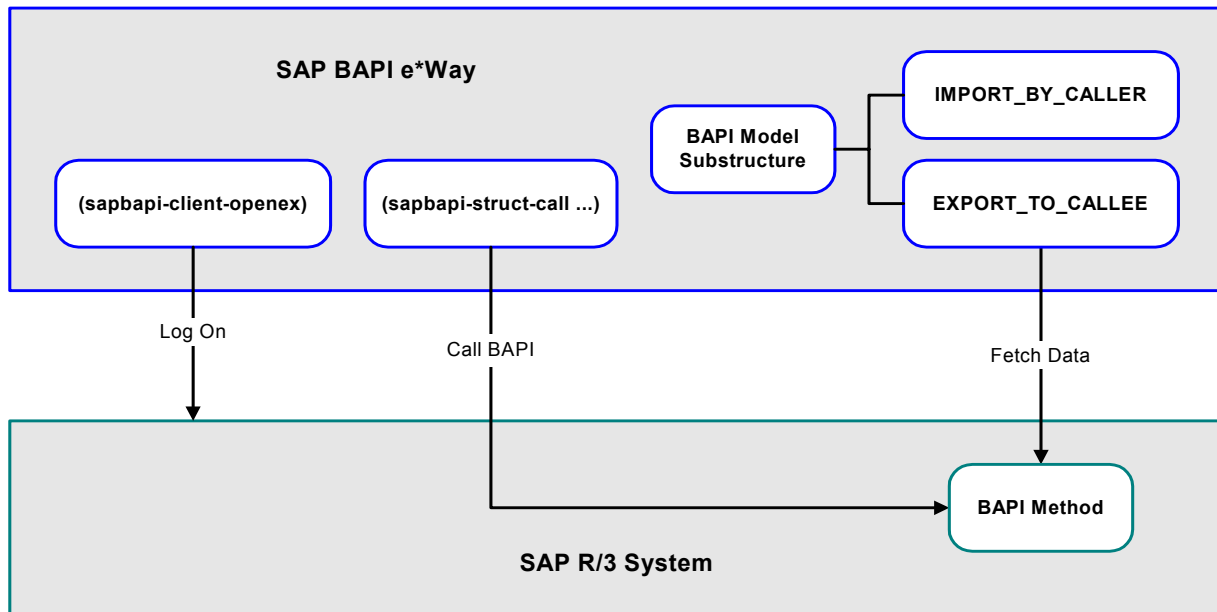
In this mode, the e\*Way typically receives data from the e\*Gate system and sends it to the SAP R/3 system by calling a specific BAPI/RFC method. In return, the called BAPI method may provide some ensuing data to be sent back to the e\*Gate system.

Before any BAPI methods on R/3 can be called, the e\*Way has to log onto the R/3 system using pre-configured parameters including:

- Host Name of the R/3 Target System
- Client
- User
- Password

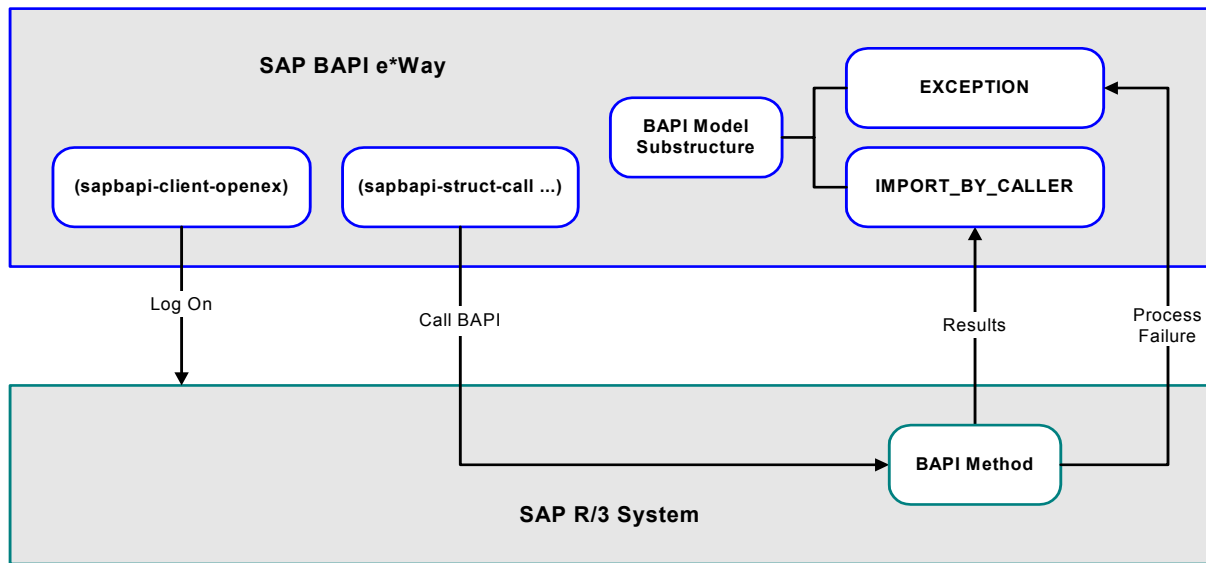
This is done via the [sapbapi-client-openex](#) function. It takes no arguments and returns Boolean true (#t) or false (#f) depending on whether it successfully connected or failed to connect with the R/3 system, respectively.

**Figure 70** RFC Client-mode Fetch



Once a client connection has been established with SAP, a BAPI/RFC method hosted on SAP can be called by the [sapbapi-struct-call](#) function. The BAPI (callee) then fetches the data from `EXPORT_TO_CALLEE`.

Figure 71 RFC Client-mode Response



If execution is successful, the outcome values supplied by the called BAPI become available in the **IMPORT\_BY\_CALLER** and/or **TABLE** parameters. On failure, however, the called BAPI may have raised an exception (available through the **EXCEPTION** node of the method substructure).

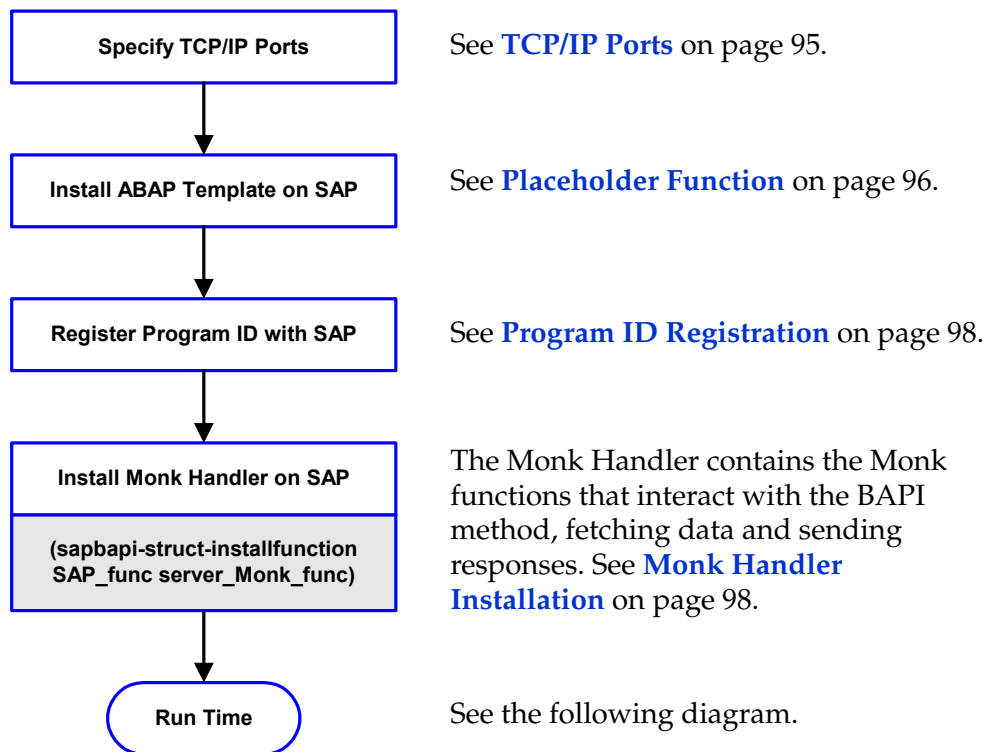
### 6.3.2 Server Mode (SAP to e\*Gate)

In the RFC Server mode, the e\*Way hosts and services a BAPI/RFC method. That is, it has installed on the SAP system one or more listeners for call requests to specific ABAP/4 function modules that are implemented by Monk scripts running in the e\*Way. Before a function can be installed, however, its precise interface—such as the **IMPORT**, **EXPORT**, and **TABLE** parameters—must be known and a corresponding message substructure generated for the method.

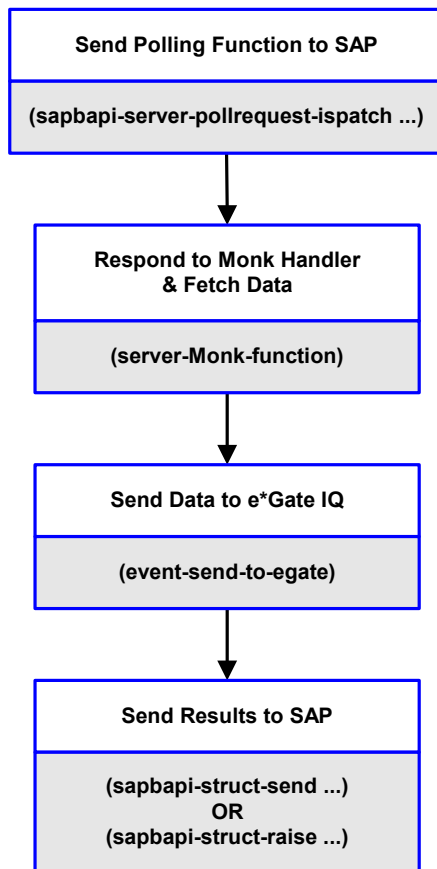
*Note:* The figures included in this section showing the SAP GUI represent a standard SAP 4.0 installation. Your screen may appear somewhat different if you are using a different version of SAP or have modified standard version. See your SAP administrator for more information.

#### Process Overview

##### Setup



## Event Transport



The e\*Way sends the Polling function to SAP (see [Polling SAP for Function Calls](#) on page 99).

When a new Event (transaction) occurs in SAP, SAP sends the Monk Handler function to the e\*Way, which then fetches the data using the Monk function [sapbapi-struct-fetch](#).

The e\*Way sends the data to the e\*Gate IQ, using the function [event-send-to-egate](#).

The e\*Way also completes the Monk Handler tasks, sending any required results to SAP using the Monk function [sapbapi-struct-send](#) or, if a failure occurs, raises an exception with SAP using the Monk function [sapbapi-struct-raise](#).

## TCP/IP Ports

The TCP/IP port numbers are controlled by SAP, and are related to the SAP Gateway Service specified in the e\*Way configuration file. On Windows, the Service Name - TCP/IP Port Number correlation can be found in the directory path C:\Winnt\system32\drivers\etc\Services; on UNIX, in /etc/services.

If the SAP GUI is not installed, this Service Name - Port Number correlation may not be accessible. In this case, the most likely correlation is as follows, where XX = 00 through 99:

| Service | Port     |
|---------|----------|
| sapgwXX | 33XX/tcp |

## Placeholder Function

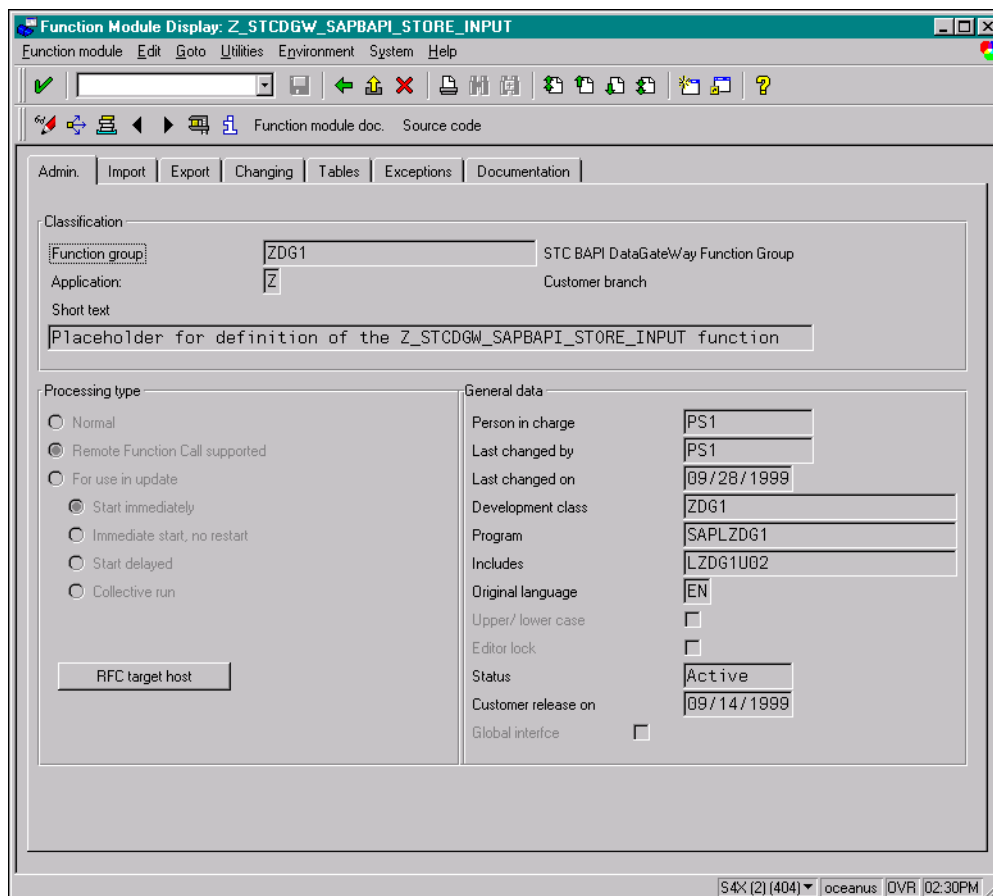
Due to the complex and precise nature of this ETD, we strongly recommend that you use the pre-defined template supplied with the e\*Way. This template, or placeholder function, is placed on SAP during installation to provide the interface definition. This function can then be converted into a working ETD by the BAPI Converter Wizard (see [The BAPI Structure Builder](#) on page 34).

The SAP BAPI e\*Way ships with the pre-defined placeholder function `Z_STCDGW_SAPBAPI_STORE_INPUT`, which receives data into the e\*Way to be stored and/or forwarded to another BAPI.

**Note:** *The function must be **Activated** and **Released** before the BAPI Converter Wizard can interrogate its definition.*

This function can be examined using transaction **SE37**:

**Figure 72** `Z_STCDGW_SAPBAPI_STORE_INPUT` (Admin Tab)





**Figure 73** Z\_STCDGW\_SAPBAPI\_STORE\_INPUT (Import Tab)

| Import parameter | Ref. field/structure | Reference type | Proposal | Optional                            | Reference                |
|------------------|----------------------|----------------|----------|-------------------------------------|--------------------------|
| BAPINAME         | ZDGBAPI - FUNCMO...  |                |          | <input type="checkbox"/>            | <input type="checkbox"/> |
| KEY01            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY02            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY03            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY04            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY05            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY06            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY07            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY08            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY09            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY10            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY11            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY12            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY13            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY14            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY15            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY16            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| KEY17            | ZDGBAPI - USRTXT     |                |          | <input checked="" type="checkbox"/> | <input type="checkbox"/> |

**Figure 74** Z\_STCDGW\_SAPBAPI\_STORE\_INPUT (Export Tab)

| Export parameters | Ref. field/structure | Reference type | Reference                |
|-------------------|----------------------|----------------|--------------------------|
| RETURNCODE        | SY - SUBRC           |                | <input type="checkbox"/> |
|                   |                      |                | <input type="checkbox"/> |

## Program ID Registration

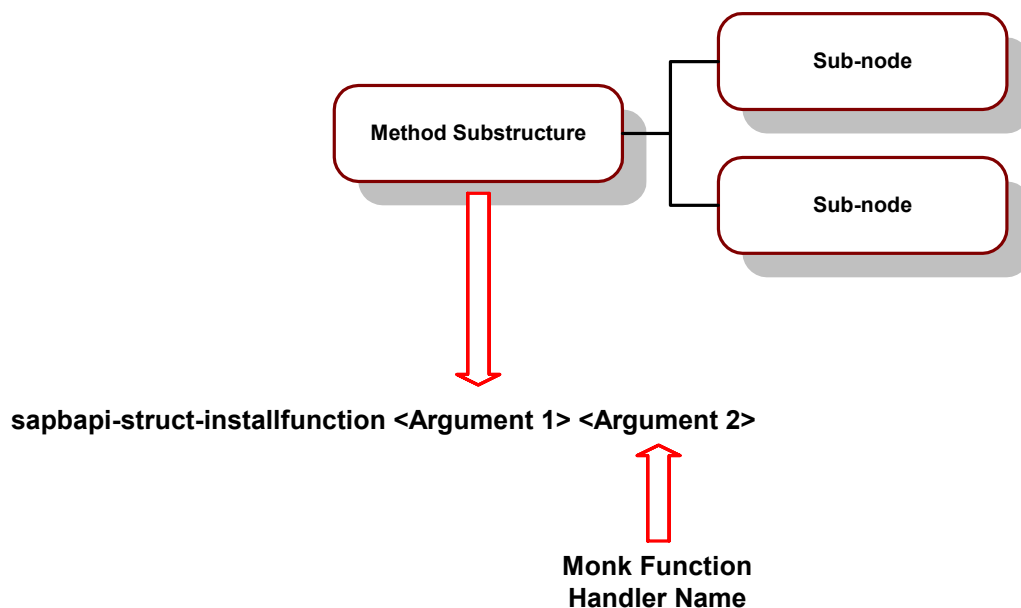
Before the e\*Way can install functions on the SAP system, it must first register its Program ID. This Program ID is associated with an SAP RFC Destination, which should have been set up previously on SAP (see [Registering the e\\*Way](#) on page 42).

Assuming the RFC Destinations are correctly setup on SAP, the [sapbapi-server-register](#) function, which takes no arguments, registers the Program ID for this e\*Way with the configured parameters such as Gateway Host Name and Program ID. The function returns Boolean true (#t) if successful.

## Monk Handler Installation

Once registered, the e\*Way can now install one or more listeners of ABAP/4 function names on the SAP system. When building a Collaboration Rule, simply drag and drop the appropriate method substructure node into the [sapbapi-struct-installfunction](#) Monk function as the first argument. The second argument required is the name of the Monk function that serves as a handler for calls to the installed ABAP/4 function name.

**Figure 75** Monk Function Handler Installation (RFC)



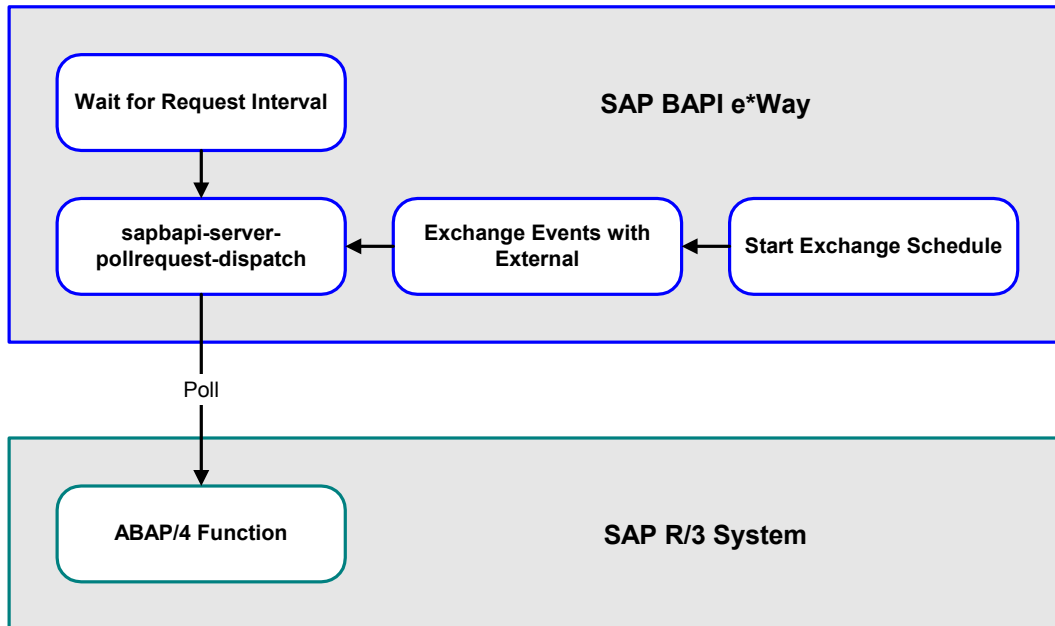
Upon successful completion, [sapbapi-struct-installfunction](#) returns a Boolean true (#t). See also [sapbapi-struct-fetch](#) on page 186 for additional information.

**Note:** The name format is a string between a pair of double quotes such as "sapbapitodgw-handler".

## Polling SAP for Function Calls

On a periodic basis, the BAPI e\*Way polls the SAP system to check for calls to any of the installed ABAP/4 functions. To accomplish this, the scheduling configuration parameters (such as **Start Exchange Data Schedule** on page 135 and **Exchange Data with External Function** on page 140) can be used to trigger calls to the function **sabapi-server-pollrequest-dispatch**. If a call has been received, the installed Monk handler corresponding to the called ABAP/4 function is dispatched automatically.

**Figure 76** Polling for Function Calls (RFC)

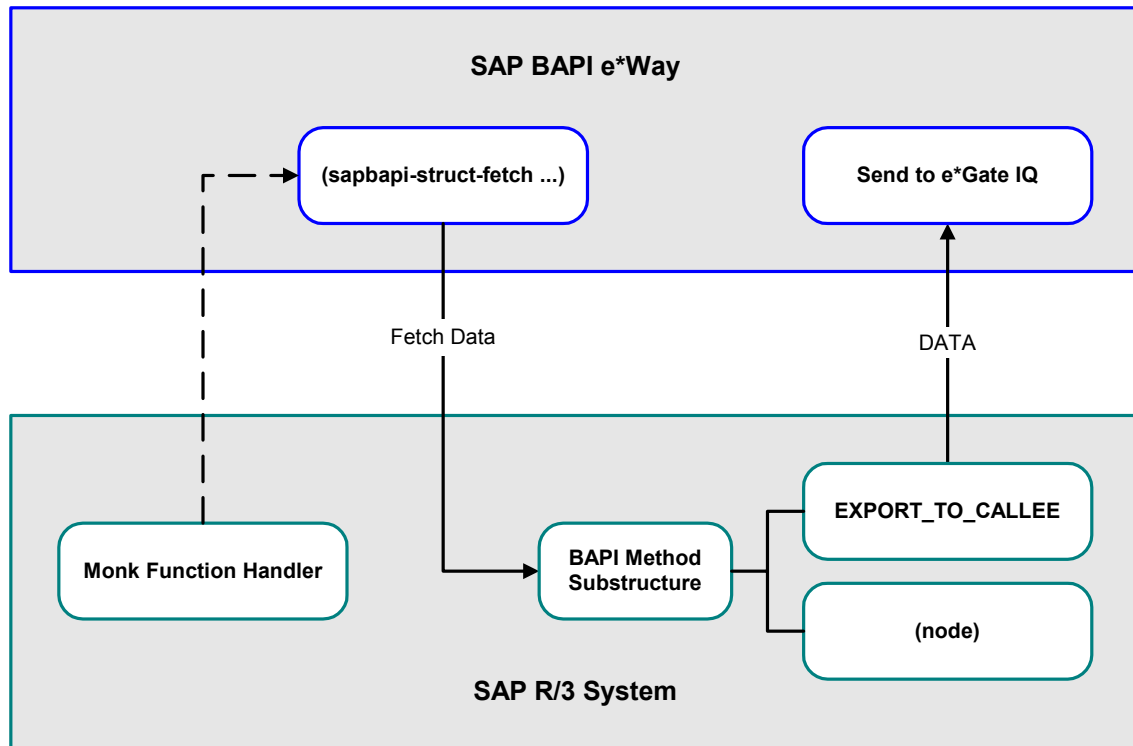


If it executes successfully, or if no activity occurs within the configured timeout (**Wait for Request Interval**) parameter, **sabapi-server-pollrequest-dispatch** returns a Boolean true (#t).

## Extracting Data

When an ABAP/4 function name is installed on R/3, the corresponding Monk function handler is dispatched to service calls to it. The first task of the Monk handler is to fetch the **EXPORT\_TO\_CALLEE** and/or **TABLE** parameters by calling the **sapbapi-struct-fetch** Monk function with the respective installed method substructure as the argument. If successful, the function returns a Boolean true (**#t**). Note that the parameters are exported to the BAPI e\*Way (callee) by the SAP R/3 (caller).

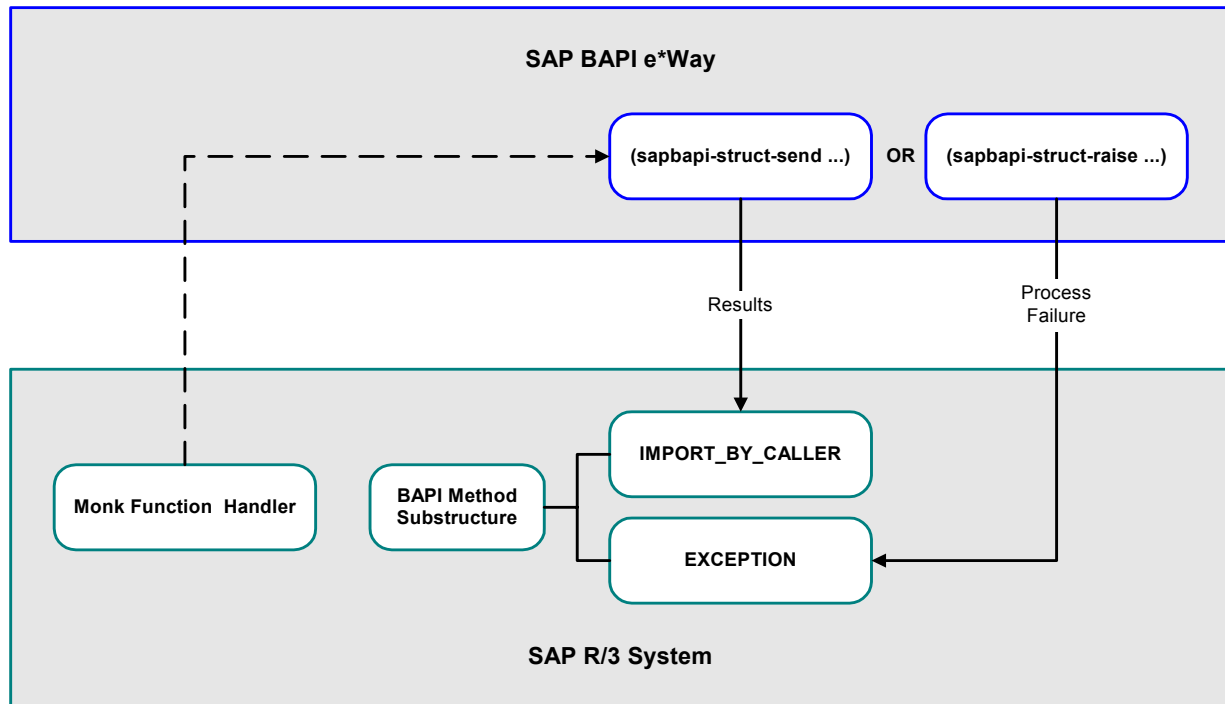
Figure 77 Extracting Data (RFC/tRFC)



## Processing Data and Sending Results

The next task of the corresponding Monk function handler is to process the imported data. If successful, any results (as specified by the ABAP/4 function interface) are inserted into the **IMPORT\_BY\_CALLER** and/or **TABLE** parameter nodes of the appropriate installed method substructure, and the latter placed into the **sapbapi-struct-send** Monk function. When successfully called, the function returns a Boolean true (#t).

**Figure 78** Processing Data (RFC)



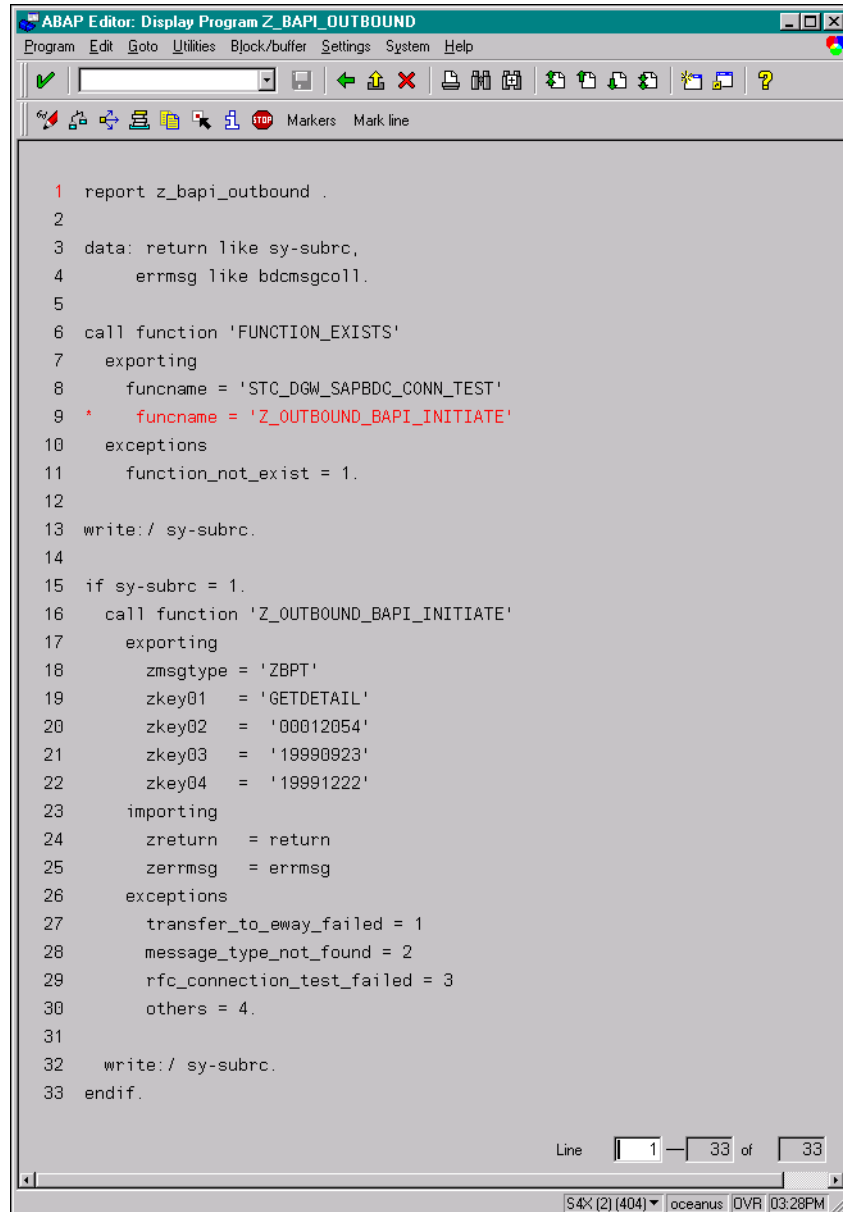
If, however, the corresponding Monk function handler fails to process the imported data, an exception is raised with the SAP caller. This is achieved by placing an exception string into the **sapbapi-struct-raise** Monk function, which subsequently inserts the string into the **EXCEPTION** node of the appropriate installed method substructure. If successful, a Boolean true (#t) is returned.

As part of cleanup, the **sapbapi-server-unregister** Monk function should be called to unregister the Program ID from the SAP system.

## Data Extraction via ABAP

Figure 79 shows the data extraction part of an example custom ABAP module, Z\_BAPI\_OUTBOUND.

Figure 79 ABAP Module Z\_BAPI\_OUTBOUND



```
1 report z_bapi_outbound .
2
3 data: return like sy-subrc,
4       errmsg like bdcmsgcoll.
5
6 call function 'FUNCTION_EXISTS'
7   exporting
8     funcname = 'STC_DGW_SAPBDC_CONN_TEST'
9   *   funcname = 'Z_OUTBOUND_BAPI_INITIATE'
10  exceptions
11    function_not_exist = 1.
12
13 write:/ sy-subrc.
14
15 if sy-subrc = 1.
16   call function 'Z_OUTBOUND_BAPI_INITIATE'
17     exporting
18       zmsgtype = 'ZBPT'
19       zkey01   = 'GETDETAIL'
20       zkey02   = '00012054'
21       zkey03   = '19990923'
22       zkey04   = '19991222'
23     importing
24       zreturn = return
25       zerrmsg = errmsg
26     exceptions
27       transfer_to_eway_failed = 1
28       message_type_not_found = 2
29       rfc_connection_test_failed = 3
30       others = 4.
31
32   write:/ sy-subrc.
33 endif.
```

Line 1 of 33

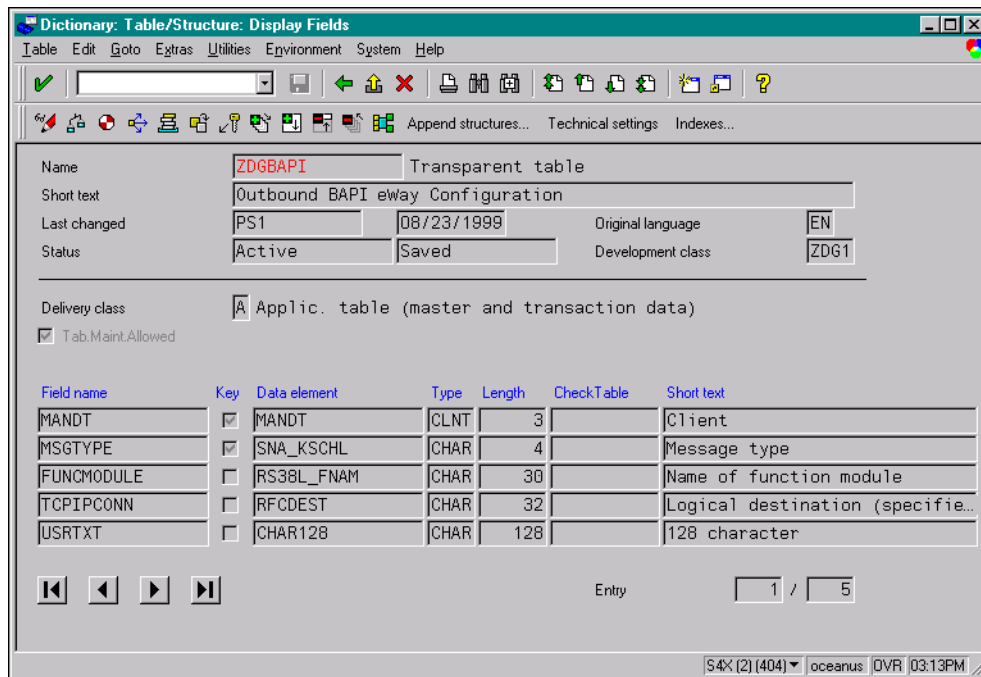
S4X (2) (404) oceanus OVR 03:28PM

In this example, a test function Z\_OUTBOUND\_BAPI\_INITIATE is called to send the data to the SAP BAPI e\*Way. In this function call, the filename, message type, and the internal data table are passed to Z\_OUTBOUND\_BAPI\_INITIATE. The return code from the latter is checked.

The input parameter **message type** tells Z\_OUTBOUND\_BAPI\_INITIATE where to send the data, or the RFC destination.

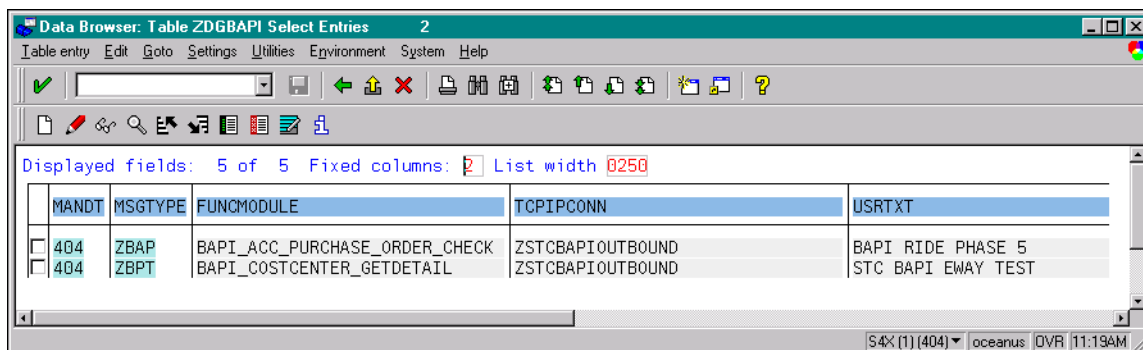
The function Z\_OUTBOUND\_BAPI\_INITIATE determines the destination from an SAP table ZDGBAPI, which is defined as follows.

**Figure 80** ZDGBAPI Definition



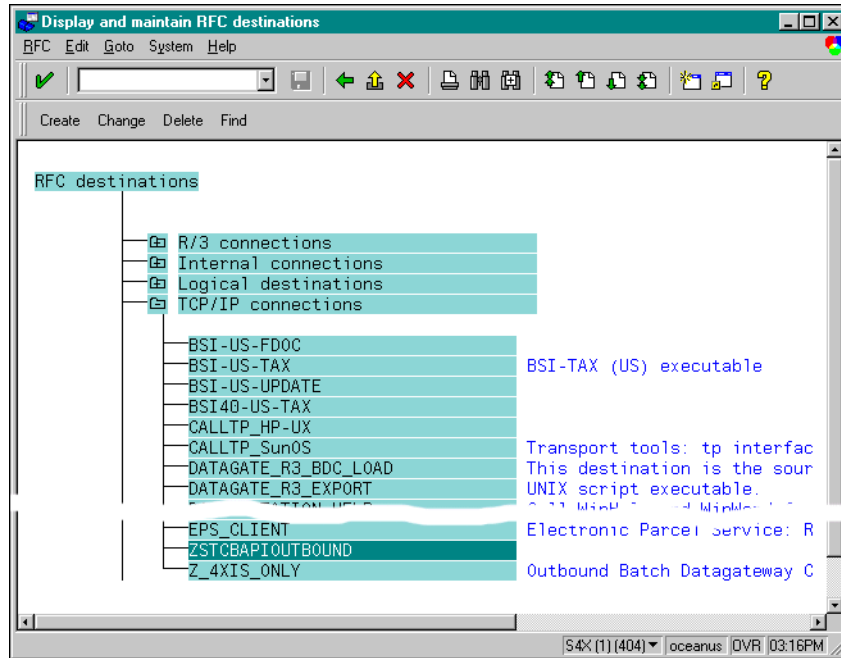
This table can be defined by the user through transaction SE11 (see [Updating SAP R/3 Objects](#) on page 62). An example ZDGBAPI table is shown below.

**Figure 81** ZDGBAPI Table Example



In this table, the message type ZOUTTEST is associated with the RFC destination (TCPIPCONN) OUTBOUND\_TEST. RFC destinations are defined with transaction SM59 as shown below (see [Registering the e\\*Way](#) on page 42).

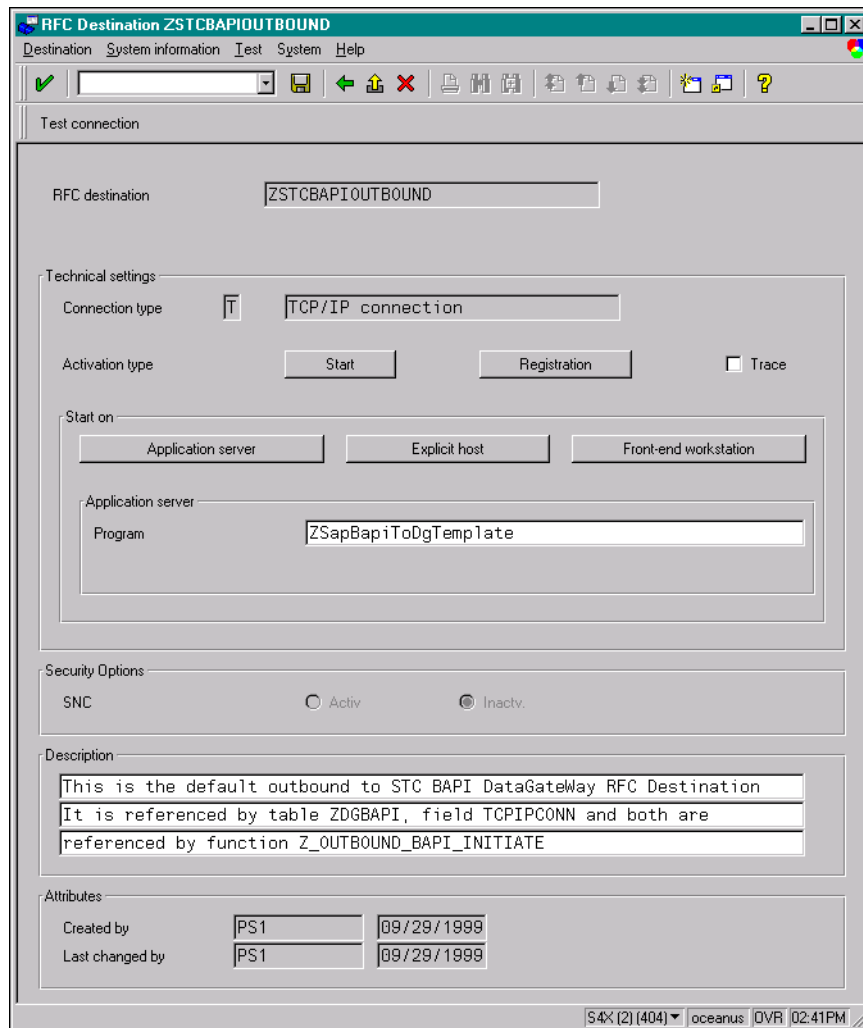
Figure 82 RFC Destinations Tree



Double clicking on the destination ZSTCBAPIOUTBOUND reveals the definition for it.



Figure 83 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\_BAPI\_INITIATE sends messages of the associated type. At e\*Way startup, an SAP BAPI e\*Way registers with the SAP Application Server using a unique Program ID.

When a custom ABAP module calls Z\_OUTBOUND\_BAPI\_INITIATE, a message type is passed together with the message to be sent. With the message type as the key, Z\_OUTBOUND\_BAPI\_INITIATE looks up the RFC destination from the ZDGBAPI table. The message is routed to the correct SAP BAPI e\*Way based on the Program ID.

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

RFCTYPE\_INT1  
RFCTYPE\_INT2  
RFCTYPE\_DATE\_1  
RFCTYPE\_DATE\_1

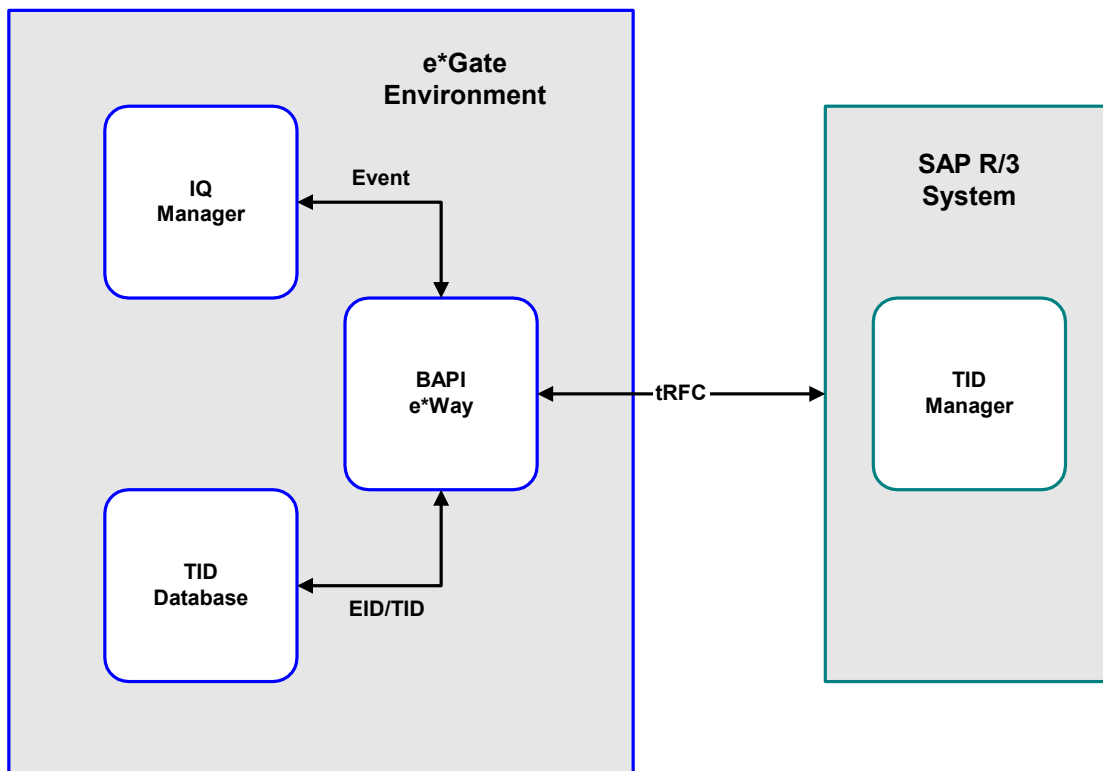
## 6.4 tRFC Transport Process

Events can be sent to the SAP R/3 host via Transactional RFC (tRFC) or regular RFC. To use the tRFC mode, the e\*Way Configuration parameter **Enforce Transactional RFC** must be enabled. Otherwise, the e\*Way sends or receives the Event via regular RFC, which has been described previously.

With tRFC, the receiving SAP system relies on a unique Transactional ID (TID) sent with the message to ascertain whether or not a transaction has ever been processed by it before. This TID is assigned by the SAP R/3 system. Every Event received from this e\*Way is checked against an internal TID database to ensure that it has not already been processed.

A receiving e\*Way exhibits the same basic behavior for Events sent to it by the SAP system, checking the TID against a TID database. Internally, the Event is assigned an Event ID (EID) to track the Event within the e\*Gate environment. When sending an Event to, or receiving an Event from, SAP R/3, there is an interchange process that occurs between EIDs and TIDs. The TIDs (assigned by SAP R/3) are stored in the e\*Way's TID database, referenced to the associated EIDs (assigned by e\*Gate). This section describes this easily-confusing process.

**Figure 84** tRFC Communications



## 6.4.1 Client Mode (e\*Gate to SAP)

As depicted in [Figure 85 on page 108](#), a subscribed Event is forwarded to the e\*Way by the e\*Gate system. If tRFC is enabled, the e\*Way associates the next TID (assigned by the SAP R/3 system) with the outbound Event and sends it via tRFC to SAP.

Using tRFC, you must first determine whether or not the EID has ever been processed by using [saptrfc-get-current-event-id](#) to attempt to find an entry for the EID.

There are now three possible scenarios: the EID associated with the Event may or may not be found; and if it is, the TID may either be reserved or not used. Each scenario involves a different procedure, as outlined below.

### EID Not Found

This is the most straightforward case, and the following procedure applies (see also [Figure 86 on page 109](#)):

- 1 Call [saptrfc-get-tid](#) to obtain a new TID from the SAP system and **reserve** it, referencing it to the Event's EID.
- 2 Send the Event to SAP by using [saptrfc-struct-call](#) to call the appropriate ABAP function (on SAP), which then fetches the Event from the e\*Way.
- 3 Mark the EID/TID pair as being **processed** using [saptrfc-commit-tid](#).

### EID Found but TID Not Used

In this case, the EID is found in the database, but the associated TID is unused (see also [Figure 87 on page 110](#)):

- 1 Call [saptrfc-get-tid](#) to mark the TID as **reserved**.
- 2 Send the Event to SAP by using [saptrfc-struct-call](#) to call the appropriate ABAP function (on SAP), which then fetches the Event from the e\*Way.
- 3 Mark the EID/TID pair as being **processed** using [saptrfc-commit-tid](#).

### EID Found and TID Reserved

In this case, the EID is found in the database, and the associated TID is **reserved**, but not **processed** (see also [Figure 88 on page 111](#)):

- 1 Call [saptrfc-get-tid](#) to **re-use** the previously-reserved TID.
- 2 Send the Event to SAP by using [saptrfc-struct-call](#) to call the appropriate ABAP function (on SAP), which then fetches the Event from the e\*Way.
- 3 Expunge the EID/TID pair from the TID database using [saptrfc-delete-tid](#).

When the [Process Outgoing Message Function](#) exits, it automatically performs a **Post Event Complete** to the e\*Gate system, indicating that the e\*Way is ready for the next Event.

Figure 85 tRFC Client Mode Process Flow

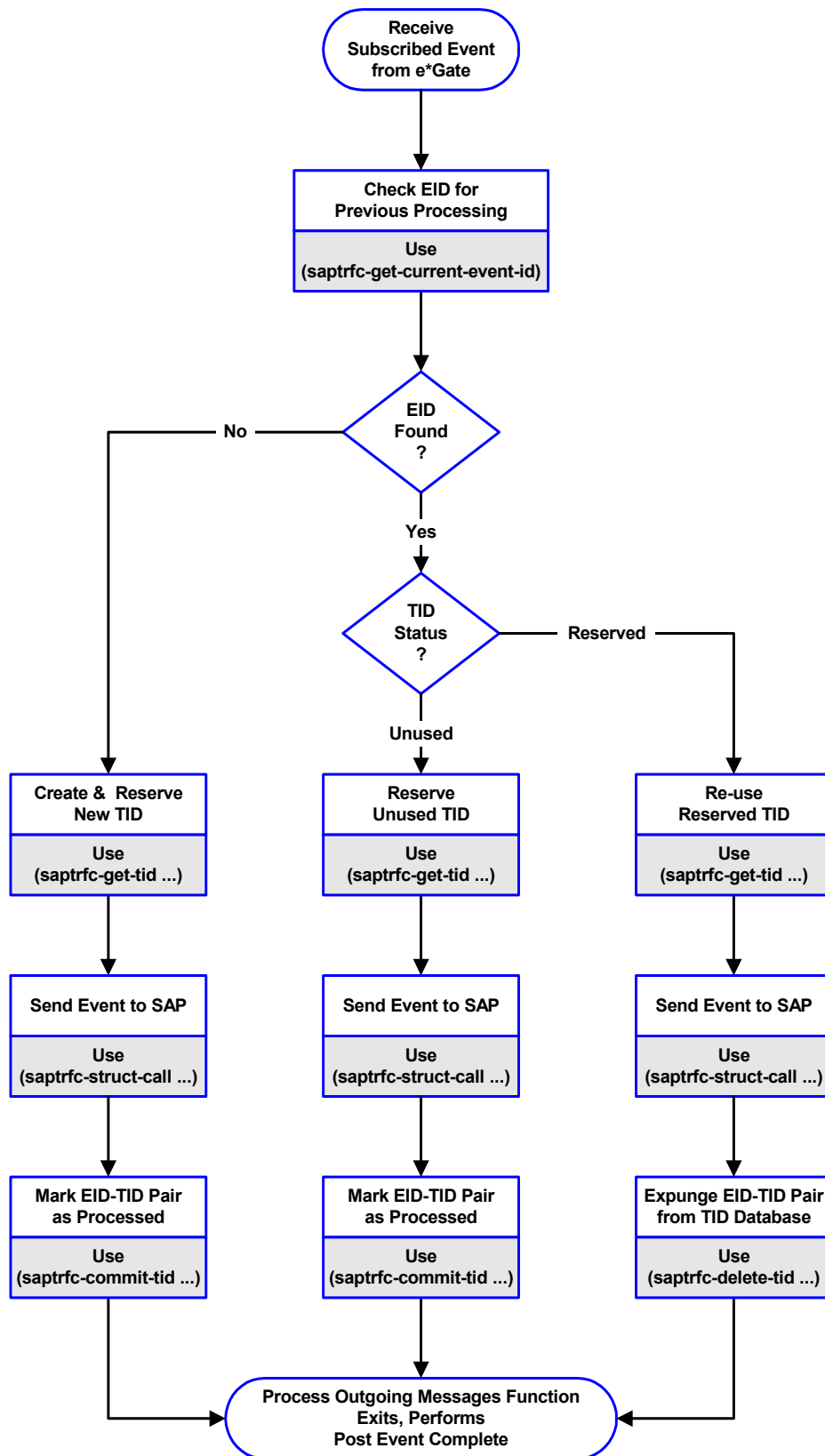


Figure 86 tRFC Scenario 1: EID Not Found

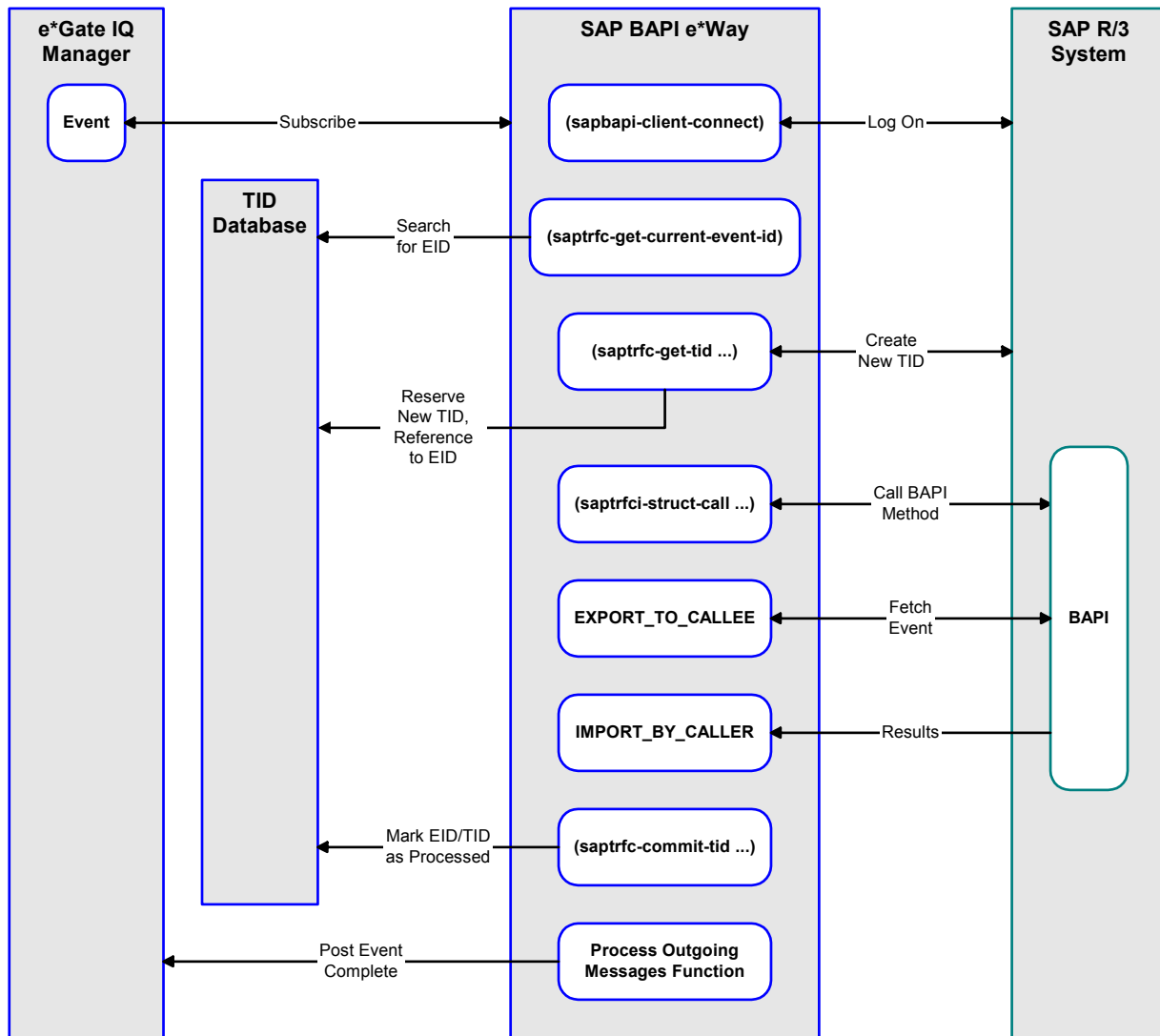


Figure 87 tRFC Scenario 2: EID Found, TID Unused

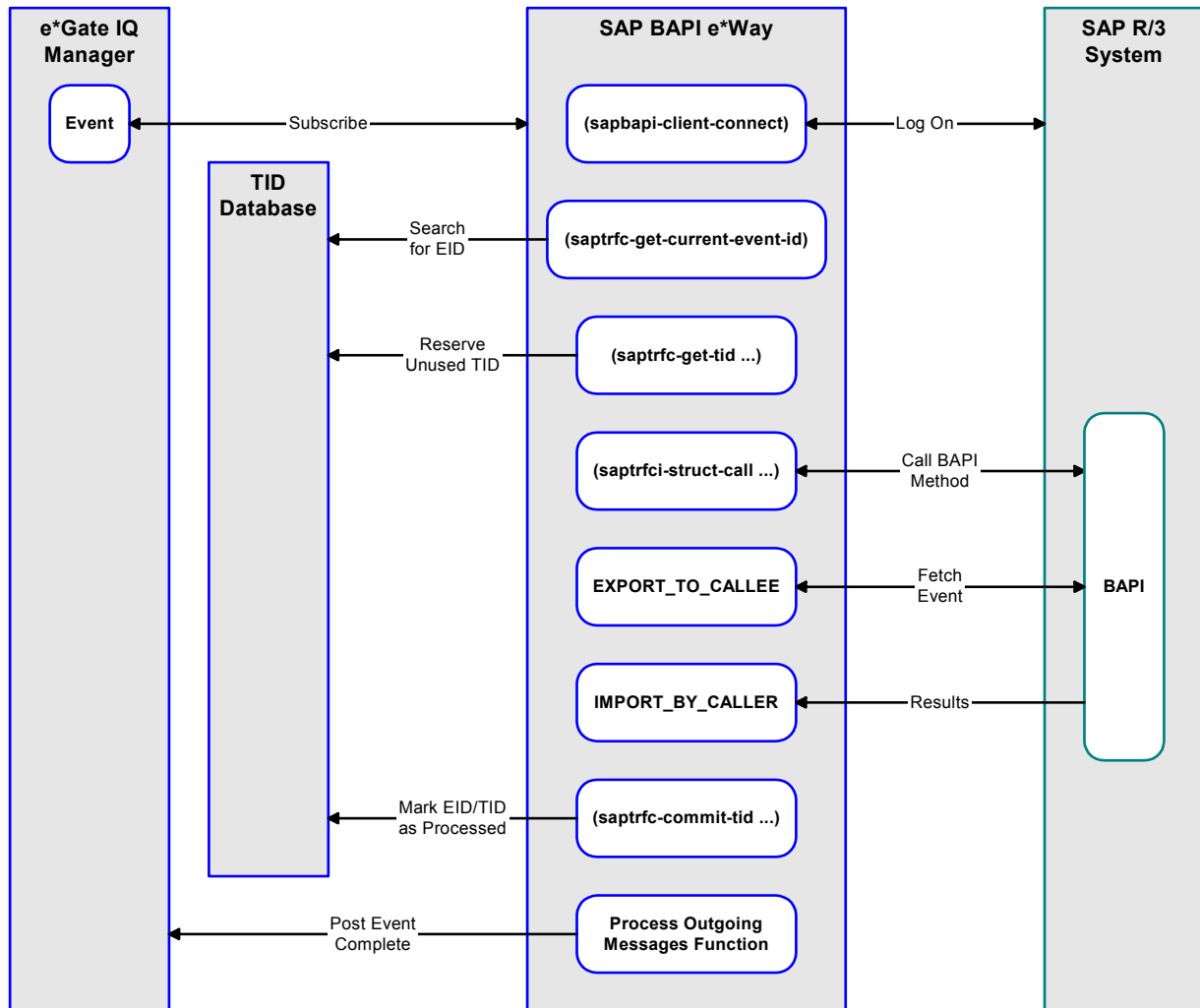
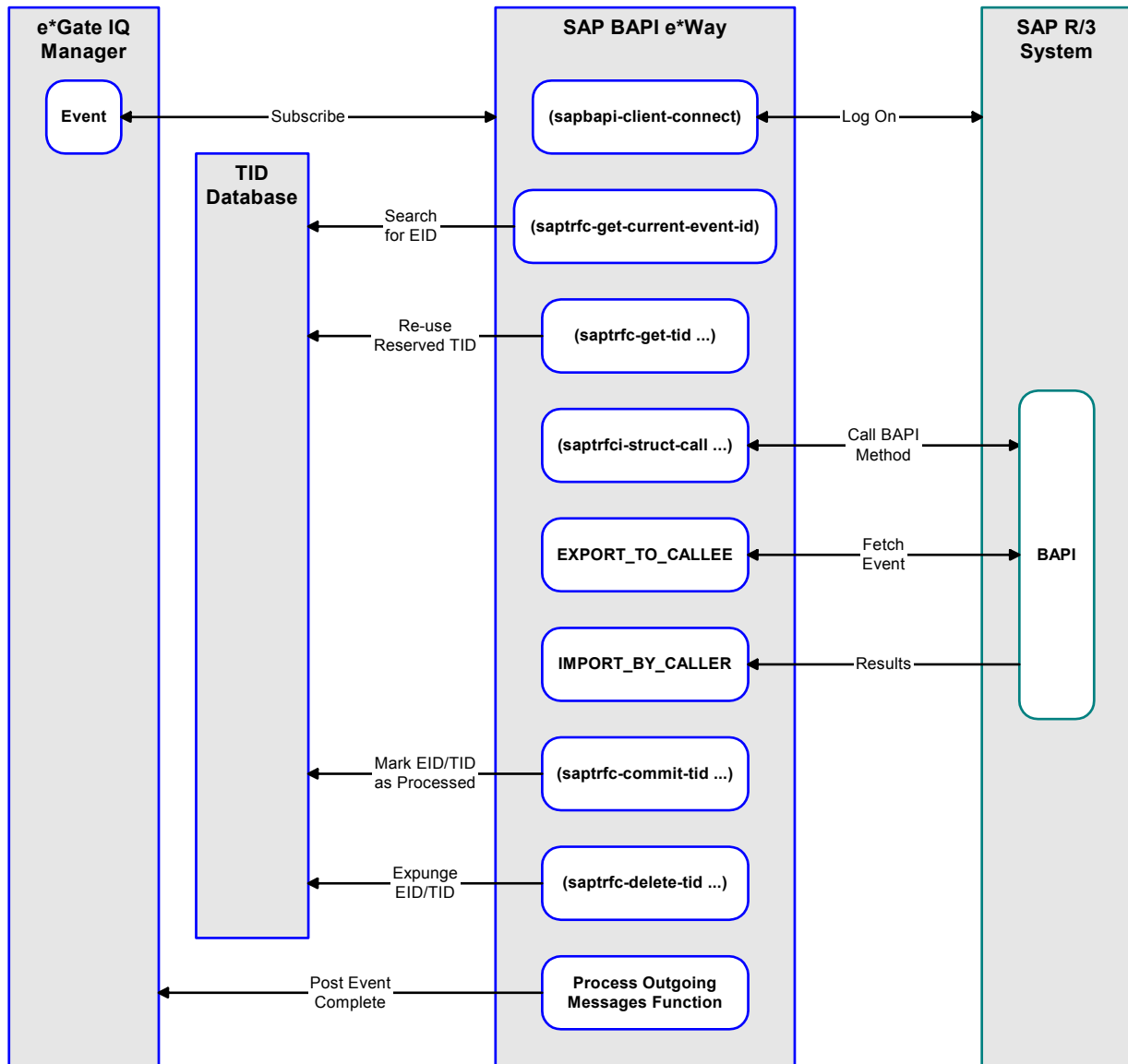


Figure 88 tRFC Scenario 3: EID Found, TID Reserved

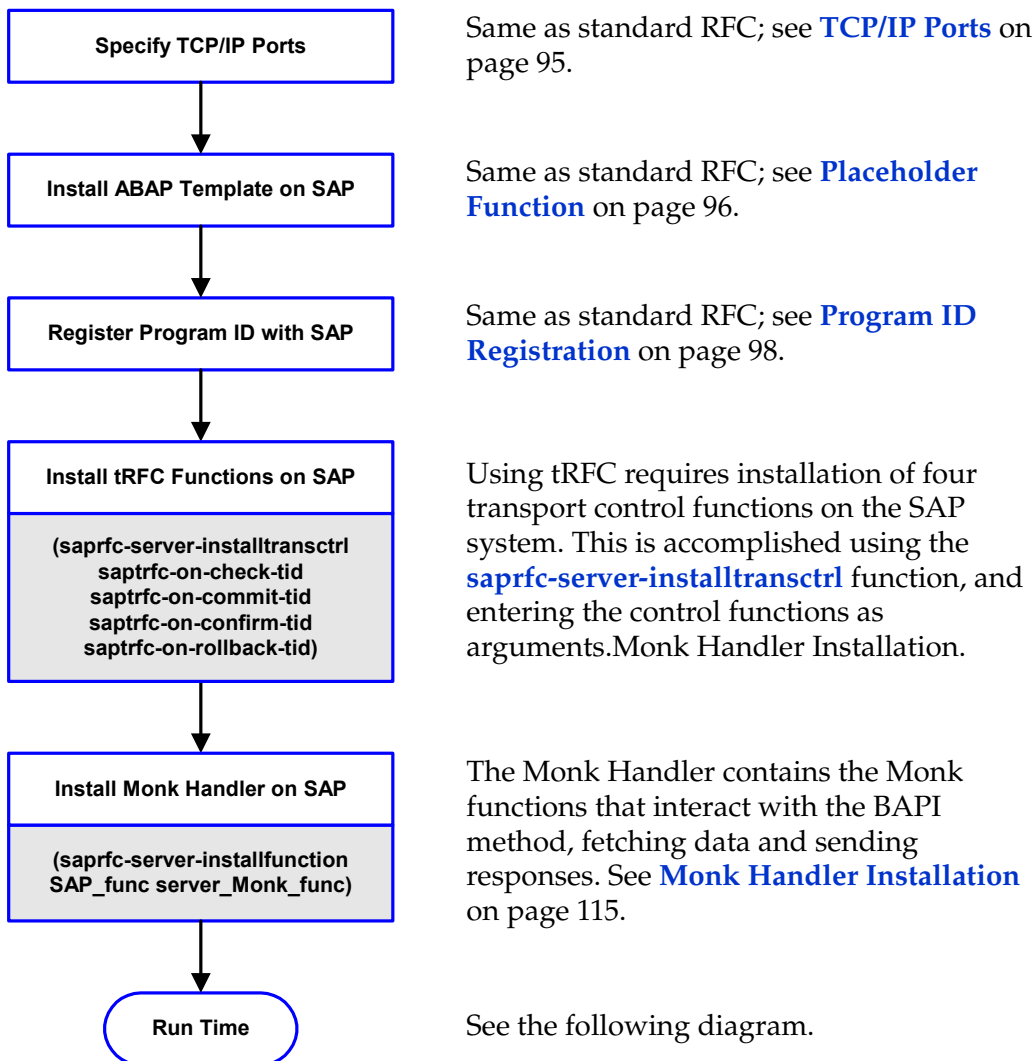


## 6.4.2 Server Mode (SAP to e\*Gate)

In the tRFC Server mode, as with the standard RFC mode, the e\*Way is hosting and servicing a BAPI/RFC method. Before any functions can be installed, its precise interface—such as the **IMPORT**, **EXPORT**, and **TABLE** parameters—must be known and a corresponding message substructure generated for the method.

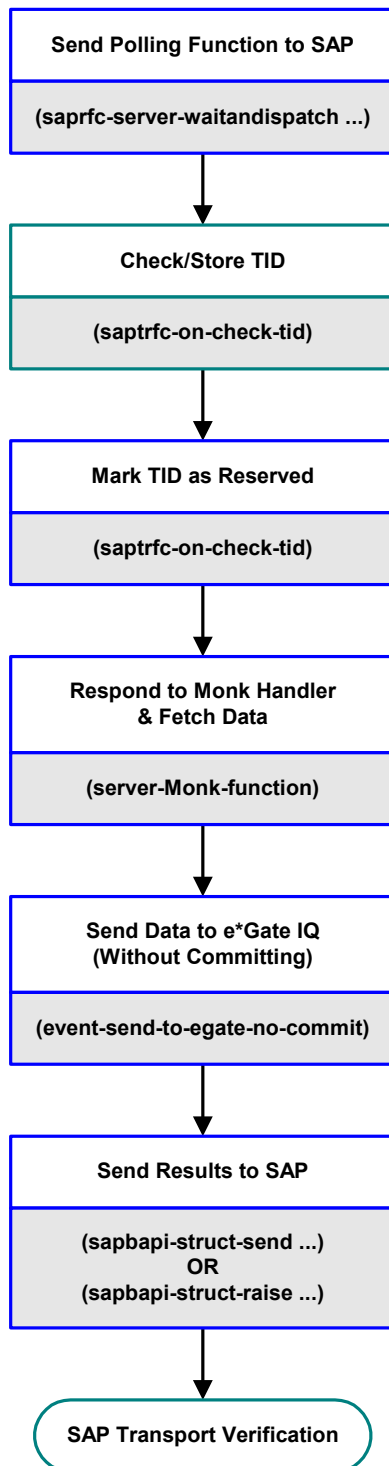
### Process Overview

#### Setup





## Event Transport



The e\*Way sends the Polling function to SAP (see [Checking SAP for Function Calls](#) on page 115).

When a new Event (transaction) occurs in SAP, SAP sends the installed Monk function [saptrfc-on-check-tid](#) to the e\*Way with a TID.

The e\*Way checks and/or stores TID in the TID database, and marks the TID as being reserved.

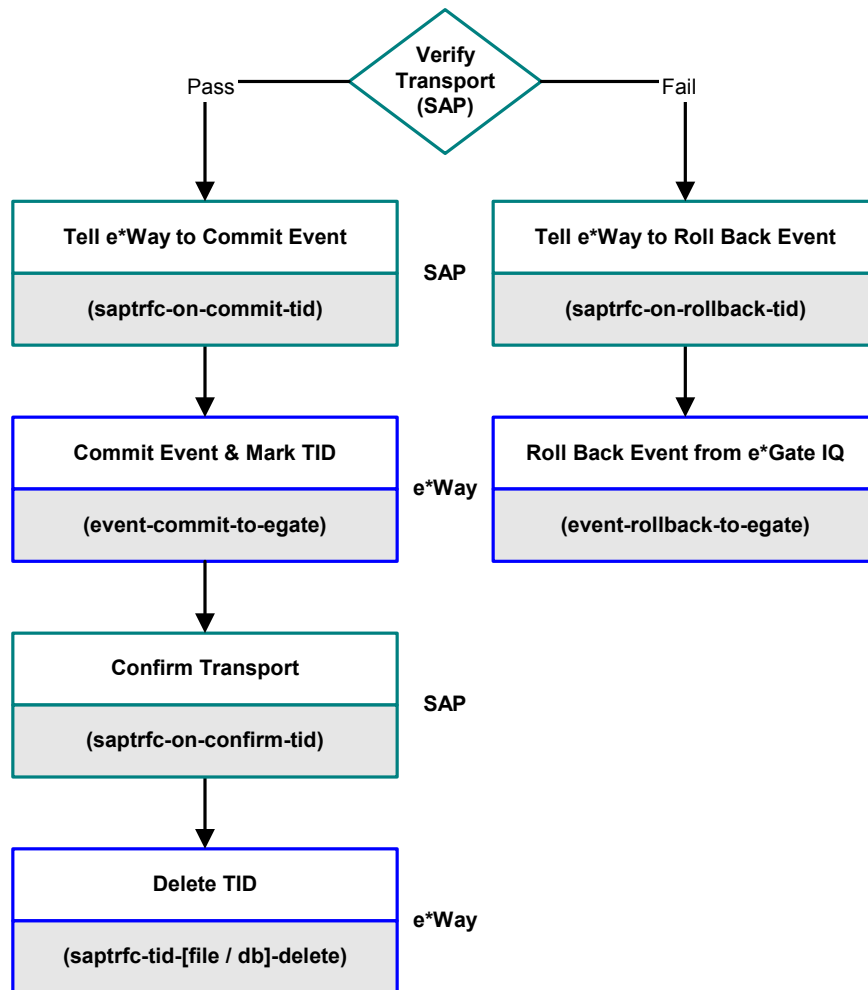
SAP sends the Monk Handler function to the e\*Way, which then fetches the data using the Monk function [sapbapi-struct-fetch](#).

The e\*Way sends the data to the e\*Gate IQ without committing, pending verification by SAP.

The e\*Way also completes the Monk Handler tasks, sending any required results to SAP using the Monk function [sapbapi-struct-send](#) or, if a failure occurs, raises an exception with SAP using the Monk function [sapbapi-struct-raise](#).

The SAP system proceeds with its verification of the transaction and transport (see the following diagram).

## Event Verification



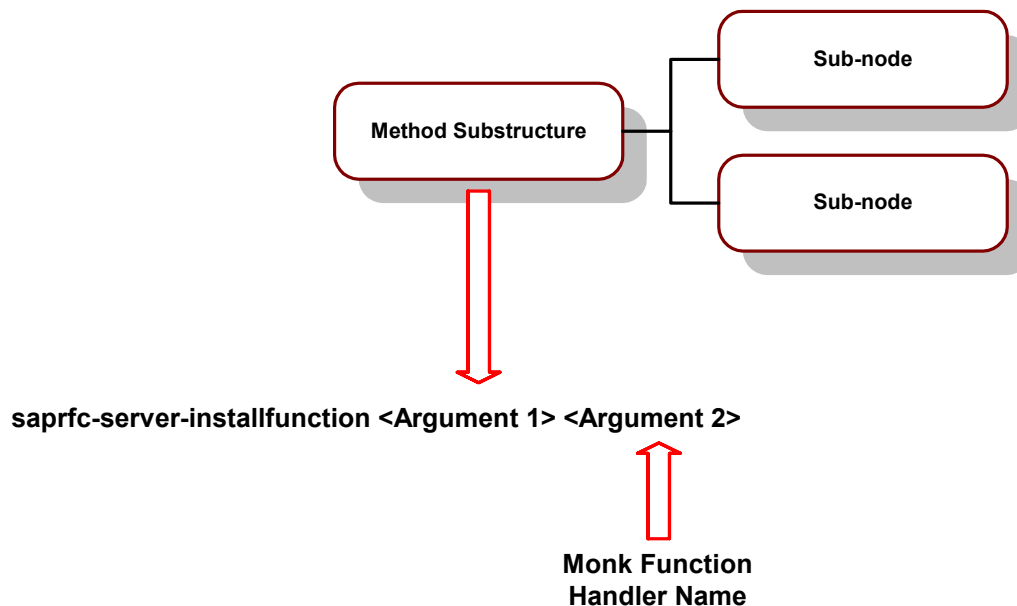
- **Pass** (see [Figure 91 on page 117](#))
  - A SAP sends the installed Monk function **saptrfc-on-commit** to the e\*Way, requesting that the Event/transaction be committed.
  - B The BAPI e\*Way commits the Event to the IQ and marks the corresponding TID as being processed using the Monk function **event-commit-to-egate**.
  - C SAP sends the installed Monk function **saptrfc-on-confirm-tid** to the e\*Way, confirming verification of the Event.
  - D The BAPI e\*Way deletes the corresponding TID using the Monk function **saptrfc-tid-file-delete** (or **saptrfc-tid-db-delete**, if using an RDBMS).
- **Fail** (see [Figure 92 on page 118](#))
  - A SAP sends the installed Monk function **saptrfc-on-rollback-tid** to the e\*Way, requesting that the Event/transaction be rolled back.
  - B The BAPI e\*Way rolls back the Event from the IQ using the Monk function **event-rollback-to-egate** and deletes the corresponding TID.

## Monk Handler Installation

Now the e\*Way can install one or more listeners of ABAP/4 function names on the SAP system. When building a Collaboration Rule, simply drag and drop the appropriate method substructure node into the **saprfc-server-installfunction** Monk function as the first argument. The second argument required is the name of the Monk function that serves as a handler for calls to the installed ABAP/4 function name.

*Note:* See [Chapter 8](#) for more information on the SAP BAPI Monk functions.

**Figure 89** Monk Function Handler Installation (tRFC)



See [sapbapi-struct-fetch](#) on page 186 for more information. Upon successful completion, **saprfc-server-installfunction** returns a Boolean true (#t).

*Note:* The name format is a string between a pair of double quotes such as "sapbapitodgw-handler".

## Checking SAP for Function Calls

The BAPI e\*Way sends the Monk function **saprfc-server-waitanddispatch** to the SAP system to wait for calls to any of the installed ABAP/4 functions. When a call is received, the installed Monk handler corresponding to the called ABAP/4 function is dispatched automatically.

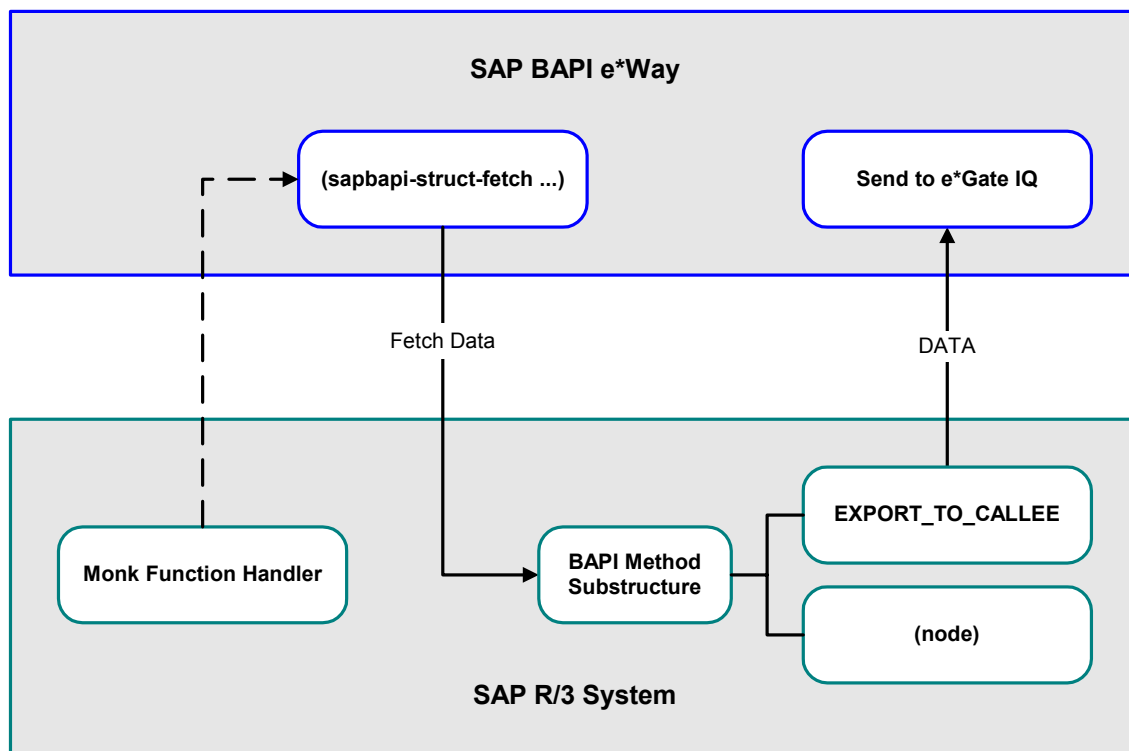
If it executes successfully, or if no activity occurs within the configured timeout (**Wait for Request Interval** parameter), **saprfc-server-waitanddispatch** returns a Boolean true (#t).

## Extracting and Processing Data

When an appropriate transaction occurs in SAP R/3, SAP sends a Monk transport function, **saptrfc-on-check-tid**, to the BAPI e\*Way with the SAP-assigned TID. The e\*Way then reserves this TID in the e\*Gate TID database.

When an ABAP/4 function name is installed on R/3, the corresponding Monk function handler is dispatched to service calls to it. As in the standard RFC case, the first task of the Monk handler is to fetch the **EXPORT\_TO\_CALLEE** and/or **TABLE** parameters by calling the **sapbapi-struct-fetch** Monk function with the respective installed method substructure as the argument. If successful, the function returns a Boolean true (**#t**). Note that the parameters are exported to the BAPI e\*Way (callee) by the SAP R/3 (caller).

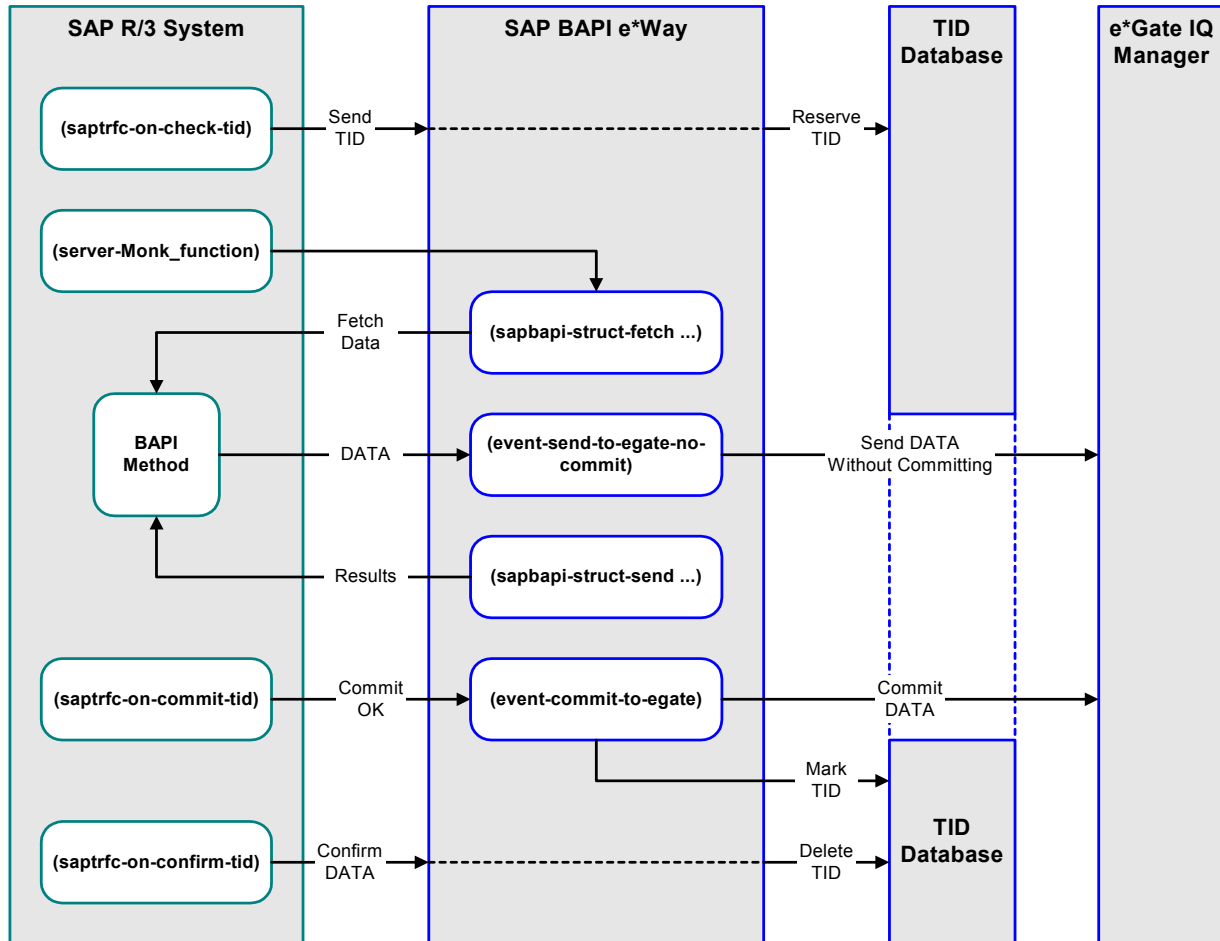
**Figure 90** Server Mode (SAP to e\*Gate) - Fetch



At this point, the process again differs from standard RFC since the Event is sent to the e\*Gate IQ *without committing*. The SAP system proceeds to verify that the tRFC transport was accomplished correctly, with two possible outcomes:

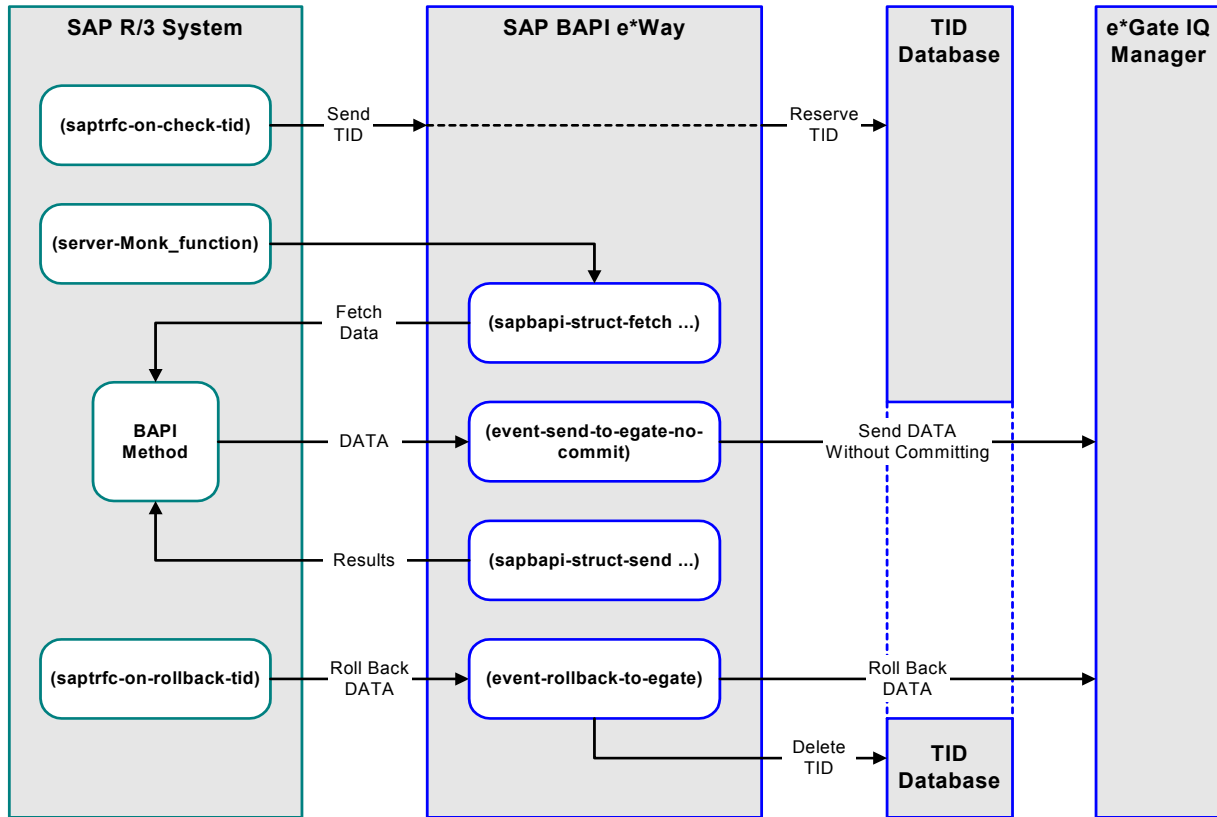
- 1 The transport is verified as correct (see Figure 91).
  - A SAP sends the installed Monk function `saptrfc-on-commit-tid` to the e\*Way.
  - B The BAPI e\*Way commits the event using `event-commit-to-egate` and marks the TID as being processed.
  - C SAP sends the installed Monk function `saptrfc-on-confirm-tid` to the e\*Way.
  - D The BAPI e\*Way deletes the TID from the TID database.

**Figure 91** Server Mode (SAP to e\*Gate) - Commit



- 2 The verification fails (see Figure 92).
  - A SAP sends the installed Monk function `saptrfc-on-rollback-tid` to the e\*Way.
  - B The BAPI e\*Way rolls back the event from the e\*Gate IQ using `event-rollback-to-egate` and deletes the TID from the TID database.

**Figure 92** Server Mode (SAP to e\*Gate) - Rollback

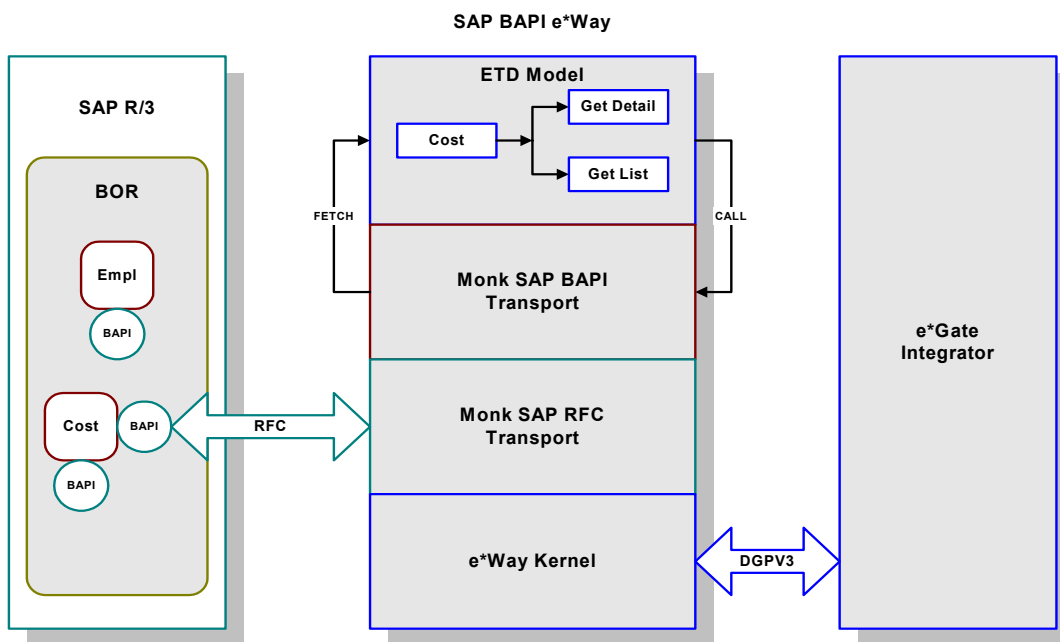


## 6.5 e\*Way Architecture

Conceptually, the e\*Way can be viewed as a multi-layered structure (see Figure 93), consisting of:

- Event Type Definition (ETD) Model layer, which translates SAP Business Objects into Event Type Definitions
- Monk SAP BAPI Transport layer, which manages communication to and from BAPIs
- 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 93** SAP BAPI e\*Way Architectural Model



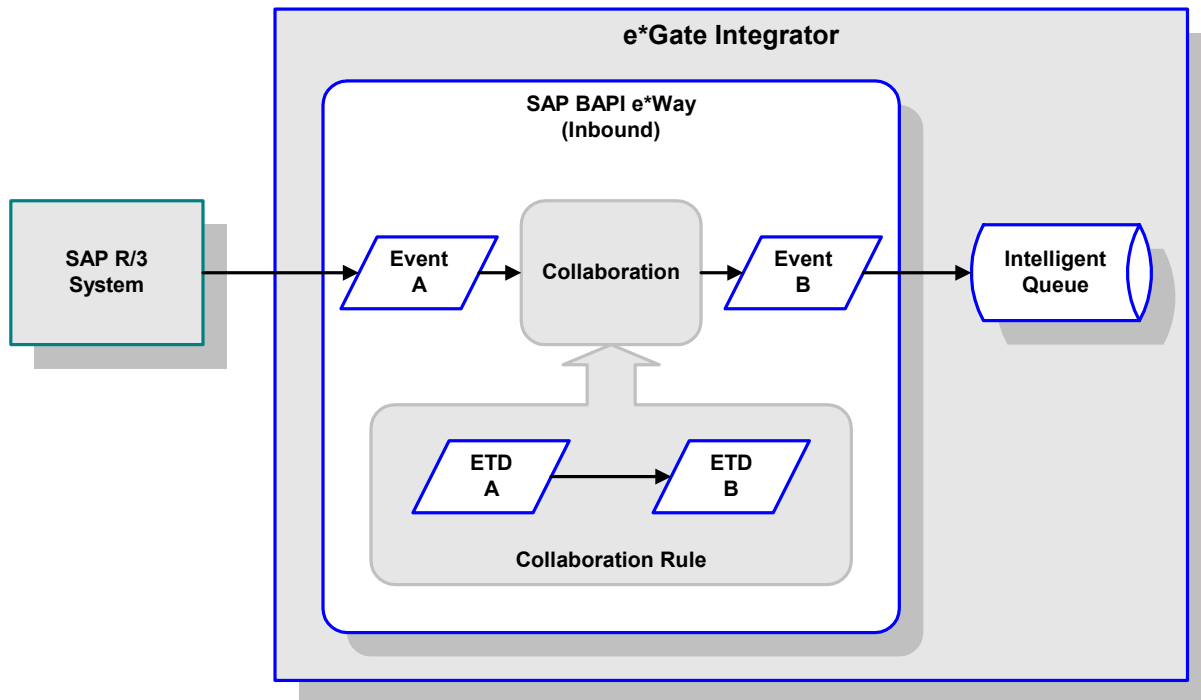
The upper layers of the e\*Way use Monk functions to perform Business Process modeling and ETD mapping, package data as e\*Gate *Events*, send those Events to Collaborations, and manage interaction with the external system. These layers are built upon an e\*Way Kernel layer that manages the basic operations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

## 6.5.1 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 94. Any Collaboration that publishes its processed Events only to an external system does *not* require *any* IQs.

**Figure 94** 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.

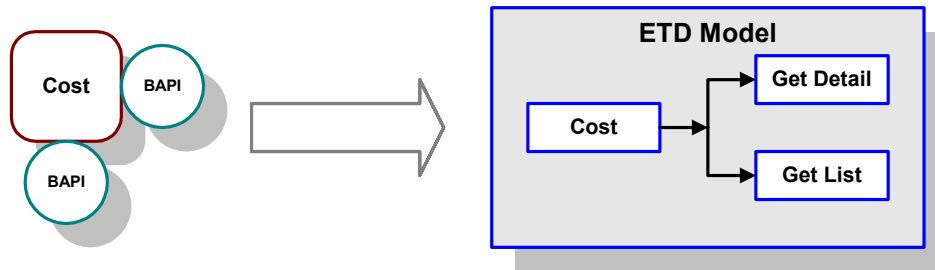
For more information on defining Collaborations, defining IQs, assigning Collaborations to e\*Ways, or configuring Collaborations to publish Events, see the *e\*Gate Integrator User's Guide*.



## 6.5.2 ETD Model Layer

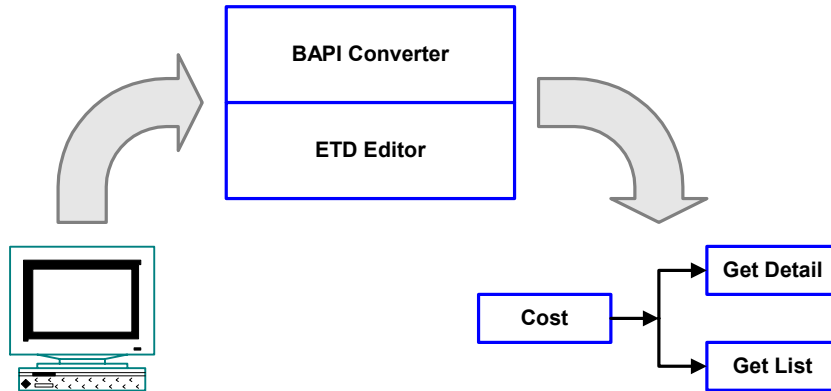
This topmost layer contains the mechanism for creating an Event Type Definition (ETD) representation of the relevant SAP Business Objects. Although this layer is primarily concerned with the modeling of an object via an ETD, it also includes all of the ancillary functions that help manage data (such as “copy” and “loop”), which are available within the Collaboration Rules Editor.

**Figure 95** ETD Model Layer



The BAPI Structure Builder enables you to build automatically an ETD representing an Object defined in SAP, using metadata taken dynamically from SAP. It consists of a BAPI Converter Wizard, which is integrated with the e\*Gate ETD Editor.

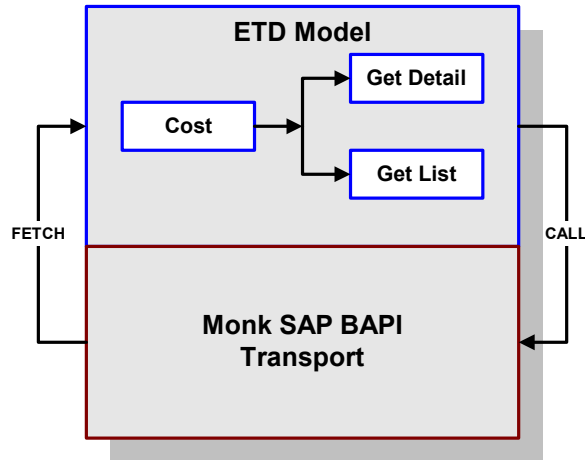
**Figure 96** Structure Creation Process



### 6.5.3 SAP BAPI Transport Layer

The ETD Model Layer provides a container for the data exchanged with a BAPI or RFC method. It is the job of the Monk SAP BAPI Transport Layer to either send or retrieve information to or from the BAPI.

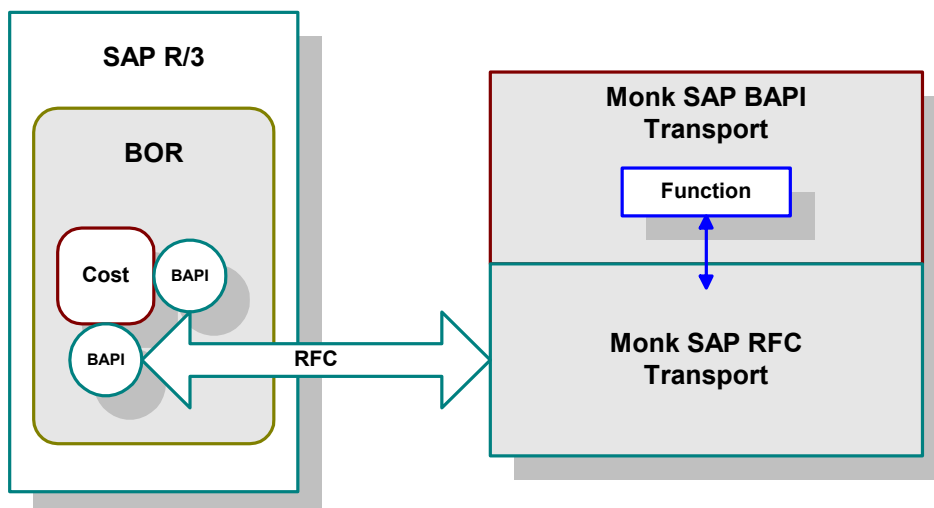
Figure 97 BAPI Transport Layer



### 6.5.4 SAP RFC Transport Layer

This layer provides direct RFC access to SAP R/3 and many of its functions have been abstracted and encapsulated into functions available in the Monk SAP BAPI Transport Layer. For more information about any Monk functions, see [Chapter 8](#). Generally, Monk RFC functions should be called only if the BAPI Monk functions cannot address a specific need.

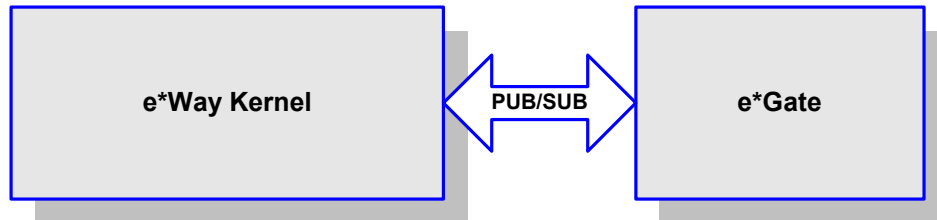
Figure 98 RFC Transport Layer



## 6.5.5 e\*Way Kernel Layer

The Monk functions in this layer allow the user's business logic to interact with the e\*Way and, thus, the e\*Gate system portal. They include functionalities such as sending a message onto the e\*Gate system for subsequent translating or routing to another destination; sending an alert to the Schema Manager if a certain error situation occurs, or informing the e\*Way that an external connection is unavailable and for the latter to keep retrying connections at a later time.

**Figure 99** e\*Way Kernel Layer

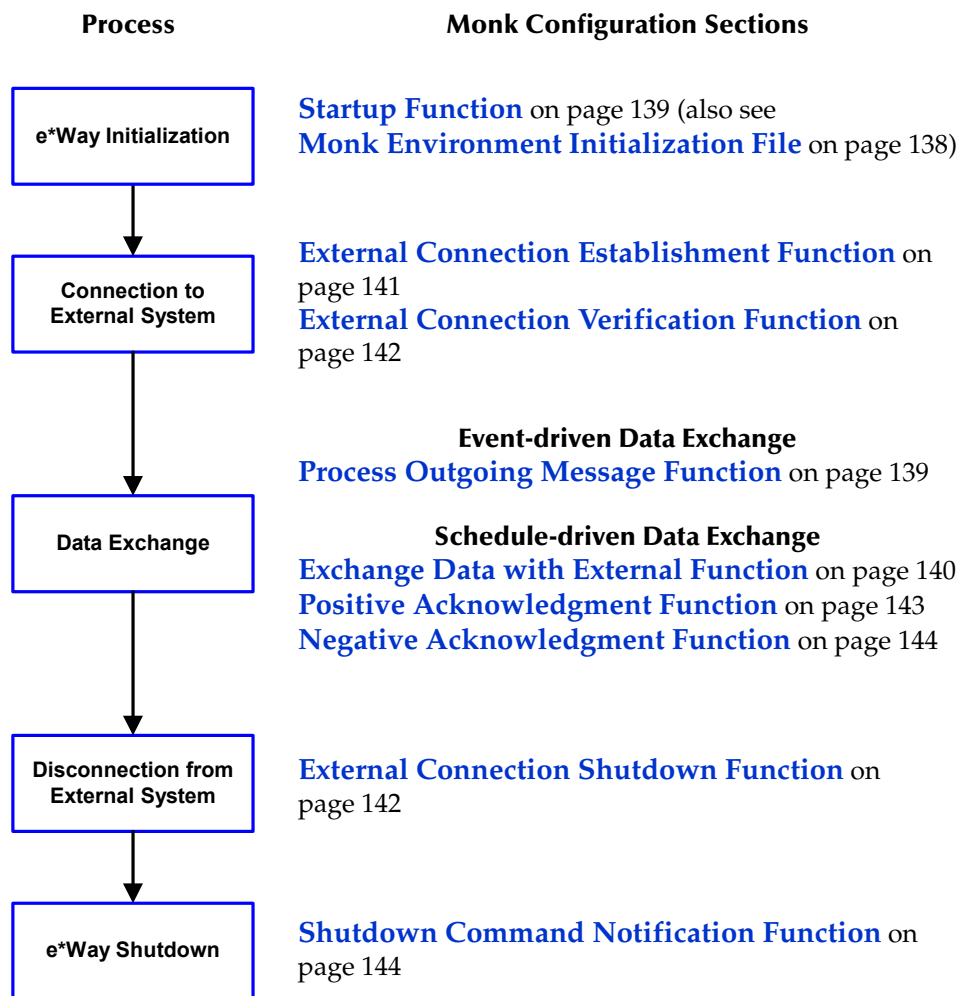


## 6.6 Basic e\*Way Processes

**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 BAPI e\*Way.

The most basic processes carried out by an e\*Way are listed in Figure 100. 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 100** Basic e\*Way Processes

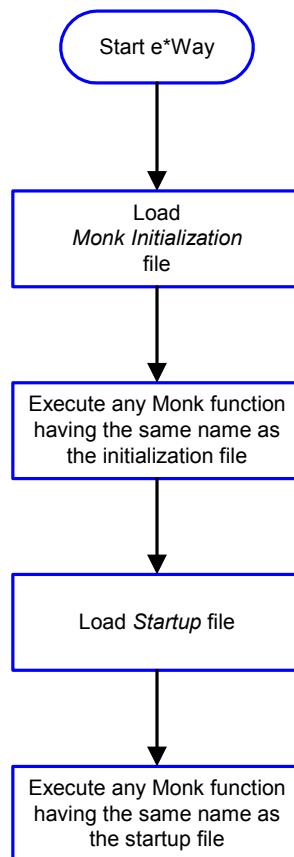


A series of diagrams on the next several pages illustrate the interaction and operation of these functions during the specified processes. Configuring the parameters associated with these functions is covered in [Chapter 7](#), while the functions themselves are described in [Chapter 8](#).

## Initialization Process

Figure 101 illustrates the e\*Way's initialization process, using the **Monk Environment Initialization File** and **Startup Function**.

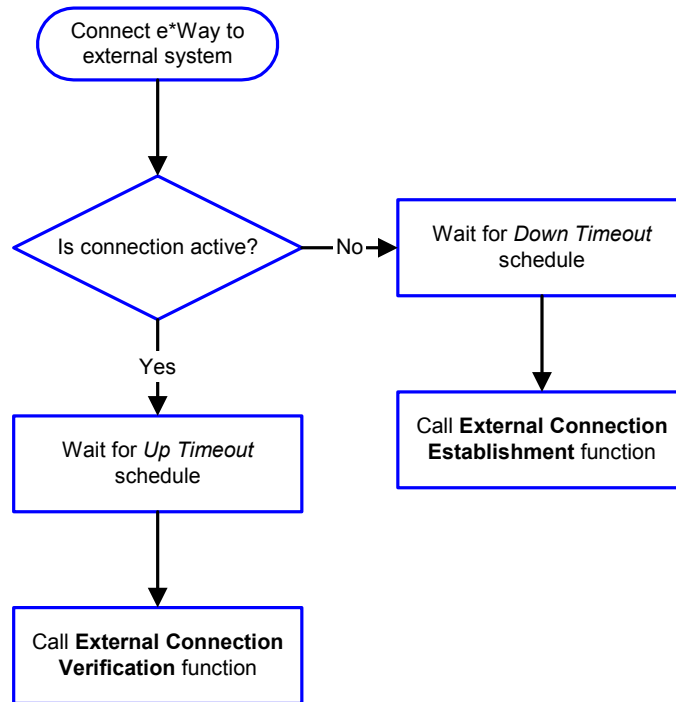
**Figure 101** Initialization Process



## Connect to External Process

Figure 102 illustrates how the e\*Way connects to the external system, using the **External Connection Establishment Function** and **External Connection Verification Function**.

**Figure 102** 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 104 on page 128](#) and [Figure 103 on page 127](#) 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 251](#) and [send-external-down on page 251](#) for more information.

## Data Exchange Process

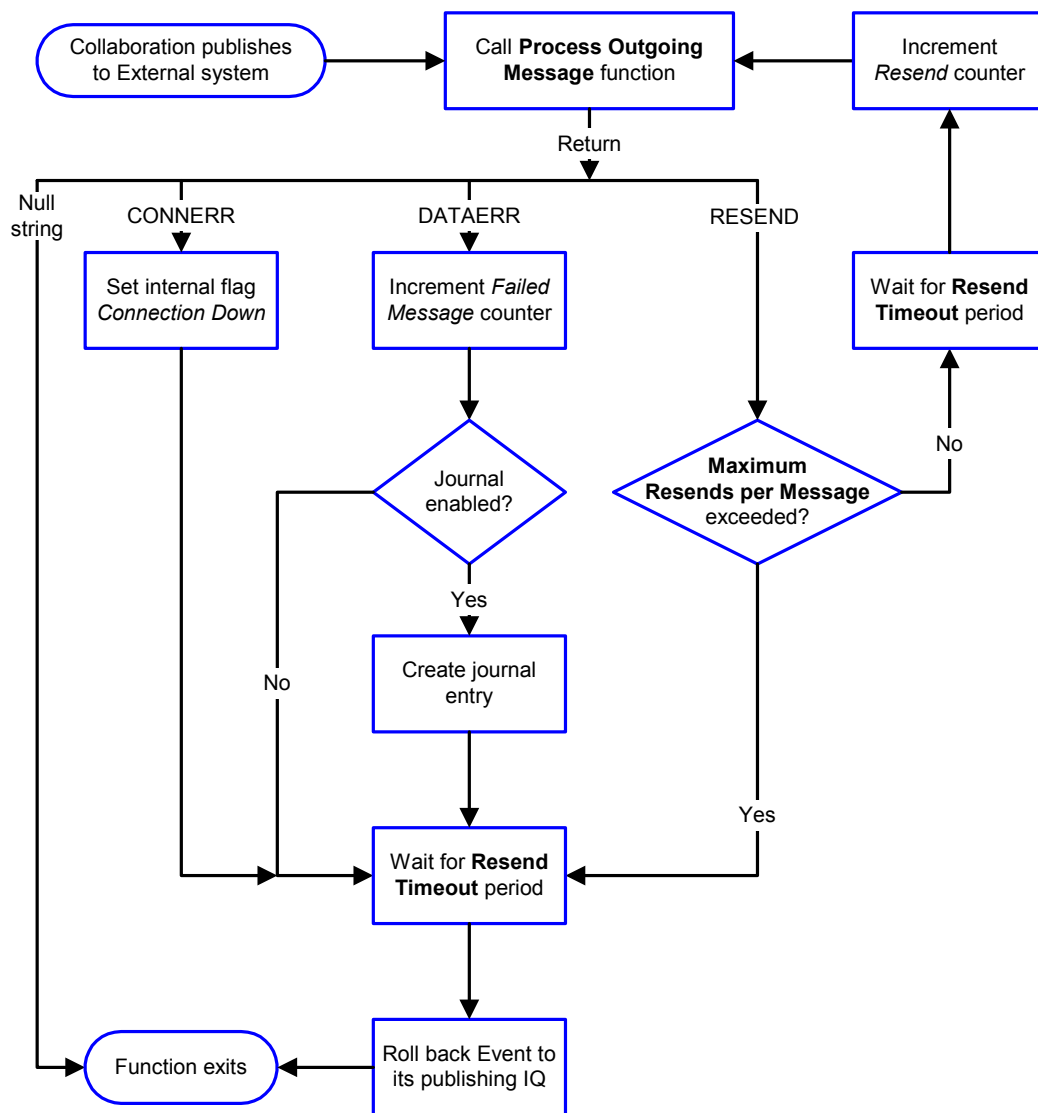
### Event-driven

Figure 103 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.

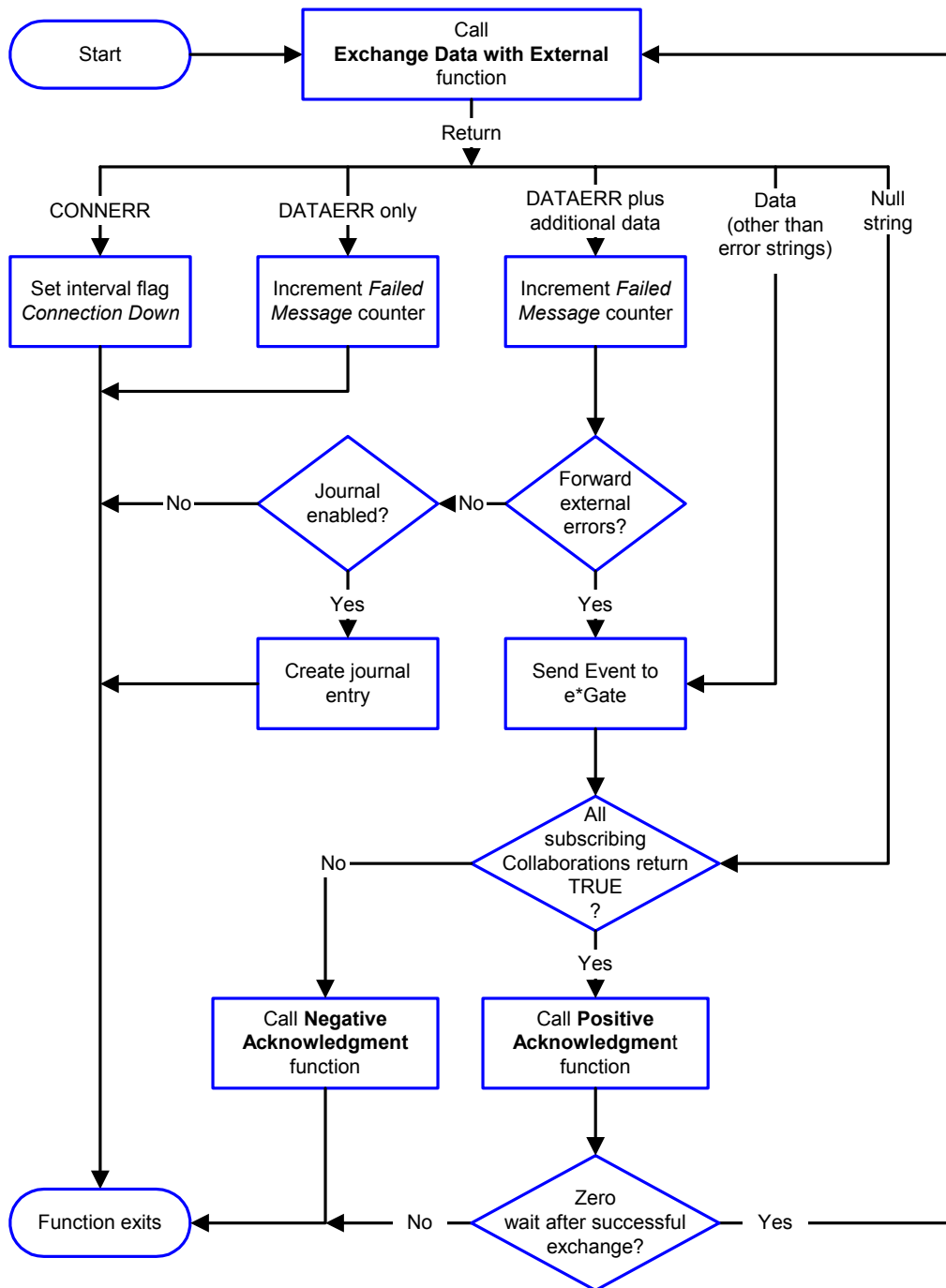
**Figure 103** Event-Driven Data Exchange Process



### Schedule-driven

Figure 104 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 104** 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

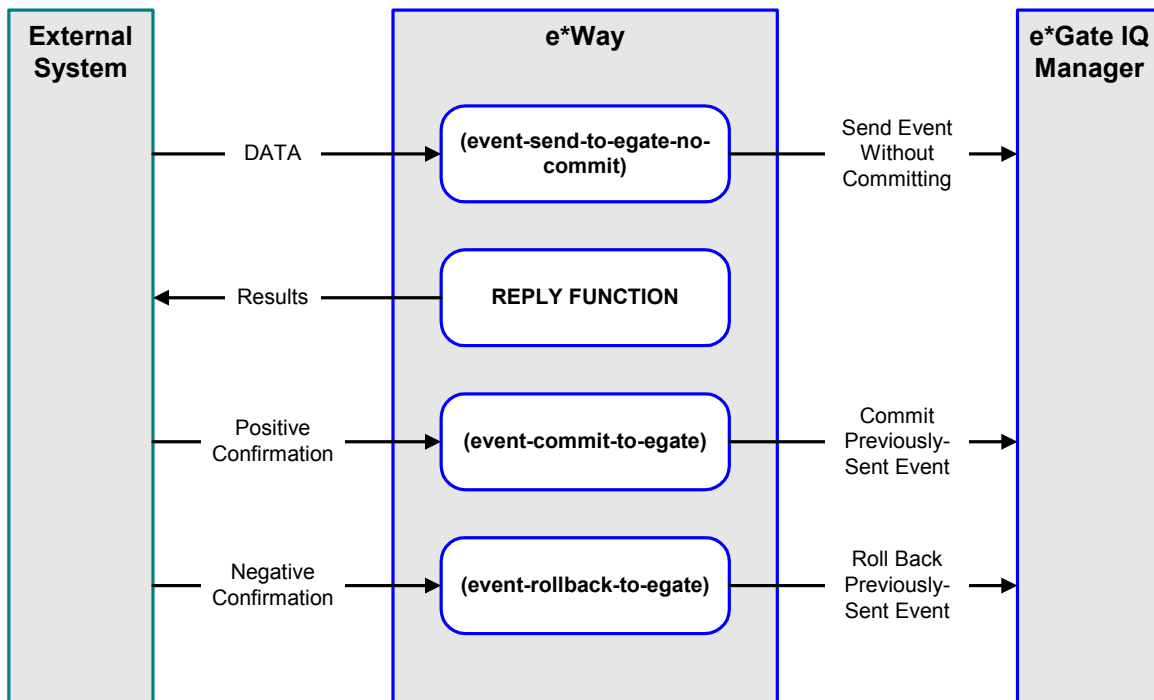
*Send Events to e\*Gate* can be implemented using any of the following Monk functions:

- **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 105):

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

**Figure 105** Send Event to e\*Gate with Confirmation

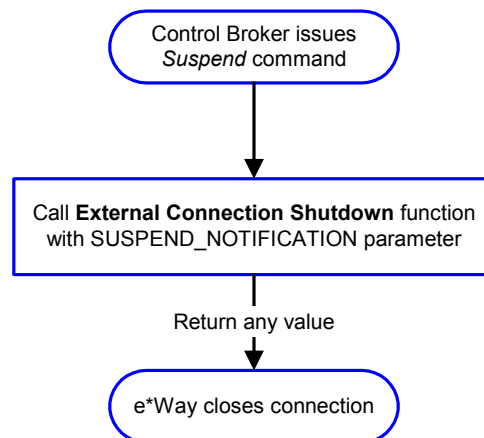


After the function exits, the e\*Way waits for the next *Start* time or command.

## Disconnect from External Process

Figure 106 illustrates how the e\*Way disconnects from the external system, using the **External Connection Shutdown Function**.

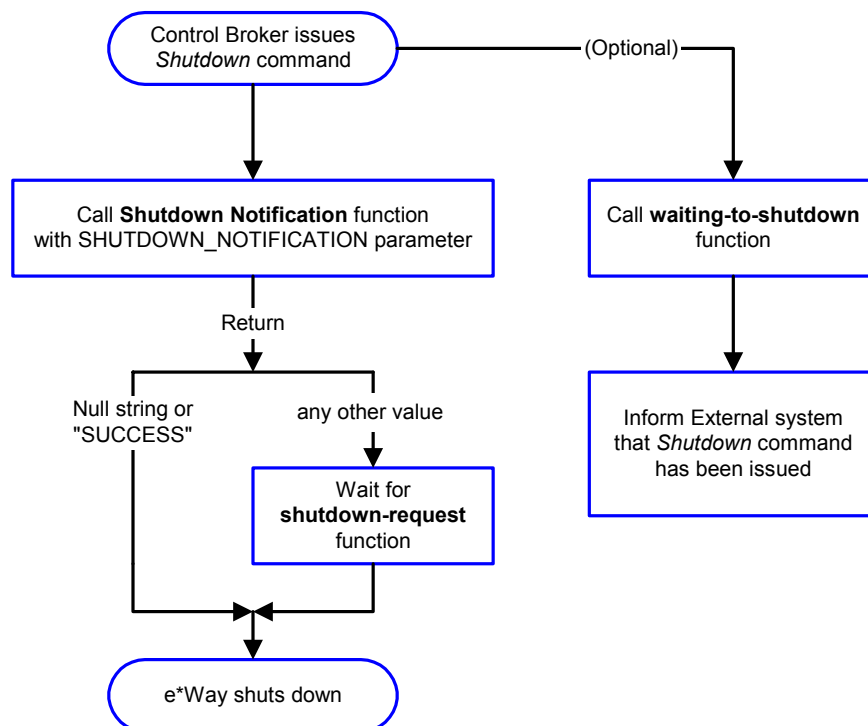
**Figure 106** Disconnect Process



## Shutdown Process

Figure 107 illustrates how the e\*Way shuts itself down, using the **Shutdown Command Notification Function**.

**Figure 107** Shutdown Process



# Configuration Parameters

This chapter describes the configuration parameters for the SeeBeyond Monk-enabled e\*Way Intelligent Adapter for SAP (BAPI).

---

## 7.1 Overview

The e\*Way's configuration parameters are set using the e\*Way Editor; see [Configuring the e\\*Way](#) on page 75 for procedural information. The default configuration is provided in `ewsapbapi.def`. The SAP BAPI e\*Way's configuration parameters are organized into the following sections:

[General Settings](#) on page 132

[Communication Setup](#) on page 134

[Monk Configuration](#) on page 137

[SAP RFC Client Setup](#) on page 146

[SAP RFC Server Setup](#) on page 153

[SAP tRFC RDBMS Setup](#) on page 159

---

## 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 Journalled 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 exit.

#### 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 that begin with the string "DATAERR" are queued to the e\*Way's configured queue. See [Exchange Data with External Function](#) on page 140 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 127 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.*

---

### 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 setting of this parameter is ignored and the e\*Way invokes the **Exchange Data with External Function** immediately
- If it is desired to invoke the **Exchange Data with External Function** again as soon as possible when data is **not** queued to e\*Gate via the return mechanism, the e\*Way Kernel Monk function **insert-exchange-data-event** can be called directly (prior to leaving the exchange function) to accomplish this
- If this parameter is set to zero, then no exchange data schedule is set and the **Exchange Data with External Function** never is called

#### See also

[Start Exchange Data Schedule](#) on page 135

[Stop Exchange Data Schedule](#) on page 136

---

### 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.

### Additional Information

- If this parameter is set to **Yes**, and the previous exchange function returned data, the e\*Way invokes the **Exchange Data with External Function** immediately
- If it is desired to invoke the **Exchange Data with External Function** again as soon as possible when data is **not** queued to e\*Gate via the return mechanism, the e\*Way Kernel Monk function **insert-exchange-data-event** can be called directly (prior to leaving the exchange function) to accomplish this
- 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**

---

## 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, regular, interval (such as weekly, daily, or every *n* seconds)

### Other Requirements

If you set a schedule using this parameter, you must also define *all* of the following parameters. If you do not, the e\*Way terminates execution when the schedule attempts to start.

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

### Additional Information

When the schedule starts, the e\*Way determines whether or not:

- it is waiting to send an ACK or NAK to the external system (using the **Positive Acknowledgment Function** or **Negative Acknowledgment Function**)
- 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, regular, interval (such as weekly, daily, or every  $n$  seconds)

---

## Down Timeout

### Description

Specifies the number of seconds for the e\*Way to wait 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 for the e\*Way to wait between calls to the **External Connection Verification Function** to verify that the connection is still up.

### 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 15.



---

## 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\ewsapbapi**.

*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 predefined options are **sapbapi-init** (the default value) and **saprfc-init**.

*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. The default option is the standard function template, **ewtosapbapi-startup**.

*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 **Start Exchange Data Schedule** 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.

The predefined options are:

- **sapbapitoew-polling**
- **sapbapitofrew-polling**
- **saptrfctoew-polling**
- **saptrfctofrew-polling**

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

### 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**. Queuing, 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. The predefined options are **ewtosapbapi-reconnect** (the default value), **ewtosaptrfc-reconnect**, and **saptrfctofrew-reconnect.dsc**.

*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. The predefined values are **sapbapi-verify-connect** (the default value) and **saptrfc-verify-connect**.

*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. The predefined options are **sapbapi-shutdown** (the default value) and **saptrfc-shutdown**.

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

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

### Required Input

A string, the inbound Event to e\*Gate. The predefined options are **sapbapi-ack** (the default value) **saptrfc-ack**.

### 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 predefined options are **sapbapi-nak** (the default value) and **saptrfc-nak**.

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

### 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. The predefined options are **ewtosapbapi-exit** (the default value), **ewtosaptrfc-exit**, **sapbapitoew-exit**, and **sapbapitofrew-exit**.



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

---

### Use Load Balancing

#### Description

From SAP R/3 release 3.0C onwards, workload balancing is available to automatically route requests to the Server (within a group of Application Servers) that currently has the best response time, as determined by an SAP Message Server. This parameter selects or deselects the load balancing feature.

#### Required Values

Yes or No. The default is No.

#### See also

[Host Name of the R/3 Target System](#) on page 146

[Target System ID for Load Balancing](#) on page 150

[Application Server Group for Load Balancing](#) on page 150

---

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

#### Description

Specifies the host name of the R/3 target system. You should *not* specify any Optional Router String here, as the value is appended to beginning of the Host Name of the R/3 Target System.

#### Required Values

A valid host name.

- If Load Balancing is used, this value represents the **SAP Message Server**
- If Load Balancing is *not* used, this value represents the **SAP Application Server**

#### See also

[Use Load Balancing](#) on page 146

[Optional Router String](#) on page 146

---

### Optional Router String

#### Description

This parameter is provided to allow access to an SAP system that is behind a firewall. The string is composed of the hostnames or IP addresses of all SAP Routers that are between this e\*Way and the SAP Gateway Host.

For example, if there are two routers, saprouter1, and saprouter2 (in order) from the e\*Way to the SAP Gateway Host, the Router String is determined as:

```
saprouter1:      204.79.199.5
saprouter2:      207.105.30.146
Router String:   /H/204.79.199.5/H/207.105.30.146/H/
```

**Note:** You must include the /H/ tokens to separate the routers, as well as at the beginning and ending of the string (as shown above).

### Required Values

A valid router string, as shown in the example above.

**Note:** This parameter is conditional and needs to be specified only to gain access to an SAP system that's behind a firewall.

See also

[Host Name of the R/3 Target System](#) on page 146

---

## System Number of the R/3 Target System

### Description

Specifies the system number of the R/3 target system, when SAP Load Balancing is not utilized.

### Required Values

A valid system number.

**Note:** This parameter is conditional, and should be configured only when **not** using SAP Load Balancing.

See also

[Use Load Balancing](#) on page 146

---

## Client

### Description

Specifies the number of the SAP Client.

### Required Values

An alphanumeric string.

**Note:** Do **not** omit any leading zeros.

---

## User

### Description

Specifies the name of the SAP Client user.

### Required Values

A valid user name.

*Note:* This parameter must be defined **before** specifying the Password.

---

## Password

### Description

Specifies the password corresponding to the specified user name.

### Required Values

A valid password.

*Note:* The User name must be specified **before** defining this parameter.

---

## Language

### Description

Specifies the language used in the SAP system.

### Required Values

DE (German), EN (English) or JA (Japanese). 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.

---

## Launch ABAP4 Debug Window

### Description

Enabling this option launches an SAP ABAP/4 Debugging window on the Participating Host for this e\*Way. The window relates to whatever RFC-enabled ABAP/4 Program is currently being called on the R/3 system.

## Required Values

Yes or No. The default is No.

**Note:** Note, this requires the SAP Front-end GUI software to be installed on the Participating Host for this e\*Way.

---

## Optional Gateway Host Name

### Description

Specifies an Optional Gateway Host Name for the R/3 Target System. You should **not** specify any Optional Router String here, as the value is appended to beginning of the Optional Gateway Host Name.

### Required Values

A string specifying the host name.

**Note:** This parameter is conditional, and should be configured only when **not** using SAP Load Balancing.

### See also

[Optional Router String](#) on page 146

---

## Optional Gateway Service

### Description

Specifies the Optional Gateway Service for the R/3 Target System. This is equivalent to a TCP/IP Port Number and can be referenced on the SAP R/3 System in the file:

- **Unix:**  
    /etc/services
- **Windows:**  
    C:\Winnt\system32\drivers\etc\Services

### Required Values

The SAP-recommended value is the string “sapgw” concatenated with the SAP System Number.

For example, with a System Number of “01”, the Gateway Service would be “sapgw01”

**Note:** This parameter is conditional, and should be configured only when **not** using SAP Load Balancing.

---

## Target System ID for Load Balancing

### Description

Specifies the ID of the Target SAP R/3 System. Typically, this is the Oracle back-end Database SID string, and can be verified using SAP transaction **S000** (main menu); menu path **System > Status > Database Data > Name** (text field).

### Required Values

A valid ID string.

*Note:* This parameter is conditional, and should be configured only when using SAP Load Balancing.

### See also

[Use Load Balancing](#) on page 146

[Host Name of the R/3 Target System](#) on page 146

[Application Server Group for Load Balancing](#) on page 150

---

## Application Server Group for Load Balancing

### Description

Specifies the name of the group of SAP Application Servers that share the workload.

### Required Values

A valid server group name string.

*Note:* This parameter is conditional, and should be configured only when using SAP Load Balancing.

### See also

[Use Load Balancing](#) on page 146

[Host Name of the R/3 Target System](#) on page 146

[Target System ID for Load Balancing](#) on page 150

---

## 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 (optionally including path information).

---

## Maximum Number of Failures Before Erroring Out

### Description

Specifies the maximum number of failures to record, after which the e\*Way shuts down.

### Required Values

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

---

## Enforce Transactional RFC

### Description

Specifies the use of Transactional RFC (tRFC) to communicate with the SAP R/3 system with unique Transaction IDs (TID).

### Required Values

Yes or No. The default is No.

### See also

[Transaction ID Verification Database](#) on page 151

[Enable RDBMS TID Management](#) on page 159

---

## Transaction ID Verification Database

### Description

Specifies the pathname to the Database file that records the disposition of all transactions outgoing from this e\*Way. The database records whether transactions are:

- C (Committed)
- U (Unprocessed or rolled back)
- R (Reserved or pending)

This parameter is ignored if tRFC is not enforced.

### Required Values

A valid pathname. If you provide a filename that is *not* an absolute path, then the value of SystemData from .egate.store is prefixed to the value of the parameter.

For example, if SystemData = \home\eGate, then:

a value such as data\SapTRFC.TIDdb

becomes \home\eGate\data\SapTRFC.TIDdb

### Additional Information

The Transaction ID management module, which includes the TID management Monk functions, is a customer-replaceable module. With the sample TID management module delivered with the SAP BAPI e\*Way, if Transactional RFC is enforced it is

required that each instance of the BAPI e\*Way needs to have its own Transaction ID Verification Database file. The result is undefined if two or more e\*Ways share a TID database file.

**See also**

[Enable RDBMS TID Management](#) on page 159



---

## 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 for the SAP R/3 system. You should *not* specify any Optional Router String here, as the value is appended to beginning of the Gateway Host Name.

#### Required Values

A valid host name.

#### See also

[Optional Router String](#) on page 153

---

### Optional Router String

#### Description

This parameter is provided to allow access to an SAP system that is behind a firewall. The string is composed of the hostnames or IP addresses of all SAP Routers that are between this e\*Way and the SAP Gateway Host.

For example, if there are two routers, saprouter1, and saprouter2 (in order) from the e\*Way to the SAP Gateway Host, the Router String is determined as:

```
saprouter1:      204.79.199.5
saprouter2:      207.105.30.146
Router String:   /H/204.79.199.5/H/207.105.30.146/H/
```

**Note:** You must include the /H/ tokens to separate the routers, as well as at the beginning and ending of the string (as shown above).

#### Required Values

A valid router string, as shown in the example above.

**Note:** This parameter is conditional and needs to be specified only to gain access to an SAP system that's behind a firewall.

#### See also

[Gateway Host Name](#) on page 153

---

## Gateway Service

### Description

Specifies the Gateway Service of the R/3 Target System that sends transactions. The e\*Way registers a Program ID with the Gateway, which listens for this e\*Way's registration. This is equivalent to a TCP/IP Port Number and can be referenced on the SAP R/3 System in the file:

- **Unix:**

`/etc/services`

- **Windows:**

`C:\Winnt\system32\drivers\etc\Services`

### Required Values

The SAP-recommended value is the string “**sapgw**” concatenated with the SAP System Number.

For example, with a System Number of “01”, the Gateway Service would be “**sapgw01**”

*Note:* This parameter is conditional, and should be configured only when **not** using SAP Load Balancing.

---

## Program ID

### Description

Specifies the Program ID for the SAP service.

### Required Values

A valid Program ID

*Note:* This value is case-sensitive, and must match **exactly** the value shown in transaction **SM59** (in the SAP Front-end GUI).

---

## Wait for Request Interval

### Description

Determines the time interval to wait for requests from the caller of a RFC function installed on this e\*Way. If a request is received within the specified interval, the Monk function installed as the handler for the called RFC function is dispatched automatically.

### Required Values

An integer between 1 and 65,535 and a unit of time. The default is 5 seconds.

---

## Wait for eGate Interval

### Description

Specify the amount of time to wait for a response to be returned by the e\*Gate system to this e\*Way.

### Required Values

An integer between 1 and 2,147,483,647 and a unit of time. The default is 5 milliseconds.

---

## Maximum Response Wait

### Description

Specify the maximum amount of time to wait for a response to be returned by the e\*Gate system to this e\*Way.

### Required Values

An integer between 1 and 2,147,483,647 and a unit of time. The default is 180,000 milliseconds.

---

## Trace

### Description

Activates or deactivates the SAP RFC Trace feature.

### 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 (optionally including path information).

---

## Maximum Number of Failures Before Erroring Out

### Description

Specifies the maximum number of failures to record, after which the e\*Way shuts down.

### Required Values

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

---

## Enforce Transactional RFC

### Description

Specifies the use of Transactional RFC (tRFC) to communicate with the SAP R/3 system with unique Transaction IDs (TID).

### Required Values

Yes or No. The default is No.

### See also

[Transaction ID Verification Database](#) on page 156

[Enable RDBMS TID Management](#) on page 159

---

## Transaction ID Verification Database

### Description

Specifies the pathname to the Database file that records the disposition of all transactions outgoing from this e\*Way. The database records whether transactions are:

- C (Committed)
- U (Unprocessed or rolled back)
- R (Reserved or pending)

This parameter is ignored if tRFC is not enforced.

### Required Values

A valid pathname. If you provide a filename that is *not* an absolute path, then the value of SystemData from .egate.store is prefixed to the value of the parameter.

For example, if SystemData = \home\eGate, then:

a value such as data\SapTRFC.TIDdb

becomes \home\eGate\data\SapTRFC.TIDdb

### Additional Information

The Transaction ID management module, which includes the TID management Monk functions, is a customer-replaceable module. With the sample TID management module delivered with the SAP BAPI e\*Way, if Transactional RFC is enforced it is required that each instance of the BAPI e\*Way needs to have its own Transaction ID Verification Database file. The result is undefined if two or more e\*Ways share a TID database file.

### See also

[Enable RDBMS TID Management](#) on page 159

---

## OnCheckTID Monk Function

### Description

This Monk function is invoked to check the Transaction ID at the beginning of a transaction. This function should either exist in the Monk Library or be explicitly loaded to the Monk environment at e\*Way startup time. It should expect a Monk string as the sole input parameter which represents the TID to be checked. This parameter is ignored if tRFC is not enforced.

The function should return a Monk string:

- “STORED” if the TID is valid and has been stored in the TID database
- “SKIP” if the transaction associated with the TID has been completed
- “FAILURE” if an error occurred

### Required Values

The default value is [saptrfc-on-check-tid](#).

---

## OnCommit Monk Function

### Description

This Monk function is invoked to commit a transaction to the local database. This function should either exist in the Monk Library or be explicitly loaded to the Monk environment at e\*Way startup time. It should expect a Monk string as the sole input parameter which represents the Transaction ID. This parameter is ignored if tRFC is not enforced.

The function should return a Monk string:

- “SUCCESS” if successfully committed to the local database
- “FAILURE” if an error occurred

### Required Values

The default value is [saptrfc-on-commit](#).

---

## OnRollback Monk Function

### Description

This Monk function is invoked to roll back the local database. This function should either exist in the Monk Library or be explicitly loaded to the Monk environment at e\*Way startup time. It should expect a Monk string as the sole input parameter which represents the Transaction ID. This parameter is ignored if tRFC is not enforced.

The function should return a Monk string:

- “SUCCESS” if rollback successful
- “FAILURE” if an error occurred

### Required Values

The default value is [saptrfc-on-rollback](#).

---

## OnConfirmTid Monk Function

### Description

This Monk function is invoked to confirm the Transaction ID at the end of a transaction. This function should either exist in the Monk Library or be explicitly loaded to the Monk environment at e\*Way startup time. It should expect a Monk string as the sole input parameter which represents the Transaction ID. This parameter is ignored if tRFC is not enforced.

The function should return a Monk string:

- "SUCCESS" if confirmation successful
- "FAILURE" if an error occurred

### Required Values

The default value is [saptrfc-on-confirm-tid](#).

## 7.7 SAP tRFC RDBMS Setup

### Enable RDBMS TID Management

#### Description

By default, a flat file is used to keep track of the transaction IDs. If you prefer to use a Relational DBMS to store the TID, and have purchased an appropriate Database (DART) e\*Way, you can select **Yes** to enable RDBMS TID Management. However, you first must run the TID database table creation script against your RDBMS before this feature can be used. The database table layouts are:

For tRFC Client mode:

| RDBMS      | Column Name | Column Type   |
|------------|-------------|---------------|
| Oracle     | eid         | VARCHAR2(132) |
|            | tid         | VARCHAR2(24)  |
| SQL Server | eid         | VARCHR(132)   |
|            | tid         | VARCHR(24)    |

For tRFC Server mode:

| RDBMS             | Column Name | Column Type  |
|-------------------|-------------|--------------|
| Oracle            | tid         | VARCHAR2(24) |
| SQL Server        | tid         | VARCHR(24)   |
| Oracle/SQL Server | status      | CHAR(1)      |

#### Required Values

**Yes** or **No**. The default is **No**.

#### Additional Information

Sample table creation scripts are located on the installation CD-ROM, under the `samples\ewsapbapi` directory:

- **Oracle:**  
    `create_tid.sql`
- **SQLServer:**  
    `create_ss_tid.sql`

#### See also

[Transaction ID Verification Database](#) on page 156

---

## Database Type

### Description

Specifies the type of relational DBMS to be used.

### Required Values

One of the following: **DB2**, **ODBC**, **ORACLE7**, **ORACLE8**, **ORACLE8i**, **SYBASE11**, **SYBASE12**. The default value is **ODBC**.

---

## Database Name

### Description

Specifies the name of the database.

### Required Values

A valid database name string.

---

## User Name

### Description

Specifies the user name for the database.

### Required Values

A valid user name string.

---

## Encrypted Password

### Description

Specifies the password associated with the specified user name.

### Required Values

A valid user password string.

---

## Client TID Table Name

### Description

Specifies the table name for the SAP tRFC Client TID database.

### Required Values

The default value is **TRFC\_CLIENT\_TID**.



---

## Server TID Table Name

### Description

Specifies the table name for the SAP tRFC Server TID database.

### Required Values

The default value is `TRFC_SERVER_TID`.

# API Functions

## 8.1 Overview

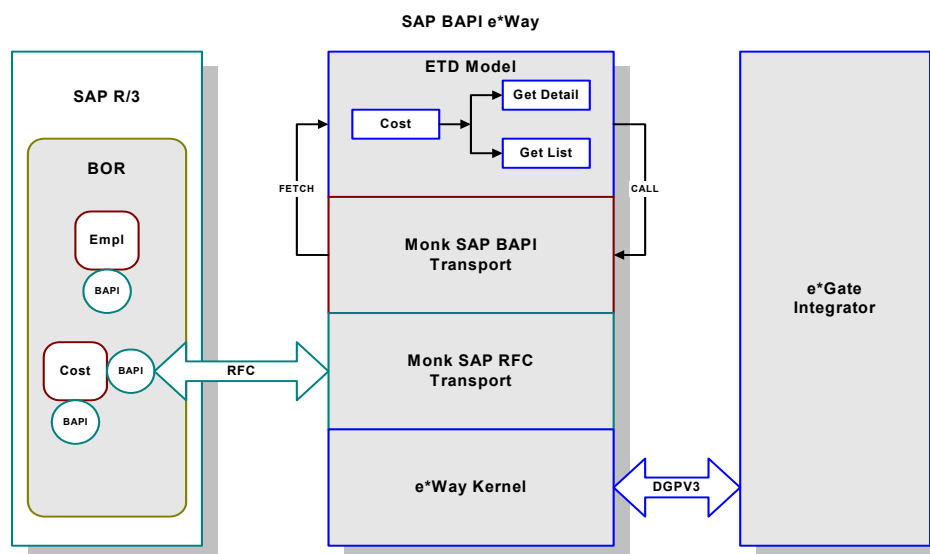
The SAP BAPI e\*Way has been designed specifically to connect e\*Gate to SAP enterprise-management software within a network of diverse hardware and software systems. Using one or more SAP e\*Ways, e\*Gate can act as a hub between SAP applications and other software systems, or between differently-configured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

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

- **BAPI 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. Most developers use the highest-level functions, which reside in the SAP BAPI Transport Layer.

**Figure 108** SAP BAPI e\*Way Architecture



## 8.1.1 Data Types and Function Templates

Data types and utility functions are referenced by one or more of the functions in this chapter. Monk standard function templates are intended to be copied and modified to control basic operations in your specific system.

[BAPI e\\*Way Data Types](#) on page 164

[BAPI e\\*Way Utility Functions](#) on page 166

[Standard Function Templates](#) on page 169

## 8.1.2 BAPI 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. BAPI Transport-layer Monk functions are described in:

[BAPI Transport Layer](#) on page 182

## 8.1.3 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 BAPI Transport Layer. Generally, Monk RFC functions should be called directly only when there is a specific need not addressed by the BAPI-Layer Monk functions.

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

[SAP RFC Client Functions](#) on page 192

[SAP RFC Server Functions](#) on page 201

[SAP tRFC Server Functions](#) on page 209

[SAP TID Database Management Functions](#) on page 218

[SAP Custom Structure Functions](#) on page 228

[SAP Parameter List Functions](#) on page 231

[SAP Table List Functions](#) on page 241

[BAPI e\\*Way Utility Functions](#) on page 166

## 8.1.4 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.

The e\*Way Kernel Monk functions are described in:

[Generic e\\*Way Functions](#) on page 247

---

## 8.2 BAPI e\*Way Data Types

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

[saprfc-conn-handle](#) on page 164

[saprfc-conn-opt](#) on page 164

[saprfc-status](#) on page 164

[saprfc-par-list](#) on page 165

[saprfc-tab-list](#) on page 165

---

### saprfc-conn-handle

#### Full Name

SAP Connection Handle.

#### Description

An 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

#### Full Name

SAP Client Connection Option Handle.

#### Description

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` described below.

---

### saprfc-status

#### Full Name

Status Code.

#### Description

Many of the Monk functions provided by the SAP BAPI 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-par-list

### Full Name

SAP Parameter List.

### Description

In an SAP Remote Function Call (RFC), input and output data are passed through named parameter lists and named table lists (see [saprfc-tab-list](#)). 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-tab-list

### Full Name

SAP Table List.

### Description

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

---

## 8.3 BAPI e\*Way Utility Functions

The BAPI e\*Way Utility functions provide basic procedures used by executable files and other Monk functions. The current set of Utility functions contains:

[sapbapi-get-laststatus](#) on page 166

[sapbapi-init](#) on page 166

[saprfc-getlasterror](#) on page 167

[saprfc-init](#) on page 167

---

### sapbapi-get-laststatus

#### Description

Returns the last status of a (sapbapi-...) Monk function.

#### Signature

(sapbapi-get-laststatus)

#### Parameters

None.

#### Returns

A string containing a status message.

#### Throws

None.

#### Location

sapbapi-get-laststatus.monk

---

### sapbapi-init

#### Description

Initializes the Generic e\*Way kernel and expands its functionality to that of an SAP BAPI e\*Way. Loads all requisite Monk functions and reads all the pertinent Configuration parameters, beyond those that are inherent to the Generic e\*Way.

#### Signature

(sapbapi-init)

#### Parameters

None.

#### Returns

The string "SUCCESS" upon success, Boolean false (#f) upon failure.

### Throws

None.

### Location

sapbapi-init.monk

### See also

[saprfc-init](#) on page 167

[saptrfc-init](#) on page 180

---

## saprfc-getlasterror

### Description

Returns a string describing details of the most recent error that occurred while running the e\*Way.

### Signature

(saprfc-getlasterror)

### Parameters

None.

### Returns

An error message string.

### Throws

None.

### Location

stc\_monksap.dll

---

## saprfc-init

### Description

Used by [sapbapi-init](#) to initialize the SeeBeyond SAP/RFC Monk Extension DLL and define SAP RFC API constants and debugging parameters.

### Signature

(saprfc-init)

### Parameters

None.

### Returns

None.

**Throws**

None.

**Location**

saprfc-init.monk



---

## 8.4 Standard Function Templates

The functions in this section provide templates for basic operations such as startup, reconnection, and exit (see note below). These functions are templates only, illustrating the structure, input, and return values required; their actual functionality is installation-dependent, and you must customize them for use in your specific system. They are contained within the file `sapbapi.monk`.

The functions are called by the default configuration parameters. See [Chapter 7](#) for more information.

**Note:** For RFC server-mode operation (i.e., SAP to e\*Way), **startup** and **reconnect** require specific BAPI/RFC Method substructures.

### 8.4.1 Standard RFC Operation

The current set of Standard RFC templates contains:

- [ewtosapbapi-startup](#) on page 169
- [ewtosapbapi-reconnect](#) on page 170
- [ewtosapbapi-exit](#) on page 170
- [sapbapitoew-polling](#) on page 171
- [sapbapitoew-exit](#) on page 171
- [sapbapitofrew-polling](#) on page 172
- [sapbapitofrew-exit](#) on page 172
- [sapbapi-ack](#) on page 172
- [sapbapi-nak](#) on page 173
- [sapbapi-nack](#) on page 173
- [sapbapi-return-empty-string](#) on page 174
- [sapbapi-shutdown](#) on page 174
- [sapbapi-verify-connect](#) on page 174

---

### ewtosapbapi-startup

#### Description

For a client-mode e\*Way, attempts to connect the BAPI e\*Way to the SAP Application server.

#### Signature

(ewtosapbapi-startup)

#### Parameters

None.

### Returns

The string “SUCCESS” upon success, “FAILURE” upon failure.

### Throws

None.

### See also

[sapbapi-client-connect](#) on page 182

[sapbapi-client-openex](#) on page 183

---

## ewtosapbapi-reconnect

### Description

For a client-mode e\*Way, attempts to reconnect the BAPI e\*Way to the SAP Application server.

### Signature

(ewtosapbapi-reconnect)

### Parameters

None.

### Returns

The string “UP” upon success, “DOWN” upon failure.

### Throws

None.

### See also

[sapbapi-client-connect](#) on page 182

[sapbapi-client-openex](#) on page 183

---

## ewtosapbapi-exit

### Description

For a client-mode e\*Way, attempts to unregister the Program ID with the SAP Application Server.

### Signature

(ewtosapbapi-exit)

### Parameters

None.

### Returns

Undefined.

### Throws

None.

### See also

[sapbapi-client-disconnect](#) on page 183

---

## sapbapitoew-polling

### Description

For a server-mode e\*Way, polls the SAP Application server to request and dispatch any installed Monk handler.

### Signature

(sapbapitoew-polling)

### Parameters

None.

### Returns

Boolean true (#t) upon success, the string "CONNERR" upon failure.

### Throws

None.

### See also

[sapbapi-server-pollrequest-dispatch](#) on page 184

---

## sapbapitoew-exit

### Description

For a server-mode e\*Way, attempts to unregister the Program ID with, and disconnect the e\*Way from, the SAP Application Server.

### Signature

(sapbapitoew-exit)

### Parameters

None.

### Returns

Undefined.

### Throws

None.

### See also

[sapbapi-server-unregister](#) on page 185

---

## sapbapitofrew-polling

### Description

For a bidirectional e\*Way, polls the SAP Application server to request and dispatch any installed Monk handler.

### Signature

(sapbapitofrew-polling)

### Parameters

None.

### Returns

An empty string "" upon success, "CONNERR" upon failure.

### Throws

None.

### See also

[sapbapi-server-pollrequest-dispatch](#) on page 184

---

## sapbapitofrew-exit

### Description

For a bidirectional e\*Way, attempts to unregister the Program ID with, and disconnect the e\*Way from, the SAP Application Server.

### Signature

(sapbapitofrew-exit)

### Parameters

None.

### Returns

Undefined.

### Throws

None.

### See also

[sapbapi-client-disconnect](#) on page 183

[sapbapi-server-unregister](#) on page 185

---

## sapbapi-ack

### Description

Positive acknowledgement template.

### Signature

(sapbapi-ack)

### Parameters

None.

### Returns

Undefined.

### Throws

None.

---

## sapbapi-nak

### Description

Negative acknowledgement template.

### Signature

(sapbapi-nak)

### Parameters

None.

### Returns

Undefined.

### Throws

None.

---

## sapbapi-nack

### Description

Alternate negative acknowledgement template.

### Signature

(sapbapi-nack)

### Parameters

None.

### Returns

Undefined.

### Throws

None.

---

## sapbapi-return-empty-string

### Description

Return empty string template.

### Signature

(sapbapi-return-empty-string)

### Parameters

None.

### Returns

Undefined.

### Throws

None.

---

## sapbapi-shutdown

### Description

Shutdown template.

### Signature

(sapbapi-shutdown)

### Parameters

None.

### Returns

Undefined.

### Throws

None.

---

## sapbapi-verify-connect

### Description

Connection verification template.

### Signature

(sapbapi-verify-connect)

### Parameters

None.

### Returns

The string "UP" upon success, "DOWN" upon failure.

## Throws

None.

## 8.4.2 Transactional RFC Operation

These functions extend the standard RFC templates described in the preceding section for tRFC operation by adding the necessary TID database procedures. They are contained within the file `saptrfc-stdver-eway-funcs.monk`.

The current set of Transactional RFC templates contains:

- [ewtosaptrfc-startup](#) on page 176
- [ewtosaptrfc-reconnect](#) on page 177
- [ewtosaptrfc-exit](#) on page 177
- [saptrfctoew-polling](#) on page 177
- [saptrfctoew-exit](#) on page 178
- [saptrfctofrew-polling](#) on page 178
- [saptrfctofrew-exit](#) on page 179
- [saptrfc-ack](#) on page 179
- [saptrfc-nak](#) on page 179
- [saptrfc-nack](#) on page 180
- [saptrfc-init](#) on page 180
- [saptrfc-return-empty-string](#) on page 180
- [saptrfc-shutdown](#) on page 181
- [saptrfc-verify-connect](#) on page 181

---

### ewtosaptrfc-startup

#### Description

For a tRFC client-mode e\*Way only, invokes [ewtosapbapi-startup](#) and attempts to start the TID database.

#### Signature

(ewtosaptrfc-startup)

#### Parameters

None.

#### Returns

None.

#### Throws

None.



---

## ewtosaptrfc-reconnect

### Description

For a tRFC client-mode e\*Way only, invokes [ewtosapbapi-reconnect](#) and attempts to reconnect the e\*Way to TID database.

### Signature

(ewtosaptrfc-reconnect)

### Parameters

None.

### Returns

None.

### Throws

None.

---

## ewtosaptrfc-exit

### Description

For a tRFC client-mode e\*Way only, invokes [ewtosapbapi-exit](#) and attempts to shut down the TID database.

### Signature

(ewtosaptrfc-exit)

### Parameters

None.

### Returns

None.

### Throws

None.

---

## saptrfctoew-polling

### Description

For a tRFC server-mode e\*Way only, invokes [sapbapitoew-polling](#) and attempts to shut down the TID database.

### Signature

(saptrfctoew-polling)

### Parameters

None.

### Returns

None.

### Throws

None.

---

## saptrfctoew-exit

### Description

For a tRFC server-mode e\*Way only, invokes [sapbapitoew-exit](#) and attempts to shut down the TID database using the same arguments as in [sapbapitofrew-exit](#).

### Signature

```
(saptrfctoew-exit)
```

### Parameters

None.

### Returns

None.

### Throws

None.

---

## saptrfctofrew-polling

### Description

For an e\*Way operating in tRFC mode, invokes [sapbapitofrew-polling](#) and attempts to shut down the TID database.

### Signature

```
(saptrfctofrew-polling)
```

### Parameters

None.

### Returns

None.

### Throws

None.

---

## saptrfctofrew-exit

### Description

For an e\*Way operating in tRFC mode, invokes [sapbapitofrew-exit](#) and attempts to shut down the TID database using the same arguments as in [sapbapitofrew-exit](#).

### Signature

```
(saptrfctpfrew-exit args)
```

### Parameters

None.

### Returns

None.

### Throws

None.

---

## saptrfc-ack

### Description

For an e\*Way operating in tRFC mode, invokes [sapbapi-ack](#) and sends a positive acknowledgement to the TID database using the same arguments as in [sapbapi-ack](#).

### Signature

```
(saptrfc-ack args)
```

### Parameters

None.

### Returns

None.

### Throws

None.

---

## saptrfc-nak

### Description

For an e\*Way operating in tRFC mode, invokes [sapbapi-nak](#) and sends a negative acknowledgement to the TID database using the same arguments as in [sapbapi-nak](#).

### Signature

```
(saptrfc-nak)
```

### Parameters

None.

### Returns

None.

### Throws

None.

---

## saptrfc-nack

### Description

For an e\*Way operating in tRFC mode, invokes [sapbapi-nack](#) and sends a negative acknowledgement to the TID database using the same arguments as in [sapbapi-nack](#).

### Signature

```
(saptrfc-nack)
```

### Parameters

None.

### Returns

None.

### Throws

None.

---

## saptrfc-init

### Description

For an e\*Way operating in tRFC mode, invokes [sapbapi-init](#) and attempts to initialize the TID database parameters.

### Signature

```
(saptrfc-init)
```

### Parameters

None.

### Returns

Error message if TID database parameters are not adequately defined.

### Throws

None.

---

## saptrfc-return-empty-string

### Description

Return empty string template.

### Signature

(saptrfc-return-empty-string)

### Parameters

None.

### Returns

None.

### Throws

None.

---

## saptrfc-shutdown

### Description

For an e\*Way operating in tRFC mode, invokes [sapbapi-shutdown](#) and attempts to shut down the TID database using the same arguments as in [sapbapi-shutdown](#).

### Signature

(saptrfc-shutdown)

### Parameters

None.

### Returns

None.

### Throws

None.

---

## saptrfc-verify-connect

### Description

For an e\*Way operating in tRFC mode, invokes [sapbapi-verify-connect](#) and attempts to verify the connection to the TID database.

### Signature

(saptrfc-verify-connect)

### Parameters

None.

### Returns

None.

### Throws

None.

---

## 8.5 BAPI Transport Layer

### 8.5.1 SAP BAPI Functions

The SAP BAPI functions make use of the SAP BAPI configuration parameters (see [Chapter 7](#)). They provide a high-level interface for the user to interact with the target SAP system specified in the configuration. The current set of SAP BAPI Transport functions contains the following:

- [sapbapi-client-connect](#) on page 182
- [sapbapi-client-disconnect](#) on page 183
- [sapbapi-client-openex](#) on page 183
- [sapbapi-server-pollrequest-dispatch](#) on page 184
- [sapbapi-server-register](#) on page 184
- [sapbapi-server-unregister](#) on page 185
- [sapbapi-struct-call](#) on page 185
- [sapbapi-struct-fetch](#) on page 186
- [sapbapi-struct-installfunction](#) on page 186
- [sapbapi-struct-raise](#) on page 187
- [sapbapi-struct-resetall](#) on page 188
- [sapbapi-struct-resetexception](#) on page 188
- [sapbapi-struct-resetexport](#) on page 189
- [sapbapi-struct-resetimport](#) on page 189
- [sapbapi-struct-resettables](#) on page 190
- [sapbapi-struct-send](#) on page 190

---

#### sapbapi-client-connect

##### Description

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

*Note:* This function is obsolete—use the function [sapbapi-client-openex](#) instead.

##### Signature

(sapbapi-client-connect)

##### Parameters

None.

### Returns

Boolean true (#t) if the function concludes successfully; and places the RFC Connection Handle into the global Monk variable `sapbapi_client_connhandle`; otherwise, false (#f).

### Throws

None.

### Location

`sapbapi-client-connect.monk`

## sapbapi-client-disconnect

### Description

Releases an RFC Client Connection to the target SAP Application Server. If successful, invalidates the RFC Connection Handle available through the global Monk variable `sapbapi_client_connhandle`.

### Signature

`(sapbapi-client-disconnect)`

### Parameters

None.

### Returns

Boolean true (#t) if the function concludes successfully; otherwise, false (#f).

### Throws

None.

### Location

`sapbapi-client-cdisonnnect.monk`

## sapbapi-client-openex

### Description

Establishes an RFC client connection to the target SAP Application Server using the specified e\*Way parameters. If successful, the RFC Connection Handle is available through the global Monk variable, `sapbapi_client_connhandle`.

### Signature

`(sapbapi-client-openex conn_par)`

### Parameters

| Name                  | Type   | Description  |
|-----------------------|--------|--|
| <code>conn_par</code> | string | Connection parameters, taken automatically from the e*Way's configuration. |

### Returns

Boolean true (**#t**) if the function concludes successfully; otherwise, false (**#f**).

### Throws

None.

### Location

sapbapi-client-openex.monk

---

## sapbapi-server-pollrequest-dispatch

### Description

Upon receiving a call, dispatches the installed Monk handler corresponding to the called ABAP/4 function.

### Signature

(sapbapi-server-pollrequest-dispatch)

### Parameters

None.

### Returns

Boolean true (**#t**) if the function concludes successfully; otherwise, false (**#f**).

### Throws

None.

### Location

sapbapi-server-pollrequest-dispstch.monk

---

## sapbapi-server-register

### Description

Registers the configured Program ID with the SAP Application server, with information (such as Gateway Host Name and Program ID) from the e\*Way configuration.

### Signature

(sapbapi-server-register)

### Parameters

None.

### Returns

Boolean true (**#t**) if the function concludes successfully; and places the RFC Connection Handle into the global Monk variable **sapbapi\_server\_connhandle**; otherwise, returns **#f** (false).



**Throws**

None.

**Location**

sapbapi-server-register.monk

---

## sapbapi-server-unregister

**Description**

Un-registers the configured Program ID with the SAP Application server.

**Signature**

(sapbapi-server-unregister)

**Parameters**

None.

**Returns**

Boolean true (#t) if the function concludes successfully; and makes the global Monk variable `sapbapi_server_connhandle` unavailable; otherwise, false (#f).

**Throws**

None.

**Location**

sapbapi-server-unregister.monk

---

## sapbapi-struct-call

**Description**

Calls an RFC-enabled ABAP Function on the SAP Application server, using information specified by the input BAPI/RFC Method substructure, which then fetches the Event from the BAPI e\*Way (operating in Client mode).

**Signature**

(sapbapi-struct-call *method\_substruct\_path*)

**Parameters**

| Name                  | Type   | Description  |
|-----------------------|--------|--|
| method_substruct_path | string | Specifies the path to a method substructure on the BAPI e*Way containing the parameters to be fetched. |

**Returns**

Boolean true (#t) if the function concludes successfully; otherwise, false (#f).

**Throws**

None.

**Location**

sapbapi-struct-call.monk

## sapbapi-struct-fetch

**Description**

Used within a Monk handler to call an RFC-enabled ABAP Function installed on the caller that fetches the **EXPORT\_TO\_CALLEE** and/or **TABLE** parameters of the specified BAPI/RFC Method substructure from the caller.

**Signature**

(sapbapi-struct-fetch *method\_substruct\_path*)

**Parameters**

| Name                  | Type   | Description  |
|-----------------------|--------|--|
| method_substruct_path | string | Specifies the path to a method substructure on the caller containing the parameters to be fetched. |

**Returns**

Boolean true (#t) if the function concludes successfully; otherwise, false (#f).

**Throws**

None.

**Location**

sapbapi-struct-fetch.monk

**See also**

[saprfc-par-add-receiving](#) on page 234

[saprfc-tab-create](#) on page 244

## sapbapi-struct-installfunction

**Description**

Installs a portal on the SAP Application server to an RFC-enabled ABAP Function using information specified by the input BAPI/RFC Method substructure, which is implemented via the given Monk function.

**Signature**

(sapbapi-struct-installfunction *method\_substruct\_path*  
*monk\_handler\_name*)

## Parameters

| Name                  | Type   | Description   |
|-----------------------|--------|---|
| method_substruct_path | string | Specifies the path to a method substructure on the caller containing the parameters to be exported during a fetch or nodes to be filled during a send or exception. |
| monk_handler_name     | string | The name of a Monk function.  |

## Returns

Boolean true (#t) if the function concludes successfully; otherwise, false (#f).

## Throws

None.

## Location

sapbapi-struct-installfunction.monk

## sapbapi-struct-raise

### Description

Raises an exception with the caller of an RFC-enabled ABAP function supporting this e\*Way. The exception string, along with any table information from the input BAPI/RFC Method substructure, is forwarded to the **caller** and into the EXCEPTION node of the appropriate installed BAPI/RFC Method substructure.

### Signature

(sapbapi-struct-raise method\_substruct\_path)

### Parameters

| Name                  | Type   | Description   |
|-----------------------|--------|---|
| method_substruct_path | string | Specifies the path to a method substructure on the caller containing the EXCEPTION node to which information is to be exported. |

## Returns

Boolean true (#t) if the function concludes successfully; otherwise, false (#f).

## Throws

None.

## Location

sapbapi-struct-raise.monk

## sapbapi-struct-resetall

### Description

By deleting stored data-nodes, resets the following from the input BAPI/RFC Method substructure: EXPORT\_TO\_CALLEE, IMPORT\_BY\_CALLER, TABLE, and/or EXCEPTION parameters.

### Signature

(sapbapi-struct-resetall *method\_substruct\_path*)

### Parameters

| Name                  | Type   | Description   |
|-----------------------|--------|---|
| method_substruct_path | string | Specifies the path to a method substructure, on either the caller or callee, containing the parameters to be reset. |

### Returns

Boolean true (#t) if the function concludes successfully; otherwise, false (#f).

### Throws

None.

### Location

sapbapi-struct-resetall.monk

## sapbapi-struct-resetexception

### Description

By deleting stored data-nodes, resets EXCEPTION from the input BAPI/RFC Method substructure.

### Signature

(sapbapi-struct-resetexception *method\_substruct\_path*)

### Parameters

| Name                  | Type   | Description  |
|-----------------------|--------|--|
| method_substruct_path | string | Specifies the path to a method substructure on the callee containing the EXCEPTION parameters to be reset. |

### Returns

Boolean true (#t) if the function concludes successfully; otherwise, false (#f).

### Throws

None.

Location

sapbapi-struct-resetexception.monk

## sapbapi-struct-resetexport

Description

By deleting stored data-nodes, resets EXPORT\_TO\_CALLEE parameters within the input BAPI/RFC Method substructure.

Signature

(sapbapi-struct-resetexport *method\_substruct\_path*)

Parameters

| Name                  | Type   | Description   |
|-----------------------|--------|---|
| method_substruct_path | string | Specifies the path to a method substructure on the caller containing the EXPORT_TO_CALLEE parameters to be reset. |

Returns

Boolean true (#t) if the function concludes successfully; otherwise, false (#f).

Throws

None.

Location

sapbapi-struct-resetexport.monk

## sapbapi-struct-resetimport

Description

By deleting stored data-nodes, resets IMPORT\_BY\_CALLER parameters within the input BAPI/RFC Method substructure.

Signature

(sapbapi-struct-resetimport *method\_substruct\_path*)

Parameters

| Name                  | Type   | Description   |
|-----------------------|--------|---|
| method_substruct_path | string | Specifies the path to a method substructure on the callee containing the IMPORT_BY_CALLER parameters to be reset. |

Returns

Boolean true (#t) if the function concludes successfully; otherwise, false (#f).

**Throws**

None.

**Location**

sapbapi-struct-resetimport.monk

## sapbapi-struct-resettables

**Description**

By deleting stored data-nodes, resets **TABLE**s from the input BAPI/RFC Method substructure.

**Signature**

(sapbapi-struct-resettables *method\_substruct\_path*)

**Parameters**

| Name                  | Type   | Description  |
|-----------------------|--------|--|
| method_substruct_path | string | Specifies the path to a method substructure on the caller containing the <b>TABLE</b> s to be reset. |

**Returns**

Boolean true (**#t**) if the function concludes successfully; otherwise, false (**#f**).

**Throws**

None.

**Location**

sapbapi-struct-resettables.monk

## sapbapi-struct-send

**Description**

Used within a Monk handler to send the **IMPORT\_BY\_CALLER** parameters and/or **TABLE** information, specified by the input BAPI/RFC Method substructure, back to the caller of a RFC-enabled ABAP Function as data being imported.

**Signature**

(sapbapi-struct-send *method\_substruct\_path*)

### Parameters

| Name                  | Type   | Description   |
|-----------------------|--------|---|
| method_substruct_path | string | Specifies the path to a method substructure on the callee from which the parameters are to be exported. |

### Returns

Boolean **#t** (**true**) if the function concludes successfully; otherwise, returns **#f** (**false**).

### Throws

None.

### Location

sapbapi-struct-send.monk

### See also

[saprfc-par-add](#) on page 231

## 8.6 RFC Transport Layer

### 8.6.1 SAP RFC Client Functions

The SAP RFC Client functions manipulate connections to the SAP client. All functions are located in `stc_monksap.dll`. The current set of SAP RFC Client functions contains the following:

- [saprfc-client-callreceive](#) on page 192
- [saprfc-client-connect](#) on page 193
- [saprfc-client-createtid](#) on page 193
- [saprfc-client-disconnect](#) on page 194
- [saprfc-client-indirectcall](#) on page 194
- [saprfc-client-openex](#) on page 195
- [saprfc-conn-abort](#) on page 195
- [saprfc-conn-createopt](#) on page 196
- [saprfc-conn-handle?](#) on page 196
- [saprfc-conn-opt?](#) on page 197
- [saprfc-conn-set-clientconnmode](#) on page 197
- [saprfc-conn-set-clientconnopt-cpic](#) on page 198
- [saprfc-conn-set-clientconnopt-r3only](#) on page 199
- [saprfc-conn-set-clientconnpar](#) on page 199
- [saprfc-conn-settrace](#) on page 200

---

### saprfc-client-callreceive

#### Description

Calls a remote function through RFC.

#### Signature

`(saprfc-client-callreceive conn_handle tab-list par-list par-list)`

#### Parameters

| Name        | Type   | Description                               |
|-------------|--------|---|
| conn_handle | handle | A valid SAP RFC Connection Option handle. |
| tab-list    | list   | A valid SAP table list.                   |
| par-list    | list   | A valid SAP import parameter list.        |
| par-list    | list   | A valid SAP export parameter 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.

## 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?](#).

*Note:* This function is obsolete—use the function [saprfc-client-openex](#) instead.

### Signature

(saprfc-client-connect *connection-option*)

### Parameters

| Name              | Type   | Description                       |
|-------------------|--------|-----------------------------------|
| connection-option | string | A valid SAP RFC Connection Option |

### Returns

An SAP RFC Connection Option handle.

### Throws

None.

## saprfc-client-createtid

### Description

Calls the target SAP system to create a valid SAP Transaction ID, which 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 | handle | A valid SAP RFC Connection Option handle. |

### Returns

This function returns a Monk string which represents 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.

## saprfc-client-disconnect

### Description

Closes the connection to the SAP Application Server.

### Signature

```
(saprfc-client-disconnect conn_handle)
```

### Parameters

| Name        | Type   | Description                              |
|-------------|--------|--|
| conn-handle | handle | 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.

## saprfc-client-indirectcall

### Description

Calls a remote function through Transactional RFC.

## Signature

(saprfc-client-indirectcall conn\_handle tab-list export-par-list\_tid  
function-name)

## Parameters

| Name            | Type   | Description                               |
|-----------------|--------|---|
| conn-handle     | handle | A valid SAP RFC Connection Option handle. |
| tab-list        | list   | A valid SAP table list.                   |
| export-par-list | list   | A valid SAP export parameter list.        |
| tid             | string | A valid SAP transaction ID.               |
| function-name   | string | The remote function to run.               |

## Returns

A status code **SAPRFC\_OK** if the operation succeeds, otherwise on failure. The reason for failure can be accessed via [saprfc-getlasterror](#).

## Throws

None.

## saprfc-client-openex

### Description

Establishes an RFC client connection with the SAP Application Server as specified in the SAP Client Connection Option. All the connection parameters are automatically taken from the e\*Way's configuration. If successful, the RFC Connection Handle is available through the global Monk variable **sapbapi\_client\_connhandle** and a Boolean **#t** is returned.

### Signature

(saprfc-client-openex)

### Parameters

None

### Returns

Boolean true (**#t**) if connection opened successfully, otherwise false (**#f**).

### Throws

None.

## saprfc-conn-abort

### Description

Aborts the SAP Client Connection.

## Signature

`(saprfc-conn-abort conn_handle reason-string)`

## Parameters

| Name          | Type   | Description                               |
|---------------|--------|---|
| conn-handle   | handle | A valid SAP RFC Connection Option handle. |
| reason-string | string | The reason for aborting the connection.   |

## 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.

## 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.

## saprfc-conn-handle?

### Description

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

### Signature

(saprfc-conn-handle? *conn\_handle*)

### Parameters

| Name        | Type   | Description         |
|-------------|--------|---------------------|
| conn-handle | handle | The object to test. |

### Returns

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

### Throws

None.

---

## saprfc-conn-opt?

### Description

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

### Signature

(saprfc-conn-opt? *connection-option*)

### Parameters

| Name              | Type   | Description         |
|-------------------|--------|---------------------|
| connection-option | string | The object to test. |

### Returns

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

### Throws

None.

---

## saprfc-conn-set-clientconnmode

### Description

Sets the client connection mode.

### Signature

(saprfc-conn-set-clientconnmode *connection-option number*)

## Parameters

| Name              | Type    | Description                                 |
|-------------------|---------|---|
| connection-option | string  | The SAP RFC Connection Option to set.       |
| number            | integer | 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.

## saprfc-conn-set-clientconnopt-cpic

### Description

Sets CPIC-specific parameters for a CPIC client connection.

### Signature

*(saprfc-conn-set-clientconnopt-cpic connection-option string string)*

### Parameters

| Name              | Type   | Description                           |
|-------------------|--------|---------------------------------------|
| connection-option | string | The SAP RFC Connection Option to set. |
| string            | string | A gateway-host name.                  |
| string            | 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.

## saprfc-conn-set-clientconnopt-r3only

### Description

Sets CPIC-specific parameters for a CPIC client connection.

### Signature

*(saprfc-conn-set-clientconnopt-r3only connection-option string string)*

### Parameters

| Name              | Type   | Description                           |
|-------------------|--------|---------------------------------------|
| connection-option | string | The SAP RFC Connection Option to set. |
| string            | string | A gateway-host name.                  |
| string            | 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.

## saprfc-conn-set-clientconnpar

### Description

Sets general client-connection parameters.

### Signature

*(saprfc-conn-set-clientconnpar connection-option string string string string)*

### Parameters

| Name              | Type   | Description                           |
|-------------------|--------|---------------------------------------|
| connection-option | string | The SAP RFC Connection Option to set. |

| Name       | Type   | Description  |
|------------|--------|--|
| string (1) | string | The client string.                                     |
| string (2) | string | A valid user name.                                     |
| string (3) | string | The password corresponding to the specified user name. |
| string (4) | 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.                                  |

---

## saprfc-conn-settrace

### Description

Activates or deactivates tracing for the specified connection option.

### Signature

*(saprfc-conn-settrace connection-option number)*

### Parameters

| Name              | Type    | Description  |
|-------------------|---------|--|
| connection-option | string  | The SAP RFC Connection Option for which to activate or deactivate tracing. |
| number            | integer | 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.



## 8.6.2 SAP RFC Server Functions

The SAP RFC Server functions control the SAP Server. All functions are located in `stc_monksap.dll`. The current set of SAP RFC Server functions contains the following:

[saprfc-server-getcallbackfailuretype](#) on page 202

[saprfc-server-getinputdata](#) on page 202

[saprfc-server-installfunction](#) on page 203

[saprfc-server-installtransctrl](#) on page 204

[saprfc-server-resetcallbackfailure](#) on page 205

[saprfc-server-sendoutputdata](#) on page 206

[saprfc-server-shutdown](#) on page 206

[saprfc-server-startup](#) on page 207

[saprfc-server-waitanddispatch](#) on page 208

---

### saprfc-server-getcallbackfailuretid

#### Description

If the **Commit**, **Rollback**, or **Confirm** Monk function installed by [saprfc-server-installtransctrl](#) returns “FAILURE”, this function can be called to obtain the ID of the failed transaction.

#### Signature

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

#### Parameters

None.

#### Returns

The transaction ID on which the callback routine has failed.

#### Throws

None.

#### Additional Information

If the **Commit**, **Rollback**, or **Confirm** Monk function installed by [saprfc-server-installtransctrl](#) returns “FAILURE”, the return status is subsequently passed to [saprfc-server-waitanddispatch](#), which returns `SAPRFC_FAILURE`.

## saprfc-server-getcallbackfailuretype

### Description

If the **Commit**, **Rollback**, or **Confirm** Monk function installed by [saprfc-server-installtransctrl](#) returns “FAILURE”, this function can be called to determine which callback routine has failed.

### Signature

(saprfc-server-getcallbackfailuretype)

### Parameters

None.

### Returns

The type of callback failure:

|                                  |   |
|----------------------------------|---|
| 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.

### Additional Information

If the **Commit**, **Rollback**, or **Confirm** Monk function installed by [saprfc-server-installtransctrl](#) returns “FAILURE”, the return status is subsequently passed to [saprfc-server-waitanddispatch](#), which returns `SAPRFC_FAILURE`.

## saprfc-server-getinputdata

### Description

Used inside a RFC service module (Monk function) to import the parameters list and tables list from the caller.

### Signature

(saprfc-server-getinputdata *par-list tab-list*)

### Parameters

| Name     | Type | Description                 |
|----------|------|-----------------------------|
| par-list | list | A valid SAP Parameter List. |
| tab-list | 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.

## saprfc-server-installfunction

### Description

Installs a set of callback routines (Monk functions) as an RFC service module.

### Signature

`(saprfc-server-installfunction string string)`

### Parameters

| Name       | Type   | Description   |
|------------|--------|---|
| string (1) | string | The name of a function known to SAP (and thus available to any RFC caller).   |
| string (2) | string | 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.

## saprfc-server-installtransctrl

### Description

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

### Signature

```
(saprfc-server-installtransctrl monk-function-check_monk-function-
commit_monk-function-rollback_monk-function-confirm)
```

### Parameters

| Name                   | Type   | Description   |
|------------------------|--------|---|
| monk-function-check    | 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. |
| monk-function-commit   | 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.  |
| monk-function-rollback | 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  |
|-----------------------|--------|--|
| monk-function-confirm | 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

A status code `SAPRFC_OK` if the operation succeeds, otherwise on failure. The reason for failure can be accessed via [saprfc-getlasterror](#).

### Throws

None.

### Additional Information

If the **Commit**, **Rollback**, or **Confirm** Monk function installed by [saprfc-server-installtransctrl](#) returns "FAILURE", the return status is subsequently passed to [saprfc-server-waitanddispatch](#), which returns `SAPRFC_FAILURE`. The function [saprfc-server-getcallbackfailureid](#) can then be called to obtain the ID of the failed transaction, or the function [saprfc-server-getcallbackfailuretype](#) can be called to determine which callback routine failed.

## saprfc-server-resetcallbackfailure

### Description

Resets the error flag set by one of the tRFC callback routines. This function must be called before [saprfc-server-waitanddispatch](#) can be used again.

### Signature

(saprfc-server-resetcallbackfailure)

### Parameters

None.

### Returns

Undefined.

### Throws

None.

### Additional Information

If the **Commit**, **Rollback**, or **Confirm** Monk function installed by [saprfc-server-installtransctrl](#) returns "FAILURE", the return status is subsequently passed to [saprfc-](#)

`server-waitanddispatch`, which returns `SAPRFC_FAILURE`. The function `saprfc-server-getcallbackfailureid` can then be called to obtain the ID of the failed transaction, or the function `saprfc-server-getcallbackfailuretype` can be called to determine which callback routine failed.

## saprfc-server-sendoutputdata

### Description

Used inside a RFC service module (Monk function); exports the **Parameters** list and **Tables** list to the caller.

### Signature

```
(saprfc-server-sendoutputdata par-list tab-list)
```

### Parameters

| Name     | Type | Description                 |
|----------|------|-----------------------------|
| par-list | list | A valid SAP Parameter List. |
| tab-list | 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.

## 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.

## saprfc-server-startup

### Description

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

### Signature

*(saprfc-server-startup string string string number)*

### Parameters

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

### Returns

An SAP RFC Connection Option handle.

### Throws

None.

## 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 RFC service module (a Monk function) installed with [saprfc-server-installfunction](#).

### Signature

```
(saprfc-server-waitanddispatch conn_handle number)
```

### Parameters

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

### Additional Information

If the [saptrfc-on-commit-tid](#), [saptrfc-on-confirm-tid](#), or [saptrfc-on-rollback-tid](#) Monk function installed by [saprfc-server-installtransctrl](#) returns "FAILURE", the return status is subsequently passed to [saprfc-server-waitanddispatch](#), which returns SAPRFC\_FAILURE.



### 8.6.3 SAP tRFC Server Functions

The SAP tRFC Server functions control the SAP Server when tRFC is enforced; otherwise, they are ignored. The current set of SAP tRFC Server functions contains the following:

- [saptrfc-commit-tid](#) on page 209
- [saptrfc-delete-tid](#) on page 210
- [saptrfc-get-current-event-id](#) on page 210
- [saptrfc-get-tid](#) on page 211
- [saptrfc-on-check-tid](#) on page 211
- [saptrfc-on-commit](#) on page 212
- [saptrfc-on-commit-tid](#) on page 213
- [saptrfc-on-confirm-tid](#) on page 213
- [saptrfc-on-rollback](#) on page 214
- [saptrfc-on-rollback-tid](#) on page 215
- [saptrfc-receive-idoc4](#) on page 215
- [saptrfc-send-idoc4](#) on page 216
- [saptrfc-struct-call](#) on page 216

#### saptrfc-commit-tid

##### Description

This Monk function marks the TID, in the file specified by the given EID, to be invalid (i.e., processed).

##### Signature

(saptrfc-commit-tid *eid*)

##### Parameters

| Name | Type   | Description   |
|------|--------|---|
| eid  | string | A Monk string that represents the current Event ID. |

##### Returns

Boolean true (#t) upon success, (#f) upon failure.

##### Throws

None.

##### Location

saptrfc-commit-tid.monk

*Note: This function is provided as a sample only.*

---

## saptrfc-delete-tid

### Description

Rolls back the transaction identified by the specified EID and also deletes the TID from the TID database.

### Signature

```
(saptrfc-delete-tid eid)
```

### Parameters

| Name | Type   | Description   |
|------|--------|---|
| eid  | string | A Monk string that represents the current Event ID. |

### Returns

Boolean true (#t) upon success, (#f) upon failure.

### Throws

None.

### Location

saptrfc-delete-tid.monk

### See also

[saptrfc-tid-db-delete](#) on page 219

*Note: This function is provided as a sample only.*

---

## saptrfc-get-current-event-id

### Description

Searches for an EID, associated with an Event coming from e\*Gate, in the e\*Gate TID database to determine whether or not the EID has ever been processed by the e\*Way.

### Signature

```
(saptrfc-get-current-event-id)
```

### Parameters

None

### Returns

An EID string.

### Throws

None.

### Location

saptrfc-get-current-event-id.monk

*Note:* This function is provided as a sample only.

---

## saptrfc-get-tid

### Description

Calls [saprfc-client-createtid](#) to get a TID created by the R/3 system. If successful, it saves the TID in a file designated by the specified EID and marks the TID as **reserved**.

### Signature

```
(saptrfc-get-tid conn_handle eid)
```

### Parameters

| Name        | Type   | Description   |
|-------------|--------|---|
| conn_handle | handle | .A valid SAP RFC Connection Option handle.          |
| eid         | string | A Monk string that represents the current Event ID. |

### Returns

Upon success, a TID string; upon failure, an empty string.

### Throws

None.

### Additional Information

If `RfcIndirectCall` fails, calling this function again does not call [saprfc-client-createtid](#), but instead returns an unused TID.

### Location

saptrfc-get-tid.monk

### See also

[saptrfc-tid-db-reserve](#) on page 222

[saptrfc-tid-file-reserve](#) on page 226

*Note:* This function is provided as a sample only.

---

## saptrfc-on-check-tid

### Description

Checks to see if the specified TID is already in the TID database. If not, stores the TID in the database.

### Signature

(saptrfc-on-check-tid *tid\_string*)

### Parameters

| Name       | Type   | Description   |
|------------|--------|---|
| tid_string | string | A Monk string that represents the current transaction ID. |

### Returns

One of the following strings:

- “SKIP” if the specified TID is already in the TID database
- “STORED” upon successfully storing the TID in the TID database
- “FAILURE” upon failure to check or store the TID in the TID database

### Throws

None.

### Location

saptrfc-on-check-tid.monk

### See also

[saptrfc-tid-db-on-check](#) on page 220

[saptrfc-tid-file-on-check](#) on page 224

## saptrfc-on-commit

### Description

When received from the SAP R/3 system, the e\*Way calls [event-commit-to-egate](#), which commits the Event to an e\*Gate IQ, and marks the TID as **processed**.

### Signature

(saptrfc-on-commit *tid\_string*)

### Parameters

| Name       | Type   | Description   |
|------------|--------|---|
| tid_string | string | A Monk string that represents the current transaction ID. |

### Returns

The string “SUCCESS” upon success, “FAILURE” upon failure.

### Throws

None.

### Location

saptrfc-on- commit.monk

### See also

[saptrfc-tid-db-on-commit](#) on page 220

[saptrfc-tid-file-on-commit](#) on page 225

[saptrfc-on-commit-tid](#) on page 213

*Note:* This function is provided as a sample only.

---

## saptrfc-on-commit-tid

### Description

When received from the SAP R/3 system, the e\*Way calls [event-commit-to-egate](#), which commits the Event to an e\*Gate IQ, and marks the TID as **processed**.

### Signature

```
(saptrfc-on-commit-tid tid_string)
```

### Parameters

| Name       | Type   | Description   |
|------------|--------|---|
| tid_string | string | A Monk string that represents the current transaction ID. |

### Returns

The string "SUCCESS" upon success, "FAILURE" upon failure.

### Throws

None.

### Location

librfc32.dll

### See also

[saptrfc-on-commit](#) on page 212

---

## saptrfc-on-confirm-tid

### Description

When received from the SAP R/3 system, the e\*Way deletes the TID from the TID database.

### Signature

```
(saptrfc-on-confirm-tid tid_string)
```

### Parameters

| Name       | Type   | Description   |
|------------|--------|---|
| tid_string | string | A Monk string that represents the current transaction ID. |

### Returns

The string "SUCCESS" upon success, "FAILURE" upon failure.

### Throws

None.

### Location

saptrfc-on- confirm.tid.monk

### See also

[saptrfc-tid-db-on-confirm](#) on page 221

[saptrfc-tid-file-on-confirm](#) on page 225

## saptrfc-on-rollback

### Description

When received from the SAP R/3 system, the e\*Way invokes [event-rollback-to-egate](#), which rolls back the Event from the IQ and also deletes the TID from the database.

### Signature

```
(saptrfc-on-rollback tid_string)
```

### Parameters

| Name       | Type   | Description   |
|------------|--------|---|
| tid_string | string | A Monk string that represents the current transaction ID. |

### Returns

The string "SUCCESS" upon success, "FAILURE" upon failure.

### Throws

None.

### Location

saptrfc-on- rollback.monk

### See also

[saptrfc-on-rollback-tid](#) on page 215

[saptrfc-tid-db-on-rollback](#) on page 221

[saptrfc-tid-file-on-rollback](#) on page 226

*Note:* This function is provided as a sample only.

## saptrfc-on-rollback-tid

### Description

When received from the SAP R/3 system, the e\*Way invokes [event-rollback-to-egate](#), which rolls back the Event from the IQ and also deletes the TID from the database.

### Signature

```
(saptrfc-on-rollback-tid tid_string)
```

### Parameters

| Name       | Type   | Description   |
|------------|--------|---|
| tid_string | string | A Monk string that represents the current transaction ID. |

### Returns

The string "SUCCESS" upon success, "FAILURE" upon failure.

### Throws

None.

### Location

librfc32.dll

### See also

[saptrfc-on-rollback](#) on page 214

[saptrfc-tid-db-on-rollback](#) on page 221

[saptrfc-tid-file-on-rollback](#) on page 226

## saptrfc-receive-idoc4

### Description

Receives an IDoc (version 4) from SAP via BAPI. It is used as part of a registered ABAP handler for the call to `IDOC_INBOUND_ASYNCHRONOUS`. On success, the value of the global Monk variable `saptrfc_global_idoc_string` is set to be the resulting IDoc string.

### Signature

```
(saptrfc-receive-idoc4)
```

### Parameters

None

### Returns

An IDoc string on success, Boolean false (#f) on failure.

### Throws

None.

### Location

saptrfc-receive-idoc4.monk

### See also

[sapbapi-struct-fetch](#) on page 186

[sapbapi-struct-resetall](#) on page 188

---

## saptrfc-send-idoc4

### Description

Sends an IDoc (version 4) string to SAP via BAPI.

### Signature

`(saptrfc-send-idoc4 tid idoc_string)`

### Parameters

| Name        | Type   | Description                 |
|-------------|--------|-----------------------------|
| tid         | string | The current transaction ID. |
| idoc_string | string | The IDoc being sent.        |

### Returns

Boolean true (**#t**) upon success, (**#f**) upon failure.

### Throws

None.

### Location

saptrfc-send-idoc4.monk

### See also

[sapbapi-struct-resetall](#) on page 188

[saptrfc-struct-call](#) on page 216

---

## saptrfc-struct-call

### Description

Calls a tRFC-enabled ABAP function on the SAP Application Server prior to sending an Event to SAP. Supplies the TID for the transaction and the input BAPI/tRFC Method substructure. The ABAP function then fetches the Event from the e\*Way.



## Signature

`(saptrfc-struct-call tid method_substruct_path)`

## Parameters

| Name                  | Type   | Description                           |
|-----------------------|--------|---------------------------------------|
| tid                   | string | The current transaction ID.           |
| method_substruct_path | path   | Path to the BAPI method substructure. |

## Returns

Boolean true (#t) upon success, (#f) upon failure.

## Throws

None.

## Location

`saptrfc-struct-call.monk`

## See also

[saptrfc-tab-countline](#) on page 243

[saptrfc-tab-getline](#) on page 245

[saptrfc-par-bcd->char](#) on page 235

## 8.6.4 SAP TID Database Management Functions

The SAP TID Database Management functions manipulate the TIDs in the TID database or file. The current set of SAP TID Database Management functions contains the following:

- [saptrfc-tid-db-bind](#) on page 218
- [saptrfc-tid-db-delete](#) on page 219
- [saptrfc-tid-db-insert](#) on page 219
- [saptrfc-tid-db-on-check](#) on page 220
- [saptrfc-tid-db-on-commit](#) on page 220
- [saptrfc-tid-db-on-confirm](#) on page 221
- [saptrfc-tid-db-on-rollback](#) on page 221
- [saptrfc-tid-db-reconnect](#) on page 222
- [saptrfc-tid-db-reserve](#) on page 222
- [saptrfc-tid-db-select](#) on page 223
- [saptrfc-tid-db-update](#) on page 223
- [saptrfc-tid-file-delete](#) on page 224
- [saptrfc-tid-file-on-check](#) on page 224
- [saptrfc-tid-file-on-commit](#) on page 225
- [saptrfc-tid-file-on-confirm](#) on page 225
- [saptrfc-tid-file-on-rollback](#) on page 226
- [saptrfc-tid-file-reserve](#) on page 226

---

### saptrfc-tid-db-bind

#### Description

Binds a TID to an EID file in a relational TID database.

#### Signature

(saptrfc-tid-db-bind)

#### Parameters

None.

#### Returns

None.

#### Throws

None.

Location

saptrfc-tid-db-bind.monk

---

## saptrfc-tid-db-delete

### Description

Deletes the specified TID from a relational TID database.

### Signature

(saptrfc-tid-db-delete *key*)

### Parameters

| Name | Type   | Description             |
|------|--------|-------------------------|
| key  | string | Key specifying the TID. |

### Returns

The status code `SAPRFC_OK` upon success, `SAPRFC_FAILURE` upon failure.

### Throws

None.

### Additional Information

To improve performance, no `commit` statement is issued; rather, `commit` statement is issued in [saptrfc-tid-db-reserve](#) once [saptrfc-client-createtid](#) returns successfully. If the system crashes before the `commit` statement is issued, you can delete the entry based on the timestamp in the `eid` field

### Location

saptrfc-tid-db-delete.monk

---

## saptrfc-tid-db-insert

### Description

Inserts a TID into a relational TID database.

### Signature

(saptrfc-tid-db-insert *param\_vector*)

### Parameters

| Name         | Type   | Description              |
|--------------|--------|--------------------------|
| param_vector | vector | Location for row insert. |

### Returns

The status code `SAPRFC_OK` upon success, `SAPRFC_FAILURE` upon failure.

### Throws

None.

### Location

saptrfc-tid-db-insert.monk

---

## saptrfc-tid-db-on-check

### Description

Marks a TID as **reserved** in a relational TID database, after the e\*Way has received [saptrfc-on-check-tid](#) from the SAP R/3 system.

### Signature

(saptrfc-tid-db-on-check *tid*)

### Parameters

| Name | Type   | Description         |
|------|--------|---------------------|
| tid  | string | The Transaction ID. |

### Returns

The status code `SAPRFC_OK` upon success, `SAPRFC_FAILURE` upon failure.

### Throws

None.

### Location

saptrfc-tid-db-on-check.monk

### See also

[saptrfc-tid-db-insert](#) on page 219

[saptrfc-tid-db-update](#) on page 223

---

## saptrfc-tid-db-on-commit

### Description

Marks the specified TID in a relational TID database as **processed**, after the e\*Way has received [saptrfc-on-commit-tid](#) from the SAP R/3 system.

### Signature

(saptrfc-tid-db-on-commit *tid*)

### Parameters

| Name | Type   | Description         |
|------|--------|---------------------|
| tid  | string | The Transaction ID. |

### Returns

The status code `SAPRFC_OK` upon success, `SAPRFC_FAILURE` upon failure.

### Throws

None.

### Additional Information

To improve performance, no `commit` statement is issued; rather, `commit` statement is issued in [saptrfc-tid-db-on-check](#).

### Location

`saptrfc-tid-db-on-commit.monk`

---

## saptrfc-tid-db-on-confirm

### Description

Deletes the specified EID/TID pair from a relational TID database, after the e\*Way has received [saptrfc-on-confirm-tid](#) from the SAP R/3 system.

### Signature

`(saptrfc-tid-db-on-confirm key)`

### Parameters

| Name | Type   | Description                  |
|------|--------|------------------------------|
| key  | string | Key specifying EID/TID pair. |

### Returns

The status code `SAPRFC_OK` upon success, `SAPRFC_FAILURE` upon failure.

### Throws

None.

### Location

`saptrfc-tid-db-on-confirm.monk`

---

## saptrfc-tid-db-on-rollback

### Description

Deletes the specified TID from a relational TID database, after the e\*Way has received [saptrfc-on-rollback-tid](#) from the SAP R/3 system.

### Signature

`(saptrfc-tid-db-on-rollback tid)`

### Parameters

| Name | Type   | Description         |
|------|--------|---------------------|
| tid  | string | The Transaction ID. |

### Returns

The status code `SAPRFC_OK` upon success, `SAPRFC_FAILURE` upon failure.

### Throws

None.

### Location

`saptrfc-tid-db-on-rollback.monk`

---

## saptrfc-tid-db-reconnect

### Description

Re-establishes connection to a relational TID database.

### Signature

```
(saptrfc-tid-db-reconnect)
```

### Parameters

None.

### Returns

None.

### Throws

None.

### Location

`saptrfc-tid-db-reconnect.monk`

### See also

[saptrfc-tid-db-bind](#) on page 218

---

## saptrfc-tid-db-reserve

### Description

Marks the TID associated with the specified EID as being **reserved**.

### Signature

```
(saptrfc-tid-db-reserve conn_handle eid)
```

### Parameters

| Name        | Type   | Description                               |
|-------------|--------|---|
| conn_handle | handle | A valid SAP RFC Connection Option handle. |
| eid         | string | The Event ID.                             |

### Returns

The status code `SAPRFC_OK` upon success, `SAPRFC_FAILURE` upon failure.

### Throws

None.

### Location

`saptrfc-tid-db-reserve.monk`

### See also

[saptrfc-client-createtid](#) on page 193

[saptrfc-tid-db-insert](#) on page 219

---

## saptrfc-tid-db-select

### Description

Selects a TID from the relational TID database row specified by the key.

### Signature

`(saptrfc-tid-db-select key)`

### Parameters

| Name | Type   | Description               |
|------|--------|---------------------------|
| key  | string | A key specifying the TID. |

### Returns

None.

### Throws

None.

### Location

`saptrfc-tid-db-select.monk`

---

## saptrfc-tid-db-update

### Description

Updates a TID in a relational TID database.

### Signature

`(saptrfc-tid-db-update param_vector)`

### Parameters

| Name         | Type   | Description              |
|--------------|--------|--------------------------|
| param_vector | vector | Location for row update. |

### Returns

The status code `SAPRFC_OK` upon success, `SAPRFC_FAILURE` upon failure.

### Throws

None.

### Location

`saptrfc-tid-db-update.monk`

## saptrfc-tid-file-delete

### Description

Deletes the specified TID from a TID file.

### Signature

`(saptrfc-tid-file-delete tid-fname eid-string)`

### Parameters

| Name       | Type   | Description    |
|------------|--------|----------------|
| tid-fname  | string | TID file name. |
| eid-string | string | Event ID.      |

### Returns

The status code `SAPRFC_OK` upon success, `SAPRFC_FAILURE` upon failure.

### Throws

None.

## saptrfc-tid-file-on-check

### Description

Marks a TID as reserved in a TID file, after the e\*Way has received [saptrfc-on-check-tid](#) from the SAP R/3 system.

### Signature

`(saptrfc-tid-file-on-check tid-fname tid-string)`



### Parameters

| Name       | Type   | Description     |
|------------|--------|-----------------|
| tid-fname  | string | TID file name.  |
| tid-string | string | Transaction ID. |

### Returns

The status code `SAPRFC_OK` upon success, `SAPRFC_FAILURE` upon failure.

### Throws

None.

## saptrfc-tid-file-on-commit

### Description

Marks the specified TID as **processed**, after the e\*Way has received [saptrfc-on-commit-tid](#) from the SAP R/3 system.

### Signature

```
(saptrfc-tid-file-on-commit tid-fname tid-string)
```

### Parameters

| Name       | Type   | Description     |
|------------|--------|-----------------|
| tid-fname  | string | TID file name.  |
| tid-string | string | Transaction ID. |

### Returns

The status code `SAPRFC_OK` upon success, `SAPRFC_FAILURE` upon failure.

### Throws

None.

## saptrfc-tid-file-on-confirm

### Description

Deletes the specified EID/TID pair from a TID file, after the e\*Way has received [saptrfc-on-confirm-tid](#) from the SAP R/3 system.

### Signature

```
(saptrfc-tid-file-on-confirm tid-fname tid-string)
```

### Parameters

| Name       | Type   | Description     |
|------------|--------|-----------------|
| tid-fname  | string | TID file name.  |
| tid-string | string | Transaction ID. |

### Returns

The status code `SAPRFC_OK` upon success, `SAPRFC_FAILURE` upon failure.

### Throws

None.

## saptrfc-tid-file-on-rollback

### Description

Deletes the specified TID from a TID file, after the e\*Way has received [saptrfc-on-rollback-tid](#) from the SAP R/3 system.

### Signature

```
(saptrfc-tid-file-on-rollback tid-fname tid-string)
```

### Parameters

| Name       | Type   | Description     |
|------------|--------|-----------------|
| tid-fname  | string | TID file name.  |
| tid-string | string | Transaction ID. |

### Returns

The status code `SAPRFC_OK` upon success, `SAPRFC_FAILURE` upon failure.

### Throws

None.

## saptrfc-tid-file-reserve

### Description

Marks the TID associated with the specified EID as being reserved.

### Signature

```
(saptrfc-tid-file-on-reserve conn-handle tid-fname tid-string)
```

### Parameters

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

| Name       | Type   | Description    |
|------------|--------|----------------|
| tid-fname  | string | TID file name. |
| eid-string | string | Event ID.      |

**Returns**

The status code **SAPRFC\_OK** upon success, **SAPRFC\_FAILURE** upon failure.

**Throws**

None.

## 8.6.5 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 a non-homogeneous structure. These functions are all located in `stc_monksap.dll`.

The current set of SAP Custom Structure functions contains the following:

[saprfc-struct-add-entry](#) on page 228

[saprfc-struct-create](#) on page 229

[saprfc-struct-handle?](#) on page 229

[saprfc-struct-install](#) on page 230

### saprfc-struct-add-entry

#### Description

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

#### Signature

*(saprfc-struct-add-entry struct-handle index-number name-string type-number length-number decimals-number)*

#### Parameters

| Name            | Type    | Description  |
|-----------------|---------|--|
| struct-handle   | handle  | Handle specifying the structure, as generated by <code>saprfc-struct-create</code> . |
| index-number    | integer | Number, beginning with 0, that specifies the element in the structure.               |
| name-string     | string  | Name of entry.   |
| type-number     | integer | Type of entry.   |
| length-number   | integer | Length of entry, in bytes.   |
| decimals-number | integer | Number of decimal digits, if <code>SAPRFC_TYPE_BCD</code> .                          |

#### Returns

A status code that evaluates to either of the following:

|                             |                                     |
|-----------------------------|-------------------------------------|
| <code>SAPRFC_OK</code>      | Operations have concluded normally. |
| <code>SAPRFC_FAILURE</code> | The function has failed.            |

#### Throws

None.

## 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-create

### Description

Creates an empty custom structure that can hold up to the specified number of entries.

### Signature

`(saprfc-struct-create number-entries)`

### Parameters

| Name           | Type    | Description   |
|----------------|---------|---|
| number-entries | integer | The number of elements to include in the structure. |

### Returns

An SAP RFC Structure handle.

### Throws

None.

### 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-handle?

### Description

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

### Signature

`(saprfc-struct-handle? struct-handle)`

### Parameters

| Name          | Type        | Description         |
|---------------|-------------|---------------------|
| struct-handle | Monk object | The object to test. |

### Returns

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

### Throws

None.

---

## saprfc-struct-install

### Description

Installs a custom structure on SAP.

### Signature

`(saprfc-struct-install struct-handle name-string)`

### Parameters

| Name          | Type   | Description   |
|---------------|--------|---|
| struct-handle | handle | Handle specifying the structure, as generated by <a href="#">saprfc-struct-create</a> . |
| name-string   | string | New name for structure.   |

### Returns

If successful, a number (`rfc_type_number`), representing the RFC type corresponding to the installed structure; if unsuccessful, 0.

### Throws

None.

## 8.6.6 SAP Parameter List Functions

The SAP Parameter List functions manipulate and check SAP parameter lists. The current set of SAP Parameter List functions contains the following:

- [saprfc-par-add](#) on page 231
- [saprfc-par-add-char](#) on page 232
- [saprfc-par-add-float](#) on page 233
- [saprfc-par-add-int](#) on page 234
- [saprfc-par-add-receiving](#) on page 234
- [saprfc-par-bcd->char](#) on page 235
- [saprfc-par-char->bcd](#) on page 236
- [saprfc-par-createlist](#) on page 236
- [saprfc-par-get](#) on page 237
- [saprfc-par-get-char](#) on page 237
- [saprfc-par-get-float](#) on page 238
- [saprfc-par-get-int](#) on page 238
- [saprfc-par-list?](#) on page 239
- [saprfc-par-pad](#) on page 239

### saprfc-par-add

#### Description

Adds a parameter of the specified type.

#### Signature

`(saprfc-par-add par-list string number string string number)`

#### Parameters

| Name       | Type            | Description                        |
|------------|-----------------|------------------------------------|
| par-list   | saprfc-par-list | A valid SAP parameter list.        |
| string (1) | string          | The type of the parameter.         |
| number (1) | integer         | The size of the parameter.         |
| string (2) | string          | The name of the parameter to add.  |
| string (3) | string          | The value of the parameter to add. |
| number (2) | integer         | The size of the parameter to add.  |

## 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.

## Notes

- 1 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*.
- 2 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.
- 3 For other parameter types, *size* is ignored.

## Location

stc\_monksap.dll

## saprfc-par-add-char

### Description

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

### Signature

```
(saprfc-par-add-char par-list string string)
```

### Parameters

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

### Description

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

### Signature

*(saprfc-par-add-float par-list string float)*

### Parameters

| Name     | Type   | Description                                |
|----------|--------|--|
| par-list | list   | A valid SAP parameter list.                |
| string   | string | The name of the parameter to add.          |
| float    | float  | The floating-point 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 par-list string number)
```

### Parameters

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

### Description

Adds a receiving parameter to the specified importing parameter list.

### Signature

```
(saprfc-par-add-receiving par-list par_type_number string  
par_size_number)
```

### Parameters

| Name            | Type            | Description                 |
|-----------------|-----------------|-----------------------------|
| par-list        | saprfc-par-list | A valid SAP parameter list. |
| par_type_number | integer         | The parameter type.         |

| Name            | Type    | Description                       |
|-----------------|---------|-----------------------------------|
| string          | string  | The name of the parameter to add. |
| par_size_number | 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.

### Notes

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

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

For other parameter types, *size* is ignored.

### Location

stc\_monksap.dll

## saprfc-par-bcd->char

### Description

Converts a binary coded decimal (BCD) value to a character value.

### Signature

`(saprfc-par-bcd->char bcd-string num-decimals)`

### Parameters

| Name         | Type    | Description                                 |
|--------------|---------|---|
| bcd-string   | string  | The binary-coded-decimal string to convert. |
| num-decimals | integer | The number of decimal digits.               |

### Returns

The character-string representation of the BCD value.

**Throws**

None.

**Location**

stc\_monksap.dll

## saprfc-par-char->bcd

**Description**

Converts a character string to a binary coded decimal (BCD) value.

**Signature**

`(saprfc-par-char->bcd char-string bcd-width [num-decimals])`

**Parameters**

| Name         | Type    | Description  |
|--------------|---------|--|
| char-string  | string  | Character string to convert.                       |
| bcd-width    | integer | The number of binary characters per decimal digit. |
| num-decimals | integer | The number of decimal digits (optional).           |

**Returns**

The binary-coded decimal (BCD) representation of the character-string value.

**Throws**

None.

**Location**

stc\_monksap.dll

## saprfc-par-createlist

**Description**

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

**Signature**

`(saprfc-par-createlist [number])`

**Parameters**

| Name   | Type    | Description  |
|--------|---------|--|
| number | integer | The initial size of the parameter list (optional). If no size is specified, a zero-length parameter list is created. |

### Returns

An empty SAP parameter list.

### Throws

None.

### Notes

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

### Location

stc\_monksap.dll

---

## saprfc-par-get

### Description

Obtains the value of the named parameter.

### Signature

*(saprfc-par-get par-list string)*

### Parameters

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

### Returns

The value of the specified parameter as a string.

### Throws

None.

### Location

stc\_monksap.dll

---

## saprfc-par-get-char

### Description

Obtains the string value of the named parameter.

### Signature

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

## Parameters

| Name     | Type   | Description                       |
|----------|--------|-----------------------------------|
| par-list | list   | A valid SAP parameter list.       |
| string   | 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-float

### Description

Obtains the floating-point value of the named parameter.

### Signature

`(saprfc-par-get-float par-list string)`

### Parameters

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

## Returns

A floating-point representation of the value of the named parameter.

## Throws

None.

## Location

stc\_monksap.dll

## saprfc-par-get-int

### Description

Obtains the integer value of the named parameter.

### Signature

`(saprfc-par-get-int par-list string)`

## Parameters

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

## Returns

An integer representation of the value of the named parameter.

## 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? par-list)
```

### Parameters

| Name     | Type | Description         |
|----------|------|---------------------|
| par-list | list | The object to test. |

### Returns

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

### Throws

None.

### Location

stc\_monksap.dll

## saprfc-par-pad

### Description

Pads an SAP field value (a Monk string) to the specified size.

### Signature

```
(saprfc-par-pad? par_type_number par_value_string par_size_number  
[decimals_number])
```

## Parameters

| Name             | Type    | Description   |
|------------------|---------|---|
| par_type_number  | integer | The parameter type.                                 |
| par_value_string | string  | A Monk string representing the initial field value. |
| par_size_number  | integer | Size desired after padding.                         |
| decimals_number  | number  | Number of decimal digits (optional).                |

## Returns

A Monk string representing the new field value.

## Throws

None.

## Notes

- 1 If the input type is `SAPRFC_TYPE_BYTE`, the input string is assumed to be the hex dump.
- 2 If the input type is `SAPRFC_TYPE_BCD` and the input value is an empty string (""), the input value is replace by "0.0" before padding.

## Location

saprfc-par-pad.dll

## See also

[saprfc-par-char->bcd](#) on page 236



## 8.6.7 SAP Table List Functions

The SAP Table List functions manipulate and check SAP Table Lists. The current set of SAP Table List functions contains the following:

- [saprfc-tab-appline](#) on page 241
- [saprfc-tab-applines](#) on page 242
- [saprfc-tab-clear](#) on page 242
- [saprfc-tab-countline](#) on page 243
- [saprfc-tab-create](#) on page 244
- [saprfc-tab-createlist](#) on page 244
- [saprfc-tab-getline](#) on page 245
- [saprfc-tab-getwidth](#) on page 245
- [saprfc-tab-list?](#) on page 246

---

### saprfc-tab-appline

#### Description

Appends a specified string to the specified table.

#### Signature

*(saprfc-tab-appline tab-list string string)*

#### Parameters

| Name       | Type   | Description  |
|------------|--------|--|
| tab-list   | list   | A valid SAP Table List name.   |
| string (1) | string | A valid SAP Table name.  |
| string (2) | 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 tab-list string string)`

**Parameters**

| Name       | Type   | Description   |
|------------|--------|---|
| tab-list   | list   | A valid SAP Table List name.  |
| string (1) | string | A valid SAP Table name.   |
| string (2) | 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-clear

**Description**

Clears the named table (i.e., deletes its content) so that the table can be reused.

**Signature**

`(saprfc-tab-clear tab-list string)`

### Parameters

| Name     | Type   | Description                  |
|----------|--------|------------------------------|
| tab-list | list   | A valid SAP Table List name. |
| string   | 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-countline

### Description

Counts the lines within the named table.

### Signature

*(saprfc-tab-countline tab-list string)*

### Parameters

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

### Returns

If successful, a number representing the number of lines within the named table; if unsuccessful, a negative number.

### Throws

None.

### Location

stc\_monksap.dll

## saprfc-tab-create

### Description

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

### Signature

`(saprfc-tab-create tab-list string number)`

### Parameters

| Name    | Type    | Description                       |
|---------|---------|-----------------------------------|
| tablist | list    | A valid SAP Table List name.      |
| string  | string  | A valid SAP Table name.           |
| number  | 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-createlist

### Description

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

### Signature

`(saprfc-tab-createlist number)`

### Parameters

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

### Returns

An empty SAP Table List.

### Throws

None.

### Notes

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

### Location

stc\_monksap.dll

## saprfc-tab-getline

### Description

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

### Signature

*(saprfc-tab-getline tab-list string number)*

### Parameters

| Name     | Type    | Description   |
|----------|---------|---|
| tab-list | list    | A valid SAP Table List name.  |
| string   | string  | A valid SAP Table name.   |
| number   | 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

If successful, the indexed row from the named table in the specified table list; if unsuccessful, a negative number.

### Throws

None.

### Location

stc\_monksap.dll

## saprfc-tab-getwidth

### Description

Obtains the width of the named table.

### Signature

`(saprfc-tab-getwidth tab-list string)`

### Parameters

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

### Returns

If successful, the width of the named table; if unsuccessful, a negative number.

### 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? tab-list)`

### Parameters

| Name     | Type | Description         |
|----------|------|---------------------|
| tab-list | list | The object to test. |

### Returns

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

### Throws

None.

### Location

stc\_monksap.dll

## 8.7 e\*Way Kernel Layer

### 8.7.1 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 basic Monk functions contains:

- [event-commit-to-egate](#) on page 247
- [event-rollback-to-egate](#) on page 248
- [event-send-to-egate](#) on page 248
- [event-send-to-egate-ignore-shutdown](#) on page 249
- [event-send-to-egate-no-commit](#) on page 249
- [get-logical-name](#) on page 250
- [insert-exchange-data-event](#) on page 250
- [send-external-up](#) on page 251
- [send-external-down](#) on page 251
- [shutdown-request](#) on page 252
- [start-schedule](#) on page 252
- [stop-schedule](#) on page 253
- [waiting-to-shutdown](#) on page 253

---

### 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 249

[event-send-to-egate-no-commit](#) on page 249

## 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 248

[event-send-to-egate-no-commit](#) on page 249

## 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 247

[event-rollback-to-egate](#) on page 248

[event-send-to-egate](#) on page 248

[event-send-to-egate-ignore-shutdown](#) on page 249

## 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 134

[Zero Wait Between Successful Exchanges](#) on page 134

---

## 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 [Exchange Data with External 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 functions
  - Z\_OUTBOUND\_BAPI\_INITIATE 69, 71, 103, 105
  - Z\_STCDGW\_SAPBAPI\_STORE\_INPUT 66, 96
- ABAP modules
  - Z\_BAPI\_OUTBOUND 68, 102
- Additional Path parameter 137
- APIs - see Monk functions
- Assigning ETDs to Event Types 40
- Autorun 21

## B

- BAPI Converter 34
- BAPI e\*Way Data Types 164–165
- BAPI e\*Way Utility Functions 166–168
- BAPI Structure Builder 34

## C

- Changing the User Name 79
- Collaboration 41, 83, 120
  - Rules 83, 84, 120
  - Service 83
- Collaboration Rules script 41
- configuration
  - Communication Setup 134–136
  - General Settings 132–133
  - Monk Configuration 137–145
  - SAP RFC Client Setup 146–152
  - SAP RFC Server Setup 153–158
  - SAP tRFC RDBMS Setup 159–161
- configuration parameters
  - Additional Path 137
  - Down Timeout 136
  - Exchange Data Interval 134
  - Exchange Data With External Function 140
  - External Connection Establishment Function 140
  - External Connection Shutdown Function 142
  - External Connection Verification Function 142
  - Forward External Errors 133
  - Journal File Name 132
  - Max Failed Messages 132

- Max Resends Per Message 132
- Monk Environment Initialization File 138
- Negative Acknowledgment Function 144
- Positive Acknowledgement Function 143
- Process Outgoing Message Function 139
- Resend Timeout 136
- Shutdown Command Notification Function 144
- Start Exchange Data Schedule 135
- Startup Function 139
- Stop Exchange Data Schedule 136
- Up Timeout 136
- Zero Wait Between Successful Exchanges 134
- configuration procedures 75
- conventions, writing 12
- Converter, BAPI 34
- Creating an e\*Way 73

## D

- Data Types 164–165
- data types
  - saprfc-conn-handle 164
  - saprfc-conn-opt 164
  - saprfc-par-list 165
  - saprfc-status 164
  - saprfc-tap-list 165
- destination templates
  - ZSTCBAPIOUTBOUND 104
- Down Timeout parameter 136

## E

- e\*Way
  - configuration 75
  - creating 73
  - Installation 21
  - Properties 74
  - Schedules 79
  - Startup Options 79
  - troubleshooting 83
- ETD Builder 34
- Event Type 40
- Event Type Definition (ETD) 34, 40
- event-commit-to-egate function 247
- event-rollback-to-egate function 248
- Events 119
- event-send-to-egate function 248
- event-send-to-egate-ignore-shutdown function 249
- event-send-to-egate-no-commit function 249
- ewtosapbapi-exit function 170
- ewtosapbapi-reconnect function 170
- ewtosapbapi-startup function 169
- ewtosaptrfc-exit Monk function 177
- ewtosaptrfc-reconnect Monk function 177

ewtosaptrfc-startup Monk function 176  
 Exchange Data Interval parameter 134  
 Exchange Data with External Function parameter 140  
 External Connection Establishment Function parameter 141  
 External Connection Shutdown Function parameter 142  
 External Connection Verification Function parameter 142

## F

File e\*Way 46, 49, 53, 56  
 Forward External Errors parameter 133  
 functions  
   BAPI Utility Functions 166–168  
   Generic 247–253  
   SAP BAPI Functions 182–191  
   SAP Custom Structure Functions 228–230  
   SAP Parameter List Functions 231–240  
   SAP RFC Client Functions 192–200  
   SAP RFC Server Functions 201–208  
   SAP Table List Functions 241–246  
   SAP TID Database Management Functions 218–227  
   SAP tRFC Server Functions 209–217  
   see also ABAP functions  
   see also Monk functions  
   Standard Function Templates 169–181

## G

Generic e\*Way Functions 247–253  
 get-logical-name function 250

## I

insert-exchange-data-event function 250  
 Installation procedure  
   e\*Way (UNIX) 25  
   e\*Way (Windows) 21  
   sample schema 27  
 InstallShield 21  
 Intelligent Queue (IQ) 42, 83

## J

Journal File Name parameter 132

## L

Load Path, Monk 137

logging options 81

## M

Max Failed Messages parameter 132  
 Max Resends Per Message parameter 132  
 message types  
   ZOUTTEST 104  
 monitoring thresholds 82  
 Monk Configuration  
   Load Path 137  
   Specifying File Names 137  
   Specifying Function Names 137  
   Specifying Multiple Directories 137  
 Monk Environment Initialization File parameter 138  
 Monk functions  
   event-commit-to-egate 247  
   event-rollback-to-egate 248  
   event-send-to-egate 248  
   event-send-to-egate-ignore-shutdown 249  
   event-send-to-egate-no-commit 249  
   ewtosapbapi-exit 170  
   ewtosapbapi-reconnect 170  
   ewtosapbapi-startup 169  
   ewtosaptrfc-exit 177  
   ewtosaptrfc-reconnect 177  
   ewtosaptrfc-startup 176  
   get-logical-name 250  
   insert-exchange-data-event 250  
   sapbapi-ack 172  
   sapbapi-client-connect 182  
   sapbapi-client-disconnect 183  
   sapbapi-client-openex 183  
   sapbapi-get-laststatus 166  
   sapbapi-init 166  
   sapbapi-nack 173  
   sapbapi-nak 173  
   sapbapi-return-empty-string 174  
   sapbapi-server-pollrequest-dispatch 184  
   sapbapi-server-register 184  
   sapbapi-server-unregister 185  
   sapbapi-shutdown 174  
   sapbapi-struct-call 185  
   sapbapi-struct-fetch 186  
   sapbapi-struct-installfunction 186  
   sapbapi-struct-raise 187  
   sapbapi-struct-resetall 188  
   sapbapi-struct-resetexception 188  
   sapbapi-struct-resetexport 189  
   sapbapi-struct-resetimport 189  
   sapbapi-struct-resettables 190  
   sapbapi-struct-send 190  
   sapbapitoew-exit 171  
   sapbapitoew-polling 171

sabbapitofrew-exit 172  
 sabbapitofrew-polling 172  
 sabbapi-verify-connect 174  
 saprfc-client-callreceive 192  
 saprfc-client-connect 193  
 saprfc-client-createtid 193  
 saprfc-client-disconnect 194  
 saprfc-client-indirectcall 194  
 saprfc-client-openex 195  
 saprfc-conn-abort 195  
 saprfc-conn-createopt 196  
 saprfc-conn-handle? 196  
 saprfc-conn-opt? 197  
 saprfc-conn-set-clientconnmode 197  
 saprfc-conn-set-clientconnopt-cpic 198  
 saprfc-conn-set-clientconnpar 199  
 saprfc-conn-set-clientconnopt-r3only 199  
 saprfc-conn-settrace 200  
 saprfc-getlasterror 167  
 saprfc-par-add 231  
 saprfc-par-add-char 232  
 saprfc-par-add-float 233  
 saprfc-par-add-int 234  
 saprfc-par-add-receiving 234  
 saprfc-par-bcd->char 235  
 saprfc-par-char->bcd 236  
 saprfc-par-createlist 236  
 saprfc-par-get 237  
 saprfc-par-get-char 237  
 saprfc-par-get-float 238  
 saprfc-par-get-int 238  
 saprfc-par-list? 239  
 saprfc-par-pad 239  
 saprfc-server-getcallbackfailuretid 201  
 saprfc-server-getcallbackfailuretype 202  
 saprfc-server-getinputdata 202  
 saprfc-server-installfunction 203  
 saprfc-server-installtransctrl 204  
 saprfc-server-resetcallbackfailure 205  
 saprfc-server-sendoutputdata 206  
 saprfc-server-shutdown 206  
 saprfc-server-startup 207  
 saprfc-server-waitanddispatch 208  
 saprfc-struct-add-entry 228  
 saprfc-struct-create 229  
 saprfc-struct-handle? 229  
 saprfc-struct-install 230  
 saprfc-tab-appline 241  
 saprfc-tab-applines 242  
 saprfc-tab-clear 242  
 saprfc-tab-countline 243  
 saprfc-tab-create 244  
 saprfc-tab-createlist 244  
 saprfc-tab-getline 245  
 saprfc-tab-getwidth 245  
 saprfc-tab-list? 246  
 saptrfc-ack 179  
 saptrfc-commit-tid 209  
 saptrfc-delete-tid 210  
 saptrfc-get-current-event-id 210  
 saptrfc-get-tid 211  
 saptrfc-init 180  
 saptrfc-nack 180  
 saptrfc-nak 179  
 saptrfc-on-check-tid 211  
 saptrfc-on-commit 212  
 saptrfc-on-commit-tid 213  
 saptrfc-on-confirm-tid 213  
 saptrfc-on-rollback 214  
 saptrfc-on-rollback-tid 215  
 saptrfc-receive-idoc4 215  
 saptrfc-return-empty-string 180  
 saptrfc-send-idoc4 216  
 saptrfc-shutdown 181  
 saptrfc-struct-call 216  
 saptrfc-tid-db-bind 218  
 saptrfc-tid-db-delete 219  
 saptrfc-tid-db-insert 219  
 saptrfc-tid-db-on-check 220  
 saptrfc-tid-db-on-commit 220  
 saptrfc-tid-db-on-confirm 221  
 saptrfc-tid-db-on-rollback 221  
 saptrfc-tid-db-reconnect 222  
 saptrfc-tid-db-reserve 222  
 saptrfc-tid-db-select 223  
 saptrfc-tid-db-update 223  
 saptrfc-tid-file-delete 224  
 saptrfc-tid-file-on-check 224  
 saptrfc-tid-file-on-commit 225  
 saptrfc-tid-file-on-confirm 225  
 saptrfc-tid-file-on-rollback 226  
 saptrfc-tid-file-reserve 226  
 saptrfctoew-exit 178  
 saptrfctoew-polling 177  
 saptrfctofrew-exit 179  
 saptrfctofrew-polling 178  
 saptrfc-verify-connect 181  
 send-external-down 251  
 send-external-up 251  
 shutdown-request 252  
 start-schedule 252  
 stop-schedule 253  
 waiting-to-shutdown 253

## N

Negative Acknowledgment Function parameter 144



## P

- Parameters
  - see also configuration parameters
- Participating Host 83
- Positive Acknowledgment Function parameter 143
- procedures
  - configuration 75
  - installation 21
- Process Outgoing Message Function parameter 139
- Properties, e\*Way 74

## Q

- Queues 42

## R

- Remote Function Call (RFC) 106
- Resend Timeout parameter 136
- RFC (Remote Function Call) 106

## S

- sample schema
  - descriptions 45
  - installation 27
- SAP BAPI Functions 182–191
- SAP Custom Structure Functions 228–230
- SAP Parameter List Functions 231–240
- SAP RFC Client Functions 192–200
- SAP RFC Server Functions 201–208
- SAP Table List Functions 241–246
- SAP TID Database Management Functions 218–227
- SAP tRFC Server Functions 209–217
- sapbapi-ack function 172
- sapbapi-client-connect function 182
- sapbapi-client-disconnect function 183
- sapbapi-client-openex function 183
- sapbapi-get-laststatus function 166
- sapbapi-init function 166
- sapbapi-nack function 173
- sapbapi-nak function 173
- sapbapi-return-empty-string function 174
- sapbapi-server-pollrequest-dispatch function 184
- sapbapi-server-register function 184
- sapbapi-server-unregister function 185
- sapbapi-shutdown function 174
- sapbapi-struct-call function 185
- sapbapi-struct-fetch function 186
- sapbapi-struct-installfunction function 186
- sapbapi-struct-raise function 187
- sapbapi-struct-resetall function 188
- sapbapi-struct-resetexception function 188
- sapbapi-struct-resetexport function 189
- sapbapi-struct-resetimport function 189
- sapbapi-struct-resettables function 190
- sapbapi-struct-send function 190
- sapbapitoew-exit function 171
- sapbapitoew-polling function 171
- sapbapitofrew-exit function 172
- sapbapitofrew-polling function 172
- sapbapi-verify-connect function 174
- saprffc-client-callreceive function 192
- saprffc-client-connect function 193
- saprffc-client-createtid function 193
- saprffc-client-disconnect function 194
- saprffc-client-indirectcall function 194
- saprffc-client-openex function 195
- saprffc-conn-abort function 195
- saprffc-conn-createopt function 196
- saprffc-conn-handle data type 164
- saprffc-conn-handle? function 196
- saprffc-conn-opt data type 164
- saprffc-conn-opt? function 197
- saprffc-conn-set-clientconnmode function 197
- saprffc-conn-set-clientconnopt-cpic function 198
- saprffc-conn-set-clientconnopt-r3only function 199
- saprffc-conn-set-clientconnpar function 199
- saprffc-conn-settrace function 200
- saprffc-getlasterror function 167
- saprffc-par-add function 231
- saprffc-par-add-char function 232
- saprffc-par-add-float function 233
- saprffc-par-add-int function 234
- saprffc-par-add-receiving function 234
- saprffc-par-bcd->char function 235
- saprffc-par-char->bcd function 236
- saprffc-par-createlist function 236
- saprffc-par-get function 237
- saprffc-par-get-char function 237
- saprffc-par-get-float function 238
- saprffc-par-get-int function 238
- saprffc-par-list data type 165
- saprffc-par-list? function 239
- saprffc-par-pad function 239
- saprffc-server-getcallbackfailureid function 201
- saprffc-server-getcallbackfailuretype function 202
- saprffc-server-getinputdata function 202
- saprffc-server-installfunction function 203
- saprffc-server-installtransctrl function 204
- saprffc-server-resetcallbackfailure function 205
- saprffc-server-sendoutputdata function 206
- saprffc-server-shutdown function 206
- saprffc-server-startup function 207
- saprffc-server-waitanddispatch function 208
- saprffc-status data type 164
- saprffc-struct-add-entry function 228

saprfc-struct-create function 229  
 saprfc-struct-handle? function 229  
 saprfc-struct-install function 230  
 saprfc-tab-appline function 241  
 saprfc-tab-applines function 242  
 saprfc-tab-clear function 242  
 saprfc-tab-countline function 243  
 saprfc-tab-create function 244  
 saprfc-tab-createlist function 244  
 saprfc-tab-getline function 245  
 saprfc-tab-getwidth function 245  
 saprfc-tab-list data type 165  
 saprfc-tab-list? function 246  
 saptrfc-ack Monk function 179  
 saptrfc-commit-tid function 209  
 saptrfc-delete-tid function 210  
 saptrfc-get-current-event-id function 210  
 saptrfc-get-tid function 211  
 saptrfc-init Monk function 180  
 saptrfc-nack Monk function 180  
 saptrfc-nak Monk function 179  
 saptrfc-on-check-tid function 211  
 saptrfc-on-commit function 212  
 saptrfc-on-commit-tid function 213  
 saptrfc-on-confirm-tid function 213  
 saptrfc-on-rollback function 214  
 saptrfc-on-rollback-tid function 215  
 saptrfc-receive-idoc4 function 215  
 saptrfc-return-empty-string Monk function 180  
 saptrfc-send-idoc4 function 216  
 saptrfc-shutdown Monk function 181  
 saptrfc-struct-call function 216  
 saptrfc-tid-db-bind function 218  
 saptrfc-tid-db-delete function 219  
 saptrfc-tid-db-insert function 219  
 saptrfc-tid-db-on-check function 220  
 saptrfc-tid-db-on-commit function 220  
 saptrfc-tid-db-on-confirm function 221  
 saptrfc-tid-db-on-rollback function 221  
 saptrfc-tid-db-reconnect function 222  
 saptrfc-tid-db-reserve function 222  
 saptrfc-tid-db-select function 223  
 saptrfc-tid-db-update function 223  
 saptrfc-tid-file-delete function 224  
 saptrfc-tid-file-on-check function 224  
 saptrfc-tid-file-on-commit function 225  
 saptrfc-tid-file-on-confirm function 225  
 saptrfc-tid-file-on-rollback function 226  
 saptrfc-tid-file-reserve function 226  
 saptrfctoew-exit Monk function 178  
 saptrfctoew-polling Monk function 177  
 saptrfctofrew-exit Monk function 179  
 saptrfctofrew-polling Monk function 178  
 saptrfc-verify-connect Monk function 181

Schedules 79  
 send-external-down function 251  
 send-external-up function 251  
 Setting Startup Options or Schedules 79  
 Shutdown Command Notification Function  
 parameter 144  
 shutdown-request function 252  
 Standard Function Templates 169–181  
 Start Exchange Data Schedule parameter 135  
 start-schedule function 252  
 Startup Function parameter 139  
 Startup Options 79  
 Stop Exchange Data Schedule parameter 136  
 stop-schedule function 253  
 Structure Builder 34

## T

tables  
     ZDGBAPI 103, 105  
 templates, destination  
     ZSTCBAPIOUTBOUND 104  
 transaction  
     SE09 60  
     SE11 62  
     SM59 42  
 Transactional ID (TID) 106  
 Transactional RFC (tRFC) 106  
 troubleshooting the e\*Way 83

## U

UNIX installation procedure 25  
 Up Timeout parameter 136  
 User name 79

## W

waiting-to-shutdown function 253  
 Windows installation procedure 21  
 writing conventions 12

## Z

Z\_BAPI\_OUTBOUND module 68, 102  
 Z\_OUTBOUND\_BAPI\_INITIATE function 69, 71,  
 103, 105  
 Z\_STCDGW\_SAPBAPI\_STORE\_INPUT function  
 66, 96  
 ZDGBAPI table 103, 105  
 Zero Wait Between Successful Exchanges parameter  
 134  
 ZOUTTEST message type 104

ZSTCBAPIOUTBOUND destination template **104**